File Number S360-29
Form Y33-9010-0

Program Logic

IBM System/360 Disk and Tape Operating Systems PL/I Subset Language Program Logic Manual

Volume 1 of 3

Program Numbers:
360N-PL-464 (DOS)
360N-PL-410 (TOS)

This publication provides information on the internal logic of the IBM System/360 DOS/TOS PL/I compiler. It is intended for use by persons involved in programming maintenance and by system programmers who wish to alter the program design. The information contained herein is not required for the use of, and the operation with, the PL/I compiler. Therefore, distribution of this publication is restricted to users with the aforementioned requirements.

The publication is divided into three volumes. Volume 1 contains the description of the compiler phases; volumes 2 and 3 contain the corresponding flow charts. The form numbers of the three volumes are:

Volume 1: Y33-9010
Volume 2: Y33-9011
Volume 3: Y33-9012

All information regarding the library subroutines of the DOS/TOS PL/I compiler is contained in the publication IBM System/360, Disk and Tape Operating Systems, PL/I Subset-Library Routines, Program Logic Manual, Form Y33-9013.

The reader must be thoroughly familiar with the IBM System/360 Disk and Tape Operating Systems and with the PL/I Subset language. A list of all publications that provide pertinent information is contained in the introduction to volume 1 of this PLM.

Restricted Distribution

First Edition

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 printer using a special print chain.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for readers' comments. If the form has been removed, comments may be addressed to IBM Laboratories, Programming Publications, 703 Boeblingen/Germany, P.O. Box 210.

IBM Confidential

CONTENTS

HOW TO USE THIS PUBLICATION. 7

ORGANIZATION OF THE PUBLICATION. 8

IBM SYSTEM/360 DOS/TOS PL/I PLM. . . 9
  Introduction. 9

LOGICAL PARTS OF THE COMPILER. 13

THE TEXT STRING DURING COMPILATION . . . 18

COMPILER INTERFACE 20
  Interface Routines used by Compiler Phases 26
  Logical IOCS for the TAPE Version . . 32
  Logical IOCS for the DISK Versions. . 33

PHASES PL/IA00, A00D, A10 (INITIALIZATION) -- AR. 34

PHASE PL/IA25 (REPLACEMENT OF KEYWORDS) -- BA 36
  Description of Routines 38

PHASE PL/IA30 (REPLACEMENT OF IDENTIFIERS) -- CA. 40
  Description of Routines 40

PHASE PL/IA35 (PICTURES) -- CZ 42
  Output Formats. 42
  Elements of Picture Strings 47
  Examples. 48
  Description of Routines 50

PHASE PL/IA45 (CHARACTER STRINGS) -- EM. 54
  Description of Routines 55

PHASE PL/IA50 (BLOCK STRUCTURE) -- FM. . 57
  Description of Routines 57

PHASES PL/IA60, A65 (SYNTAX CHECK I AND II) -- GL, GW 62
  Functional Description. 63
  Description of Routines 64

PHASE PL/IB10 (DECLARATION SCAN I) -- HM. 67
  Description of Routines 67

PHASE PL/IB15 (DECLARATION SCAN II) -- IM. 70
  Description of Routines 70

PHASE PL/IB20 (SYMBOL TABLE CONSTRUCTION I) -- KA 73
  Description of Routines 74

PHASE PL/IB25 (FILE DECLARATIONS) -- L$. 79

PHASE PL/IB30 (SYMBOL TABLE CONSTRUCTION II) -- MA. 84
  Description of Routines 84

PHASE PL/IB40 (STRUCTURE MAPPING) -- MZ. 86
  Description of Routines 89

PHASE PL/IB70 (CONTEXTUAL DECLARATIONS) -- OA 92
  Communication with Other Phases . . . 92
  Description of Routines 94

PHASE PL/IB75 (EXTERNAL ENTRY NAMES FOR IMPROPERLY GENERATED BUILT-IN FUNCTIONS) -- OR. 98
  Description of Routines 98

PHASE PL/IB80 (IMPLICIT DECLARATIONS) -- PA 100
  Description of Routines 101
  Subroutines 103

PHASE PL/IB90 (PRESTATEMENT GENERATION) -- QA 107
  Description of Routines 107

PHASE PL/IB92 (ATTRIBUTE TABLE COMPRESSION) -- RA. 109
  Description of Routines 109

PHASE PL/IB95 (ARRAY TABLE CONSTRUCTION) -- SA 111
  Description of Routines 111

PHASE PL/IB97 (EXTERNAL NAME TABLE CONSTRUCTION) -- SM 113
  Description of Routines 113
  Subroutines 114

PHASE PL/IC00 (SYMBOL TABLE LISTING) -- TM 116
  Description of Routines 116

PHASE PL/IC25 (IF STATEMENT) -- TZ . . .118
  Statements and Macros put out by C25. 118
  Description of Routines 120

PHASE PL/IC30 (PROCESSING CONSTANTS I) -- WA 123
  Description of Routines 124

PHASE PL/IC35 (BLOCK SORTING) -- VA, VB. 127
  Description of Routines 127

PL/IC50 (I/O SCAN I) -- XB, XC 131
  Description of Routines 132

PHASE PL/IC55 (I/O SCAN II) -- YA. . . .138
  Description of Routines 140

PHASE PL/IC60 (I/O SCAN III) -- YS, YT .144
Description of Routines 145

PHASE PL/IC65 (I/O SCAN IV) -- $A, $B. .148
Description of Routines 148

PHASE PL/IC85 (DO STATEMENT I) -- $O . .151
Description of Routines 153

PHASE PL/IC86 (DO STATEMENT II) -- Z9. .155
Functional Description. 157
Description of Routines 158

PHASE PL/IC95 (NEW INTERFACE) -- AV. . .161
New Interface 161

GENERAL DESCRIPTION OF PL/I PHASES D00 - D11 163
Input 163
Common Service Routines 172
Buffer Concept and Phase Layout . . .174

PHASE PL/ID00 (STATEMENT DECOMPOSITION) -- AZ, BA. 175
Description of Routines 175

PHASE PL/ID05 (CONVERSION, PRECISION, STORAGE TYPE) -- DC 180
Description of Routines 180

PHASE PL/ID10 (MACRO GENERATION I) -- EP. 188
Part 1 of D10 -- EP 188
Part 2 of D10 (PREMAC) -- GF. 189
Part 3 of D10 194

PHASE PL/D11 (MACRO GENERATION II) -- HM. 197
Description of Routines 198

PHASE PL/ID15 (EVALUATION OF SUBSCRIPTS) -- JA 205
Description of Routines 209

PHASE PL/ID17 (LIBRARY CALLS FOR BUILT-IN FUNCTIONS I) -- LA 213
Built-in Functions Processed. 214
In-Line Functions 219
Description of Routines 220

PHASE PL/ID20 (BUILT-IN FUNCTIONS II) -- OK 225
Description of Routines 226

PHASE PL/ID40 (ON, SIGNAL, REVERT, AND STOP) -- OQ, OU 230
Functional Description. 230
Description of Routines 234

PHASE PL/I D70 (PROCESSING CONSTANTS II) -- PK 238

PHASE PL/ID75 (GENERATION OF I/O MACROS I) -- QP 246

PHASE PL/ID80 (GENERATION OF I/O MACROS II) -- RF. 249

PHASE PL/IE25 (ERROR DIAGNOSTIC) -- SA .251
Description of Routines 252

GENERAL DESCRIPTION OF THE GENERATOR PHASES (PL/IE50, PL/IE60, PL/IE61) . . .254
Model-Instruction Dictionary. 255
Format of the Instructions. 258
The Macros and the Generated Code . .259

PHASE PL/IE50 (CODE GENERATION I) -- TA. 285
Description of Routines 285

PHASE PL/IE60/61 (CODE GENERATION II) -- UA, UB 286
Description of Routines 286

PL/IF25 (SORTING CONSTANTS AND VARIABLES) -- W9. 288
Description of Routines 289

PHASE PL/IF35 (OPTIMIZATION OF CONSTANTS) -- YA. 293
Storage Areas 293
Description of Routines 294

PHASE PL/IF75 (STORAGE ALLOCATION) -- Y0. 296
Description of Routines 297

PHASE PL/IF90 (BUILDING OF OFFSET TABLE) -- AA. 300
Functional Description. 300
Description of Routines 301

GENERAL DESCRIPTION OF PHASES F95 - G55 305
Phases F95 - G15. 305
Phases G20 - G55. 305
Phases G20 - G55. 306

PHASE PL/IF95 (HANDLING OF OFFSETS) -- AN. 307
Functional Description. 308
Description of Routines 310

PHASE PL/IG00 (LABEL HANDLING) -- BF . .313
Description of Routines 315

PHASE PL/IG01 (LABEL OFFSETS) -- CA. . .319

PHASE PL/IG15 (FINAL OFFSET PREPARATION) -- CH. 320

PHASES PL/IG17, B, D, E, R, S (FILE GENERATION) -- DJ 328

PL/IG17 (CARD, PRINT, UNBUFFERED FILES) 329
Card Files. 329
Print Files 330
Unbuffered Tape Files 330
Unbuffered Disk Files 331

PL/IG17B (Buffered Tape Files) 331

PL/IG17D, E (Buffered Consecutive Disk Files 333

PL/IG17R, S (Regional Disk Files) 334

PHASE PL/IG20 (FILE MODULE) -- DP. . . .335

PHASE PL/IG25 (GENERATION OF ESD CARDS) -- EH. 339
Description of Routines 340

PHASE PL/IG30 (GENERATION OF TXT AND RLD CARDS) -- FH. 342

PHASE PL/IG31 (FINAL DIAGNOSTIC) -- GA .348

PHASE PL/IG40 (LISTING OF COMPILER OUTPUT) -- GF 349

PHASE PL/IG55 (FINAL OUTPUT) -- HA . . .355

APPENDIX A. SYNTAX NOTATION OF PL/I INPUT STREAM. 358

APPENDIX B. SYNTAX NOTATION OF PL/I OUTPUT STREAM 366

APPENDIX C. LIBRARY ROUTINES 374

APPENDIX D. DTF TABLES 376

INDEX. 419

HOW TO USE THIS PUBLICATION

In the majority of cases, a PLM is used to analyze a specific error that caused a compile-time dump or an erroneous result. The following is therefore intended to assist the programmer in obtaining from this dump all information he requires to locate the specific section of the PLM in which he is interested. Using the descriptive text, the flow charts, and the program listing, he can then find out what error caused the compiler to produce the dump or the erroneous result so that he may take the appropriate corrective action.

Conventions on Register Usage

1. If an interface routine is called, registers 0 and 1 serve as parameter registers (refer to the description of the individual interface routines in the section Compiler Interface).
2. Register 9 serves as input area register for IJSYSIN during phases A00, A00D, and A25.
3. Register 10 serves as output area register for IJSYSPH during phases A00, A00D, and G55 (for punching).
4. Register 11 serves as output area register for IJSYSLS during all listing phases.
5. Register 12 is used for any reference to the communication region.
6. Register 13 is not used by the phases, but as save area register for LIOCS.
7. Register 14 serves as return register in case of subroutine calls.
8. Register 15 is used both as base register in the phases and as entry point register when calling a subroutine.

Entry Points in the Communication Region

Register 12 points to the beginning of the communication region. The absolute address of entry points in the communication region can be found in the Linkage Editor storage map.

The following entry points in the communication region are of interest in case of a compile-time dump:

KSAVE1: This area contains return addresses of the last interface call in the following order:

register 14: points to the routine that was last active.

register 15
register 0
register 1
register 2z

K5PH: This 8-byte area normally contains the name of the phase currently in storage. The phase name is stored as follows:

```
P  L  /  I  x  x  x  b
D7 D3 61 C9 yy yy yy 40
```

The last four bytes contain the actual phase name, e.g., E25 or, in hexadecimal notation, C5F2F540. Phases D00, D05, D10, and D11 form an exception. For these phases, the name can be found at X'108' (register 12).
It should be noted that the actual phase currently in storage may be either C95 or D11 if K5PH contains the name C95. To determine which phase is actually in storage, locate the start address of the phase and compare it with the listing.

KTETA: If the contents of KTETA are less than those of KTETA+4, SYS002 is currently used as text input medium and SYS003 as output medium.

IJKMTS: Contains the start address of the table space.

IJKMBL: Contains the buffer length for text I/O.

IJKMBS: Contains the start address of the buffer area.

IJXA04: Is the address of the table directory (TABTAB).

For detailed information on the format of the communication region refer to the section Compiler Interface.

Note: The interface routines are used by all phases. Therefore, they are not described in each phase, but in the separate section Compiler Interface. For a list of all interface routines refer to Figures 7 and 8 of that section. The names of interface routines start either with IJK or Z.

ORGANIZATION OF THE PUBLICATION

Due to its size, this book has been divided into three volumes. For the reader's convenience, volume 1 contains all of the descriptive text, whereas volumes 2 and 3 contain the flow charts. Thus, the text and the corresponding flow chart(s) may be used synoptically. The form numbers of the three volumes are as follows:

Volume 1: Y33-9010
Volume 2: Y33-9011
Volume 3: Y33-9012

The individual phases are presented in the order of their appearance within the compiler. The compiler interface (which, most probably, will have to be looked up quite frequently in many of the phases) is described in a separate section to make it stand out. The appendices provide reference information taken out of the corresponding phase description to improve the readability of the text and to make the information easily accessible.

The heading of each phase description gives the phase name, the function (in parentheses), and -- separated by two dashes -- the identification of the corresponding general flow chart, e.g.,

PHASE PL/IA45 (CHARACTER STRINGS) -- EM

In the description of the individual routines of a phase, the flow chart for the routine, if any, is indicated by the flow chart identification, separated from the routine name by two dashes, rom the routine name by two dashes, e.g.,

INIT1 -- XY

The use of the individual flow chart symbols is explained in detail at the beginning of each of the flow chart volumes.

Figures are numbered sequentially, starting at 1 in each section.

Related Publications

PL/I Subset Language Specifications, Form C28-6809

IBM System/360, Disk and Tape Operating Systems, PL/I Programmers Guide, Form C24-9005

IBM System/360, Disk and Tape Operating Systems, PL/I Subset-Library Routines, Program Logic Manual, Form Y33-9013

IBM System/360, Disk Operating System, PL/I DASD Macros, Form C24-5059

IBM System/360, Disk Operating System, System Control and System Service Programs, Form C24-5036

IBM System/360, Tape Operating System, System Control and System Service Programs, Form C24-5034

IBM System/360, Disk Operating System, Supervisor and Input/Output Macros, Form C24-5037

IBM System/360, Tape Operating System, Supervisor and Input/Output Macros, Form C24-5035

IBM System/360, Disk Operating System, System Generation and Maintenance, Form C24-5033

IBM System/360, Tape Operating System, System Generation and Maintenance, Form C24-5015

IBM System/360, Disk Operating System, Performance Estimates, Form C24-5032

IBM System/360, Tape Operating System, Performance Estimates, Form C24-5020

IBM System/360, Disk Operating System, Operating Guide, Form C24-5022

IBM System/360, Tape Operating System, Operating Guide, Form C24-5021

INTRODUCTION

The DOS/TOS PL/I compiler is designed to compile source programs written in the PL/I Subset language. A set of library subroutines that are part of the component is used as control routine for the execution of PL/I programs in the DOS/TOS environment.

The language implemented is the language described in the SRL publication PL/I Subset Language Specifications, Form C28-6809. Further restrictions and implementation-defined features are listed in the SRL publication IBM System/360 Disk and Tape Operating Systems, PL/I Programmer's Guide, Form C24-9005. This publication also describes the Disk and Tape Operating Systems as the environment of the PL/I compiler.

The DOS/TOS PL/I compiler is a multiphase, multi-pass compiler. Input to the compiler is read from the logical unit SYSIPT. The compiler output is produced on the logical unit SYSLST. Object programs are produced on SYSPCH or SYSLNK. Three work files are used by the compiler. All three work files may be either on tape (DOS and TOS) or on disk (DOS only). On DOS, a second compiler version that allows SYSIPT, SYSLST, and SYSPCH to be 2311 DASD extents is available. The version used is determined at system generation time. The compiler version that allows system logical units to be DASD extents requires 12K of main storage. Switching between tape and disk work files on DOS is automatic at open time.

Parts of the first phase (PL/I) remain in main storage as a control routine during execution of the other phases of the compiler. Their function is the execution of I/O operations for work files and interphase communication. A special smaller control routine is used during execution of the extremely long phases D00 to D10 which do not use the table file SYS001.

The PL/I library is a set of relocatable routines and transient core-image library routines. The library is used at object time for:

1. Monitoring object program execution,
2. Performing input/output operations,
3. Performing object time conversions, and
4. Built-in functions.

The relocatable library routines are cataloged into the relocatable library and loaded by the autolink feature. Six library routines are cataloged into the core-image library. These routines are loaded at execution time into a transient area of the PL/I library to perform functions that are not frequently used, e.g., opening of files, etc. Their phase names start with $ to ensure storage in the privileged region of the core-image library. An additional routine ($$BPLOSE) is to be executed in the systems logical transient area when closing PL/I files.

For detailed information on the library subroutines refer to the library subroutines PLM named on the cover page.

The storage used by the compiler is divided into the following 4 parts (see Figure 1):

1. Control routine
2. Compiler phases
3. Table area
4. Buffer area

```
    0 ┌──────────────────────────────────┐
      │                                  │
      │                                  │
      │ Control Routine                  │
      │                                  │
      │                                  │
      ├──────────────────────────────────┤
      │                                  │
      │                                  │
      │ Compiler Phases                  │
      │                                  │
      │                                  │
6.75K ├──────────────────────────────────┤
      │                                  │
      │ Table Area 6*256 bytes           │
      │                                  │
8.25K ├──────────────────────────────────┤
      │                                  │
      │ Buffer Area 7*256 bytes          │
      │                                  │
  10K └──────────────────────────────────┘
```

Figure 1. Storage Used by PL/I Compiler

The last part of the control routine area is the table directory. Part of this area can be overlaid by the first phases that use only a few of the tables. The table area is used for processing by compiler phases that have no table handling. Some phases use less than 7 buffers and can therefore use part of the buffer area.

Phase Name	PL/I Module Name	F u n c t i o n	Phase Length	Phase End	Maint. Area	Tab. Use	Buff. No.
PL/I	IJXA00	DOS control routine and initialization	5016	5016	1896	N	0
	IJXA00D	DOS control routine and initialization with system files on disk	5426	5426	3534	N	0
	IJXA00	TOS control routine and initialization	4224	4224	2688	N	0
PL/IA10	IJXA10	Overlay for tape work files (DOS only)	898	898	NA	N	0
PL/IA25	IJXA25	Elimination of blanks and comments, replacement of keywords	6004	9118	1280	N	4[4]
PL/IA30	IJXA30	Replacement of identifiers	3124	6238	674	T	7
PL/IA35	IJXA35	Pictures	5700	8256	704	N	5
PL/IA45	IJXA45	Character-string replacement	3234	5950	962	T	7
PL/IA50	IJXA50	Scan block structure	3012	5728	1184	N	7
PL/IA60	IJXA60	Syntax 1	5120	7676	772	N	7
PL/IA65	IJXA65	Syntax 2	4748	7304	1144	N	7
PL/IB10	IJXB10	Declaration scan 1	2264	4980	1932	T	7
PL/IB15	IJXB15	Declaration scan 2	3528	6244	668	T	7
PL/IB20	IJXB20	Symbol table construction 1	3704	6420	492	T	7
PL/IB25	IJXB25	File declarations	3804	6484	428	T	7
PL/IB30	IJXB30	Symbol table construction 2 (diagnostic)	2592	5308	1604	T	7
PL/IB40	IJXB40	Symbol table construction 3 (structures, etc.)	2604	5320	1592	T	7
PL/IB70	IJXB70	Symbol table construction 4 (contextual declarations)	3660	6376	536	T	7
PL/IB75	IJXB75	BUILTIN versus contextual declarations	1568	4284	2628	T	7
PL/IB80	IJXB80	Symbol table construction 5 (implicit declarations)	3492	6208	704	T	7
PL/IB90	IJXB90	Prestatement generation 1	3072	5788	1124	T	7
PL/IB92	IJXB92	Prestatement generation 2	3196	5912	1000	T	7
PL/IB95	IJXB95	Array table construction	1736	4416	2304	T	7
PL/IB97	IJXB97	External name table construction	2736	5452	1460	T	7
PL/IC00	IJXC00	Symbol table listing	3230	5946	966	T	7
PL/IC25	IJXC25	IF scan	2956	5672	1280	T	7
PL/IC30	IJXC30	Constant processing 1	3020	5736	1176	T	7
PL/IC35	IJXC35	Block sorting	3084	5764	956	T	7
PL/IC50	IJXC50	I/O scan 1	3558	6274	638	T	7
PL/IC55	IJXC55	I/O scan 2	3684	6400	512	T	7
PL/IC60	IJXC60	I/O scan 3	3804	6520	392	T	7
PL/IC65	IJXC65	I/O scan 4	3748	6464	448	T	7
PL/IC85	IJXC85	DO scan 1	3276	5956	956	T	7
PL/IC86	IJXC86	DO scan 2	3676	6356	556	T	7
PL/IC95	IJXC95	Switch to small control routine	1032	3748	3164	T	7
PL/ID00	IJXD00	Statement decomposition	5472	6976	2368[1]	N	3.5
PL/ID05	IJXD05	Conversion, precision, storage types	7400	8888	712[1]	N	2.5
PL/ID10	IJXD10	Macro generation 1	6856	8360	984[1]	N	3.5
PL/ID11	IJXD11	Macro generation 2	4226	7026	2318	N	3.5
PL/ID15	IJXD15	Evaluation of subscripts	3784	5500	2948	N	7
PL/ID17	IJXD17	Generation of linkage to library	5082	7798	1162	N	5
PL/ID20	IJXD20	Special built-in functions	5184	7900	1060	N	5
PL/ID40	IJXD40	ON generation	3940	6656	2304[1]	N	5
PL/ID70	IJXD70	Constant processing 2 (conversion)	4344	7060	620	T/2	7
PL/ID75	IJXD75	I/O macro generation 1	3716	6432	2016	N	7
PL/ID80	IJXD80	I/O macro generation 2	2632	5348	3100	N	7

Figure 2. List of Phases (Part 1 of 2)

PL/IE25	IJXE25	Main Diagnostic	3762	6478	434	T	7
PL/IE25A	IJXE26	Messages	1200	6470	-	-	-
PL/IE25B	IJXE27	Messages	1200	6470	-	-	-
PL/IE25C	IJXE28	Messages	1200	6470	-	-	-
PL/IE25D	IJXE29	Messages	1200	6470	-	-	-
PL/IE25E	IJXE30	Messages	1200	6470	-	-	-
PL/IE25F	IJXE31	Messages	1200	6470	-	-	-
PL/IE25G	IJXE32	Messages	1200	6470	-	-	-
PL/IE25H	IJXE33	Messages	1200	6470	-	-	-
PL/IE25I	IJXE34	Messages	1200	6470	-	-	-
PL/IE25J	IJXE35	Messages	960	6230	240	-	-
PL/IE50	IJXE50	Code generation 1	5308	8024	424	N	7
PL/IE60	IJXE60	Code generation 2	4688	7404	1044[2]	N	7
PL/IE60A	IJXE61	Macro library (overlay)	2933	7217	1231[2]	-	-
PL/IF25	IJXF25	Sorting of variables and constants	3860	6576	592	T[3]	6
PL/IF35	IJXF35	Optimization of constants	3032	5748	1164	T	7
PL/IF75	IJXF75	Storage allocation	2708	5424	1488	T	7
PL/IF90	IJXF90	Construction of offset table	2214	4930	1982	T	7
PL/IF95	IJXF95	Code generation for offset > 4K	2360	5076	1836	T	7
PL/IG00	IJXG00	GOTO optimization	3914	6630	464	T	7
PL/IG01	IJXG01	Insertion of label offsets	2216	4896	2766	T	7
PL/IG15	IJXG15	Final offset preparation	3488	5204	420	T	7
PL/IG17	IJXG17	File generation 1	5060	7776	1952	N	2
PL/IG17B	IJXG17B	File generation 2 (DTFMT)	4902	7618	2210	N	2
PL/IG17D	IJXG17D	File generation 3 (DTFSD)	4854	7570	2258	N	2
PL/IG17E	IJXG17E	File generation 4 (DTFSD)	3262	5978	3750	N	2
PL/IG17R	IJXG17R	File generation 5 (REGIONAL(1))	4770	7486	2234	N	2
PL/IG17S	IJXG17S	File generation 6 (REGIONAL(3))	5446	8162	1566	N	2
PL/IG20	IJXG20	Produce file module, rearrange SYS001	1912	4628	2284	T	7
PL/IG25	IJXG25	Generate ESD	3148	5864	1048	T	7
PL/IG30	IJXG30	Generate TXT, RLD, END	2624	5340	1572	T	7
PL/IG31	IJXG31	Final diagnostic	2838	5454	4170	N	1
PL/IG40	IJXG40	Object code listing	4420	7136	1824	N	5
PL/IG55	IJXG55	Final output	4402	7118	1842	N	3

[1] Includes dynamic stack.
[2] Includes 10-byte parameter from PL/IE25.
[3] Shifted up one buffer.
[4] 2 buffers are used by program at the beginning of the phase.

Figure 2. List of Phases (Part 2 of 2)

Figure 2 lists all phases including their function, length, and maintenance area. The entry in the column Tab. Use specifies whether the table area is used for table handling (T) or for other purposes (N). The number of 256-byte blocks used as buffers is given in the last column.

The starting point of the compiler is assumed to be zero in this list. The DOS version not supporting system files on disk is assumed in this table. The maintenance area includes the area required for the control routine.

If more than 10K are available to the compiler, the remaining storage is used to increase the table area (maximum used is 64K) and the buffer length (maximum 1536 bytes per buffer). This increases the compiler performance considerably.

The I/O flow during compilation is shown in Figure 3.

PL/I object programs including library subroutines, IOCS modules, and static storage form one or more phases. Automatic storage is allocated beginning at the end of the longest problem program phase up to the end of storage available to background programs. Start and end addresses of automatic storage are taken from the DOS/TOS communication region and are handled by a PL/I library subroutine.

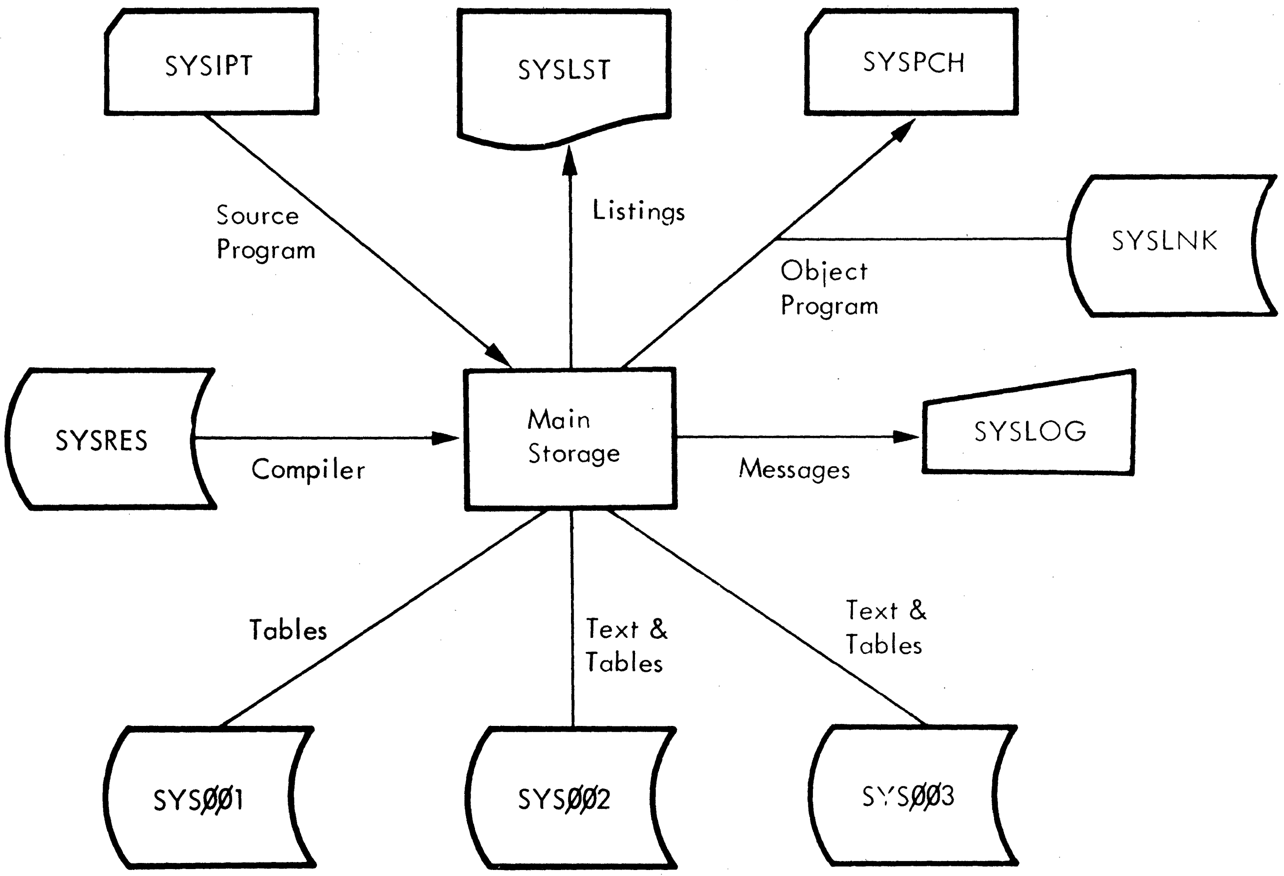


Figure 3. I/O Flow During Compilation

The compiler is built up of about 70 phases, which may be grouped into five logical parts referred to as packages.

Package 1 (Phases A25 - C00)

In this package, the programmer-written source text is transformed into a text string, the format of which is oriented to the logical structure of a PL/I program. This means that language elements such as statements, prefixes, identifiers, delimiters, etc. are translated into a representation that permits the relatively simple recognition of that association.

Redundant information (blanks and comments) is deleted from the text string. The non-executable DECLARE statements are also deleted. The information contained therein is transferred to the corresponding identifiers in the text string where they occur.

The program string is syntactically checked and diagnostie information for errors, if any, is inserted.

The syntax of the PICTURE attribute is checked and the information required either for further processing or during execution at object time is provided.

A symbol table is constructed. It is listed if listing is specified in the OPTION job control statement.

The compiler also constructs tables for character constants, names, files, external names, and arrays.

Package 2 (Phases C25 - C95)

As the result of the processing in package 1, the source text is now a statement-oriented text string.

This package of phases processes the IF, I/O, and DO statements. Processing of this group of statements requires special phases since these statements all possibly contain expressions, the handling of which involves a considerable programming effort. The above statements are scanned and the expressions prepared for further processing in package 3.

The IF statements are expanded into simple statements that can be processed in package 3. Branch and label-definition macros are generated.

The I/O statements are semantically checked, and DO loops are generated for repetitive specifications. For all I/O statements containing the FILE option, the identity of the information given in the file declaration (from the FILE table) and that in the FILE option is checked. The I/O statements are then prepared to be sequentially processed in package 3.

The DO statements are decomposed into simple statements. Branch and label-definition macros similar to those in the IF phase are generated and inserted in the program string.

In addition, blocks are ordered sequentially in this package.

Package 3 (Phases D00 - D80)

All executable statements are processed in this package. The statements that were preprocessed in package 2 are now finally processed. The result of this processing is a text string consisting of elements that do not refer to statements but to separate operations. The text elements that represent these operations are called macros.

Array and structure assignments are decomposed.

Expressions are reordered in reverse Polish notation. The necessity for data type conversions is determined and the conversions are prepared by macros. In addition, macros are generated to give each variable the storage type required for particular operations, e.g., register, working storage, etc. Registers are allocated for operands that are to be registers. The appropriate library call macro is generated for built-in functions implemented by library routines.

Subscripts are evaluated. If the subscripts are constants, they are evaluated at compile time. Otherwise, the appropriate macro is generated for use at object time.

ON entries that contain the ON and prefix information are generated to be included in static storage.

Conversion of constants is performed at compile time.

Package 4 (Phases E25-E61)

If errors are detected in the program string, the corresponding diagnostic messages are printed, if specified.

Assembler-type code is generated from the macros. The selection of the macros depends on the type of the macro, the storage class of the operands of the macro, and further information contained in the macros.

A model instruction dictionary is used to furnish additional information independent of the information contained in the macro.

Indirect addressing is assigned for operands that have the attributes external, parameter, or controlled.

Package 5 (Phases F25-G55)

This package is referred to as the assembler of the compiler because its functions are similar to those of an assembler.

Storage is allocated for variables and constants.

Constants are optimized.

Final machine instructions are generated by changing the format of the assembler instructions and by replacing the operands of the assembler instructions by base register and displacement.

Code for branches and addressing beyond the scope of 4K-blocks is generated.

The required tables are generated for each file.

Note: The logical flow of the compiler is illustrated in Figure 1.

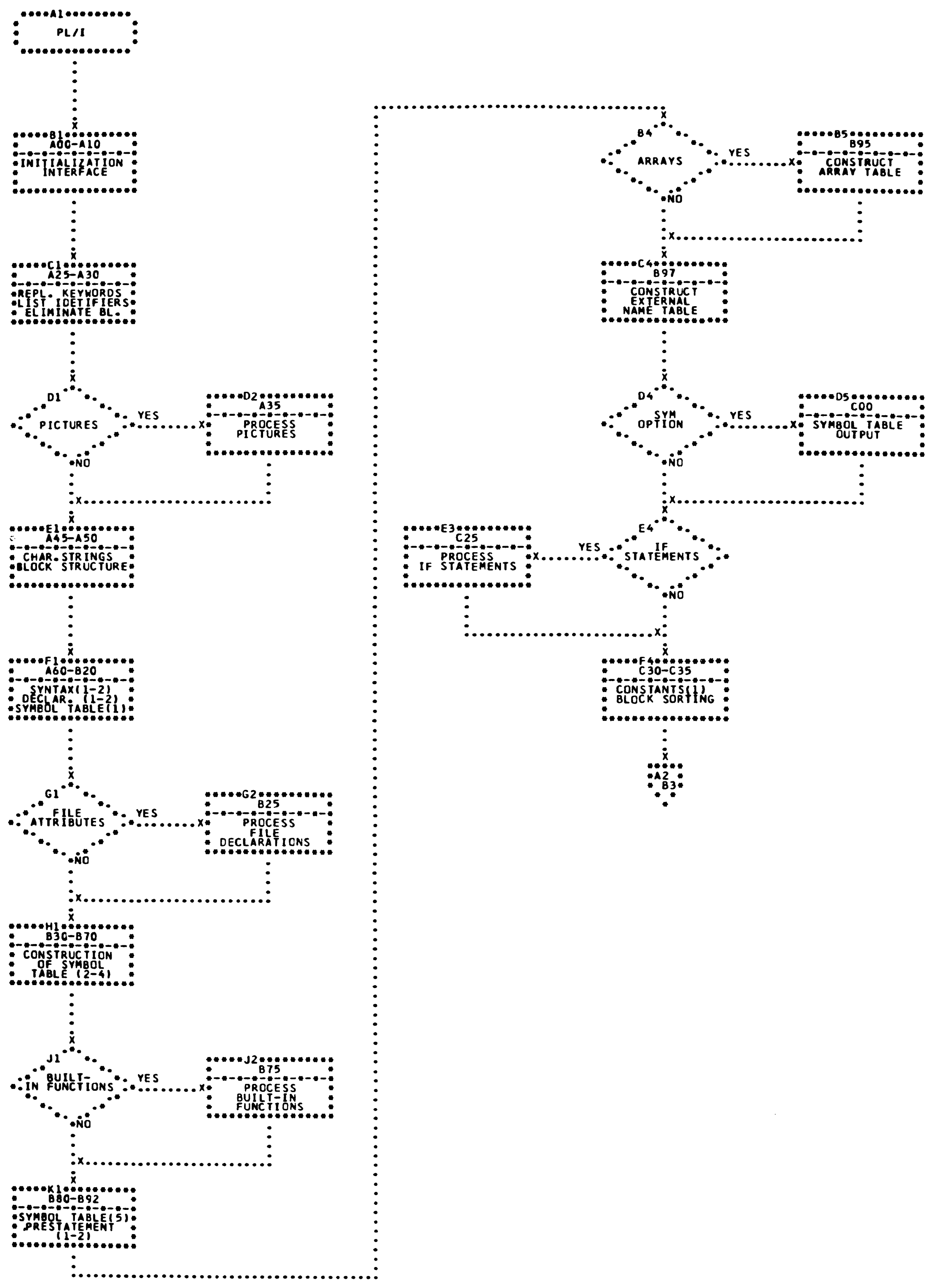


Figure 1. Logical Flow of the DOS/TOS PL/I Compiler (Part 1 of 3)

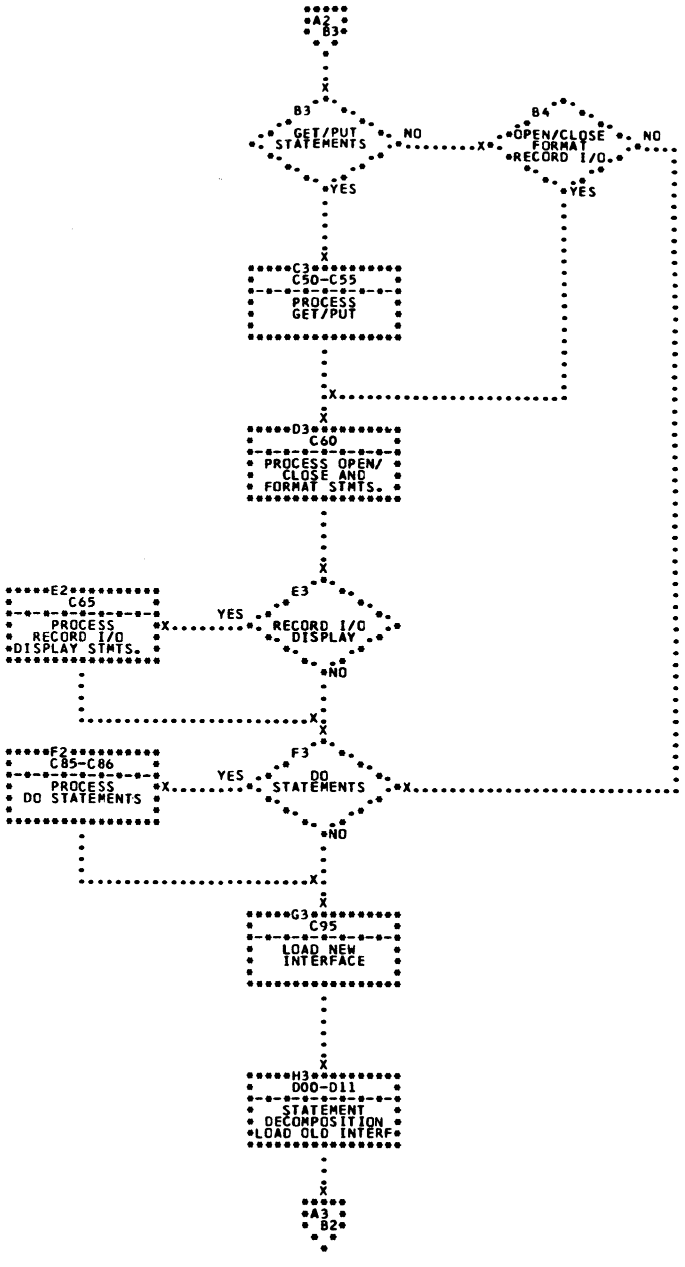

A2 B3
B3 GET/PUT STATEMENTS
NO
YES
B4 OPEN/CLOSE FORMAT RECORD I/O
NO
YES
C3 C50-C55 PROCESS GET/PUT
D3 C60 PROCESS OPEN/ CLOSE AND FORMAT STMTS.
E2 C65 PROCESS RECORD I/O DISPLAY STMTS.
YES
E3 RECORD I/O DISPLAY
NO
F2 C85-C86 PROCESS DO STATEMENTS
YES
F3 DO STATEMENTS
NO
G3 C95 LOAD NEW INTERFACE
H3 D00-D11 STATEMENT DECOMPOSITION LOAD OLD INTERF
A3 B2

Figure 1. Logical Flow of the DOS/TOS PL/I Compiler (Part 2 of 3)

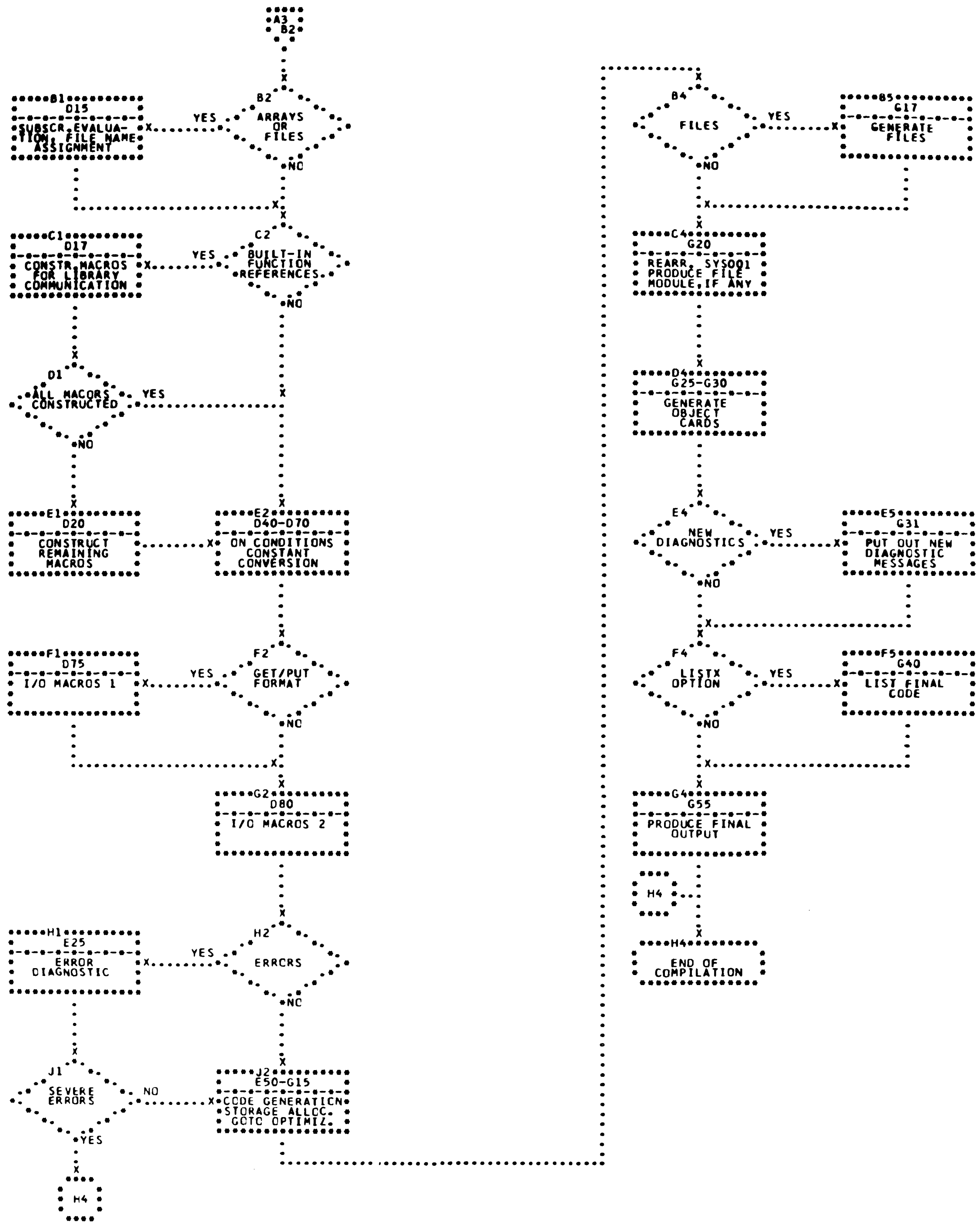

A3 B2
B2 ARRAYS OR FILES
YES
NO
B1 D15 SUBSCR.EVALUATION, FILE NAME ASSIGNMENT
C2 BUILT-IN FUNCTION REFERENCES
C1 D17 CONSTR.MACROS FOR LIBRARY COMMUNICATION
D1 ALL MACROS CONSTRUCTED
E1 D20 CONSTRUCT REMAINING MACROS
E2 D40-D70 ON CONDITIONS CONSTANT CONVERSION
F2 GET/PUT FORMAT
F1 D75 I/O MACROS 1
G2 D80 I/O MACROS 2
H2 ERRORS
H1 E25 ERROR DIAGNOSTIC
J1 SEVERE ERRORS
J2 E50-G15 CODE GENERATION STORAGE ALLOC. GOTO OPTIMIZ.
H4
B4 FILES
B5 G17 GENERATE FILES
C4 G20 REARR. SYS001 PRODUCE FILE MODULE, IF ANY
D4 G25-G30 GENERATE OBJECT CARDS
E4 NEW DIAGNOSTICS
E5 G31 PUT OUT NEW DIAGNOSTIC MESSAGES
F4 LISTX OPTION
F5 G40 LIST FINAL CODE
G4 G55 PRODUCE FINAL OUTPUT
H4 END OF COMPILATION

Figure 1. Logical Flow of the DOS/TOS PL/I Compiler (Part 3 of 3)

THE TEXT STRING DURING COMPILATION

The general concept for the representation of the text string is that the text string consists of text elements whose first byte (the key) contains the meaning of the element. The keys may be either X'En' or X'Fn'. X'En' is used for text elements with fixed length; X'Fn' is used for text elements with variable length. For the latter category, the two following bytes give the length of the element.

During compilation, the text passes the following five main states:

1. Source text (phases A25 - B95)
2. Statement-oriented text (phases C25 - E25)
3. Macros (phases E50 - E61)
4. Assembler code (phases F25 - G15)
5. Final output (phases G17 - G55)

Source Text

This is the initial status of the text string. The source program is taken as it is written by the programmer.

After deleting redundant information, e.g., blanks and comments, and translating the machine-dependent external code into an internal code, the individual language elements are replaced. First, the identifiers that look like keywords are replaced by 3-byte keys. The remaining identifiers are replaced by 3-byte internal names. Delimiters are replaced during the syntax phases by their 3-byte keys.

DECLARE statements are deleted from the text string. The information contained therein is partially transferred to pre-statements that are constructed to precede the statements.

Statement-oriented Text

At this stage, a statement may consist of the following items:

1. Each statement is introduced by a 6-byte statement identifier with the key X'E0'.
2. The statement identifier may be followed by a table that contains the attributes of the declared variables. The attribute table has the key X'F4'.
3. Items 1 and 2 (where item 2 is optional) may be followed by a table of the constants declared in the corresponding statement. The constant table has the key X'F3'.
4. The statement body consists of a sequence of 3-byte elements, each of which represents either an identifier or a keyword.
5. Each statement is terminated by a 6-byte "end of statement" (EOS) has the key X'EA'.
6. The statement (consisting of the elements listed under items 1 through 5) may be followed by 2-byte error indicators giving the errors that were detected in the preceding statement. The error indicator has the key X'EB'.

This form of the text string is changed by deleting the statement attribute tables and replacing each statement body by a sequence of macros. The replacement of the statement bodies is performed in several steps. This means that specific phases process only specific statements, whereas the remaining statements are passed unchanged to the next phase. At this stage, the status of the text string is therefore not uniform.

For some operations, generated variables are used as additional required storage, e.g., for the result of an operation. Definitions of such generated variables (with the key X'F0') are inserted into the text string.

For a limited time, additional information may be inserted into the text string, e.g., to mark an element as interesting or not interesting for some other phase(s).

Macros

The statement body is replaced by one or more macros. Each macro represents a particular operation. Macros have the key X'F2'. The format of the individual macros is fixed (see General Description of Phases E50 - E61). The macros contain the information required for generating the assembler code.

The definition of the individual macros is such that each macro is either associated with a fixed set of code, or the selection of the required code is possible only by means of the operands of the macro.

The error indicators are deleted from the text string at the same time the macros are replaced by assembler code.

Assembler Code

After the assembler code has been generated, the text string consists of the following:

1. Statement identifiers as just described.
2. Assembler code.
3. Generated variables as just described.
4. Constant tables as just described.
5. End of statement as just described.

Assembler code elements have the key X'F6'. Two types of instructions are used: machine instructions and pseudo instructions for communication with the assembler (phases F25 - G55). The machine instructions refer to the IBM System/360 machine instructions, to which they are equal except for the format of the operands. The format of the assembler code is described under General Description of the Phases E50 - E61.

The constant tables and generated variables are deleted from the text string after storage allocation. The first three bytes of all assembler code elements (X'F6xxxx') are also deleted.

After storage has been allocated, it is possible to replace the operands of the assembler code by base register and displacement. Thus, the assembler instructions are expanded by insertion of the address of a symbolic given operand (base and displacement) after the corresponding operand. Most of the pseudo instructions furnish information for this change and are deleted after the expansion. Only the instructions defining or reserving storage (DC X, DS) remain in the text string.

The static storage for the program is given a format similar to the pseudo instructions and is joined to the program string that consists of the assembler instructions.

This format of the text string is the last step on the way to the final output.

Final Output

The final output of the compiler consists of two modules, each of which consists of ESD, TXT, and RLD cards, and an END card. The first module is produced for all of the file declarations; the second module is produced for the program with the static storage. The TXT cards are generated from the assembler instructions and the static storage.

The system file accommodating the final output of the compiler depends on the options specified in the OPTION job control statement.

COMPILER INTERFACE

The logical IOCS provided by the DOS/TOS is used for input and output of data during a compilation. For this purpose as well for loading a new phase, the compiler control routines (interface) are provided to communicate between the compiler phases and the operating system. The interface mainly consists of subroutines to be called by the individual phases. Each subroutine causes DOS/TOS to perform a specific function requested by a phase.

These subroutines form the main body of the compiler control program, which contains a communication region used by the phases. Some of the subroutines, together with the communication region, are part of phase A00/A00D and reside in storage throughout the compilation. (For exceptions refer to phase C95.) The main functions of these subroutines are:

1. To load a new phase from the core-image library on SYSRES.
2. To handle the input text stream on SYS002 or SYS003;
3. To handle the output text stream on SYS002 or SYS003;
4. To write information on SYS001 for intermediate storage;
5. To read information intermediately stored on SYS001.

Alternating from phase to phase, the logical units SYS002 and SYS003 serve as input or output medium. The three logical units SYS001, SYS002, and SYS003 must always be assigned to physical units of the same device type (disk or tape). The device type may be changed from job to job.

The internal communication area (interphase communication region) provided in the control program is used for communication between different phases.

Macro instructions may be used in a compiler phase to branch through a branching vector in the interphase communication region to one of the interface routines in the compiler control program.

Some compiler phases require data input or output in addition to that mentioned above. These functions pertain to the input of the source program, output of listings, writing the object module either on SYSPCH or on SYSLNK for compile-and-go. A special routine is provided for each of these functions. It is assembled together with the phase requesting the function. The functions of these routines and the names of the logical I/O units used are listed below:

1. Input of PL/I source program from SYSIPT;
2. Output listing of source program on SYSLST;
3. Output listing of the offset table on SYSLST;
4. Output listing of error messages on SYSLST;
5. Output listing of source program symbols and external references on SYSLST;
6. Output listing of generated object program on SYSLST;
7. Output of generated object module on SYSLNK;
8. Output of generated object module on SYSPCH.

The logical unit SYSLNK must always be assigned to a physical unit of the same device type as SYSRES (disk or tape). The device type is fixed at system generation time. SYSIPT, SYSLST, and SYSPCH may be assigned to different device types. The assignment of these three units may be changed from job to job. The file specifications for these units are of the type DTFCP, which provides device independence. The user can control the bypassing of some output for listings or object modules by means of appropriate parameters in the OPTION card.

Some special control routines can be inserted into a compiler phase by means of appropriate macro instructions. These routines serve for input and output of table information on the device assigned to SYS001 and for moving a record of any length into the available storage area.

Storage Layout During Compilation

Storage allocation during compilation is illustrated in Figure 1. It is assumed that at least 10K (excluding the storage required by the DOS/TOS) is available for compilation of PL/I programs. The area occupied by the DOS/TOS is followed by an area of 2.6K for the compiler control program and logical IOCS routines used by it. The table directory (184 bytes) which contains information on tables used during compilation is contained in this area. The area provided for the compiler phases is 4K bytes long. It is followed by the Table Area and the Buffer Area. If more than 10K bytes are available for the PL/I compiler, the entire additional storage area is allocated to the Table and Buffer Areas.

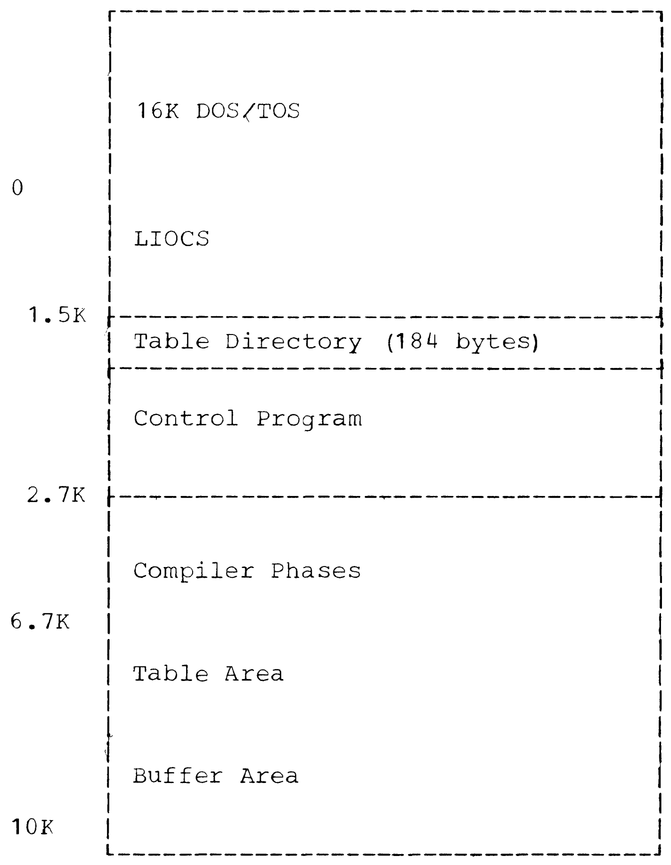


Figure 1. Storage Layout During Compilation (for 16K)

As shown in Figure 1, the begin address of the Table Area is always 6.7K bytes higher than the start address of the storage available during compilation. The length of the Table Area and the start address of the Buffer Area are calculated in the Initialization routine of this phase as follows: The Buffer Area (see Figure 2) is partitioned into seven buffers of equal length. The first five buffers serve as work areas for the compiler phases. The remaining two buffers are used as input and output areas for overlapped processing of text information. The length of the Buffer Area is the sum of the individual buffers plus 8 bytes. (These 8 bytes serve a special use during compilation.) The length of a single buffer depends on the total storage available during compilation. The minimum length is 256 bytes, which results in a minimum Buffer Area length of 256 x 7 + 8 bytes = 1800 bytes.

The minimum buffer length is always taken for an available storage size from 10K to 14K. The minimum storage for both the Table and the Buffer Area is: 10K - 2.5K - 4K - 184 bytes = 3.4K. Thus, the minimum length of the Table Area is 3.4K - 1800 bytes = 1600 bytes.

For the tape version, the length of a single buffer is extended by 256 bytes for each additional 4K available storage until the length reaches 1536 bytes (for 30K storage). This is shown in Figure 3. If more than 30K is available, the buffer length remains at 1536 bytes and the entire additional storage is allocated to the Table Area to reduce the time required for compilation.

For the disk version, the buffer length increases similarly (see Figure 3). To avoid unused track space as far as possible, the maximum buffer length for the disk version is 1536 bytes.

Overall Storage Requirements about 3.3K								
Table Area	Buffer Area							
	Work Area 1	Work Area 2	Work Area 3	Work Area 4	Work Area 5	8 Bytes	I/O Area 1	I/O Area 2

Figure 2. Table and Buffer Areas

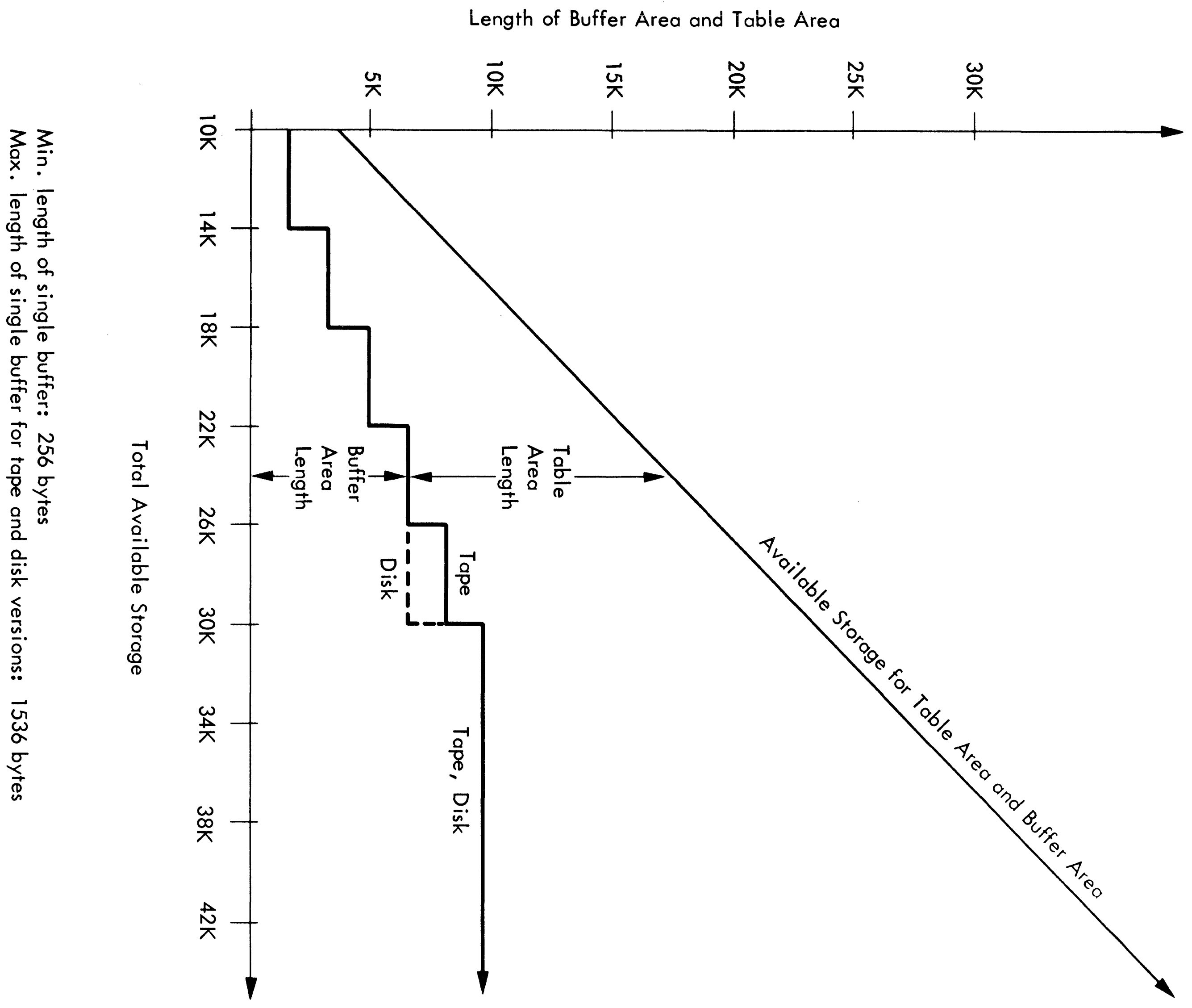


Figure 3. Partitioning of Storage for Buffer and Table Areas

Communication with the Control Routines

The routines of the compiler interface that remain in storage together with the interphase communication region are called by use of special macro instructions. The expansion of each of these macro instructions contains a branch to the corresponding routine through a branching vector in the interphase communication region.

The main purpose of the communication region is to accommodate information to be exchanged between phases. It is part of the control section IJXA01 in phase A00 or A00D and assembled by the macro instruction IJKCO INTERF. If the parameter INTERF is omitted, a dummy section for this region is assembled. This is done in every compiler phase to cause each symbol specified in the communication region to be assembled in the compiler phase without storage being assigned to it. During the Initialization routine, the start address of the communication region is loaded into register 12. If a USING instruction is given at the beginning of each phase, this register can be used as base register for addressing the communication region.

The interphase communication region shown in Figure 4 can be logically divided into four parts. The first part is the branching vector that contains branch instructions to the individual interface routines always contained in storage. Most of the macro instructions provided for use in the compiler phases generate a branch to this branching vector.

		TRANSFER VECTOR
	B	IJKAGI READ RECORD FROM TEXT INPUT IN OVERLP
	B	IJKAGO READ RECORD FROM TEXT OUTPT IN OVERLP
	B	IJKANT GET RECORD IDENTIFICATION (I,O,T)
	B	IJKAPH ROUTINE FOR END OF PHASE
	B	IJKAPI WRITE RECORD ON TXT INPUT IN OVERLP
	B	IJKAPO WRITE RECORD ON TEXT OUTPUT IN OVERLP
	B	IJKAPT ROUTINE FOR POINTW
	B	IJKARN RESET END IDENTIFICATION FOR SYS001
	B	IJKAWT WAIT FOR COMPLETION (I,O,T)
	B	IJKATIN READ RECORD FROM SYS001 IN NONOVERLP
	B	IJKATOUT WRITE RECORD ON SYS001 IN NONOVERLP
	B	IJKAMN MOVE RECORD NORMALLY
	B	IJKAGINO READ RECORD FROM TEXT INPUT IN NOOV
	B	IJKAGONO READ RECORD FROM TEXT OUTPUT IN NOOV
	B	IJKAPINO WRITE RECORD FROM TEXT INPUT IN NOOV
	B	IJKAPONO WRITE TEXT RECORD ON TXT OUTPUT IN NOOV
	B	IJKAPTR ROUTINE FOR POINTR
		COMMUNICATION BYTES
IJKMLB	DC	8F'0' LIBRARY USAGE BYTES
IJKMBS	DS	F BUFFER AREA START ADDRESS
IJKMPR	DS	F BUFFER FOR PRINT REGISTER
IJKMPC	DS	F BUFFER FOR PUNCH REGISTER
IJKMTT	DC	A(IJXA0J) ADDRESS OF TABTAB
IJKMTS	DC	A(IJXA0M) ADDRESS OF TABLE SPACE
IJKMJT	DC	F'0'JOB INFORMATION BITS
		BIT 0 SYSIN HAS BEEN CALLED
		BIT 1 ERRORS IN CURRENT COMPILATION
		BIT 2 ARRRAYS IN CURRENT COMPILATION
		BIT 3 STRUCTURES IN CURRENT COMPILATION
		BIT 4 ARRAY EXPRESSIONS IN CURRENT COMPILATION
		BIT 5 I/O IN CURRENT COMPILATION
		BIT 6 FILE ATTRIBUTE IN CURRENT COMPILATION
		BIT 7 INITIAL ATTRIBUTE IN CURRENT COMPILATION
		BIT 8 DEFINED ATTRIBUTE IN CURRENT COMPILATION
		BIT 9 SYSPRINT HAS BEEN CALLED
		BIT 10 PICTURE ATTRUBUTE IN CURRENT COMPILATION
		BIT 11 INDICATES MAIN PROCEDURE
		BIT 12 CALLS FOR LIBRARY ROUTINES IN COMPILATION
		BIT 13 DO LOOPS IN CURRENT COMPILATION
		BIT 14 TABLE DICTIONARY ON SYS001
		BIT 15 OUTPUT LISTING STARTED
		BIT 16 TYPE OF WORK FILE MEDIA DURING COMPILATION 0 FOR TAPE
		BIT 17 IF ON, SYS002 IS THE CURRENT TEXT INPUT MEDIUM
		BIT 18 INDICATES ONSYSLOG
		BIT 19 FORMAT LABELS IN CURRENT COMPILATION
		BIT 20 BUILT-IN FUNCTIONS IN CURRENT COMPILATION
		BIT 21 NEED FOR INDIRECTLY CALLED LIBRARY ROUTINES
		BITS 22 - 26: RESERVED
		BIT 27 SKIP BIT FOR PHASE C25
		BIT 28 SKIP BIT FOR PHASES C50-C55
		BIT 29 SKIP BIT FOR PHASE C85
		BIT 30 SKIP BIT FOR PHASES C60-C65
		BIT 31 SKIP BIT FOR PHASE C65
IJKTAB	DC	A(0) RECORD IDENTIFIER FOR TABTAB ON SYS001
IJKMVC	DC	H'256' VARIABLE COUNTER
IJKMNN	DC	H'0' INTERNAL NAME OF THE ADDRESS CONSTANT FOR THE ORIGIN OF COMPILATION
IJKMWC	DC	H'0' COUNTER FOR GENERATED VARIABLES WITH UNKNOWN ATTRIBUTES
IJKCSL	DC	H'0' LENGTH OF CHARACTER STIRNGS
IJKPAG	DC	H'1' PAGE NUMBER FOR LISTING
IJKDCW	DC	Y(0) DECLARED VARIABLE COUNTER INCL. CONST.

Figure 4. Assembly Listing of the Interface Communication Region (Part 1 of 2)

```
┌────────┬───┬──────────────────────────────────────────────────────────────────┐
|IJKMIP  |DS |5H INTERPHASE COMMUNICATION BYTES                                  |
|IJKMBL  |DC |Y(256) BUFFER LENGTH                                              |
|IJKMBC  |DC |X'00' BLOCK COUNTER                                               |
|IJKMCH  |DC |CL6'CL2-0'                                                        |
├────────┼───┼──────────────────────────────────────────────────────────────────┤
|        |   |TABLE KTETA FOR INTERFACE HOUSEKEEPING                            |
|        |   ├──────────────────────────────────────────────────────────────────┤
|KTETA   |DC |A(KSYS002) POINTER FOR TEXT INPUT                                 |
|        |DC |A(KSYS003) POINTER FOR TEXT OUTPUT                                |
|KSYS001 |DC |A(IJSYS01) ADDRESS OF FILE TABLE FOR SYS001                       |
|        |DC |F'0' RESERVED FOR LINE NUMBER. MAINTENANCE                        |
|        |DC |F'0' END KEY FOR INFORMATION ON SYS001                            |
|        |DC |X'18000000' INDICES                                               |
|KSYS002 |DC |A(IJSYS02) ADDRESS OF FILE TABLE FOR SYS002                       |
|        |DS |F ADDRESS OF I/O AREA FOR SYS002                                  |
|        |DC |F'0' END KEY FOR INFORMATION ON SYS002                            |
|        |DC |X'18000000' INDICES                                               |
|KSYS003 |DC |A(IJSYS03) ADDRESS OF FILE TABLE FOR SYS003                       |
|        |DS |F ADDRESS OF I/O AREA FOR SYS003                                  |
|        |DC |F'0' END KEY FOR INFORMATION ON SYS003                            |
|        |DC |X'18000000' INDICES                                               |
├────────┼───┼──────────────────────────────────────────────────────────────────┤
|        |   |SAVE AREAS                                                        |
|        |   ├──────────────────────────────────────────────────────────────────┤
|        |DS |0D                                                                |
|KSAVE1  |DS |F SAVE AREA FOR REGISTER                                          |
|KSAVE2  |DS |F SAVE AREA FOR REGISTER                                          |
|KSAVE3  |DS |F SAVE AREA FOR REGISTER                                          |
|KSAVE4  |DS |F SAVE AREA FOR REGISTER                                          |
|KSAVE5  |DS |F SAVE AREA FOR REGISTER                                          |
|KSAVE6  |DS |F SAVE AREA FOR REGISTER                                          |
|KSAVE7  |DS |F SAVE AREA FOR REGISTER                                          |
|KSAVE8  |DS |F SAVE AREA                                                       |
|KSAVE9  |DS |F SAVE AREA                                                       |
├────────┼───┼──────────────────────────────────────────────────────────────────┤
|        |   |SYMBOLIC NOTATION FOR REGISTERS                                   |
|        |   ├──────────────────────────────────────────────────────────────────┤
|ZREG0   |EQU|0                                                                 |
|ZREG1   |EQU|1                                                                 |
|ZREG2   |EQU|2                                                                 |
|ZREG3   |EQU|3                                                                 |
|ZREG10  |EQU|10                                                                |
|ZREG11  |EQU|11                                                                |
|ZREG12  |EQU|12                                                                |
|ZREG14  |EQU|14                                                                |
|ZREG15  |EQU|15                                                                |
├────────┼───┼────────────────────────────────────────────────┬─────────────────┤
|        |   |TABTAB DESCRIPTION                              |                 |
|        |   |     NAME     TABLE DESCRIPTION                 | PHASE TO PHASE  |
├────────┼───┼────────────────────────────────────────────────┼─────────────────┤
|ZTAB00  |EQU|000  CARTAB   CHARACTER CONSTANT TABLE          | A45       G15   |
|ZTAB01  |EQU|008  NAMTAB   NAME TABLE                        | A25       C00   |
|ZTAB02  |EQU|168  SYMTAB   SYMBOL TABLE                      | B10       F90   |
|ZTAB03  |EQU|024  FILTAB   FILE TABLE                        | B25       G17   |
|ZTAB04  |EQU|032  EXTTAB   EXTERNAL NAME TABLE               | B97       G25   |
|ZTAB05  |EQU|040  ARYTAB   ARRAY INFORMATION TABLE           | B95       D15   |
|ZTAB07  |EQU|056  DSTAB    DS TABLE                          | F25       F90   |
|ZTAB08  |EQU|064  CONTAB   CONSTANT TABLE                    | F35       G15   |
|ZTAB11  |EQU|088  OFFTAB   FINAL OFFSET TABLE                | F90       G15   |
|ZTAB16  |EQU|128  CARDS    CARDS FOR FINAL OUTPUT            | G20       G55   |
|ZTAB18  |EQU|144  FORMTAB  FORMAT LABEL TABLE                | C60       D15   |
|ZTAB21  |EQU|016  LITAB    LIOCS TABLES                      | A00       G55   |
|ZTAB19  |EQU|152  CONEQU   EQUATE TABLE                      | F35       F90   |
|ZTAB19  |EQU|152  LABTAB   LABEL OFFSET TABLE                | G00       G25   |
└────────┴───┴────────────────────────────────────────────────┴─────────────────┘
```

Figure 4. Assembly Listing of the Interface Communication Region (Part 2 of 2)

The second part consists of communication bytes. These are various DC entries where information is exchanged from phase to phase. This part further includes entries for the work buffer length, the start address of the Table and Buffer Areas, and the TABTAB address. The entry IJKMJT contains job information bits which indicate special features of the source program to be compiled, e.g., structures, etc.

The third part is a register save area used by the individual interface routines.

The fourth part is a string of EQU statements specifying register names and offsets of TABTAB entries.

Note: The base register (register 15) is saved by the subroutines. Therefore, no reloading of the base address is required in the compiler phase after a macro instruction has been issued.

Housekeeping on Work Files

The functions of SYS002 and SYS003 (text input and output, respectively) are normally switched at the end of a compiler phase. This switching is done by means of the table KTETA, which is part of the communication region and contains file specification information for SYS001, SYS002, and SYS003. The format of this table is shown in Figure 4. The table contains one 4-word entry for each of the work files. The contents of each 4-word entry are described below.

The first word contains the address of the file definition table. The second word contains the address of the I/O area used (for SYS002 and SYS003 only). For overlapped I/O operation, the same I/O area is always assigned to one of SYS002 and SYS003. The third word contains the record identifier for the last record written on the file. It is changed whenever the end key for the information written must be saved. The first byte of the fourth word contains housekeeping flag-bytes (see Figure 5). Bytes 3 and 4 are used to accommodate the available track length.

The first two words of KTETA contain pointer addresses. Each address points to one of the 4-word entries for SYS002 and SYS003. The first one of these pointers represents text input, the second represents text output. Switching of the I/O functions for these units simply consists of an exchange of these first two words in KTETA.

The use of this table is discussed in more detail in the description of the individual control routines. One of the main functions of the control routines is the setting, resetting, and testing of flag bits in KTETA.

Bit 0	Index for writing
Bit 1	Index for end of file
Bit 2	Index for first read call
Bit 3	Index for rewinding
Bit 4	Index for checking
Bit 5	Index for POINTW
Bit 6	Index for NOTE
Bit 7	Index for POINTR

Figure 5. Flag Bits Used in KTETA

The information to be exchanged between phases is stored in the form of tables written on SYS001. A communication table, referred to as TABTAB and following the Interface area, is provided for accessing these tables. Each table is pointed to by an 8-byte entry in TABTAB. Each entry contains the information shown in Figure 6.

BYTES	MEANING
1	Bit 0 = 1 indicates that the table is on SYS001 Bit 1 = 1 indicates that that table is in storage Bit 2 = 1 indicates that transfer to or from SYS001 has been started
2--4	Identifier of the first table record on SYS001
5--6	Number of records on SYS001 for the table
7--8	Length of a record on SYS001

Figure 6. Format of Entries in TABTAB

Two special routines (ZTIN and ZTOUT) are provided for reading and writing tables or part thereof on SYS001. If these routines are to be used, the entry for record length must have been specified by the compiler phase. The housekeeping on the other TABTAB entries is explained in the discussion of the individual routines.

Interface Structure for DOS/TOS Versions

The structure of the interface differs according to the DOS/TOS version used. The differences are as follows:

1. The work files used for the tape version are of the form DTFMT, MTMOD. The same is used for the disk version if the work files are assigned to tapes. DTFSD, SDMOD is used if the work files are assigned to disks. For the disk version, file tables and modules for DTFSD, SDMOD are loaded. During the

initialization, the type of work files is tested and, if necessary, tables and modules for tape work files are loaded (phase A10) to overlay the previous ones. Thus, the user may change his assignments for work files on tape or disk from job to job if he used the disk version.

2. If the work files are assigned to disk, a flag bit is set in the interphase communication region during Initialization, and a conditional branch instruction in the control routine for text input (IJKAGI) is changed to an unconditional branch.

3. For the disk version, the file parameter DISK=YES is always specified for the file IJSYSLN. For the 32K disk version, the same parameter DISK=YES is specified for the files IJSYSIN, IJSYSIS, and IJSYSPH.

4. The output listing header lines differ for the disk and tape versions. This implies differences in the listing phases.

LIOCS Modules Used by the Interface

The logical LIOCS modules used by the compiler are included during the Linkage Editor run; they are not assembled together with the phases.

The module name for the files IJSYSIN, IJSYSLS, and IJSYSPH is IJJCP0 for the tape version and the 16K disk version; it is IJJCPD0 for the 32K disk version.

The module name for the file IJSYSLN is IJJCP0 for the tape version and IJJCPD0 for both disk versions.

The module name for work files is IJGWZNZZ for disk work files, and IJFWZNZZ for tape work files.

The work file module is always in storage; the other modules are in storage only together with the phases needing them, and overlaid by other phases.

INTERFACE ROUTINES USED BY COMPILER PHASES

There are two classes of routines. The first class comprises routines that remain in storage during the entire compilation (with the exceptions described in phase C95). They are called by macro instructions in the compiler phases.

The names of the routines and the corresponding macro instructions are listed in Figure 7.

Routine	Calling Macro Instruction
IJKAWT	IJKWT
IJKANT	IJKNT
IJKAPTR* IJKAPT*	IJKPTR IJKPT
IJKAPH	IJKPH
IJKAGI* IJKAGO* IJKAGINO* IJKAGONO* IJKAPI* IJKAPO* IJKAPINO* IJKAPONO*	IJKGI IJKGO IJKRI IJKRO IJKPI IJKPO IJKWI IJKWO
IJKAMN	IJKMN
IJKATIN	Contained in ZTIN (see Figure 8)
IJKATOUT	Contained in ZTOUT (see Figure 8)
*=Routine with more than one entry point.	

Figure 7. Interface Routines Called by Macro Instructions

The routines listed in Figure 7 internally use the following subroutines: KGETNOTE, KREAD, KCHECK, and K2CHECK. (The last two names are entry points of the same routine.)

Note: KCHECK and K2CHECK are entry points of the same routine.

The second class of routines comprises all routines that can be assembled in the phase either directly or by means of macro instructions. These routines are called inside the phases by appropriately branching to them. The names of these routines and the corresponding macro instructions to include them are listed in Figure 8.

Routine	Macro for Assembly
ZTIN	IJKTI
ZTOUT	IJKTO
ZMO	IJKMO
ZRCD	-
ZPRNT	-
ZLEDI	-
ZPCH	-

Figure 8. Interface Routines Assembled In-Line either Directly or by Macro Instructions

The source programs for ZRCD, ZPRNT, ZLEDI, and ZPCH are part of the corresponding phase source program.

The functions of all interface routines and the corresponding macro instructions are explained in the following sections.

KCHECK, K2CHECK -- AF

When this subroutine is called, register 0 (KCHECK) or register 2 (K2CHECK) contains the address of a work file item in KTETA. If necessary, the subroutine issues a CHECK macro instruction for this work file. For correct housekeeping on the record identifier in the LIOCS, this CHECK macro instruction must be given only once after each read or write operation. The check index (a flag bit in KTETA) is used to check whether the CHECK macro instruction is required.

KGETNOTE -- AF

When this subroutine is called, register 2 contains the address of a work file item in KTETA. The subroutine performs some housekeeping and issues a NOTE macro instruction. If a first call is performed after a write operation, the record identifier obtained by NOTE is saved in KTETA together with the information for available track length. If a further call is performed after a write operation, no further NOTE is issued, but the information saved in KTETA is returned as for the preceding call. If a call is performed just after repositioning of the work file (in IJKAPH), a zero is returned in register 1 for the record identifier. This zero, if used in calling IJKAPT or IJKAPTR, causes a POINTS macro instruction to be issued.

KREAD -- AF

KREAD merely is the expansion of a work file read macro instruction.

IJKAWT -- AG

This routine is called by a compiler phase to wait for the completion of a read or write operation on a work file. On return, all register contents are unchanged. The macro for calling is IJKWT with one of the parameters I, O, or T.

The parameters I, O, and T specify text input, text output, and table medium, respectively. The macro expansion is a load instruction loading the address of a work file item in KTETA into register 0, and a branch-and-link instruction that branches to the branching vector in the interphase communication region. An example for the macro expansion is:

```
  L  ZREG0,KTETA
BAL  ZREG14,32(ZREG12)
```

IJKAWT performs the wait function by using the subroutine KCHECK.

IJKANT -- AG

This routine is called to obtain the actual record identifier. This may be repeated several times after a read or write operation. On return, register 0 contains the available track length (useful after writing on disk only), register 1 contains or the record identifier. The other registers remain unchanged. The macro for calling is IJKNT with one of the parameters I, O, or T.

The parameters I, O, and T specify text input, text output, and table medium, respectively. An example for the macro expansion is:

```
  L  ZREG0,KTETA+4
BAL  ZREG14,8(ZREG12)
```

IJKANT performs the NOTE function by using the subroutines KCHECK and KGETNOTE.

IJKAPTR, IJKAPT -- AG

This routine performs a POINTW (IJKAPT), a POINTR (IJKAPTR) or a POINTS operation (see description below) on a work file. On return, all register contents are unchanged. The macro instructions for calling are either IJKPT or IJKPTR with one of the parameters I, O, or T.

The parameters I, O, and T specify text input, text output, and table medium, respectively.

If one of these macros is given in a compiler phase, register 1 must contain the record identifier of the record to be pointed to (as obtained after a NOTE). If IJKPT is given, register 0 must contain the available track length only if a write command follows this pointing. An example for the macro expansion of IJKAPTR is:

```
  L  ZREG0,KTETA+4
BAL  ZREG14,64(ZREG12)
```

The first instruction loads the address of a work file item in KTETA into register 0, the second instruction branches to the branching vector. An example for the macro expansion of IJKAPT is:

```
STH  ZREG0,KSAVE7
  L  ZREG0,KTETA+4
BAL  ZREG14,24(ZREG12)
```

The first instruction saves the contents of register 0, which may be the available track length. The two other instructions are as shown for IJKAPTR.

The routine first calls the subroutine KCHECK. Then a test is made to determine whether the record identifier in register 1 is zero. If it is zero, a POINTS is issued, otherwise a POINTR or a POINTW, depending on the calling macro instruction.

Eventually, some flag bits are reset if a point was done with the actual end key stored in KTETA.

IJKAPH, KREP -- AH

The routine IJKAPH is normally used at the end of a compiler phase. It fetches a new compiler phase if requested by the calling program and repositions SYS001 and/or SYS002, if required. Moreover, the functions of SYS001 and SYS002, as regards text input and output, can be switched.

If rewinding or switching is requested by the calling program, register 0 must contain a specified number according to the following convention:

<register 0> = 0	No rewinding, no switching
<register 0> = 1	Rewind input medium, no switching
<register 0> = 2	Rewind output medium, no switching
<register 0> = 3	Rewind both media, no switching
<register 0> = 4	No rewinding, switching
<register 0> = 5	Rewind input medium, switching
<register 0> = 6	Rewind output medium, switching
<register 0> = 7	Rewind both media, switching

If a new compiler phase has to be fetched, register 1 must contain the address of a 4-byte character string that contains the last three character bytes of the phase name (right-aligned). Note that all compiler phase names differ in the last three characters only.

The routine first rewinds the text media, if necessary, using the subroutine KREP. It then switches their functions, if required. Finally, some housekeeping is done and a FETCH macro instruction is given if a new phase is required by the calling program.

If a write operation was the last operation performed on a text medium, the actual end key for this medium is saved prior to rewinding.

The routine IJKAPH can be called by the keyword macro instruction:

```
IJKPH NEWPH=,REW=ALL|I|O,SWITCH=NO
```

If a new phase is required, the keyword NEWPH must be specified followed by an equal sign and the three ending characters of the phase name. If NEWPH is not specified, no phase is fetched, and the routine returns to the calling program. For rewinding, the keyword REW may be specified followed by an equal sign and one of the parameters NO, ALL, I, or O. The meaning of these parameters is:

NO	No medium must be rewound
ALL	Both media must be rewound
I	The actual input medium must be rewound
O	The actual output medium must be rewound

If REW= with a parameter is not specified in the macro instruction, <u>both media</u> are automatically rewound. For switching of functions, the keyword SWITCH is specified followed by an equal sign and one of the parameters YES or NO. The meaning of these parameters is:

YES	Switching is performed
NO	Switching is not performed

If SWITCH= with a parameter is not specified, switching is done automatically.

IJKAGI, IJKAGO, IJKAGINO, IJKAGONO -- AI, AJ

This routine is used to read records from a work file medium. It can be called by various macro instructions. Each macro instruction provides a branch to a specific entry point by means of the branching vector. The correspondence is:

Macro Instruction	Entry Point	Function
IJKGI	IJKAGI	Overlapped input from text input medium.
IJKGO	IJKAGO	Overlapped input from text output medium.
IJKRI	IJKAGINO	Non-overlapped input from text input medium.
IJKRO	IJKAGONO	Non-overlapped input from text output medium.

When one of these macro instructions is given in a compiler phase, register 1 must contain the address of the area where the new record is required. For IJKRI and IJKRO, this is the input area for reading; for IJKGI and IJKGO, this is the work area. Note that overlapped input means that the record is available after returning from this routine, whereas non-overlapped input means that reading in the indicated area is started on return from the routine.

Some indices and pointers for KTETA are set first, depending on the entry point used. Then (only on the disk version, see Initialization) a test is made to determine whether a POINTW operation for the same file preceded this call. If so, a dummy read is performed to position the medium for reading.

For overlapped working, a test is made to determine whether reading in the overlapped mode has already been started. If this is not the case, a first record is read into the input area.

Before moving this record into the work area, the routine waits for completion of the preceding read operation. Finally, a new reading is started, and the routine returns to the calling program.

For non-overlapped working, the routine checks for completion of any previous operation. Then, a new read operation into the input area indicated in register 1 is started.

Note: Each starting of a new read operation is preceded by a test to determine whether the work file medium is positioned at the end of the information written on it. If it is, the routine returns without having started a new read operation.

The entry point at box G2 in flowchart AI is used by the routine IJKATIN to read a table record from IJSYS001 in non-overlapped mode.

The routine normally returns with register 0 set to 0. However, if no more records are available, register 0 contains a 1. All other registers are unchanged.

IJKAPI, IJKAPO, IJKAPINO, IJKAPONO -- AK

This routine writes records on a work file. It can be called by various macro instructions. Each macro instruction provides a branch to a specific entry point by means of the branching vector. The correspondence is shown below.

Macro	Entry Point	Function
IJKPI	IJKAPI	Overlapped output on text input medium.
IJKPO	IJKAPO	Overlapped output on text output medium.
IJKWI	IJKAPINO	Non-overlapped output on text input medium.
IJKWO	IJKAPONO	Non-overlapped output on text output medium.

When one of these macro instructions is given in a compiler phase, register 1 must contain the address of the area from where the new record has to be read. Each macro expansion is a branch-and-link instruction that branches to the branching vector. For IJKWO and IJKWI, this is the output area for writing; for IJKPI and IJKPO, this is the work area.

Non-overlapped output means that writing from the output area is started on return from the routine. Overlapped output means that the output record is first moved from the work area to the output area used by the interface. Output is then started from there before returning.

The routine sets some indices and pointers in the table KTETA depending on the entry point used. It checks for completion of any previous operation on the same file. A test is made to determine whether a POINTW is required for the file to position the medium at the end of the information actually written on it. If so, a POINTW is issued with the end key saved in KTETA. This end key is saved on each NOTE after a write operation. No POINTW is required if a write or rewind command was given last. This concept allows the compiler phases to interrupt the writing by some intermixed reading.

After this test, a record is moved from the work area to the output area if overlapped working was requested. Writing is then started and the routine returns with all register contents unchanged.

IJKAMN -- AL

This routine is used to move a record of any length from one area to another. Overlapping of the form that the start address of the TO area lies inside the FROM area is not allowed. The routine first moves single 256-byte records until a field shorter than 256 bytes remains to be moved. The residual moving length is then calculated and moving is performed. If the whole length on calling is zero, no moving is performed by the routine.

IJKAMN can be called in the source program of the compiler phase by the macro instruction IJKMN. The macro expansion consists of a branch-and-link instruction that branches to the branching vector in the interphase communication region, e.g.:
    ZREG14,44(ZREG12)
The following register contents must have been provided:

register 0 - total field length,
register 1 - start address of the TO field,
register 2 - start address of the FROM field.

On return from the routine, registers 1 and 2 contain the end addresses +1 of the corresponding fields. The content of register 0 is undefined. All other registers remain unchanged.

IJKATIN -- AL

This routine is called by ZTIN in the compiler phase to read a record from SYS001 in non-overlapped mode. On calling, the address of the input area is contained in register 1. This address and the maximum table record length are supplied to the subroutine KREAD. The routine then branches to some entry point of IJKAGINO. There is no macro instruction to call this routine.

IJKATOUT -- AL

This routine is called by ZTOUT in the compiler phase to write a record on SYS001 in non-overlapped mode. On calling, the address of the output area is contained in register 1, the length is contained in register 0. The register contents are supplied to the routine IJKAWI. The routine then branches to some entry point of IJKAPINO. There is no macro instruction to call this routine.

ZTIN -- AM

The symbolic start address of this routine is ZTIN. It can be called into the source program of a compiler phase by the macro instruction IJKTI. On branching to the routine, the following register contents must have been provided:

<register 0>	=	Number of records to be read,
<register 1>	=	Start address of the read-in area of the records,
<register 2>	=	Relative address of the entry in TABTAB.

A table can be read from SYS001 in several steps, i.e., by branching to ZTIN several times for the same table. If a table is to be read in from its beginning, bit 2 of the corresponding TABTAB entry must be set to zero prior to branching to the routine. In all other cases, it must be set to one.

The routine first tests whether bit 2 of the TABTAB entry is zero. If it is, the following steps are performed:

1. If a write operation was the last operation on SYS001, the last record written is first identified by a NOTE macro instruction; the identifier is saved for writing on SYS001 at a later time.

2. Bits 1 and 2 of the TABTAB entry are set to 1.

3. SYS001 is repositioned according to the record identifier found in the TABTAB entry, provided that this record identifier is less than the identifier for the last record written on SYS001. If the record identifier is higher, compiling is terminated with an error dump. Then the reading of the single records is started.

The routine normally returns to the calling program when reading of the last record has been started. Thus, a limited overlapping of I/O and processing is possible.

A test is made for each record to be read to check that it is not located beyond the end of information written on SYS001. If this test matches for each record, the routine returns with register 0 set to 0. If this test does not match for a new reading, ZTIN is terminated immediately and returns with register 0 set to 1.

The routine ZTIN performs its I/O functions by calling routines in the compiler control program. The record length is the physical record length for each record read.

ZTOUT -- AN

The symbolic start address of the routine is ZTOUT. It can be called into the source program of a compiler phase by the macro instruction IJKTO. On branching to the routine, the following register contents must have been provided:

<register 0>	=	End address of the output area+1,
<register 1>	=	Start address of the output area,
<register 2>	=	Relative address of the entry in TABTAB.

If the beginning of a table is to be written on SYS001, bit 2 of the corresponding TABTAB entry is set to zero prior to branching to ZTOUT. In this case, the identification for the first record written is saved in TABTAB after this record has been written on SYS001. In all other cases, bit 2 of the TABTAB entry is set to 1.

Prior to starting the write operation, the routine checks whether the end address in register 0 is less than or equal to the start address or whether the record length RL stored in the TABTAB entry is not in the range 18 < RL < 3500. The compilation is terminated with an error dump if these conditions occur.

After the first record has been written on SYS001, bit 2 in the TABTAB entry is checked for zero. If it is zero, the following is done prior to the next write operation:

1. The record identification for the record just written is saved in the TABTAB entry;
2. The number of records written for the table on SYS001 is set to 1 in the TABTAB entry;
3. Bits 0 and 2 of the flag byte in TABTAB are set to 1.

The length of the records written is determined by the TABTAB entry. The routine normally returns after having started the writing of the last record. The length of this record is at least 18 bytes.

ZRCD -- AO

This routine is used to read source cards and to print lines on SYSLST. Reading and printing is done in overlapped mode.

This routine uses ZPRNT as a ubroutine for printing. Each line is 120 characters long. For reading, the routine contains the two I/O areas used for overlapped reading.

On each call but the first, the routine writes a line and then reads a card. On the first call, only the first card is read. On each return from the routine, register 10 points to the start address of an alternative input area where a card has been read in. Register 11 points to the start address of an alternative output area where the record for output on SYSLST can be built.

The logical IOCS used for reading is of the type DTFCP, CPMOD. The branch address for the end-of-file condition is ELC010 in phase A25.

ZLEDI -- AO

This routine is used to write a record on SYSLNK in non-overlapped mode. Only one output area of the length 322 bytes is used. This output area is located in the Buffer Areas otherwise used as I/O areas for text input and output. Its start address is equal to the start address of the two Buffer Areas. Each output record is 322 bytes long (4 cards). The first two bytes of the output record contain the number of logical records (4) and the length of a single logical record (80), respectively. This control information is provided by the phase.

On return from ZLEDI, the output is completed and a new record can be built in the output area.

ZPCH -- AO

This routine is used to produce a record on SYSPCH in overlapped mode. The length of each output record is 81 characters, the first character being a control character for stacker selection.

Two 80-byte I/O areas are specified for overlapped output in the uppermost Buffer Area otherwise used as I/O area for text input and output. No text input or output is performed during execution of a phase that contains ZPCH. Register 10 is used to point to the I/O area where the next record for output on SYSPCH can be built by the phase. This register must not be changed by any phase using ZPCH.

On each return from the routine, register 10 points to an alternate I/O area.

Prior to the first branch to ZPCH, the compiler phase must issue the macro instruction IP to load register 10 with the address of the first I/O area where the phase can build the first output record. The expansion of the IP macro instruction is:

```
L    ZREG10,IJKMPC
LA   ZREG10,1(ZREG10)
```

IJKMPC is an entry in the communication region and contains the output area address for the first output on SYSPCH. This address is stored during initialization.

ZPRNT, HESUB -- AP

The routine ZPRNT is used for any output on SYSLST. The routine automatically provides header lines and subheader lines on each page, using the subroutine HESUB.

Each header line contains compiler name, program number, change level, job name, and page number. The content of the subheader line depends on the compiler phase.

Each line is 120 characters long. Writing of single lines is done in overlapped mode. Therefore, two I/O areas are specified in the fifth work buffer. Register 11 is used as I/O register to point to the output area where the next record can be built by the phase for output on SYSLST. This register must not be changed by the phase containing ZPRNT. On each return from the routine, register 11 points to an alternate I/O area. The register content is saved in the entry IJKMPR in the interphase communication region at the end of the initialization and at the end of any listing phase.

Prior to the first branch to this routine, the compiler phase must issue the macro instruction IJKIL. The expansion of this macro instruction is:

```
L   ZREG11,IJKMPR
LA  ZREG11,1(ZREG11)
OI  IJKMJT+1,X'01'
```

The first instruction loads the register with the saved content. The register then points to the control character position. The second instruction causes the register to point to the first position of information to be given out. The last instruction sets a flag bit for printing in the interphase communication region, this flag bit is tested at phase end in the routine IJKAPH for end of phase.

ZMO -- AQ

This routine is used to move a record of any length from a FROM field to a TO field. The two fields may overlap. The routine is assembled in the source program of a compiler phase by the macro instruction IJKMO.

It can be called in the phase by branching to ZMO. There is the following convention on register contents for calling:

register 0	contains the field length,
register 1	contains the start address of the TO field,
register 2	contains the start address of the FROM field.

On return, registers 1 and 2 contain the end address+1 of the corresponding fields. The content of register 0 is undefined. All remaining registers are unchanged.

The routine first tests whether the start address of the TO field is lower than that of the FROM field. In this case, the routine IJKMN is called for moving. If the start address of the TO field is higher than that of the FROM field, the field is moved step by step from the right to the left. The move length for a single step is thereby calculated according to the following formula: Move Length = Min (256, field length, address difference). This move length implies a correct moving for any field overlapping.

LOGICAL IOCS FOR THE TAPE VERSION

Work Files SYS001, SYS002 and SYS003

The first control section of phase A00 contains the DTFMT tables for these work files . The name of this control section is IJXA00.

The file specifications are the same for all three work files, e.g.,

```
IJSYS01  DTFMT  BLKSIZE   = 3072
                DEVADDR   = SYS001
                EOFADDR   = K001EOF
                NOTEPNT   = YES
                RECFORM   = UNDEF
                TYPEFILE  = WORK
                MODNAME   = IJFWZNZZ
```

For IJSYS002 and IJSYS003, the blocksize entry is BLKSIZE = 1536.

The address of the input or output area is specified in the expansion of the respective READ or WRITE macro instruction. The file specification implies nonoverlapped working. Thus, overlapping, if any, is done by the control routines.

The logical IOCS module is not assembled in phase A00; it is included during linkage editing.

DTFCP Files for SYSLST, SYSIPT, SYSPCH and SYSLNK

For device independence, the logical IOCS used for these files is of the type DTFCP, CPMOD.

The file specifications for the print file IJSYSLS are assembled in phase A00 in the control section IJXA01. This DTFCP table remains in storage throughout the entire compilation (for exceptions see phase C95). The file specifications are:

```
IJSYSLS  DTFCP  DEVADDR   = SYSLST
                IOAREA1   = KTETA
                IOAREA2   = KTETA
                RECSIZE   = 121
                TYPEFLE   = OUTPUT
                IOREG     = 11
```

The module IJJCP0 is included in all printing phases during linkage editing.

The addresses of the two I/O areas are fixed in the Initialization routine. Output for listing is always done in overlapped mode, using two I/O areas and register 11.

The file specifications for the input file IJSYSIN are assembled in phase A00 in the control section IJXA06. This DTFCP table serves for input during phase A25. It is further used for closing the file IJSYSIN in the last compiler phase. It is therefore written on SYS001 in phase A30 in order to save storage during the other compiler phases. The file specifications are:

```
IJSYSIN   DTFCP   DEVADDR   = SYSIPT
                  IOAREA1   = KTETA
                  IOAREA2   = KTETA
                  RECSIZE   = 81
                  EOFADDR   = ELCO10
                  TYPEFLE   = INPUT
                  IOREG     = 9
```

The module IJJCP0 for this input and for listing is included in phase A00 during linkage editing. It is not overlaid during phase A25. Input is done in overlapped mode using two input areas and register 9. The input areas are fixed in the Initialization routine.

The file specifications for the punch output file IJSYSPH are assembled in the control section IJXA06. After this file is opened in the Initialization routine, the file table is written on SYS001 to save storage. It is reloaded into storage by phase G55 for punching and closing the file. The file specifications are:

```
IJSYSPH   DTFCP   DEVADDR   = SYSPCH
                  IOAREA1   = KTETA
                  IOAREA2   = KTETA
                  RECSIZE   = 81
                  TYPEFLE   = OUTPUT
                  IOREG     = 10
```

The module IJJCP0 for this output is included in phase G55 during linkage editing. Output is done in overlapped mode using two output areas and register 10. The output areas are fixed in the Initialization routine.

The file specifications for IJSYSLN are assembled in the control section IJXA06. After this file is opened in the Initialization routine, the file table is written on SYS001 to save storage. It is reloaded into storage by phase G55 for link file output and for closing the file. The file specifications are:

```
IJSYSLN   DTFCP   DEVADDR   = SYSLNK
                  IOAREA1   = KTETA
                  RECSIZE   = 322
                  TYPEFLE   = OUTPUT
```

The module IJJCP0 for this file is the same as for the punch file. Output is done in non-overlapped mode. The address of the output area is fixed in the Initialization routine. The first two bytes of the output record contain the number of logical records (4) and the length of a single record (80), respectively.

LOGICAL IOCS FOR THE DISK VERSIONS

Work Files SYS001, SYS002 and SYS003

Disks will normally be used as work files for the disk versions. Therefore, the first control section of phase A00 (DOS 16K) or A00D (DOS 32K) contains the DTFSD tables for these work files. This control section is named IJXA00 (DOS 16K) or IJXA00D (DOS 32K).

The logical IOCS module named IJGWZNZZ and used for these work files on disk is not assembled in phase A00 or phase A00D, but included during linkage editing. During the Initialization routine, the configuration is tested for tape media for these work files. If tapes are specified, phase A10 is loaded at the location of the tables and the module for the work files (see phase A10). The file specifications are the same for all three disk work files, e.g.,

```
IJSYS001  DTFSD   BLCKSIZE = 3072
                  DEVICE   = 2311
                  EOFADDR  = K001EOF
                  NOTEPNT  = YES
                  RECFORM  = UNDEF
                  TYPEFLE  = WORK
                  DELETFL  = NO
                  MODNAME  = IJGWZNZZ
```

For IJSYS002 and IJSYS003, the blocksize entry is BLCKSIZE = 1536.

DTFCP Files for SYSIPT, SYSLST, SYSLNK, and SYSPCH

The handling of these files is similar to that of the tape version. The exception is the parameter DISK = YES in some file specifications. This parameter is always given for the link file IJSYSLN. For the other three files, it is only entered for DOS 32K. Whenever I/O functions are requested for a file with the parameter DISK = YES, the module IJJCPD0 instead of IJJCP0 is included during linkage editing.

PHASES PL/IA00, A00D, A10 (INITIALIZATION) -- AR

The first phase loaded during a PL/I compilation is either PL/IA00 (for DOS 16K and TOS) or PL/IA00D (for DOS 32K). Phases A00 and A00D contain

the interphase communication region,

the interface, and

the initialization routine.

Phases A00 and A00D differ in the DTF tables and modules.

The PL/I compilation starts with the Initialization routine that performs some preparation for the entire compilation. If the disk version is used and tape drives are assigned to work files SYS001, SYS002, and SYS003, the Initialization routine calls phase A10 to overlay the DTF tables and module for disk work files.

Storage Map of Interface A00 for Disk 16K

The object module of A00 consists of 3 control sections. The first section (IJXA00) contains the LIOCS tables for work files and the module IJGWZNZZ, which is called from the relocatable library by the Linkage Editor.

The second section (IJXA01) contains the Save Area for LIOCS, the interphase communication region, the control routines which are always in storage, the LIOCS table for SYSLST, the area reserved for TABTAB, and the CP module, which is called from the relocatable library by the Linkage Editor.

The third section (IJXA06) contains the LIOCS tables for IJSYSIN, IJSYSPH, and IJSYSLN and the Initialization routine.

After opening the files, the tables for IJSYSPH and IJSYSLN are written on SYS001 during the initialization; they are reloaded into storage by the final output phases of the compiler. Thus, the storage area can be overlaid by all other compiler phases. The same applies to the table for IJSYSIN after phase A25.

The load point for most phases is IJXA05 + 4. Exceptions may be looked up in the Linkage Editor mapping for the compiler.

For a schematic representation for the storage layout for phase A00 see Figure 1.

```
┌────────┬──────────────────────────────────┐
│IJXA00* │LIOCS tables for work files       │
│        ├──────────────────────────────────┤
│        ├──────────────────────────────────┤
│        │SD module IJGWZNZZ**              │
│        ├──────────────────────────────────┤
├────────┼──────────────────────────────────┤
│IJXA01  │Save area for LIOCS (72 bytes)    │
│        ├──────────────────────────────────┤
│        │Interphase communication region   │
│        ├──────────────────────────────────┤
│        │                                  │
│        │Control routines that are always  │
│        │         in storage               │
│        │                                  │
│        │                                  │
│        ├──────────────────────────────────┤
│IJXA04  │LIOCS table for IJSYSLS           │
│        ├──────────────────────────────────┤
│IJXA05  │TABTAB                            │
│        ├──────────────────────────────────┤
│        ├──────────────────────────────────┤
│        │CP module IJJCP0***               │
│        ├──────────────────────────────────┤
├────────┼──────────────────────────────────┤
│IJXA06  │LIOCS table for IJSYSIN           │
│        ├──────────────────────────────────┤
│        │LIOCS table for IJSYSPH           │
│        ├──────────────────────────────────┤
│        │LIOCS table for IJSYSLN           │
│        ├──────────────────────────────────┤
│        │                                  │
│        │Initialization                    │
│        │                                  │
├────────┴──────────────────────────────────┤
│*   for 32K disk : A00D                    │
│**  for tape     : MT module IJFWZNZZ      │
│*** for 32K disk : IJJCPD0                 │
└───────────────────────────────────────────┘
```

Figure 1. Storage Map of IJXA00 for 16K Disk

Initialization for Tapes -- AS

This routine has two principal functions: storage allocation during PL/I compilation and opening of files. Storage allocation and some housekeeping functions depend on the type of the work file. Therefore, the routine shown in flow chart AS is used both in the tape version and in the disk version with tape work files. Opening of files is identical for the disk and tape versions.

For tape work files, the initialization begins with rewind commands for SYS001, SYS002, and SYS003. All program mask bits are set to 1; they will not be reset during the compilation. The buffer length is calculated and stored in the communication region entry IJKMBL. The formula for the calculation is as follows:

Buffer length = 1536 if available core storage more than 30K

$$\text{Buffer length} = 256\left(1+\text{FLOOR}\left(\frac{\text{AVC} - 10\text{K}}{4\text{K}}\right)\right)$$

where AVC is available core storage.

The remaining part of the initialization is identical for the tape and disk versions. It is discussed here together with the disk versions.

Initialization for Disk Versions -- AT, AU

As mentioned under Initialization for Tapes, part of this initialization is common to both the disk and the tape versions. The initialization has two principal functions: storage allocation during PL/I compilation and opening of files.

If the work files are tapes, the routine flowcharted in AS is used for storage allocation. If disks are used, the following is done: The program mask bits are set to 1. They will not be reset during the compilation. The buffer length is calculated and stored in the communication region entry IJKMBL. The rules for calculation are given below.

Available Core Storage (AVC)	Buffer Length
AVC ≥ 30K	1536
22K ≤ AVC < 30K	1024
18K ≤ AVC < 22K	768
14K ≤ AVC < 18K	512
10K ≤ AVC < 14K	256

Initialization for Tape and Disk Versions -- AU

The addresses of the I/O areas for SYS002 and SYS003 are calculated and stored in KTETA. These addresses depend on the uppermost available core storage address and the buffer length. The addresses for the files IJSYSLS, IJSYSIN, IJSYSPH and IJSYSLN are calculated and stored in the corresponding DTFCP tables.

The addressing concept is as follows: The two output areas for printing lie in succession in the fifth work buffer, beginning at its lower limit. The two input areas for reading lie in succession in the first output area otherwise used for SYS002. The output areas for the link and punch files lie in corresponding succession also in the first output area otherwise used for SYS002. The length of the tables depends on whether or not the parameter DISK = YES is specified.

The address of the entire Buffer Area is then calculated and stored in the interphase communication region entry IJKMBS.

The opening of files depends on the Job Control options given for the list, punch, and link files. If none of the options for these files are given, the job is terminated and a warning message is produced on SYSLST.

If any option is specified, the corresponding files are opened with their work files. Before fetching the next compiler phase (A25), the contents of registers 10 and 11 are saved in the communication region, and the LIOCS tables for the files IJSYSLN and IJSYSPH are written on SYS001.

PHASE PL/IA25 (REPLACEMENT OF KEYWORDS) -- BA

Phase A25 has the following functions:

1. to read the PL/I source program into storage;
2. to list the source program if the LIST option is on;
3. to count the statements and to print the number of the first statement per printed line;
4. to eliminate the comments from the source text and to replace them by one blank;
5. to replace (if the 48-character set is used) the combinations period-period and comma-period by colon and semicolon, respectively, and the alphabetic operators GT, GE, NE, NG, NL, LE, LT, NOT, OR, AND, and CAT by their 60-character equivalents. No replacements are performed within quotes. Moreover, the combination comma-period is not replaced, i.e., not interpreted as an end-of-statement delimiter if it is followed by a digit;
6. to translate the source text into the internal code shown in Figure 1;
7. to replace all identifiers that physically look like PL/I keywords by 3-byte keys (all other identifiers are replaced in phase A30);
8. to eliminate redundant blanks from the source text (the remaining blanks are eliminated in phase A30); and
9. to terminate the source program by the 3-byte end-of-program key (FFFFFF).

Phase Input

The input of this phase is the PL/I source program, which is provided in card-image format. Each card consists of 80 columns. Column 1 must be blank except for the last card, which is the DOS/TOS end-of-data-file card. Columns 2 to 72 are assumed to contain the source text. Columns 73 to 80 are not used.

Input Processing

To translate the source text into the internal code, a translate table of 256 bytes is used. This table describes an

left half-byte right	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	G	W	b	|											
1	1	H	X	;												
2	2	I	Y	:												
3	3	J	Z	(												
4	4	K	$	)												
5	5	L	#	,												
6	6	M	@	'												
7	7	N	-	+												
8	8	O		-												
9	9	P		¬												
A	A	Q	"	*												
B	B	R	.	/												
C	C	S	¢	=												
D	D	T	%	>												
E	E	U	?	<												
F	F	V	!	&												

Note: The free space in this table may be occupied by an extended character set, e.g., KATAKANA. The characters E0 to FF are reserved for internal keys.

Figure 1. Table Describing the Internal Code

isomorphism from the external into the internal code.

The cards are read one by one into a card area. The source text is then scanned by means of several translate-and-test tables. An end-of-card mark (X'FF') is set in column 73.

TRT table 1 contains non-zero function bytes for alphabetic characters, semicolon, quotation mark, slash, end-of-card mark and -- if the 48-character set is used -- for comma and period. This table is used to scan statements, the beginning of which has already been detected.

TRT table 2 contains only one non-zero function byte for the end-of-card mark. It is used to scan for the end of a comment or a string constant. Function bytes for asterisk or quotation mark are moved into the table as required.

TRT table 4 contains zero function bytes for all alphanumeric characters. All other bytes are ≠ 0. This table is used to scan for the end of an identifier.

TRT table 5, which is used to find the first significant character of a statement, contains a zero function byte only for the blank. All other characters except slash and end-of-card mark are mapped into the same non-zero function byte. Blanks and comments are replaced by one blank.

The Keyword Table

Whether or not an identifier is a keyword is determined by means of the keyword table. All keywords of equal length are grouped together. The groups are arranged according to their lengths. Each group is preceded by two bytes that contain the length and the number of keywords the group consists of. The keyword table forms the first part of the identifier table and, due to its size, is divided into two records. Every keyword is assigned a current number, starting at one for each record. This number and the record number (0 or 1) are part of the key the keyword is replaced by. X'80' is added to the current number if the keyword begins with the letter I - N.

The contents of records 0 and 1 of the keyword table are shown in Figures 2 and 3. Some space is left open in the table for additional keywords. This space is filled with blanks.

A table of the same format as the keyword table is used to replace text written in the 48-character set.

Phase Output

The text output is a continuous stream of delimiters, identifiers, constants, and keywords, the last being represented by 3-byte keys of the following format:

```
byte 0: identifier key = X'E1'
byte 1: record number (0 or 1)
byte 2: bit  0=1:  the corresponding key-
                   word begins with the
                   letter I,J,K,L,M, or N.
        bits 1-7:  current number. Number-
                   ing starts at 1 for each
                   record.
```

First Record:											
B	0001	KEY	0097	LOW	00A8	SINH	003F	ELSE	0051	INDEX	00F4
E	0002	SET	0018	SUM	0029	MARK	00C0	SIZE	0052	PRINT	0075
A	0003	BIT	0019	ALL	002A	CHAR	0041	READ	0053	ROUND	0076
F	0004	END	001A	ANY	002B	HIGH	0042	OPEN	0054	FIXED	0077
X	0006	GET	001B			BOOL	0043	EDIT	0055	FLOAT	0078
L	0008	PUT	001C	SIGN	0034	PROD	0044	PAGE	0057	LABEL	00F9
V	0009	ABS	001D	CEIL	0035	POLY	0045	LINE	00D8	ENTRY	007A
R	000A	MAX	009E	LOG2	00B6	DATE	0046	INRO	00D9	BEGIN	007B
U	000B	MIN	009F	ATAN	0037	FILE	0047	FROM	005A	KEYED	00FC
		MOD	00A0	TAND	0038	TIME	0048	SKIP	005D	LEAVE	00FE
DO	000D	EXP	0022	SIND	0039	NULL	00C9	MAIN	00E0	WHILE	007F
GO	000E	LOG	00A3	COSD	003A	CALL	004A				
TO	000F	TAN	0024	TANH	003B	ADDR	004B	TRUNC	0070		
BY	0010	SIN	0025	SQRT	003C	GOTO	004D	LOG10	00F1		
IF	0091	COS	0026	ERFC	003D	THEN	004E	ATAND	0072		
ON	0021	ERF	0027	COSH	003E	STOP	0050	ATANH	0073		

Figure 2. Contents of the Keyword Table (Record 0)

Second Record:							
WRITE	0101	NOSIZE	01A3	BUFFERS	0141	PRECISION	0163
CLOSE	0102	MEDIUM	01A5	REWRITE	0142	BACKWARDS	0164
INPUT	0183	CREATE	0126	DISPLAY	0143	KEYLENGTH	01E5
BASED	0104						
REPLY	0105	FORMAT	0127	OPTIONS	0145		
ERROR	0107	DIVIDE	0128	RETURNS	0146	SEQUENTIAL	0167
FLOOR	0108	SUBSTR	0129	ENDFILE	0147	UNBUFFERED	0168
LEAVE	0189	REPEAT	012A	KEYFROM	01C9	ZERODIVIDE	0169
		SYSIPT	012C	OVERLAY	014A	CONVERSION	016A
UNSPEC	0110	SYSLST	012D	ENDPAGE	014B	NOOVERFLOW	01EB
VERIFY	0111	SYSPCH	012E			CONTROLLED	016C
STRING	0113	COLUMN	012F	INTERNAL	01CC		
RECORD	0114			EXTERNAL	014D	CONSECUTIVE	016F
LOCATE	0195			REGIONAL	014F	ENVIRONMENT	0170
UPDATE	0116	DECIMAL	0135	ONSYSLOG	0150	NOUNDERFLOW	01F1
STREAM	0118	PICTURE	0126	OVERFLOW	0152		
BINARY	0119	BUILTIN	0137	BUFFERED	0156	NOCONVERSION	01F3
STATIC	011A	ALIGNED	0138	TRANSMIT	0158	EXTENTNUMBER	0174
PACKED	011B	INITIAL	01B9	PAGESIZE	0159	NOZERODIVIDE	01F5
OUTPUT	011C	INDEXED	01BA				
DIRECT	011D	DECLARE	013B	CHARACTER	015E	FIXEDOVERFLOW	0177
SYSTEM	011F	POINTER	013C	AUTOMATIC	015F	INDEXMULTIPLE	01F8
RETURN	0120	DEFINED	013D	KEYLENGTH	01E0	NOFIXEDOVERFLOW	01FB
REVERT	0121	NOLABEL	01BE	PROCEDURE	0161		
SIGNAL	0122	DYNDUMP	013F	UNDERFLOW	0162		

Figure 3. Contents of the Keyword Table (Record 1)

DESCRIPTION OF ROUTINES

Initialization -- BB

This is the beginning of the main routine. It initializes pointers, switches, etc., and reads in the first card of the source text.

ELCO -- BC

Secondary entry point: ELCO10

This is part of the main routine. It scans for the first significant character of the first or next statement (comments and blanks are bypassed). Control is transferred to FSTN after the first character has been found.

Secondary entry point: ELCO10
This entry point is used if the DOS/TOS end-of-data-file card has been reached. The phase is terminated and IJKPH is called to read in the next phase.

FSTN -- BD

This is part of the main routine. It counts the statements and moves the number of the first statement per line into print positions 1 - 6, right-aligned. It performs the scan over the statement by means of TRT table 1. When a non-zero function byte is found, the preceding source text is moved into the output buffer, and control is transferred to one of the following routines depending on the character found.

FSLA -- BE

If a slash is found, the next character is tested for *. If it is not, the routine returns 'false' to 4(LINK). If it is an asterisk, the end of a comment is searched for. The comment is replaced by a blank and the routine returns 'true' to (LINK).

FCOM -- BF

This subroutine is called if the 48-character set has been specified and a comma has been found in the source text. If the comma is followed by a period that is not followed by a digit, control is transferred to FSEM. Otherwise, control is returned to the calling program.

Secondary entry point: FSEM
This entry point is used to move a semicolon into the output area. Control is transferred to ELCO.

FQUO -- BG

The subroutine scans for the end of a string constant and moves the constant into the output buffer.

FPER -- BH

This subroutine is called if the 48-character set has been specified and a period has been found in the source text. If the period is followed by another period, the two periods are replaced by a colon.

FCMB -- BI

Calling sequence:

```
BAL   LINK,FCMB
DC    XL1'character 1'
DC    XL2'character 2'
```

Input parameter:
R1 points to the card area (character 1).

Output parameters:
R1 points to the character following character 1 in the source text.
RR = R1-1 if return 'false'.

The routine tests if character 1 in the card area is followed by character 2. If necessary, a new card is read and character 1 is moved into column 1 of the card area. The routine returns 'true' to 6(LINK) if character 1 is followed by character 2. Otherwise, it returns 'false' to 2(LINK).

FNCA -- BK

Secondary entry points: ZRCD, FNCA05

This subroutine reads a new card into the card area (if an end-of-data-file condition arises, control is transferred to ELCO10) and prints the preceding card. The new card is moved into print positions 20 - 99. All other positions are cleared to blank. If column 1 of the new card does not contain a blank, print positions 7 to 20 are filled with asterisks. Then pointers, switches, etc., are initialized, and control is returned to the calling routine.

FKEW -- BL, BM

This subroutine is called if the first character of an identifier has been found in the source text (card area). After the length of the identifier has been determined (if necessary, new cards are read in), the identifier is compared with all keywords of equal length. If a matching entry is found, the corresponding 3-byte key is moved into the output buffer. Otherwise, the identifier is moved unchanged unless the 48-character set has been specified and the identifier is a 48-character operand. In the latter case, the 60-character equivalent is moved into the output buffer.

FTKW -- BN

Input parameters:

RLEN =	length of identifier.
PTAB =	address of one of the two records of the keyword table or of the 48-character operands table.
PID =	address of the identifier to be compared.

Output parameters:

RKEY =	current number of the keyword the identifier matches with (if any match has been found).
PID =	unchanged.
RLEN =	unchanged.

The subroutine compares the identifier with every keyword of equal length. If no matching entry is found, the routine returns 'false' to 0(LINK). Otherwise, it returns 'true' to 4(LINK). RKEY is initialized with 1 and increased accordingly whenever a keyword or a group of keywords is skipped.

FKBU -- BO

Input parameters:
RKEY: current number of the keyword (part of the 3-byte key the keyword is replaced with).
KEY: = X'E100..' or X'E101..' , depending on whether the keyword is contained in the first or the second record.

RKEY is stored in the third byte of KEY, which is then moved into the output buffer.

Secondary entry point: FWBU
This entry point is used to move source text into the output buffer. If the entire text does not fit into the buffer, the buffer is filled with the first part of the text to be moved. The buffer is then put out on text medium. The remaining part of the text is moved into the buffer, left-justified.

Input parameters:

RR	= address of source text
R1	= end address of source text + 1
POUT	= pointer of output buffer
BULIM	= end address of output buffer

Output parameters:

RR	= R1
POUT	= next free address in output buffer

PHASE PL/IA30 (REPLACEMENT OF IDENTIFIERS) -- CA

Phase A30 has the following functions:

1. To build the name table NAMTAB and to put it onto SYS001.

2. To replace all identifiers that were not replaced in phase A25 by 3-byte keys.

3. To eliminate all redundant blanks from the source text.

4. To put out the LIOCS table for SYSIPT as the third record of LITAB (the first two records were put out in phase A00).

5. If the keyword PICTURE appears in the source text, the job-information bit 10 is set to 1.

6. Phase A35 is skipped if the keyword PICTURE does not appear in the source text.

7. Phase A45 is skipped if there are no character strings in the source text.

Phase Input and Output

The text input is a continuous stream of delimiters, constants, and identifiers. The identifiers physically identical with keywords were replaced by 3-byte keys in phase A25.

The text output is a continuous stream of delimiters (blanks have been eliminated), constants, and identifiers. The identifiers are represented by 3-byte keys of the following format:

```
byte 0 : identifier key = X'E1'
byte 1 : record number (0 or 1 if keyword,
         otherwise ≥ 2)
byte 2 : bit 0 = 1 : the corresponding
         identifier begins with one of the
         letters I-N
         bit 1-7 : current number starting
         at 1 for each record.
```

Interface with Other Phases

1. The begin and end address of the LIOCS table for SYSIPT is referred to by the external symbols IJKA06 and IJKA07, respectively. The table was generated by phase A00 and is still in storage.

2. The addresses of the two records of the keyword table are referred to by the external symbols REC1 and REC2. The keyword table was generated by phase A25 and is still in storage.

3. The third record of LITAB is noted and the information stored in KSAVE8. It is used in phase G55.

4. The second record of NAMTAB is noted and the information stored in IJKMIP+4. It is used in phase B25.

Format of the Name Table NAMTAB

The name table consists of at least two records, each of which is 1024 bytes long. The first two records are the keyword table described in phase A25. All other records consist of up to 127 entries and are terminated by an end-of-record key (X'FF'). The individual entries have the following format (which differs from that of the first two records):

L = length of the following identifier - 1
I = the identifier itself in internal code

DESCRIPTION OF ROUTINES

Initialization -- CB

This is the beginning of the main routine. The LIOCS table for SYSIPT is put out on SYS001 as the third record of LITAB. IJKNT is called and the NOTE information is stored in KSAVE8. It is tested whether any option other than LIST is specified. If not, phase G31 is called to terminate the compilation.

The keyword table is put out on SYS001 as the first two records of NAMTAB. IJKNT is called and the NOTE information is stored in IJKMIP+4. Pointers, switches, etc., are initialized and two records of text input are read. The record information table RECT is built up. It contains information on the records of NAMTAB which are built up in the table space. RECT has the format shown below:

```
┌────┬────┬────┬────┬────┬────┐──   ─────┬─┐
│ A  │ C  │ N  │ A  │ C  │ N  │        A │F│
└────┴────┴────┴────┴────┴────┤──   ─────┴─┘
```

A = address of the begin of one of the records
C = current pointer (initial value = A. Subsequently it points to the next free entry within the record).
N = record number
F = end of table = X'FF'
The last A before F is used as end address of the last record.

FIDE -- CC, CD, CE

This is part of the main routine. The input is scanned by means of a TRT table for digits, letters, periods, blanks, quotation marks, identifier keys, and end-of-program key. All other characters are bypassed. If one of the characters listed is found, the bypassed text is moved into the output buffer. It is tested whether the input pointer points to an address of the first buffer (FPIN) , and one of the following actions is taken depending on what character is found:

1. End-of-program key:
   Control is passed to FEND.
2. Quotation mark:
   The end of the string is searched for and the entire string is moved into the output buffer. The scan continues.
3. Identifier key:
   The identifier key is moved unchanged into the output buffer.
4. Blank:
   This and all following blanks are skipped. They are not moved into the output buffer.
5. Letter:
   The identifier beginning with the letter found is compared with all identifiers contained in the name table. If a matching entry is found, the identifier is replaced by the corresponding key. If not, the identifier is incorporated into the table and replaced in the source text by the identifier key. The current number of the identifier key is the number of the entry in the relative record. If the table space is full, the identifier is moved unchanged into the output buffer, and the overflow switch is set to 1.
6. Digit:
   If the digit found is the first digit of a floating-point constant, the E is replaced by the corresponding 3-byte key. If it is followed by a B (binary constant) the B is replaced accordingly.

FEND -- CF

This is the end of the main routine. It is called if the end-of-program key is found. The key is moved into the output buffer, and the buffer is put out on text output medium. The name table is put out on SYS001 in records of 1024 bytes.

If the overflow switch is on, e.g., if not all identifiers have been replaced yet, control is transferred to INIT04 for a further pass over the source text. If the overflow switch is off, the next phase is called. Phase A50 is called if there are no character strings in the source program. Phase A45 is called if the keyword PICTURE does not appear. Phase A35 is called in all other cases.

FPIN -- CG

FPIN is called each time the input pointer is increased. This routine ensures that the input pointer always points to an address within the first of the two input buffers. Whenever the input pointer exceeds its range, the contents of the second buffer are moved into the first one, and a new record is read into the second buffer in overlapped mode.

The end of a program is indicated by the end-of-program key X'FF'. If, however, the ending quotation mark of a string constant is missing, this key cannot be detected since a character string may contain any of the 256 characters. It is therefore necessary to test for an end-of-file condition after every call of IJKGI. If the end of the file has been reached, the last record in the first buffer is processed and control is transferred to FEND the next time FPIN is invoked.

FMBU -- CH

Secondary entry point: FMBUS

Input parameters:
RR : (general register) address of source text.
source text + 1.
POUT : pointer of output buffer.
BULIM : end address of output buffer. Output parameters:

Output parameters:
RR : = R1.
POUT : points to the next free address in the buffer.

This routine moves the source text into the output buffer. If not all the bytes to be moved fit into the buffer or if they do exactly fit, the buffer is filled with the first part of the text to be moved. The buffer is then put out on text output medium and the rest, if any, is moved into the buffer, left-justified.

PHASE PL/IA35 (PICTURES) -- CZ

This phase, which is called by phase A30 if the identifier PICTURE is detected in the input stream, has the following functions:

1. To check whether a picture is syntactically correct;

2. To determine the precision and the attributes of the data item represented by the picture and to pass this information to subsequent phases;

3. To transform each decimal fixed-point and decimal floating-point picture into an "Edit Pattern" and a "Pseudo Program" and to produce additional information on precision, sign characters, etc., of the corresponding data item. This is done to considerably reduce the library subroutine requirements and to speed up the object time picture editing.

Phase Input

The source text used as input is in the format described as output in phase A30. Thus, there are no blanks between syntactical units, and all identifiers are represented by 3-byte keys.

OUTPUT FORMATS

The output format of the individual PICTURE items is described in the following.

Character-String Pictures

Since character-string pictures can only contain the picture character X, it is possible to determine the length of the data item and then to eliminate the picture. The output format of character-string pictures is shown in Figure 1.

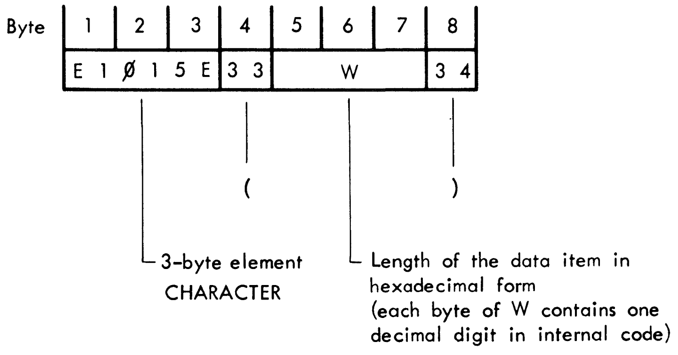

Byte
1
2
3
4
5
6
7
8
E 1 Ø 1 5 E
3 3
W
3 4
(
)
3-byte element CHARACTER
Length of the data item in hexadecimal form (each byte of W contains one decimal digit in internal code)

Figure 1. Output Format of Character-String Pictures

Sterling Pictures

The output format of sterling pictures is shown in Figure 2.

Decimal Fixed Pictures (Zoned)

If a decimal fixed picture contains the picture characters 9 and V only, it is possible to determine the precision of the data item and then to eliminate the picture. This data type is given the internal attribute ZONED. The output format of zoned decimal fixed pictures is shown in Figure 4.

Decimal Fixed Pictures (Zoned (T))

If a decimal fixed picture contains the picture characters 9, V, and T only, it is possible to determine the precision of the data item and then to eliminate the picture. This data type is given the internal attribute ZONED (T). The output format of zoned (T) decimal fixed pictures is shown in Figure 5.

Other Decimal Fixed Pictures

The output format of decimal fixed pictures other than zoned or zoned (T) is shown in Figure 6.

Decimal Float Pictures

The output format of decimal float pictures is shown in Figure 8.

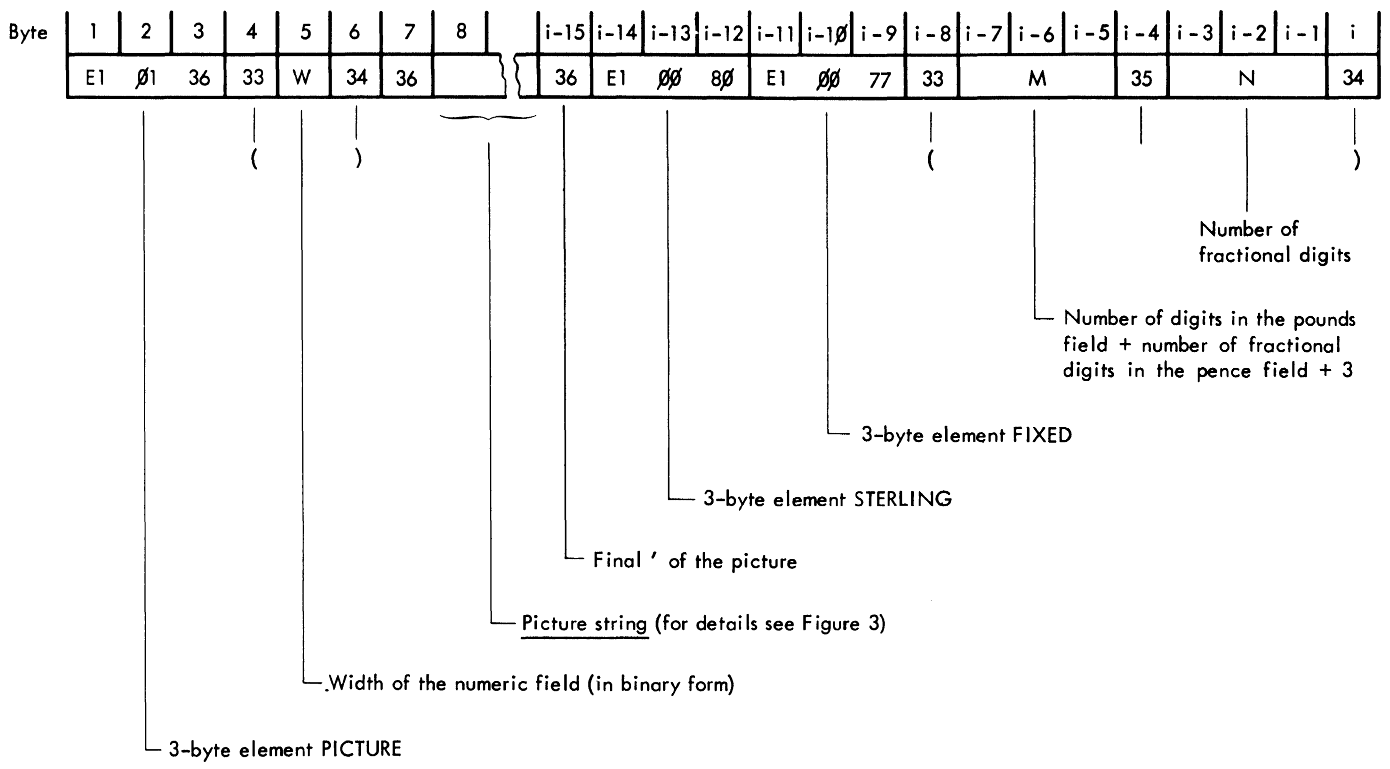

Byte
1
2
3
4
5
6
7
8
i-15
i-14
i-13
i-12
i-11
i-1Ø
i-9
i-8
i-7
i-6
i-5
i-4
i-3
i-2
i-1
i
E1
Ø1
36
33
W
34
36
36
E1
ØØ
8Ø
E1
ØØ
77
33
M
35
N
34
(
)
(
)
Number of fractional digits
Number of digits in the pounds field + number of fractional digits in the pence field + 3
3-byte element FIXED
3-byte element STERLING
Final ' of the picture
Picture string (for details see Figure 3)
Width of the numeric field (in binary form)
3-byte element PICTURE

Figure 2. Output Format of Sterling Pictures

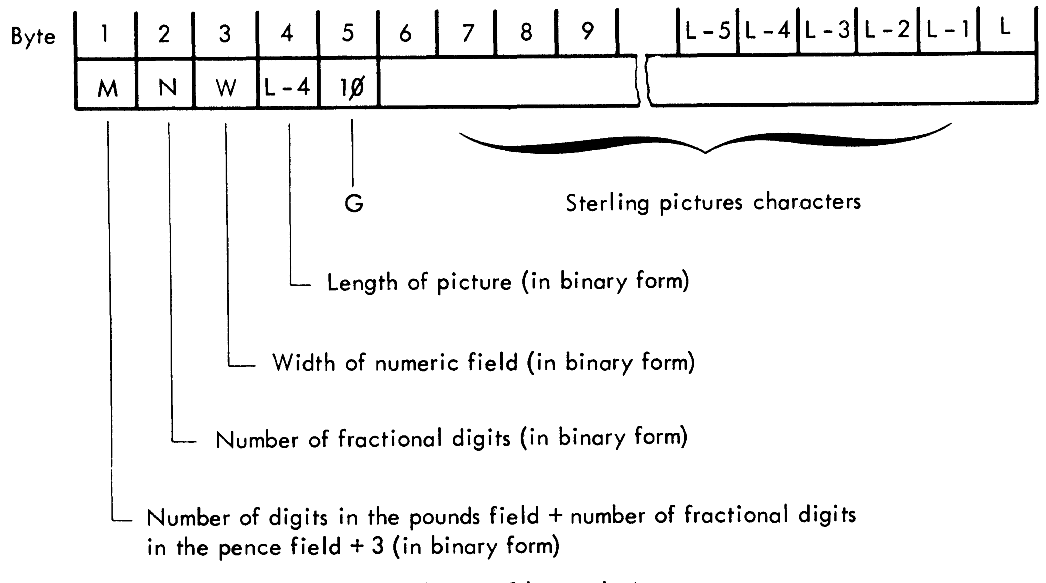

Byte
1
2
3
4
5
6
7
8
9
L-5
L-4
L-3
L-2
L-1
L
M
N
W
L-4
1Ø
G
Sterling pictures characters
Length of picture (in binary form)
Width of numeric field (in binary form)
Number of fractional digits (in binary form)
Number of digits in the pounds field + number of fractional digits in the pence field + 3 (in binary form)

Figure 3. String Format of Sterling Pictures

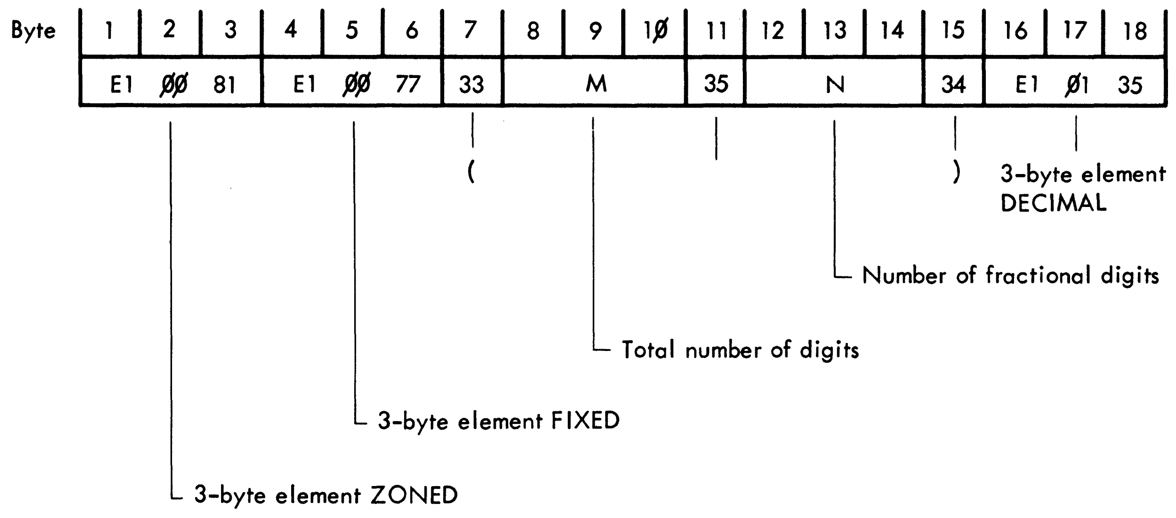


Figure 4. Output Format of Zoned Decimal Fixed Pictures

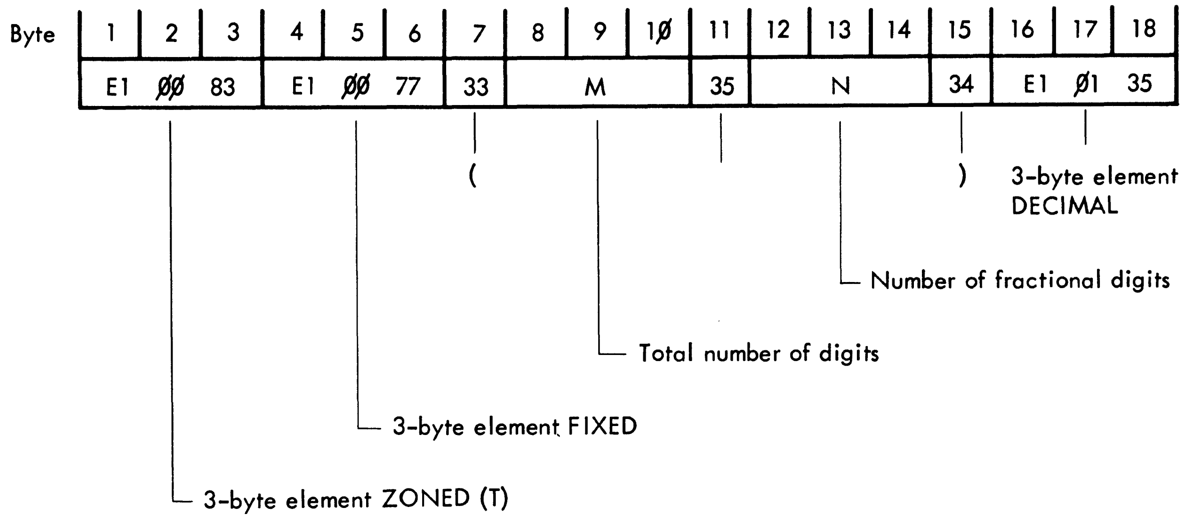


Figure 5. Output Format of Zoned (T) Decimal Fixed Pictures

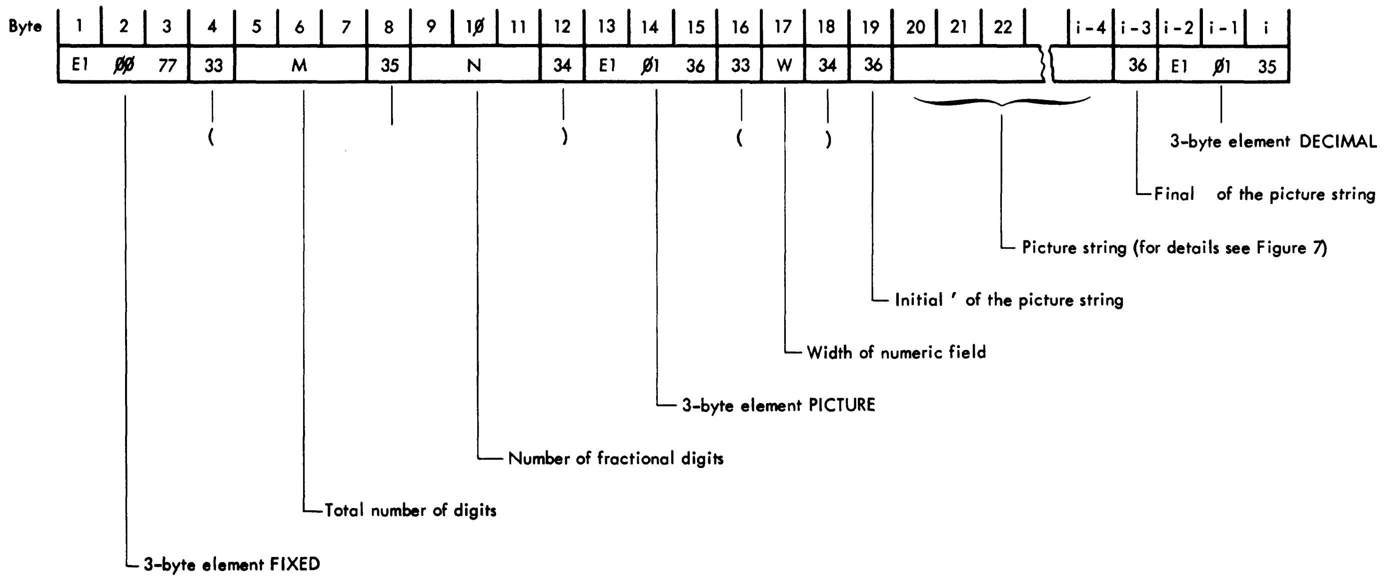

Byte
1 2 3 4 5 6 7 8 9 1Ø 11 12 13 14 15 16 17 18 19 20 21 22 i-4 i-3 i-2 i-1 i
E1 ØØ 77 33 M 35 N 34 E1 Ø1 36 33 W 34 36 36 E1 Ø1 35
(
)
(
)
3-byte element DECIMAL
Final ' of the picture string
Picture string (for details see Figure 7)
Initial ' of the picture string
Width of numeric field
3-byte element PICTURE
Number of fractional digits
Total number of digits
3-byte element FIXED

Figure 6. Output Format of Decimal Fixed Pictures Other than Zoned or Zoned (T)

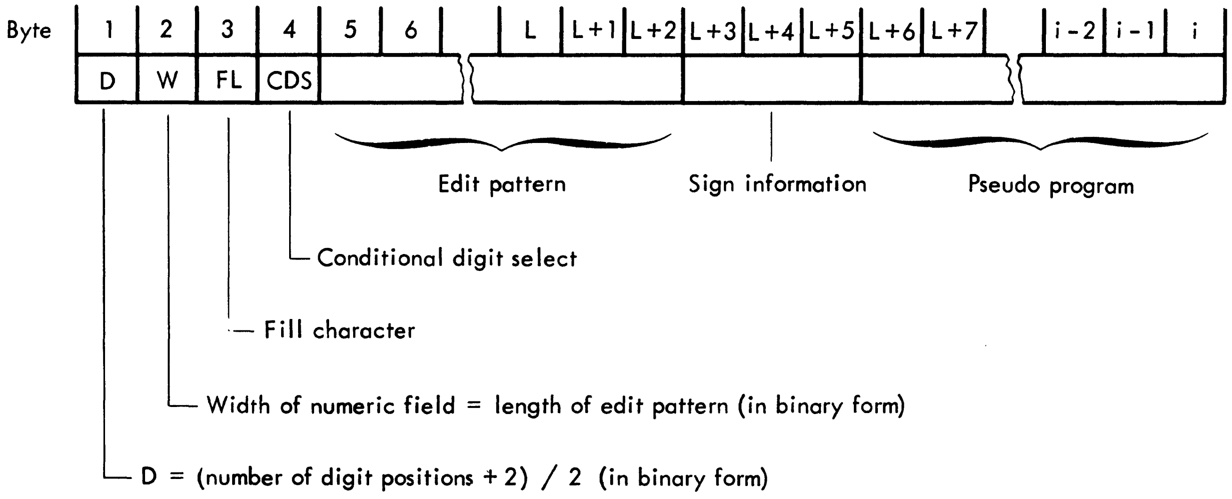

Byte
1 2 3 4 5 6 L L+1 L+2 L+3 L+4 L+5 L+6 L+7 i-2 i-1 i
D W FL CDS
Edit pattern
Sign information
Pseudo program
Conditional digit select
Fill character
Width of numeric field = length of edit pattern (in binary form)
D = (number of digit positions + 2) / 2 (in binary form)

Figure 7. String Format of Decimal Fixed Pictures Other than Zoned or Zoned (T)

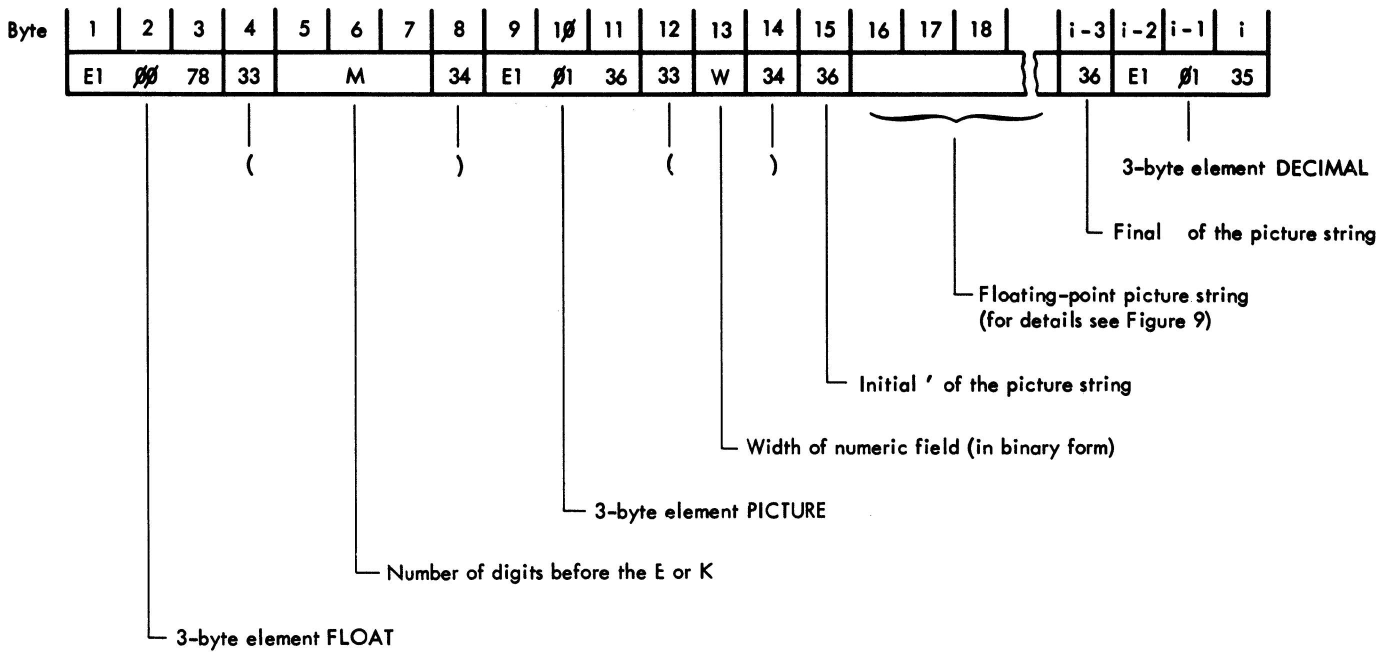

Byte
1 2 3 4 5 6 7 8 9 1ø 11 12 13 14 15 16 17 18 i-3 i-2 i-1 i
E1 øø 78 33 M 34 E1 ø1 36 33 W 34 36 36 E1 ø1 35
(
)
(
)
3-byte element DECIMAL
Final ' of the picture string
Floating-point picture string (for details see Figure 9)
Initial ' of the picture string
Width of numeric field (in binary form)
3-byte element PICTURE
Number of digits before the E or K
3-byte element FLOAT

Figure 8. Output Format of Decimal Float Pictures

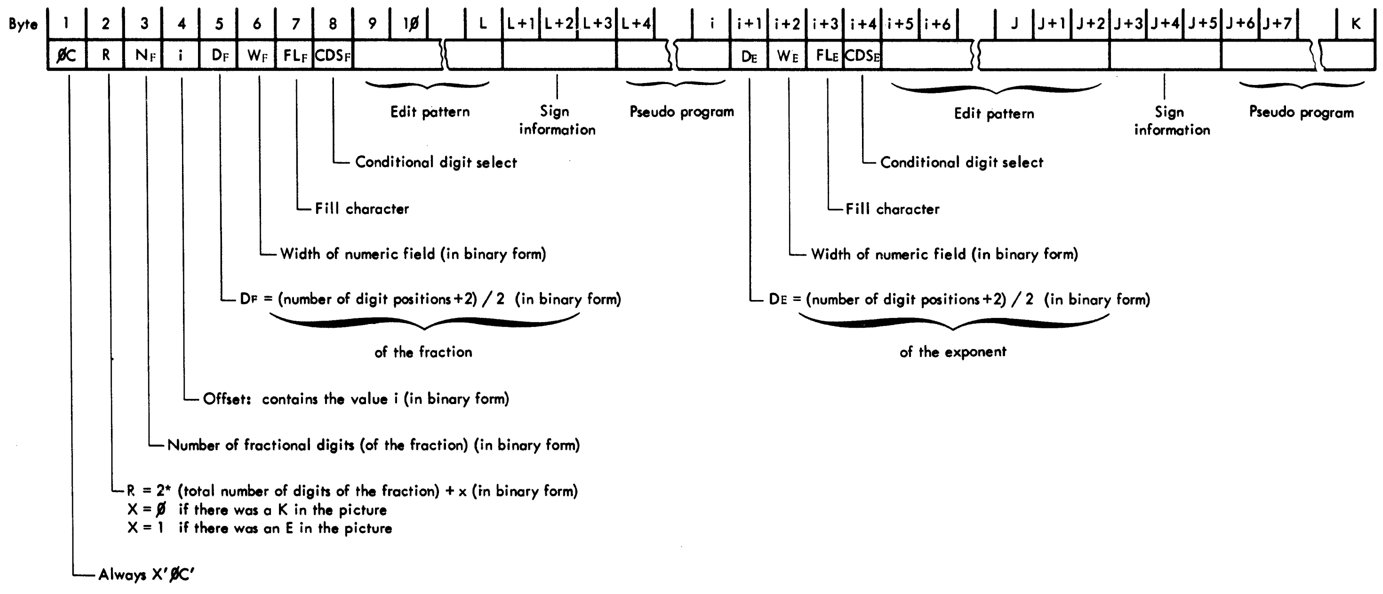

Byte
1 2 3 4 5 6 7 8 9 1ø L L+1 L+2 L+3 L+4 i i+1 i+2 i+3 i+4 i+5 i+6 J J+1 J+2 J+3 J+4 J+5 J+6 J+7 K
øC R NF i DF WF FLF CDSF DE WE FLE CDSE
Edit pattern
Sign information
Pseudo program
Conditional digit select
Fill character
Width of numeric field (in binary form)
DF = (number of digit positions+2) / 2 (in binary form)
of the fraction
Offset: contains the value i (in binary form)
Number of fractional digits (of the fraction) (in binary form)
R = 2* (total number of digits of the fraction) + x (in binary form)
X = ø if there was a K in the picture
X = 1 if there was an E in the picture
Always X'øC'
Edit pattern
Sign information
Pseudo program
Conditional digit select
Fill character
Width of numeric field (in binary form)
DE = (number of digit positions+2) / 2 (in binary form)
of the exponent

Figure 9. String Format of Decimal Float Pictures

ELEMENTS OF PICTURE STRINGS

The fill character (FL, FLF, and FLE) is blank if the picture (subfield) contains no asterisk. It is asterisk if the picture (subfield) contains an asterisk.

The conditional digit select character (CDS, CDSF, and CDSE) is X'20' if the precision (of the subfield) is even and blank if it is odd.

Edit Pattern

The edit pattern is used in EDMK instructions (at object time) in some library subroutines.

Sign Information

The sign information is a (pseudo) instruction specifying where to test for which sign. There are 5 different pseudo instructions containing sign information.

1. NSS - No Sign Specified
   The general format of the NSS pseudo instruction is shown below.

```
byte    0   X'40'
bytes 1-2   not relevant
```

   The instruction is generated if none of the picture characters S + - DB CR T I R appears.

2. TSS - Test for Static Sign
   The general format of the TSS pseudo instruction is shown below.

```
byte 0  X'00'
byte 1  offset of the last byte of the
        drifting string

byte 2 - for $
       b for +
       - for -
       C for CR
       D for DB
```

   It is generated if a static sign appears in the picture. If C1 appears in the specified byte, it is assumed that the data is negative. Otherwise, it is assumed to be positive.

3. TDM - Test for Drifting Minus
   The general format of the TDM pseudo instruction is shown below.

```
byte 0  X'00'
byte 1  offset of the last byte of the
        drifting string
byte 2 - for $
       b for +
       - for -
       C for CR
       D for DB
```

   It is generated if a string of drifting "-" appears in the picture. If "-" appears in the specified field, the data is assumed to be negative. Otherwise, it is assumed to be positive.

4. TDP - Test for Drifting Plus
   The general format of the TDP pseudo instruction is as shown for the TDM instruction, but with X'D0' in the first byte. It is generated if a string of drifting "+" appears in the picture. If "+" appears in the specified field, the data is assumed to be positive. Otherwise, it is assumed to be negative.

5. TOS - Test for Overpunched Sign
   The general format of the TOS pseudo instruction is shown below.

```
byte 0  X'F0'
byte 1  specifies offset of sign byte
byte 2  X'D0' for T
        X'F0' for I
        X'D0' for R
```

   It is generated if a T, I, or R appears in the picture. If the zone in the specified byte is identical to the zone in byte 3 of the instruction, the data is assumed to be negative. Otherwise, it is assumed to be positive.

Pseudo Program

The pseudo program is a series of (pseudo) instructions used by the library for editing (at object time). There are 5 different pseudo instructions.

1. IZR - Insert Zero
   The general format of the IZR pseudo instruction is shown below.

```
byte 0  X'00'
byte 1  offset
```

   The instruction causes the insertion of a zero in the byte with the offset d.

2. IST - Insert Static Character
   The general format of the IST pseudo instruction is shown below.

```
byte    0   X'04'
byte    1   offset
bytes 2-3   C1 and C2.
            For S : C1 = +    C2 = -
                + : C1 = +    C2 = b
                - : C1 = b    C2 = -
                S : C1 = $    C2 = $
```

   The instruction inserts C1 into the specified byte if the data value is greater than or equal to zero. Otherwise, it inserts C2.

3. <u>IDR - Insert Drift Character</u>
   The general format of the IDR pseudo instruction is as shown for the IST instruction, but with X'08' in the first byte. If the data value is greater than or equal to zero, C_1 is inserted into the byte with the offset l. Otherwise, C_2 is inserted. The value of l is determined as follows: if register 1 was set in an EDMK instruction, l is equal to the contents of register 1 minus 1. Otherwise, l is equal to address of the byte with the offset d.

4. <u>IZN - Insert Zone</u>
   The general format of the IZN pseudo instruction is shown below.

```
byte   0   X'0C'
byte   1   offset
bytes 2-3  Z1 and Z2.
       For T : Z1 = X'C0'   Z2 = X'D0'
           I : Z1 = X'C0'   Z2 = X'F0'
           R : Z1 = X'F0'   Z2 = X'D0'
```

   If the data value is greater than or equal to zero, the zone of Z_1 is inserted into the zone of the specified byte. Otherwise, the zone of Z_2 is inserted.

5. <u>EOP - End of Pseudo Program</u>
   The general format of the EOP pseudo instruction is X'14' if the picture does not contain 9, T, I, or R. X'10' is generated in all other cases. It indicates the end of the pseudo program. If the byte contains X'14' and the data value is zero, the entire field is filled with the fill character FL (or FLF or FLE).

EXAMPLES

The examples in Figure 11 show the original picture and the resulting edit pattern, sign information, pseudo program, fill character, and conditional digit select. The notation used in these examples is as shown in Figure 10.

Symbol	Meaning	
DS	Digit select	
SS	Significance start	
NSS	Pseudo instruction	Sign information
TSS	Pseudo instruction	
TDM	Pseudo instruction	
TDP	Pseudo instruction	
TOS	Pseudo instruction	
IZR	Pseudo instruction	Instructions for the pseudo program
IST	Pseudo instruction	
IDR	Pseudo instruction	
IZN	Pseudo instruction	
14	Pseudo instruction	
10	Pseudo instruction	
$	Picture character	
9		
V		
.		
.		
b	Blank	
0	0, 1, 2, . . .	
1		
.		

Figure 10. Notation Used in Picture Examples

ORIGINAL PICTURE	OUTPUT		ORIGINAL PICTURE	OUTPUT	
S 9 9 . V 9 9	Fill char.	b	$ * * T	Fill char.	*
	Cond. digit sel.	DS		Cond. digit sel.	b
	Edit pattern	b SS SS . SS SS		Edit pattern	b DS DS DS
	Sign inform.	TSS Ø -		Sign inform.	TOS 3 DØ
	Pseudo program	IST Ø + - IZR 1 1Ø		Pseudo program	IST Ø $ $ IZR 3 IZN 3 CØ DØ 1Ø
$ $ $ $ 9 9	Fill char.	b	$ $ $ $, 9 9 9 V 9 9	Fill char.	b
	Cond. digit sel.	b		Cond. digit sel.	b
	Edit pattern	b DS DS SS SS SS		Edit pattern	b b DS SS , SS SS SS SS SS
	Sign inform.	NSS		Sign inform.	TSS Ø -
	Pseudo program	IDR 3 $ $ 1Ø		Pseudo program	IST Ø + - IDR 4 $ $ 1Ø
+ + +	Fill char.	b	S S S , S S S . V S S	Fill char.	b
	Cond. digit sel.	DS		Cond. digit sel.	b
	Edit pattern	b DS DS		Edit pattern	b DS DS , DS DS DS . SS SS
	Sign inform.	TDP 2 2		Sign inform.	TDM 7 7
	Pseudo program	IDR 2 + - 14		Pseudo program	IDR 7 + - IZR 8 14
* * V *	Fill char.	*	+ + + 9	Fill char.	b
	Cond. digit sel.	b		Cond. digit sel.	b
	Edit pattern	DS DS DS		Edit pattern	b DS SS SS
	Sign inform.	NSS		Sign inform.	TDP 2 2
	Pseudo program	14		Pseudo program	IDR 2 + b 1Ø
9 , V 9	Fill char.	b	S S S B , B S S	Fill char.	b
	Cond. digit sel.	DS		Cond. digit sel.	DS
	Edit pattern	SS SS		Edit pattern	b DS DS b , b DS DS
	Sign inform.	NSS		Sign inform.	TDM 7 7
	Pseudo program	IZR Ø 1Ø		Pseudo program	IDR 7 + - 14

Figure 11. Examples of Picture Transformation in Phase A35

DESCRIPTION OF ROUTINES

Pointers, Storage Areas, and Flags

Phase A35 uses the following pointers, storage areas, and flags:

PIN	input pointer
POUT, OPOINT	output pointer
WAP, XP, PUN	work area pointers
PP	pseudo program pointer
EP	edit pattern pointer
PIP13	contains expanded picture
EDIP	contains edit pattern
PSEUP	contains pseudo program
SIN	contains sign information

Name of flag	Set on for
$	$
+	+
-	-
S	S
T	T
I	I
R	R
CR	CR
DB	DB
Z	Z
*	*
9	9
V	V
EK	E or K
PCB	period, comma, B
Digpos	scanned digit position character
H	to check that a static S + - appears only to the left or right of a subfield or that a DB or CR appears only to the extreme right of a subfield
Sign	sign character (S + - T I R CR or DB)
U	zero suppression requested
$ drift	drifting string consists of $
+ drift	drifting string consists of +
- drift	drifting string consists of -
S drift	drifting string consists of S

Notation and Terms Used in Routine Descriptions

M,m	total number of digit positions
N,n	number of digit positions after the (implied) decimal point
W,w	width of numeric field
FL	fill character
CDS	conditional digit select
"item code"	a number assigned to a picture character (for example: $ has the "item code" 1, + has the "item code" 2, etc.)
(Open)	A routine is called open if control is transferred to it by 1. a simple B instruction, in which case control is also returned by a B instruction, or 2. some in-line coding that requires a separate description.
(Closed)	A routine is called closed if control is transferred to it by a BAL instruction. Control is returned by a BR instruction in this case.

PIP1 -- DA, DB

PIP1 is the "master program" of this phase. It initializes pointers, registers, and other items and reads the first 2 records into input buffers 1 and 2. It scans the input stream until X'E1013636' (the internal representation of PICTURE') is encountered outside a character-string constant. EXPA stores the "expanded" picture in PIP13.

If the picture is a character-string picture, STRI is called. If it is a sterling picture, STER is called. These routines process the pictures and return control to PIP1, which continues the scan.

If the picture is a fixed-decimal picture, DEC is called. Control is then transferred to DEC. If DEC determines that the picture is ZONED or ZONED (T) (see Figures 4 and 5), the 3-byte element DECIMAL is put out, and the scan is continued. Otherwise, P2 is called and FIXED (M,N) is put out. P1 is then called and puts out PICTURE (W), quote, the first 2 bytes of the picture string, and -- indirectly (by calling other routines) -- the remainder of the picture string. The scan then continues.

If the picture is a decimal-float picture, DEC is called which processes the fraction part of the picture. Control is then returned to PIP1 and the values m, n, w, and d of the fraction are stored. Then DEC732 is called which processes the exponent part. After control has been returned to PIP1, M, W, and D of the exponent are stored and w of the entire picture is computed. FLOAT (M) PICTURE (W) is put out. TPEP is called which transforms the fraction part of the picture into an edit pattern and a pseudo program. Then a quote and the first 6 bytes of the picture string are put out. PAT puts out the next 2 bytes of the picture string (see Figure 9), the edit pattern, the sign information, and the pseudo program of the fraction part. Control returns to PIP1, and d and w of the exponent are put out. TPEP is called again to process the exponent part. PAT is called, which puts out the next 2 bytes of the picture string, the edit pattern, the sign information, and the pseudo program of the exponent part. Then, DECIMAL is put out and the scan continues.

<u>BANN (Closed) -- DZ</u>

The following instructions are stored in PSEUP:

1.	IST	(EP)	$	$	if current picture character is $
2.	IST	(EP)	+	b	if current picture character is +
3.	IST	(EP)	b	-	if current picture character is -
4.	IST	(EP)	+	-	if current picture character is S

The following sign information is stored:

1.	NSS			if current picture character is $
2.	TSS	(EP)	b	if current picture character is +
3.	TSS	(EP)	-	if current picture character is -
4.	TSS	(EP)	-	if current picture character is S

<u>BLUE (Closed) -- DO</u>

Adds 1 to N if the V-flag is on.

<u>DEC (Closed) -- DE, DF, DG, DH</u>

1. It checks whether a decimal fixed-point picture or a subfield of a decimal floating-point picture is syntactically correct.

2. It determines the precision of the data item corresponding to the picture, determines the width of the numeric field, and computes D = (number of digit positions+2)/2.

3. If the picture is a zoned decimal fixed picture, ZONED FIXED (m,n) is put out, and control is returned to PIP1.

4. If the picture is a zoned (T) decimal fixed picture, ZONED (T) FIXED (m,n) is put out, and control is returned to PIP1.

5. If the picture is not a decimal fixed-point picture or a subfield of a decimal floating-point picture, control is returned to PIP1.

<u>DPTE (Closed) -- DO</u>

Signals error if the Digpos-flag is off.

<u>DRIFT (Closed) -- DV</u>

1. Sets "drift switch" on.

2. Sets "item code" to

   1 if currently scanned character is $
   2 if currently scanned character is +
   3 if currently scanned character is -
   4 if currently scanned character is S

3. Returns to (LINK) if it is none of the characters indicated under item 2.

4. If the "drift switch" is on, the currently scanned character and the "item code" are stored.

5. If the "drift switch" is off, ("item code"-1)*4 and the currently scanned item are stored.

6. Returns to 4(0,LINK).

<u>ECAV (Open) -- DX</u>

1. Sets the SS-flag on if the V-flag is off.

2. If the V-flag is on and the SS-flag is off, it puts out an IST instruction for each period, comma, or B between the current PICTURE character and the V.

3. Returns before setting the S-flag on.

<u>EXPA (Open) -- DC, DD</u>

Expands a picture using replication factors. The expanded picture is stored in PIP13. An error is indicated if the expanded picture contains more than 255 characters. After the expansion, the picture is converted to external representation.

Examples:

Unexpanded picture	Expanded picture
'(5)9.V9'	'99999.V9'
'(3)S,V(2)9'	'SSS,V99'
'(0)Z(4)9'	'9999'

A zero replication factor followed by a picture character is interpreted as shown in example 3.

<u>F(Y) (Closed) -- DR</u>

Scans the field "separator-1" if R0 = 0.
Scans the field "separator-2" if R0 ≠ 0.

FPIN (Closed) -- EB

Controls the reading of the input stream, i.e., FPIN is called each time the input pointer is increased (only exception: when scanning a replication factor). Two buffers are used. If the input pointer passes the end of the first buffer, the contents of the second buffer are moved into the first, and a new record is read into the second buffer. The input pointer is modified accordingly. Additionally, all the input text (with the exception of pictures) is put out again.

FSI, FSI1 (Closed) -- DY

1. Sets the FF-flag on.

2. Adds a "Significance Start" character (X'21') to the edit pattern.

3. Adds an IZR (EP) to the pseudo program if the above "Significance Start" character is the first one in this edit pattern.

HAM (Closed) -- DS

1. Increases WAP by 1. Adds 1 to M and N if the currently scanned digit position character was preceded by a V.

2. Returns to 4(0,LINK) if the Z-flag is on.

3. Returns to (LINK) if the Z-flag is off.

HTE (Closed) -- DO

Signals error if the H-flag is on.

JTRNA1 (Closed) -- EC

Output routine. Register BYZ contains the number of bytes to be put out, register PIN the starting address. One output buffer is used.

If the string to be put out fits into the remaining free space of the output buffer, it is moved there, and BYZ is added to POUT (thus updating the output pointer). If the string is too long, the string length required to fill the buffer is moved there. The contents of the buffer are written on the output medium, and POUT is reset to the begin address of the buffer. BYZ is reduced by the number of bytes moved into the buffer and PIN is added to that number. This procedure is continued until the entire string has been moved.

PAT (Closed)

This routine is called by PIP1. It puts out FL, CDS, Edit Pattern, Sign Information, and Pseudo Program of a subfield. An additional ' is put out if it is a decimal fixed-point field.

RFDF (Closed) -- DS

Signals error if any Drift flag and/or "normal" flag (specified by the "item code") is on.

Examples:

Error is signalled if

1. the "item code" specifies $ and the $-flag is on.

2. the "item code" specifies + and the $ drift-flag is on.

RFT (Closed) -- DP

Returns to (LINK) if the flag specified by the "item code" is on; otherwise, it returns to 4(0,LINK).

SEAV (Closed) -- DO

1. Scans for period and/or comma and/or B and/or V following a (potential) drifting character. Stops scanning after the first non-editing character is encountered.

2. Signals error if any editing character occurs without the Digpos-flag having been set on.

3. The PCB-flag is set on if one or more of the editing characters have been detected.

4. VTE is called if a V is encountered. If the Digpos-flag is off, the Q-flag is set on. Then HTE is called.

SITE (Closed) -- DO

Signals error if the sign-flag is on. Otherwise, the sign-flag is set on.

STER (Open) -- DI, DJ, DK, DL

1. Checks whether a sterling picture is syntactically correct.

2. Determines the precision of the data item corresponding to the picture:

   M = number of digit positions in the pounds field + 3 + N.
   N = number of fractional digits in the pence field.

3. Determines the width of the numeric field : W.

4. Puts out PICTURE (W) 'picture string' STERLING FIXED (M,N).

STRI (Open) -- DN

Processes character-string pictures, e.g., pictures that only contain the picture character X. Puts out CHARACTER (W). Deletes the picture.

SUP (Closed) -- DP

1. Returns to the address in R1 if the character being scanned is neither * nor Z. Otherwise, the *-flag or Z-flag is set on, respectively.

2. Signals error if any flag is on that represents a digit position character other than the one being scanned.

TDS (Closed) -- EA

Generates the sign information:

TDM	(EP)	a	if the currently scanned character is - or S.
TDP	(EP)	a	if the currently scanned character is +.

a = length of the drifting string - 1

TPEP (Closed) -- DT, DU, DV, DW

Transforms a decimal picture subfield into edit pattern, sign information, and pseudo program.

The relationship between the input of TPEP and the output produced can be seen in the section Examples in the description of this phase.

Note: If the routine detects an invalid picture character, all storage occupied by this phase is dumped. (The error will probably be in DEC, or in one of the routines called by DEC).

VTE (Closed) -- DO

Signals error if the V-flag is on. Otherwise, the V-flag is set on and the U-flag is set off.

WTE (Closed) -- DO

Signals error if the W-flag is set on. Otherwise, it returns to (LINK).

Z9 (Closed) -- DQ

1. Scans a shilling field or a pence field.

2. Returns to 4(0,LINK) if ZZ or Z9 or 99 is detected.

3. Returns to (LINK) if Z or 9 is detected.

4. Signals error if the character being scanned is neither 9 nor Z.

5. Sets the Z-flag off if a 9 is detected (provided the Z-flag was on). Signals error if the Z-flag is off and the character being named is a Z (if there is a Z in a sterling subfield, all digit position characters in the preceding subfield must be Zs).

Error -- DM

Since there are more than 50 error possibilities in a picture, only one error message is produced by putting out a 3-byte element ERROR of the general hexadecimal format E10082.

If an error is detected, processing of the picture is terminated. The message code 01 (declaration in error) will be given in the symbol table listing.

PHASE PL/IA45 (CHARACTER STRINGS) -- EM

This phase has the following functions:

1. To scan character string constants for correct format and precision;

2. To eliminate character strings from the source text and to replace them by two 3-byte keys;

3. To retranslate the character strings (except the picture strings) from the internal to the external code, and to collect them in the character string table CARTAB which is written on SYS001;

4. To optimize the character strings within the limits of table space, e.g., to cause two identical character strings to appear only once in the table.

This phase is skipped if no character strings appear in the source text.

Phase Input and Output

The text input is a continuous stream of delimiters, constants, and identifiers, the latter being represented by 3-byte keys.

The character-string constants have the following format:

```
┌──          ──┐
│ ┌─────────┐  │  ┌─────────────────────────────┐
│ │  ( I )  │  │  │'character          string'  │
│ └─────────┘  │  └─────────────────────────────┘
└──          ──┘
```

I = decimal integer specifying replication (1 to 3 digits).

The character string is a sequence of at least one character. (A quotation mark is represented by two adjacent quotation marks.) All characters, except those of picture strings, are represented by the internal code. If the character-string constant is followed by a B (in the form of a 3-byte key), it is interpreted as a bit-string constant. Picture strings appear in the following context:

PICTURE	(	length of field	)	'character-string'

Note: The field length must not be interpreted as a replication factor.

The text output is a continuous stream of identifiers (3-byte keys), delimiters, and constants. The character-string constants are represented by two adjacent 3-byte keys of the following format:

K	O	K	E	L

K = character-string-constant key = X'E3' (1 byte)

O = offset in the character string table (2 bytes)

E = error byte : X'00' = no error
bit 0 = 1 : diagnostic message E055I
bit 1 = 1 : diagnostic message E056I
bit 2 = 1 : diagnostic message E067I

L = length of the character string (1 byte).

Note: The error byte is cleared in phase C30 so that the length occupies two bytes as follows:

K	O	K	L

The Character-String Table:

The character string table (CARTAB, ZTAB00) is written on SYS001. The record length is equal to the buffer length. The last record may be shorter. The length of the table is stored in IJKCSL.

The table consists of a continuous stream of characters (in external code). Each single character string can therefore be found by its address (offset) relative to the beginning of the table and by the length of the string. The character strings are optimized as shown below. Assume that the following 4 character-string constants appear in a source program:

'SPARGELDER', 'GELD', 'ERBE', 'DERB'.

This will cause the following character-string table to be built:

```
┌─────────────────────────┐
│S P A R G E L D E R B E│
└─────────────────────────┘
```

The character-string constants would be represented as follows:

```
X'E30000E3000A'
X'E30004E30004'
X'E30008E30004'
X'E30007E30004'
```

DESCRIPTION OF ROUTINES

Initialization -- EN

This is the beginning of the main routine. It initializes pointers, switches, etc., and reads input text into two buffers.

FSCA -- EO ,EQ

This is part of the main routine of this phase. It performs the scan over the source text by means of several TRT tables. If a marked character is found, the preceding source text is moved into the output buffer. FPIN is called to check the input pointer, and control is transferred depending on the character found.

FIDE -- EP

This is part of the main routine. It is called if an identifier key has been found. If it is a PICTURE key that appears in the correct context of picture strings, all the text preceding the begin quote of the string is moved into the output buffer, the switch CONVSW is set to one to indicate that the string is not to be retranslated, and, after calling FPIN, control is transferred to FSTR. All other keys are moved into the output buffer, FPIN is called, and the scan continues. If the key INITIAL is found, the entire INITIAL list is skipped.

FREP -- ER

This is part of the main routine. It is called if a left parenthesis has been found in the source text. If the parenthesis is followed by 1 to 3 digits, a right parenthesis, and a quote, i.e., if a replication factor is found, the decimal integer is converted to binary and control is transferred to FSTR05 to process the string constant. Otherwise, the left parenthesis is bypassed, and the scan continues.

FSTR -- ES, ET

Input parameter:

R1	:	Points to the beginning quotation mark of a character string within the input text.

The routine increases R1 by one, sets REPL to 1 and RLEN to 0. REPL and RLEN are the parameters for the following routine segment labeled by FSTR05.

Secondary entry point: FSTR05

Input parameters:

R1	:	address of the character string in the input text.
REPL	:	replication factor.
RLEN	:	number of digits specified for the replication factor + 2. RLEN equals zero if no replication factor has been specified.

The input text is scanned by means of a TRT table for a quote indicating the end of the character string. If the string is a bit string, it is moved unchanged into the output buffer. The character string is moved into CHST. Strings exceeding the length of 255 are truncated on the right. For two successive quotes, one quote is moved into CHST. One blank is moved if the begin quote is immediately followed by the end quote. Control is then transferred to FCTA.

FCTA -- EU,EV

This is part of the main routine.

Input parameters:

REPL	:	replication factor.
RLEN	:	length of basic string.
CONVSW	:	switch to indicate whether the string must be retranslated (=0) or not (=1).
CTAB	:	address of the character-string table in storage.
CTAB1	:	current pointer to this table.
ADABS	:	address that must be subtracted from CTAB1 to obtain the offset in the character-string table, a part of which may already have been put out on SYS001.
TABE	:	end address of the character-string table in storage.
R1ST	:	address of the end quote of the character string.

If the replication factor is 0, it is ignored and set to one. If necessary, the basic string is retranslated into the external code. The string is expanded according to the replication factor. Strings exceeding the length of 255 are truncated on the right.

The string is then compared with all sequences of characters of equal length in the character-string table. If a matching entry is found, the same offset is used in the key, which is moved into the output buffer. Otherwise, the string is moved into the table. If it does not fit into it, the table is filled by a first part of the string and put out on SYS001. ADABS is reduced by the length of this output, and the remaining bytes of the string are moved to the beginning of the table. Control then returns to FSCA and the scan continues.

FFIN -- EW

This is the end of the main routine. The end-of-program key is moved into the output buffer, and the buffer is put out on text output medium. The character string table (or the last part of it) is put out on SYS001. The length of the character-string table is stored in IJKCSL. IJKPH is then called to fetch the next phase (A50).

FPIN -- EX

FPIN is called each time the input pointer has been increased and ensures that the input pointer always points to an address within the first of the two input buffers. Whenever the input pointer exceeds its range, the contents of the second buffer are moved into the first one, and a new record is read into the second buffer.

Note: The end of the program is indicated by the end-of-program key X'FF'. If, however, the end quote of a string constant is omitted, this key cannot be detected since a character string may contain any of the 256 characters. It is therefore necessary to test for end of file after every call of IJKGI. If the end of file has been reached, the last record in the first buffer is processed, and FFIN is called the next time FPIN is invoked.

FMBU -- EY

Input parameters:
RR : address of source.
R1 : end address of source + 1.
POUT : pointer of output buffer.
BUFOL : end address of output buffer.

Output parameters:
RR : = R1.
POUT : points to the next free address in the buffer.

This routine moves text into the output buffer. If all bytes to be moved do not fit into the buffer or if they do exactly fit, the buffer is filled with the first part of the text to be moved and its contents are written on output medium. The remaining bytes, if any, are moved to the begin of the buffer.

This phase scans the block structure of the source program. Therefore, the statements PROCEDURE, BEGIN, IF, DO, and END as well as the keywords THEN and ELSE must be recognized.

Each statement is given a 6-byte end-of-statement (EOS) key, which contains a level, a block, and a statement number. Each assignment statement is given a special key (SET key).

All statement keywords are translated into internal representation (see Figure 1). For the keyword THEN, an EOS key is generated. For the keyword ELSE, an "ELSE statement" containing a statement key and an EOS key is generated.

If an error is found in the block structure (more PROCEDURE or BEGIN statements than END statements or vice versa), the source text is truncated after the last correct END statement.

Internal Representation of Statement Identifiers

Each statement identifier is replaced by a 3-byte key in internal code. This key has the following format:

byte 0: E0
byte 1: undefined
byte 2: identification (see Figure 1)

DESCRIPTION OF ROUTINES

Symbols used in flow charts

POUT	=	pointer output area
PCA	=	pointer communication area
IBUFL	=	length of the I/O buffers
PIN	=	pointer input area
BUFB1	=	start address of the first input buffer
GROUT	=	end address of the output buffer
BUFEND	=	end address of the input area
PTA	=	pointer table area
TABA	=	start address of the table area
TABE	=	end address of the table area + 1

JEPLA1 -- FO

The routine skips the prefix lists and labels preceding a statement. It is tested whether a parenthesized list preceding a statement is followed by a colon. The prefix list is translated into a mask. The statement counter is increased according to the number of statements processed. The counter value is inserted into the EOS key.

Entry parameter:
PIN = address of the first byte of the statement

Return parameters:
HR4 = address of the first byte of a statement which is not yet put out.
PIN = address of the first byte after the first identifier of a statement.

Byte 2	Statement	Byte 2	Statement
03	DUMP	21	REVERT
04	OVERLAY	22	ON
05	PROCEDURE	23	STOP
06	BEGIN	30	CLOSE
07	END (PROCEDURE)	31	OPEN
08	END (BEGIN)	32	DISPLAY
09	CALL	33	GET
0A	GOTO	34	PUT
0B	ENTRY	35	FORMAT
0C	RETURN	36	READ
0D	NOP	37	WRITE
0E	SET	38	LOCATE
0F	EXPRESSION	39	REWRITE
10	IF	40	DECLARE
11	ELSE	41	INITIAL SCALAR
12	DO	42	INITIAL ARRAY
13	END (DO)	43	FILE
20	SIGNAL	44	ARRAY

Figure 1. Contents of Byte 2 of the 3-Byte Statement Identifier Key

JASSA1 -- FP

The program tests whether the actual statement is an assignment statement. If so, the SET key is inserted before the statement.

If the identifier preceding the statement is IF, control is transferred to JPIF. If the identifier is the statement keyword, it is replaced by the corresponding key. Otherwise, the SET key is generated.

Entry parameters:
PIN = address of the first byte after the first identifier of a statement.
HR4 = address of the label identifier preceding the statement, if any.

Return parameters:
PIN = start address of the statement identifier.
HR4 = address of the first byte of a statement that is not yet put out.

JSTAA1 -- FQ

Secondary entry points: JSTAA3, JSTAE2, JSTAE4

The routine compares the identifier preceding a statement with a list of statement keywords contained in KEYTAB. This table contains a 4-byte entry for each keyword. The first two bytes contain the keyword itself; the other two bytes contain a relative branch address.

If the identifier is one of the keywords PROCEDURE, BEGIN, ENTRY, IF, ELSE, DO, END, GOTO, or DECLARE, control is transferred to one of the routines JPRO, JENT, JPIF, JELS, JPDO, JEND, JGOT or JDLA.

All statement identifiers are translated into internal representation by means of the table CODTAB, which contains a 4-byte entry for each keyword. The first two bytes contain the keyword itself; the second two bytes contain the internal representation of the statement identifier.

Entry parameters:
PIN = start address of the statement identifier.
BYZ = 0 or length of a parenthesized list follows the identifier.
HR4 = start address of the label identifier preceding the statement, if any; otherwise HR4 = PIN.

JPROA1 -- FR

Secondary entry point: JPROB1

The routine is called by JSTA and processes the PROCEDURE and the BEGIN statement, respectively. The level counter is increased by 1. It may not be greater than three because only three levels are allowed. The block counter is increased by 1. It may not be greater than 63 because only 63 blocks are allowed.

For each PROCEDURE, BEGIN, or DO statement, a pointer ENDZ is increased by 1. Corresponding to the status of ENDZ, it is entered in a push-down table ENDTAB, whether it is a begin block (0) or a DO group (1). The evaluation of this table and reducing of ENDZ by 1 is done by the routine JEND.

Entry parameters:
PIN = start address of the statement identifier.
HR4 = start address of the first label identifier, if any, preceding the statement.

Return parameters:
PIN = unchanged.
HR4 = start address of the label identifier. If more than one label is given, the last label is pointed to.

JENTA1 -- FS

The routine is called by JSTAA1 and processes the ENTRY statement. Only the label preceding the statement is checked. Entry and return parameters are the same as in JPRO.

JPIFA1 -- FT

The program is called by JSTAA1 and processes the IF statement. An IF statement has the form:

IF <u>expression</u> THEN <u>unit 1</u> ELSE <u>unit 2</u>;

An equal sign and parenthesized lists may occur in <u>expression</u>. Since there is no difference in appearance of the logical equal sign and the arithmetical equal sign, the IF statement can be differentiated from the assignment statement only by the keyword THEN. The statement identifier is replaced by the internal representation. The keyword THEN is replaced by an EOS key.

Entry parameters:
PIN = start address of the statement identifier.
HR4 = start address of the label identifier preceding the statement. If there is no label, HR4 = PIN.

Return parameters:
PIN = address of the next byte after THEN.
PIN = unchanged if no IF statement is encountered.

JELSA1 -- FU

The routine is called by JSTAA1 and processes the keyword ELSE. ELSE is followed by a semicolon only if unit is a NOP statement. To facilitate the statement scan for the following phases, the ELSE key is concluded by an EOS key.

An ELSE keyword has the form:

statement; ELSE identifier or
statement; ELSE (prefix): identifier or
statement; ELSE ;

i.e., ELSE can only be followed by an identifier, a left parenthesis, or a semicolon. In the source text, the keyword ELSE is replaced by:

Key	(6 bytes)
EOS	(6 bytes)

The statement number in EOS is the same as in the previous statement.

Entry parameter:
PIN = HR4 = start address of the identifier ELSE.

Return parameter:
PIN = address of the byte following the identifier.

JPDOA1 -- FV

The routine is called by JSTAA1 and processes the DO statement. A DO statement must be recognized to be able to differentiate the END statement into block ends and group ends.

The statement identifier is replaced by the internal representation. In the internal buffer ENDTAB, a 1 for marking group end is entered. A zero is entered for block end.

Entry parameters:
PIN = start address of the statement identifier.
HR4 = start address of the first label preceding the statement.

Return parameters:
PIN = Unchanged HR4 = Unchanged

JENDA1 -- FW

The routine is called by JSTAA1 and processes the END statement. An END statement has the format: END; No other format is permitted in the DOS/TOS PL/I compiler. In the PL/I language, the END statement for a block end is the same as for a group end. Internally, the two types of END are coded differently. Pointer ENDZ points to the last entry in ENDTAB (see JPROA1 -- FR), thus showing the type of END. The level counter LEV is decreased by one at the end of a block.

Entry parameters:
PIN = start address of the statement identifier.
HR4 = start address of the label identifier, if any.

Return parameter:
PIN = address of the semicolon.

JLACA1 -- FX

The routine checks the label preceding a PROCEDURE or ENTRY statement. Only one label must precede each of these statements. The following errors may appear:

1. No label: pseudo label is inserted.
2. More than one label: all labels except the last are ignored.

Entry parameters:
HR4 = start address of the (possibly first) label.
PIN = start address of the statement identifier. If no label appears, HR4 = PIN.

Return parameters:
PIN = unchanged.
HR4 = unchanged if no error is detected.
HR4 = PIN if error 1 is detected. HR4 = start address of the last label if error 2 is detected.

JEOSA1 -- FY

The routine is called at each statement end. It generates the EOS key and puts out an error list, if necessary. When JEOS is called, PIN points to the semicolon. The statement itself is already in the output area or on the output medium.

On return, PIN points to the first byte of the new statement. If no more statements follow, i.e., if the end of the source text is reached, PIN points to the end-of-source-text mark.

It is tested whether PIN is still inside the first input buffer. If it is not, it is tested whether PIN is still inside the last buffer because incorrect statements can cause PIN to run out of the input area. In this case, an error message is given. Otherwise, the contents of buffers 2 - 4 are moved into buffers 1 - 3 and a new record is read into buffer 4. It must therefore be avoided that a statement or a single identifier is divided by the end of the input area. This is done as long as PIN is outside the first buffer.

The EOS key has the following format:

byte	0	EOS key
byte	1	error indicator
byte	2	level number
byte	3	block number
byte	4-5	statement number

If an incorrect statement is discovered, and error message is generated in the source text. The error message has the following format: The first bit of the error indicator in the EOS key is set to 1. Two bytes are inserted after the key for every error in the source text; byte 1 contains the error key, byte 2 contains the error number.

JERRA1 -- FZ

The routine is called if an error is detected. Up to eight error messages per statement are stored. Additional errors are ignored.

JEOSA1 puts out the error messages into the source text following the statement in error. The error table (ERRTAB) entries have the following format:

byte	1	= error key (X'EB')
byte	2	= number of errors
bytes	3-10	= special error keys

Entry parameter:
HR0 = special error key (1 byte)

MOVEA1 -- F0

The subroutine moves any number of bytes from a FROM field to a TO field. The FROM and TO fields may overlap.

Entry parameters:
HR0 = number of bytes to be moved
HR1 = address of the TO field
HR2 = address of the FROM field
BYZ is used as auxiliary register.

JCHAA1 -- F1

The subroutine is used to find a character in the source text. Searching is performed up to the end of the statement. If the end is reached, PIN contains the address of the semicolon as return parameter. If the end of the source program is reached before the character is found, an error message is given. An EOS key is inserted.

Entry parameters:
PIN = start address of the search region.
BYZ = character to search for (1 byte right-justified).

Return parameters:
PIN = address of the character found or of the end of statement.
BYZ = PIN new - PIN old.

JSKPA1 -- F2

The subroutine searches for the end of a parenthesized expression. All internal pairs of parentheses are skipped.

Entry parameter:
PIN = address of the first left parenthesis.

Return parameters:
PIN = address of the next byte after the last right parenthesis.
HR0 = PIN old.
BYZ = PIN new - PIN old.

JTRNA1 -- F3

The subroutine moves information into the output buffer and controls the pointer for this buffer. When the pointer exceeds the scope of the buffer, the text is put out on output medium.

Entry parameters:
PIN = start address of the information to be put out.
BYZ = length of the information.
POUT = next free address in the output buffer.

Return parameters:
PIN new = PIN old + BYZ.
POUT = next free address in the output buffer.

JGOTA1 -- F4

The routine is called by JSTA and processes the GOTO statement. The statement identifier for the GOTO statement may be written with or without a blank between GO and TO. The key is the same for both forms.

JDLAA1 -- F5

The routine is called by JSTA and processes the DECLARE statement. If a label list precedes the statement, it is removed from the source text.

Entry parameters:
PIN = start address of the statement identifier.
HR4 = start address of the label identifier preceding the statement. If there is no label, HR4 = PIN.

Return parameters:
PIN = unchanged.
HR4 = PIN.

JFIXA1 -- F6

The program scans the prefix lists and generates a mask. This mask has the following format:

```
bit 0 : 0 = NO ZERODIVIDE
        1 = ZERODIVIDE
bit 1 : 0 = NO UNDERFLOW
        1 = UNDERFLOW
bit 2 : 0 = NO OVERFLOW
        1 = OVERFLOW
bit 3 : 0 = NOFIXEDOVERFLOW
        1 = FIXEDOVERFLOW
bit 4 : 0 = NOCONVERSION
        1 = CONVERSION
bit 5 : 0 = NO SIZE
        1 = SIZE
bit 6 :     reserved
bit 7 :     reserved
```

Entry parameter:
PIN = address of the left parenthesis.

Return parameters:
PIN = address of the colon after the prefix list.
FIXMSK = mask.

JSSAA1 -- F7

The routine generates a statement attribute of 3 bytes and inserts it into the source text immediately after the statement identifier. The statement attribute contains the following information:

byte 0	prefix mask (see JDLAA1 -- F5)
byte 1	number of the actual block
byte 2	number of the embracing block

Byte 1 is set to zero in phase B90.

JEOPA1 -- F9

This routine checks if the end of the source text has been reached. If it has, the end-counter ENDZ is checked, and the output area is cleared.

Entry parameter:
PIN = input pointer

JBETA1 -- F8

This routine generates the end-table.

PHASES PL/IA60, A65 (SYNTAX CHECK I AND II) -- GL, GW

The two syntax phases, A60 and A65, may be considered as one logical phase.

The first syntax phase, A60, processes all statements except READ, WRITE, GET, PUT, FORMAT, which are processed by the second syntax phase A65.

Phases A60 and A65

- check each statement for syntactical errors (exception: DECLARE statement).
- substitute 3-byte keys for symbols as follows:

byte 1 :	key S'E2'
bytes 2-3:	program-internal code for the respective symbol (see Figure 1 in phase A25).

- substitute elements of variable length for all constants (except character string constants) as shown in Figure 1.

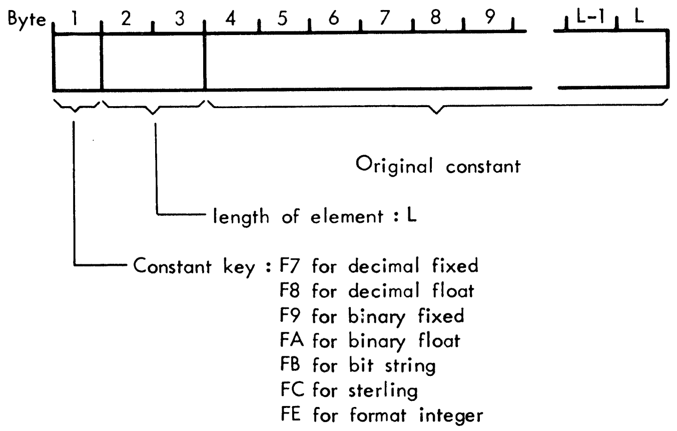


Figure 1. Substitution of Variable-Length Elements for Constants

The preceding phases have:

1. eliminated all blanks and comments,
2. substituted an end-of-statement delimeter for each semicolon as follows:

   a. if no error has been detected in the statement:

Byte(s)	Contents
1	end-of-statement key X'EA'
2	indicator no error X'00'
3	level number
4	block number
5-6	statement number

   b. if an error has been detected in the statement:

Byte(s)	Contents
1	end-of-statement key X'EA'
2	error indicator X'40' or X'80'
3	level number
4	block number
5-6	statement number
7	error key X'EB'
8	error number
9	error key X'EB'
10	error number, etc., (up to 8 errors)

3. substituted an end-of-statement delimiter for each keyword THEN;
4. placed an end-of-statement delimiter after each keyword ELSE;
5. substituted a special key for each character string constant as follows:

Byte(s)	Contents
1	character-string constant key X'E3'
2-3	offset to begin of character-string constant table
4	character-string constant key X'E3'
5-6	length of the constant

6. substituted 6-byte elements for all statement identifiers as follows:

Byte(s)	Contents
1	statement identifier key X'E0'
2	not used in this phase
3	number specifying the statement identifier
4	prefix information
5-6	not used in this phase

7. placed a 6-byte element ASSIGN in front of each assignment statement as shown under item 6.
8. substituted a 3-byte key for each program element appearing as an identifier or keyword as follows:

Byte(s)	Contents
1	identifier key X'E1'
2-3	offset to a table

9. processed and eliminated the prefix option lists.

Note: Steps 3 and 4 have left the program non-recursive. All statements may now be processed independently of each other. Push-down stacks are reduced in size (since recursion only occurs in expressions).

Output of Phases A60 and A65

With the exception of the DECLARE statement and the declarative portions of PROCEDURE and ENTRY statements, the output stream consists of 3-byte elements and variable-length elements. Any ambiguities resulting from the fact that keywords are not reserved have been clarified. This is illustrated by the following example:

```
   DO IF=BEGIN TO END WHILE(DISPLAY);
```

Since all identifiers making up the above statement are potential keywords, the syntax phases must detect the real keywords (in this case DO, TO, WHILE).

The first byte of each 3-byte element substituted for a keyword now contains X'EF' instead of X'E1'

An error message is generated for each detected syntactical error. This message is attached in coded form to the end-of-statement delimiter as shown in Figure 3.

FUNCTIONAL DESCRIPTION

1. Scanning Syntactical Units (Linguistic Functions)
   A "Linguistic Function" (abbreviation LF) is a routine which returns a Boolean value. The value of the LF is determined as follows: Within the LF, a "linguistic expression" is written, which syntactically describes a pattern of the source string. This linguistic expression is said to define the LF. During execution,the LF examines the source text for the occurrence of the pattern described by the LF's linguistic expression. If the pattern is found, the LF yields a TRUE value; if not, it yields a FALSE value. (These are quotations from the SLANG Language Tutorial Manual. Edition 1, 3-18-64, pages 36-37).

2. Processing Syntactical Units
   Whenever a syntactical unit has been recognized, 3-byte elements and variable-length-elements are substituted for symbols and constants.

3. Detection of Syntactical Errors
   After a statement has been identified by scanning and comparing the statement identifier, it is checked for conforming to the syntactical rules. If an error is detected, a message specifying the nature of the error is generated.

The syntactical scan is based on the assumption that the complete statement is contained in the four input buffers. Two pushdown stacks, three pointers and three LF utility routines are used.

Push-Down Stacks

LPDL	used to store the linkage.
PPDL	used to store the value of the input pointer

Pointers

PDLI	A symbolic register used as a pointer to LPDL and PPDL. This pointer is moved by the routines BEGLF, EXTRUE, and EXFALS.
PIN	A symbolic register used as input pointer.
POUT	A symbolic register used as output pointer.

LF Utility Routines

The following 3 routines enable recursion during the syntactical scan (see flow charts HH and GT).

BEGLF	Initiated upon entry into an LF. The current value of the input pointer PIN is saved and the linkage information contained in LINK is stored.
EXTRUE	Initiated if an LF yields TRUE. This routine fetches linkage information from LPDL, adds 4 to it, and returns to the resulting address.
EXFALS	Initiated if an LF yields FALSE. This routine restores PIN (i.e. fetches from PPDL the value which was stored there when the LF was initiated), fetches linkage information from LPDL, and returns to the provided address.

Note: After a TRUE exit, PIN points to the character following the examined syntactical unit. After a FALSE exit, PIN points to the same character it was pointing to when the LF was initiated.

```
 1.  INTEG    BAL    UTIL,BEGLF     Linguistic Utility Routine.
 2.           ST     PIN,INTEG1     Store begin of integer.
 3.           BAL    LINK,DIGIT     Digit?
 4.           B      EXFALS         No. Return FALSE.
 5.  INTEG2   BAL    LINK,DIGIT     Yes. Another digit?
 6.           B      INTEG3         No. End of integer.
 7.           B      INTEG2         Yes. Try again.
 8.  INTEG3   L      R1,PIN         Compute the address of
 9.           BCTR   R1,0           last digit of integer.
10.           L      R2,INTEG1      Load start address.
11.           BAL    LINK,STORIT    Call storing routine.
12.           B      EXTRUE.        Return TRUE
```

Step 1: When initiating BEGLF, the symbolic register UTIL is used instead of LINK. This saves LINK.

Steps 3-7 comprise the "linguistic expression".

Figure 2. Linguistic Utility Routine

Example for Syntactical Scan

An integer is assumed to be defined (using the Backus-Naur form (BNF)) as follows:

<integer> ::= <digit>|<integer><digit>

This means an integer is a string consisting of more than 0 digits. The above BNF definition gives the base for the "linguistic expression" as illustrated in the program shown in Figure   which scans an integer, notes the address of the first digit, the address of the last digit and calls another routine with these addresses as parameters.

Note: Although the BNF definition of an integer is recursive, the routine shown in Figure 7 is not recursive. This is correct because the integer could be defined as <integer> :: = min 1 <digit> by using an extended BNF. Recursion has been avoided to improve the phase performance.

Syntactical Definition of Input and Output Stream

The syntactical definition (metalanguage) of the input and output stream is given in Appendices A and B.

Skipping of Phases

To save compilation time, certain phases following the syntax phase are skipped if the statement which they process does not occur in the source program. Skipping of phases is prepared and specified by the syntax phases A60 and A65 as follows:

Bits 3 to 7 of byte IJKMJT+3 specify skipping of certain subsequent phases. If one of these bits is set to 0, the associated phase is skipped. At the beginning of the syntax phase, all 5 bits are set to zero. The occurrence of specific statements causes the syntax phases to set the associated bit to 1 as shown below.

Statement	Bit No. Set to One:
CLOSE	6
DISPLAY	6,7
FORMAT	6
GET	4
IF	3
LOCATE	6,7
OPEN	6
READ	6,7
REWRITE	6,7

DESCRIPTION OF ROUTINES

(Open) A routine is called open if control is transferred to it by

1. a simple B instruction, in which case control is also returned by a B instruction, or
2. some in-line coding that requires a separate description.

(Closed) A routine is called closed if control is transferred to it by a BAL instruction. Control is returned by a BR instruction in this case.

SYN1 -- GM

This routine is the "master program" of the phase.

1. PIN and POUT are initialized and the four input buffers are filled.

2. PDLI is reset. PIN is stored in CREAT1. PIN is moved until a statement identifier key (x'E0') is found. Then 6 is added to PIN so that it points to the first character of the statement body. The statement-processing routines are activated.

3. If the statement returns TRUE (entry SYN166 = statement conforms with syntactical rules), it is tested whether PIN points to the end-of-statement (EOS) delimiter.

4. If PIN points to the EOS delimiter, the last part of the statement (the start address is in CREAT1, the end address - 1 is in PIN) is put out, and EOST is called. SYN1 continues with step 2.

   If PIN does not point to the EOS delimiter, ERROR is called(the logical end of the statement body is not followed by an EOS). The last part of the statement is put out, and PIN is moved until an EOS or the end-of-program mark (EOP) is detected. If an EOS is encountered, EOST is called. SYN1 continues with step 2. If an EOP is encountered, TEPHA is called to terminate the phase.

5. If the statement returns FALSE (the statement does not conform with syntactical rules, or is not processed in this phase), INPT is called. INPT moves PIN until an EOS or EOP is encountered.

Note: Whenever control returns from EOST, PIN points to the first byte of the next statement.

BUBU (Closed) -- GO

Puts out a string. The start address of this string is in CREAT1, the end address -1 is in PIN.

CARFB (Closed)

This routine is called by several linguistic functions.

R1 contains the address of the 3-byte element. If this element is not identical to that starting at 0(PIN), the routine returns FALSE to (LINK). Otherwise, the byte at 0(PIN) is replaced with X'EF' (key for "Keyword") and PIN is incremented by 3. BUBU is called. The routine returns TRUE to 4(0,LINK).

EOST, JEOSA1 (Closed) -- GQ

Arranges the contents of the input buffers 1 to 4. The currently scanned EOS is located in input buffer 1 (this is done by moving and by reading new records). Puts out the EOS and the error codes attached to it. Any additionally generated error codes are also put out.

INPT (Open) -- GN

1. If PIN points to an EOS, control is passed to SYN157. (SYN157 is a label associated to SYN1 step 4 - see description of SYN1).

2. If PIN points to an EOP, TEPHA is called.

3. If PIN points to an E-key, PIN is incremented by 3.

4. If PIN points to an F-key, the contents of the two bytes following this F-key are added to PIN.

5. Otherwise, PIN is incremented by 1 and INPT starts again with step 1.

JSLCA1 (Open) -- GU

Tests the statement for excessive length. (The appropriate EOS must be located in the first 4 input buffers). If the statement exceeds the permitted length, the statement body is deleted. The statement now consists of the statement identifier and the EOS attached with error codes. The next statement is positioned starting in input buffer 1.

JTRNA1 (Closed) -- GR

Output routine. Register BYZ contains the number of bytes to be put out; register PIN contains the start address.

One output buffer is used.

1. If the (remaining) length of the output string does not exceed the available space of the output buffer, the complete (remaining part of the) string is moved into the buffer. The output pointer is updated by adding BYZ to POUT.

2. If the length of the output string exceeds the available buffer space, an appropriate part of the string is moved to the buffer. The contents of the buffer are written onto the output

medium. POUT is reset to the start address of the buffer. BYZ is decremented by the number of bytes moved into the buffer. PIN is incremented by this number. JTRNA1 starts again with step a.

LKW (Closed)

This routine is called by several linguistic functions.

Input parameters:
R2: table address
R3: address of the LF to be initiated if the search is successful. Must be 0 if no LF is to be initiated.
R4: length of source pattern
R5: address of a 3-byte element

Looks up the table (address defined by R2) for a pattern (length defined by R4) that is identical to that located in 0(PIN). Returns FALSE to 0(LINK) if the search was unsuccessful. Otherwise, BUBU is called to put out a string. A 3-byte element is created by using the rightmost byte in R5 as first byte and the "function value" of the table as second and third bytes. The 3-byte element is put out. If R3 contains 0, LKW returns TRUE to 4(0,LINK). If R3 contains an address of an LF, this LF is initiated and depending on the value of the LF, LKW returns TRUE or FALSE.

Example:

```
        DS    0F
TABLE   DC    X'2'       Length of pattern
        DC    X'5'       Number of elements
                         in the table
        DC    X'3A3A'    1st argument
        DC    X'0AFA'    1st "function value"
        DC    X'393C'    2nd argument
        DC    X'07EE'    2nd "function value"
        DC    X'3E3C'        etc.
        DC    X'07F1'
        DC    X'3D3C'
        DC    X'07F2'
        DC    X'4040'
        DC    X'03EA'
        DC    X'0'       end of table

Parameters               R2  :  A(TABLE)
                         R3  :  A(0)
                         R4  :  A(2)
                         R5  :  X'000000E2'
```

Assumed input:

```
┌───────────────────────────────────────────┐
│ ──┬────┬──┬──┬──┬──┬──┬──┬───────────     │
│ ..│....│E1│03│61│3D│3C│01│05│......        │
│ ──┴────┴──┴──┴──┴─┬┴──┴──┴──┴───────────  │
│                   ∧└─>source  pattern     │
│                   │                       │
│  ┌────────────────┘                       │
│  PIN points to this character             │
└───────────────────────────────────────────┘
```

In this case LKW performs the following:

The table lookup is successful. BUBU is called to put out a string ending at X'61'. A 3-byte element is created (X'E207F2') and put out. PIN points to X'01'. LKW returns TRUE.

TARI1 (Closed)

This routine is called by several linguistic functions.

If the rightmost byte in R1 is identical with the byte at 0(PIN), BUBU is called. A 3-byte element consisting of the leftmost 3 bytes in R1 is put out. PIN is incremented by 1. TARI1 returns TRUE to 4(0,LINK).
Otherwise, TART1 returns FALSE to (LINK).

TEPHA (Open) -- GP

Puts out the contents of the output buffer. Further actions depend on the utilization of TEPHA.

If the routine is used in phase A60, it returns control to the compiler control program, indicating the next phase to be initiated. This is A65 if one of the following statements occurred in the source program: READ, WRITE, GET, PUT, FORMAT. Otherwise, phase B10 IS initiated.

If the routine is used in phase A65, it returns control to the compiler control program, indicating that the next phase to be initiated is B10.

ERROR(M), JERRA1(M) -- GS

This routine fills an error table with up to 8 errors per statement. If the same error is detected more than once for one statement, the error appears only once in the error table.

Each detected error causes an error message to be generated, represented internally as a one-byte number. This number is attached to the End-of-Statement delimiter.

In this phase, all declarations given explicitly in DECLARE statement parameter lists and label declaration lists are collected in a declaration pool. The pool is written on SYS001.

Declaration Pool

In the declaration pool all declarations belonging to one block are collected in a group which is written on SYS001 if the end of the block is reached.

Three block levels are allowed. For each level a buffer is defined in the table area. The declarations are collected in the buffer indicated by the level counter. If a buffer overflows, it is written before the end of the block is reached.

The first four bytes in each buffer contain special information concerning the block.

byte 0:	block number
byte 1:	block level
byte 2:	block number of the embracing block
byte 3:	mark if the record is the last of the block.

This information is put in front of each record. In phase B20 the records of the pool which are on SYS001 are ordered by ascending block number and written on SYS002 or SYS003.

The information entered in the declaration pool is classified in three groups:

1. label declaration lists

2. parameter lists

3. DECLARE statements

A label declaration list starts with an identifier key. A label constant or an entry name may be entered in such a list. A label constant consists of 4 bytes:

byte 0 :	identifier key
bytes 1-2:	user name (coded in phase A25)
byte 3 :	colon

An entry name consists of:

byte 0 :	identifier key
bytes 1-2:	user name (coded in phase A25)
bytes 3-4:	attribute ENTRY (optionally, data attributes specified by the user as attributes describing the returned value).

The end of an entry name is indicated by an EOS key (6 bytes).

A parameter list starts with a parameter key (1 byte). This key is followed by the internal representation of the left parenthesis (3 bytes).

The user-defined parameter names follow (3 bytes each, coded in phase A25) separated by the internal representation of the comma (3 bytes) and closed by the right parenthesis (3 bytes).

A DECLARE statement starts with a declare key (1 byte). The whole statement follows. It is scanned syntactically in phase B20.

DESCRIPTION OF ROUTINES

Note: The routines JERRA1, MOVEA1, JCHAA1, and JTRNA1 are described in phase A50. The corresponding flow charts are FZ, F0, F1, and F3, respectively.

Symbols used in flow charts:

PCA :	pointer for communication area
EOS :	end-of-statement key
EOPR :	end-of-program key
STATAB:	table of addresses of routines that process PROCEDURE, BEGIN, ENTRY, DECLARE, and END
BLZ :	block counter
ERRCOD:	error code
LEV :	level counter
EOSC :	end of record on SYS001

Initialization -- HN

JELAA1 -- HO

This routine scans the statement labels. If a label is found, it is entered in the declaration pool.

Entry parameter:
PIN = address of the first byte of a statement.

Return parameter:
PIN = address of the statement-identifier key.

JSTBA1 -- HP

This routine scans the statement identifiers and searches for the identifiers PROCEDURE, BEGIN, ENTRY, DECLARE, and END. If one of these identifiers is found, the program branches to special routines that process these statements. All other statements are written unchanged.

Entry parameter:
PIN = address of the statement-identifier key.

JPCRA1 -- HQ

Secondary entry point: JPCRE1

This routine, called in JSTBA1, processes the PROCEDURE statement. The PROCEDURE statement opens a new block. Therefore, the level and block counter are increased by 1.

The following information is entered into the declaration pool:

1. The last label is given the attribute ENTRY.

2. If the procedure has data attributes, they are associated with the last label.

3. The end of this attribute list is indicated by the EOS key.

4. Four bytes of information concerning the block are entered in the declaration pool.

   byte 0: block number
   byte 1: block level
   byte 2: block number of embracing block
   byte 3: indicates last record of block

5. If the PROCEDURE has a parameter list, a key is entered in the pool. The list follows unchanged.

The PROCEDURE statement, with the exception of the data attributes, is written unchanged.

The first PROCEDURE statement in a source program may have the attribute OPTIONS followed by an option list in parentheses. The options in this list are separated by commas.

The following options may appear:

MAIN: It specifies the MAIN procedure.
ONSYSLOG: It specifies that object time diagnostics will be written on SYSLOG.

If these options appear, special bits in the communication area are set.

A PROCEDURE statement with the attribute OPTIONS must not have a parameter list or data attributes.

JOPTA1 -- HR

This routine processes the OPTIONS attribute.

JCPLA1 -- HS

This routine checks the parameter list for identical parameters.

Entry parameters:
HR4 = PIN = address of the left parenthesis.
HR3 = 0.

Return parameters:
PIN = address of the right parenthesis.
HR3 = length of the parameter list.

JENTA1 -- HT

This routine is called in JSTBA1 and processes the ENTRY statement.

An ENTRY statement differs from a PROCEDURE statement in that it does not open a new block. The entry name is internal to the embracing block. Therefore, the entry name is moved into the declaration pool of the embracing block. This is done in routine JELA.

Entry parameter:
PIN = start address of the statement identifier.

Return parameter:
PIN = address of the EOS key.

JBEGA1 -- HU

This routine is called in JSTBA1 and processes the BEGIN statement. For the scope of declarations, the BEGIN statement has the same function as the PROCEDURE statement.

Entry parameter:
PIN = start address of the statement identifier.

Return parameter:
PIN = address of the EOS key.

JDCSA1 -- HV

This routine is called by JSTBA1 and processes the DECLARE statement. The entire statement is moved unchanged into the declaration pool.

JENDA1 -- HW

This routine is called by JSTBA1 and processes the END statement. An END statement closes a block.

The level counter (LEV) is decreased by one. When the end of a block is reached, three bytes containing X'FFFFFF' are moved into the declaration pool for that block, and the declaration pool is written on a work file.

Entry parameter:
PIN = start address of the statement identifier.

Return parameter:
PIN = address of the EOS key.

<u>JSLCA1 -- HX</u>

This routine checks the length of a statement. A statement must not be longer than 3 buffers. If a statement with an error message is detected, the statement is deleted except for the statement-identifier key and the EOS key.

Entry parameter:
PIN = address of the statement identifier.

Return parameter:
PIN = unchanged.

Registers used:
HR0, HR1, HR2, HR4.

<u>JEOSA1 -- HY</u>

This routine is called at the end of each statement. The error indicator contained in the EOS key is tested to determine if an error exists. An error list is written, if necessary.

When JEOSA1 is called, PIN points to the first byte of the EOS key. The statement itself except for the EOS key is already on the output medium or in the output buffer.

When returning, PIN points to the first byte of the new statement. If no other statement follows, i.e., if the end of the source text is reached, PIN points to the end-of-source-text key.

The program uses four buffers for the input stream. If PIN is beyond the first buffer, the remainder of the input stream is moved to the left, and a new record is read into the last buffer.

Entry parameter:
PIN = address of the first byte of the EOS key.

Return parameter:
PIN = address of the first byte of the new statement.

<u>Subroutine JDEPA1 -- HZ</u>

This routine checks the length of the declaration pool. If necessary, the declaration pool is written onto the work file. A record counter is increased by 1.

Entry parameters:
HR1 = start address of the information to be transferred into the pool.
BYZ = number of bytes to be transferred.
PTA = address of the first free byte in the pool.

Return parameter:
PTA = address of the next free byte in the pool.

PHASE PL/IB15 (DECLARATION SCAN II) -- IM

This phase scans the DECLARE statements for syntactical errors. In phase B10, all declarations were collected in a declaration pool and written on a work file. Phase B15 reads the pool, sorts the records according to ascending level numbers, and scans the DECLARE statements. The output is written onto TXTIN of the previous phase.

For some declarations, special statements are generated in the source text (see items 1 to 3 below). The previous phase leaves the last record of the source text in the output buffer. The output medium is not rewound. Therefore, the statements generated in this phase are attached to the end of the last record.

1. ARRAY

ASK	= array statement key	(6 bytes)
VN	= variable name. If the array is a part of a structure, a full qualification is made.	(3 bytes)
CAN	= current array number	(3 bytes)
	bounds	(9 bytes)
EOS	= end-of-statement key	(6 bytes)

2. FILE

FSK	= file statement key	(3 bytes)
VN	= variable name	(3 bytes)
CFN	= current file number	(3 bytes)
file description		uncoded
EOS	= end-of-statement key	(6 bytes)

3. INITIAL

ISK	= initial statement key	(6 bytes)
VN	= variable name If the initial item is a part of a structure, a full qualification is made.	(3 bytes)
LIL	= length of list	(3 bytes)
	initial list	uncoded
EOS	= end-of-statement key	(6 bytes)

There are two ISK's, one for scalar and one for array initialization.

The following is entered in the declaration pool:

1. ARRRAY - The array attribute followed by the current array number and the number of contained elements.

2. FILE - The file attribute followed by the current file number.

3. INITIAL - The initial attribute only.

DESCRIPTION OF ROUTINES

Symbols used in flow charts

PCA	:pointer communication area
FINO	:current file number
ZI	:integer constant
ERRCOD	:error code

Note: The following routines used in this phase are described as follows:

MOVEA1	A50	JCHAA1	A50
JERRA1	A50	JTRAA1	B90

Initialization -- IN

JSRTA1 -- IO

This routine sorts the declaration pool. The sorted records are moved into the table area. After the syntactical scan of the DECLARE statements, they are written on the input work file of the previous phase. Each record starts with a special word:

Byte 0: block number
Byte 1: level number
Byte 2: block number of the embracing block
Byte 3: indicates if the record is the last of the block.

The records are sorted in ascending order of level numbers.

LVA = actual level number
LVM = maximum level number

JSCNA1 -- IP

This routine scans the declaration pool. The information entered in the declaration pool is classified in three groups:

1. label declaration lists
2. parameter lists
3. DECLARE statements

A label declaration list starts with an identifier key. A label constant or an entry name may be entered in such a list.

A label constant consists of 4 bytes:
byte 0 : identifier key
bytes 1-2: user name (coded in phaseA25)
byte 3 : colon

An entry name consists of:
byte 0 : identifier key
bytes 1-2: user name (coded in phase A25)
bytes 3-4: attribute ENTRY

(Optionally): data attributes specified by the user as attributes describing the returned value.

An entry name is always closed by an EOS key (6 bytes).

A parameter list starts with a parameter key (1 byte). This key is followed by the internal representation of the left parenthesis (2 bytes).

The user-defined parameter names follow (3 bytes each, coded in phase A25), separated by the internal representation of the comma (3 bytes) and closed by the right parenthesis (3 bytes).

A DECLARE statement starts with a declare key (1 byte) which is followed by the declaration. It is scanned for syntactical errors in routine JDECA1.

Entry parameter: PST = start address of the pool.

JDECA1 -- IQ-IW

This routine is called in JSCNA1 and scans the DECLARE statement for syntactical errors. The identifiers are separated in programmer-defined names and attributes. Attributes are coded internally. Parentheses are separated in such that mark factorization and such that include precisions or lists. Some attributes get a special treatment (see flow charts IQ-IW). If a syntactical error is detected, a NOP statement followed by an error message is generated in the source text.

Entry parameter:
PST = address of the first byte to be processed.

Return parameter:
PST = address of the first byte after the end-of-statement key.

The syntactical scan is performed by means of a two-dimensional matrix of addresses. Depending on the preceding symbol, the routine branches to corresponding routines at a new symbol (see Figure 1). The following routines may be called:

```
JDE1A1  JDE1A4JEODA1A1
JDE1A2  JDE2A1  JSEFA1
JDE1A3  JDE2A3  JDCDA1
```

Subroutine JATRA1 -- IX

This subroutine recognizes the attributes and sets the internal representations of the attributes into the declaration pool.

The external representation of all attributes is stored in a table (ATTAB). After each attribute there is a byte with the internal coding. This byte with a common attribute key is moved into the declaration pool.

All data attributes have a 1 in the first four bits of the internal coding.

Entry parameter:
PST = start address of the attribute.

Return parameters:
PST = address of the first character after the attribute.
ATKEY + 1 = internal coding of the actual attribute.

Subroutine JSIPA1 -- IY

This subroutine searches for the end of a parenthesized expression; all internal pairs of parentheses are skipped.

Entry parameter:
PST = address of the first left parenthesis.

Return parameters:
PST = address of the next byte after the last right parenthesis.
HR0 = PST old. BYZ = PST new - PST old.

JTRIA1 -- IZ

This routine moves information into the output buffer and controls the pointer for this buffer. If the pointer exceeds the scope of the buffer, the contents of the buffer are written on the actual input work file. The output is made in the non-overlapped mode.

Entry parameters:
HR1 = start address of the information to be written.
BYZ = length of the information.
PIT = next available address in the output buffer.
BUFI = end address of the output buffer.
POUTI= start address of the output buffer.

Return parameters:
PLT = next available address in the output buffer
BYZ = 0.
HR1 = HR1 old + BYZ old.

JCVTA1 -- JA

This routine converts an unpacked decimal integer constant to binary representation. The decimal number may have up to 9 digits.

Entry parameters:
HR1 = start address of the decimal constant.

Return parameters:
HR1 = value of the converted constant.
HR2 = number of digits in the decimal constant.

S \ T	identifier 0	number 2	left parenthesis 4	right parenthesis 6	comma 8	semicolon 10
empty or comma 0	name S = 2	structure	factorization open S = 6	error	error	error
name or right parenthesis 2	attribute S = 4	error	dimension S = 4	factorization close S = 2	delimiter S = 0	end of statement
attribute 4	attribute S = 4	error	precision	factorization close S = 2	delimiter S = 0	end of statement
left parenthesis 6	name S = 2	structure	factorization open S = 6	error	error	error
T = next symbol in the source program S = last symbol in the source program						

Figure 1. Two-dimensional Matrix of Addresses (SWITAB)

PHASE PL/IB20 (SYMBOL TABLE CONSTRUCTION I) -- KA

This phase constructs the symbol table for all explicitly declared variables and label constants. The input for this phase is the declaration pool constructed in phase B15.

Symbol Table

The symbol table consists of n+1 parts, where n is the number of blocks in the source program. Each part is attached to one block and contains all items declared explicitly. The last part contains all items declared contextually and implicitly. This part is constructed in phases B70 and B80.

The parts of the symbol table are separated from each other by a scope chain which contains the number of the embracing block. The start addresses of the parts are entered in the scope table. If the symbol table is written on a work file, each part starts with a new record. The first record number of each part is also entered in the scope table.

For each programmer-defined variable or label constant an entry of 20 bytes is made in the symbol table. The format of this entry is shown in Figure 1. The entries are used in phase B90 to build the statement attribute table.

Scope Table

An entry of 6 bytes is entered in the scope table for each block. The format of this entry is as follows:

Byte 0 :	Number of records belonging to this block.
Bytes 1-3:	NOTE information of the record in which the symbol table for this block starts.
Bytes 4-5:	If the symbol table is in core storage, relative start address of the symbol table for this block.

If a block has no declarations, the entry is given the data for the embracing block. Since the number of records belonging to one block is restricted to 255 and each record contains the declarations for 12 variables, the total number of declared variables for one block is restricted to 3060. This restriction is valid only for the minimum configuration. If the table space and the buffer area are increased, the number of declared variables increases at the same rate.

```
Bytes 0 - 1:

User-defined name (coded in phase A30)
----------------------------------------
Bytes 2 - 3:

Internal representation of the name
----------------------------------------
Byte 4:

Bits 0-3: Reserved
Bits 4-7: Internal length of the variable
----------------------------------------
Byte 5:

Bit 0: 1 = STATIC       0 = AUTOMATIC
Bit 1: 1 = CONTROLLED
Bit 2: 1 = POINTER
Bit 3: 1 = EXTERNAL     0 = INTERNAL
Bit 4: 1 = DEFINED
Bit 5: 1 = PARAMETER
Bit 6: 1 = BUILTIN
Bit 7: 1 = CONSTANT     0 = variable
Bit 7: 1 = contextual
           ENTRY        0 = declared ENTRY
----------------------------------------
Byte 6:

Bits 0-1:00 = not a structure element
         01 = structure element
         10 = minor structure
         11 = major structure

Bit 2  : 1 =  PACKED    0 = ALIGNED
Bit 3  : 1 =  Array
Bit 4  : 1 =  FILE
Bit 5  : 1 =  LABEL
Bit 6  : 1 =  ENTRY name
Bit 7  : 1 =  zoned decimal (T)
----------------------------------------
Byte 7:

Bit 0  : 1 =  PICTURE
Bit 1  : 1 =  sterling
Bit 2  : 1 =  arithmetic data
Bit 3  : 1 =  string data
Bit 4  : 1 =  bit string;
              0 = character string
Bit 5  : 1 =  FIXED;    0 = FLOAT
Bit 6  : 1 =  BINARY;   0 = DECIMAL
Bit 7  : 1 =  zoned decimal

If it is a structure, bits 4-7 contain
the lefthang.
```

Figure 1. Entries in the Symbol Table for Programmer-defined Variables and Label Constants (Part 1 of 2)

Byte 8: If string: Bits 0-7: length of the string if arithmetic: scale FLOAT or FIXED BINARY: bits 0-7: w scale FIXED DECIMAL: bits 0-3: w bits 4-7: d
Byte 9: Bits 0-1: block level Bits 2-7: block number
Byte 10: if structure or element of structure: level number

Byte 11:
 	
if structure: boundary of the structure
if array : current array number
if FILE : current file number

Bytes 12-13: If array : number of array elements if structure: length of the structure
Bytes 14-15: if DEFINED : name of the base variable if BASED : name of the pointer if minor structure or structure element : origin relative to the major structure
Bytes 16-17: if numeric field: offset of the picture string
Byte 18: if numeric field: length of the data
Byte 19: Number of actual block (only for checking entry names in phases B30 and B40. It will not appear in the attribute table).

Figure 1. Entries in the Symbol Table for Programmer-defined Variables and Label Constants (Part 2 of 2)

Mask Table MSKTAB

For each PL/I attribute, the mask table contains a mask of 8 bytes. Each mask is divided into two parts. The first part declares which bits in the symbol table are to be set on or off if a variable is declared with some attribute. The second part is used to check conflicting attributes. It contains a 1 in each position where a specific attribute may not appear.

The mask-table is used as follows: The corresponding mask of an attribute is put together with all other masks of the attributes previously declared for the same variable. The first part of a mask is put together by an OR instruction in register R1, the second part in register R2. If the declaration of a variable is complete, i.e., if all given attributes are composed, the mask parts in R1 and R2 are 'anded'. The result is 0 if no conflicting attributes have occurred.

The format of the mask table is shown in Figure 2. The masks are shown in hexadecimal notation.

Treatment of Errors in Variable Declarations

If an error occurs in a declaration, it is treated in the following manner:

1. The name is given the value 00 as internal representation.

2. If the name in the source text is replaced by the internal representation (see phase B80), all statements in which the name occurs are flagged.

3. The name gets an error message in the symbol table listing (see phase C00). This message is entered in byte 11 of the symbol table.

DESCRIPTION OF ROUTINES

JSCOA1 -- KC

This routine processes the block heading. It is called if a new part of the symbol table is opened. In the declaration pool, constructed in phase B15, all declarations belonging to one block are collected in a group. Each group starts with a new record and may contain more than one record. At the beginning of each group, there are four bytes containing the following information:

byte 0:	block number
byte 1:	block level
byte 2:	block number of the embracing block
byte 3:	mark if the record is the last of the group.

			Mask							
		Attribute	First Part				Second Part			
0	00	parameter	00	04	00	00	01	9A	00	00
8	01	FILE	00	00	08	00	01	EA	F6	FF
16	02	INITIAL	01	00	00	00	00	6E	CE	00
24	03	DEFINED	00	08	00	00	01	D6	0E	00
32	04	dimension	00	00	10	00	00	02	8A	00
40	05	CONTROLLED	00	40	00	00	01	AA	08	00
48	06	POINTER	00	20	00	00	00	42	2C	FF
56	07	colon	00	01	00	00	00	00	33	00
64	08	LABEL	00	00	04	00	01	2A	AA	FF
72	09	PICTURE	00	00	00	80	00	22	8C	08
80	0A	ALIGNED	00	00	00	00	00	22	2E	E7
88	0B	ENTRY	00	00	02	00	01	4A	FC	00
96	0C	BUILTIN	00	02	00	00	01	FC	FD	FF
104	0D	INTERNAL	00	00	00	00	00	16	08	00
112	0E	EXTERNAL	00	10	00	00	00	06	00	00
120	0F	PACKED	00	00	20	00	00	22	0E	E7
128	F0	BINARY	00	00	00	22	00	22	0C	58
136	F1	DECIMAL	00	00	00	20	00	22	0C	1A
144	F2	FIXED	00	00	00	24	00	22	0C	18
152	F3	FLOAT	00	00	00	20	00	22	0C	5C
160	F4	BIT	00	00	00	18	00	22	0C	E7
168	F5	CHARACTER	00	00	00	10	00	22	0C	6F
176	16	STATIC	00	80	00	00	00	4E	08	00
184	17	AUTOMATIC	00	00	00	00	01	DE	08	00
192	18	precision	02	00	00	00	00	00	00	00
200	19	ERROR	10	00	00	00	FF	FF	FF	FF
208	1A	null	00	00	00	00	00	00	00	00
216	1B	ZONED	00	00	00	01	00	22	8C	18
224	1C	ZONED (T)	00	00	01	00	00	22	8C	18
232	1D	STERLING	00	00	00	C0	00	22	8C	18
240	1E	major structure	00	00	C0	00	01	02	1E	FF
248	1F	minor structure	00	00	80	00	01	F2	7E	FF
256	20	element of structure	00	00	40	00	00	D2	A8	00
		arithmetic	00	00	00	20				
		erase data attribute	FD	DF	F0	00				
		precision	02	00	00	00				

Figure 2. Format of Mask Table

These four bytes are stored in an intermediate storage SSCOPE.

The actual position of the pointer PST pointing to the symbol table is entered into the scope table.

Abbreviations:

PARAM =	parameter-list key	(1 byte)
PARZ =	counter for parameters	(2 bytes)
LAREC =	key for last record	(1 byte)
EOREC =	end-of-record key	(3 bytes)

JLABA1 -- KE

This routine processes the statement-label constants and the entry names.

A statement label has the form:

Identifier	(3 bytes)
Colon	(1 byte)

An entry name has the form:

Identifier	(3 bytes)
Attribute ENTRY	(2 bytes)

Optional data attributes:

EOS key	(6 bytes)

JDCLA1 -- KF

Secondary entry points: JDCLD2, JDCLK2

This routine processes the DECLARE statement. Since attributes may be nested, a DECLARE statement is first scanned to the EOS key. At this time an intermediate table AHSTAB is constructed. The data is entered starting at the end of the table in range of its appearance.

The following information may appear:

1. User-defined names:
   Starting with the identifier key, length 3 bytes. Processing: (see Subroutine JNAMA1).

2. Structure level:
   Starting with a number, length up to 3 bytes. Processing: The integer is converted from decimal to binary and saved in a current level storage.

3. Attributes:
   Starting with an attribute key, length 2 bytes. Processing: The two bytes are entered into AHSTAB.

4. Precision:
   Starting with a left parenthesis. Processing: The precision is converted and entered into AHSTAB (see JPREA1).

5. Left parenthesis:
   Special key, length 1 byte. Processing: A parenthesis counter is increased. The current level is stored in the internal buffer LEVPDS. The key is entered into AHSTAB.

6. Right parenthesis:
   Special key, length 1 byte. Processing: The parenthesis counter is decreased. The key is entered into AHSTAB.

7. Comma:
   Special key, length 1 byte. Processing: The actual level is reloaded from the internal buffer.

8. End of statement:
   Starting with an EOS key, length 6 bytes. Processing: See JATAA1.

JSATA1 -- KH

This routine scans the attributes. Normally, only the attributes are entered into AHSTAB; however, some attributes are given a special treatment:

1. ARRAY: code X'04'
   Four bytes following the attribute are entered in AHSTAB.

2. FILE: code X'01'
   One byte following the attribute is entered in AHSTAB.

3. PICTURE: code X'09'
   Nine bytes following the attribute are entered in AHSTAB.

4. CONTROLLED: code X'05'
   Five bytes following the attribute are entered in AHSTAB.

5. DEFINED: code X'03'
   Three bytes following the attribute are entered in AHSTAB.

JPREA1 -- KI

This routine converts the precision given in the source text to a 2-byte form and stores it in the intermediate table AHSTAB.

A precision has one of the following forms:

(w) or (s) or (s,d)

where w, s, and d are unsigned decimal integer constants having the following range of values:

$1 \leq w \leq 255,\ 0 \leq s,\ d \leq 15.$

The result of the conversion has the following form:

Byte 0: precision key
Byte 1: binary value of the precision.

The last bit of the key (byte 0) declares the form of the precision. If the form is (w) or (s), the bit = 0. If the form is (s,d), the bit = 1.

Entry parameter:
PIN = address of the left parenthesis of the precision.

Return parameter:
PIN = address of the byte after the right parenthesis.

JATAA1 -- KJ - KN

This routine processes the intermediate table AHSTAB and generates the symbol table SYMTAB. The items which may be entered in AHSTAB and their processing are described below.

1. Attributes
   Representation: 2 bytes

byte 0	=	attribute key
byte 1	=	specification

   Byte 1 addresses an entry in a mask table MSKTAB (see Figure 2 in phase B20). Parts 1 and 2 of the mask are taken from MSKTAB and added with an OR instruction to the already existing information in registers 1 and 2.

   Some attributes get additional treatment.

   a. Dimension
      Representation: additional 4 bytes

byte 2	= reserved
byte 3	= current array number
bytes 4-5	= number of array elements

      Bytes 3-5 are stored in a special location.

   b. FILE
      Representation: additional 1 byte

byte 2	= current file number

      Byte 2 is stored in a special location.

   c. PICTURE
      Representation: additional 9 bytes

byte 2	= left parenthesis
byte 3	= binary length of data
byte 4	= right parenthesis
byte 5	= string constant key
bytes 6-7	= offset of the string constant
byte 8	= string constant key
bytes 9-10	= length of the string constant

      Bytes 3, 6 7, and 10 are stored in a special location.

   d. CONTROLLED
      Representation: additional 5 bytes

byte 2	= left parenthesis
byte 3	= identifier key
bytes 4-5	= name of the pointer coded in phase A30
byte 6	= right parenthesis

Bytes 4 and 5 are stored in a special location.

e. DEFINED
Representation: additional 3 bytes

byte 2	= identifier key
bytes 3-4	= name of the base variable

Bytes 3 and 4 are stored in special location.

2. Precision
Representation: additional 2 bytes

= byprecision key
= byprecision in binary form

There are two different keys. If the precision is of the form (w), bit 7 of the key is 0. If the form is (s,d), bit 7 is 1.

3. Name
Representation: additional 4 bytes

byte 0	= identifier key
bytes 1-2	= user-defined name coded in phase A30
byte 3	= current structure level 1.

First, if necessary, the default attributes are added in routine JDFAA1. A test on conflicting attributes follows. If there are no conflicts, the entry in the symbol table is constructed. Finally, registers 1 and 2 are reloaded from the internal buffer.

4. Right parenthesis:
Representation: 1 byte

A parenthesis counter is increased and the contents of registers 1 and 2 are moved into the internal buffer.

5. Left parenthesis:
Representation: 1 byte

The parenthesis counter is decreased and the contents of registers 1 and 2 are restored from the internal buffer.

At points 4 and 5, the functions of the right and left parenthesis are reversed, because construction of AHSTAB in routine JDCLA1 begins at the bottom of the table and the processing sequence is inverted.

Entry parameter:
PAHS = address of the first byte in AHSTAB to be processed.

JTRLA1 -- KO

This routine processes the block trailing. It is called if a part of the symbol table is closed.

If the end of a group in the declaration pool is reached, 4 bytes are moved into the symbol table. The first 2 bytes get a mark specifying the end of the part. The second 2 bytes contain the number of the embracing block.

If the source text contains file declarations, or if a table overflow occurs, the part of the symbol table is written on a work file.

Abbreviations used in this routine:

PST	= Pointer symbol table
IJKMTS	= Start address of table area
SWTOV	= Switch table overflow
IJKMBC	= Block counter
SSCOPE	= Storage for scope information
BSCOPE	= Scope chain
TTEXT	= Relative TABTAB entry for external table
SCOTAB	= Start address of scope table
IJKMTT	= Start address of master table TABTAB.

Subroutine JNAMA1 -- KP

This subroutine moves the user-defined name and the current level number into AHSTAB. If no structure level is given, zero is inserted.

Entry parameter:
PIN = start address of the name.

Return parameter:
PIN new = PIN old + 3.

Note: The total number of names declared in one DECLARE statement is restricted to 65. This restriction is valid for the minimum machine configuration. If the table space is increased by 20 bytes, the number is increased by 1 name.

Subroutine JAHSA1 -- KQ

This subroutine transfers information to an intermediate table AHSTAB and controls the pointer PAT for this table.

The table is built in the buffer area and uses three buffers. Construction of the table starts at the end.

Since a DECLARE statement cannot be longer than three buffers and the AHSTAB cannot contain more than one statement, an overflow cannot occur.

Entry parameters:
PIN = start address of the information to be transferred.
BYZ = number of bytes

Return parameter:
PIN new = PIN old + BYZ.

<u>Subroutine JPCOA1 -- KR</u>

This subroutine controls the input pointer PIN and inserts a new record in the declaration pool, if necessary.

Generally, it is possible to process the information sequentially. But because identifiers or correlated expressions must not be divided by the buffer end, two input buffers are used. When pointer PIN reaches the second buffer, the contents of the second input buffer are moved into the first and a new record is read.

<u>JPUTA1 -- KS</u>

The routine writes the symbol table. It is called if a table overflow occurs or if the current source text contains a file declaration. The symbol table is divided into parts. Each part contains all declarations given for one block of the source program.

The scope table SCOTAB contains an entry for each part.

SCOTAB+4 = relative start address of a part
SCOTAB+2 = relative end address of a part

If the symbol table is written, each part starts with a new record. The following information is moved into the scope table:

SCOTAB+0 = number of records belonging to this part (1 byte).
SCOTAB+1 = record identification for the record (3 bytes)
SCOTAB+4 = 00 (2 bytes)

<u>JCWTA1 -- KT</u>

This routine converts an unpacked decimal integer constant to binary representation. The decimal number may have up to 9 digits.

Entry parameter:
PIN = start address of the decimal constant.

Return parameters:
HR1 = value of the converted constant.
HR2 = number of digits of the decimal constant.
PIN = address of the first byte after the decimal constant.

This phase has the following functions:

1. to perform the syntactical scan of the file declarations;
2. to test the file declarations for conflicting or missing attributes and options;
3. to build up the file table FILTAB and to replace the file declaration statements by NOP statements.

Notes: Phase B25 is skipped if there are no file declarations in the source program. The information required to point to the third record of NAMTAB has been stored in IJKMIP+4 in phase A30. The internal name of the first file has been stored in IJKMIP in phase B20.

Phase Input and Output

The input is a string of 3-byte elements and/or elements of variable length.

The file declaration statements have the following format:

FSK	VN	CFN	file description	EOS

where FSK = file statement key = X'E00043' (3 bytes)
VN = variable name (3 bytes)
CFN = current file number (3 bytes)
EOS = end-of-statement key (6 bytes)

The output differs from the input only in that the file declaration statements have been replaced by NOP statements.

The File Table

This table (FILTAB;ZTAB03) is written on SYS001 (recordsize = length of one entry = 20 bytes). Each entry has the format shown in Figure 1.

BYTE	MEANING	
0-1	internal name	
2	bit 0	1 = RECORD, 0 = STREAM
	bit 1	1 = INPUT
	bit 2	1 = OUTPUT
	bit 3	1 = UPDATE
	bit 4	1 = PRINT
	bit 5	1 = STREAM
	bit 6	1 = KEYED
	bit 7	1 = BACKWARDS
3	bit 0	1 = DIRECT, 0 = SEQUENTIAL
	bit 1	1 = CONSECUTIVE
	bit 2	1 = REGIONAL(1)
	bit 3	1 = REGIONAL(3)
	bit4-6	not used
	bit 7	1 = UNBUFFERED,
		0 = BUFFERED
4	bit 0	1 = KEYLENGTH
	bit 1	1 = F
	bit 2	1 = V
	bit 3	1 = U
	bit 4	1 = BUFFERS(2)
		0 = BUFFERS(1)
	bit 5	1 = LEAVE
	bit 6	1 = NOLABEL
	bit 7	1 = VERIFY
5	keylength	
6	000 - 244 = SYS000 - SYS244	
	251 = SYSIPT	
	252 = SYSLST	
	253 = SYSPCH	
7	X'10' = 2540 (card reader or punch)	
	X'11' = 1442 (card reader or punch)	
	X'12' = 2520 (card reader or punch)	
	X'13' = 2501 (card reader)	
	X'20' = 1403 (printer)	
	x'21' = 1404 (printer)	
	X'22' = 1443 (printer)	
	X'23' = 1445 (printer)	
	X'40' = 2400 (tape)	
	X'80' = 2311 (disk)	
8- 9	blocksize	
10-11	recordsize	
12-19	not used	

Figure 1. Format of File Table Entries

To scan the file declarations for conflicting attributes and options, every attribute is assigned to a bit position of a bit string of 32 bits. The mapping is identical to bytes 2-4 of the file table. The last byte contains the following:

bit 0	:	1	= F with recordsize
bit 1	:	1	= card reader or punch
bit 2	:	1	= printer
bit 3	:	1	= tape
bit 4	:	1	= disk
bit 5	:		not used
bit 6	:	1	= ENVIRONMENT
bit 7	:	1	= MEDIUM

In addition, every attribute and option is assigned to a bit string consisting of two substrings of 32 bits. In the first substring, all bits except that of the characteristic bit position, which may be 0 or 1, are zero. In the second bit string, a bit is set to 1 only if it is the characteristic bit position of a conflicting attribute or option. All the bit strings of attributes and options appearing in the file declaration are OR-ed.

If the logical product (AND) of the resulting two substrings is ≠ 0, the file declaration contains conflicting attributes and/or options. Conflicts in attributes and/or options are illustrated in Figure 2 (X means conflict).

Errors

Errors found in this phase may cause one of the error messages 188-216. For the individual messages, refer to the SRL publication IBM System/360, Disk and Tape Operating Systems, PL/I Programmer's Guide, Form C24-9005.

The name of a file is set to 0 in the file table if the corresponding file declaration contains an error of the severity T. Statements in which incorrect file names occur are not flagged.

Initialization -- LA

This is the beginning of the main routine. It initializes pointers, switches, etc., and reads input text into 4 buffers.

FSCN -- LB

This is part of the main routine. It performs the general scan over the source text.

Note: A file declaration statement is not preceded by any label.

FFIL -- LC

This is part of the main routine. It scans the file-declaration statement for acceptable attributes by means of an attribute table that has the following format:

K	0000	B	B

where K = last two bytes of the 3-byte key the keyword is represented by;

B = bit string (see the section The File Table).

The table is terminated by X'FF'. When the routine is entered, the general registers R4 and R5 are cleared. They are then OR-ed with every bit string of a file attribute found in the statement. Any element that is not a file attribute is ignored. FERR is called to note error message 189. If the ENVIRONMENT attribute is found, control is passed to FENV. Reaching the EOS key causes control to be transferred to FFIT.

FENV -- LD

This is part of the main routine. It scans the options of the ENVIRONMENT attribute by means of an options table that has the following format:

K	A	B	B

where K = last two bytes of the 3-byte key the keyword is represented by:

A = address relative to FENV of the routine processing the option, i.e.,

FBUF	for	BUFFERS
FMED	for	MEDIUM
FFIX	for	F
FUVN	for	U/V
FREG	for	REGIONAL
FKEL	for	KEYLENGTH

B = bitstring (see the section The File Table).

The table is terminated by X'FF'.

The bit strings of the option found are OR-ed into general registers R4 and R5. Then control is transferred to one of the abovementioned routines. Any element that is not an option found before reaching the right parenthesis of the ENVIRONMENT attribute is ignored. FERR is called to note error message 189, and control is transferred to FNOP to bypass a possibly following specification, e.g., (14).

	STREAM	RECORD	INPUT	OUTPUT	UPDATE	PRINT	DIRECT	KEYED	SEQUENTIAL	BUFFERED	UNBUFFERED	BACKWARDS	EXTERNAL	ENVIRONMENT	CONSECUTIVE	REGIONAL (1)	REGIONAL (3)	F (B)	F (B,R)	V	U	BUFFERS (1)	BUFFERS (2)	LEAVE	NOLABEL	VERIFY	MEDIUM	Card reader+punch	Printer	Tape	Disk	KEYLENGTH
STREAM		X			X		X	X	X	X	X	X				X	X		X	X	X											
RECORD	X					X																										
INPUT				X	X	X																							X			
OUTPUT			X		X							X																				
UPDATE	X		X	X		X						X																X	X	X		
PRINT		X	X		X			X				X								X	X							X				
DIRECT	X								X	X	X				X				X	X	X		X					X	X	X		
KEYED	X					X																						X	X	X		
SEQUENTIAL	X						X									X	X															
BUFFERED	X						X				X																					
UNBUFFERED	X						X			X						X	X		X	X		X	X					X	X			
BACKWARDS	X			X	X	X														X								X	X		X	
EXTERNAL																																
ENVIRONMENT																																
CONSECUTIVE							X									X	X															X
REGIONAL (1)	X								X		X				X		X											X	X	X		X
REGIONAL (3)	X								X		X				X	X												X	X	X		
F (B)																			X	X	X											
F (B,R)	X						X				X							X		X	X							X	X			
V	X					X	X				X	X						X	X		X							X	X			
U	X					X	X											X	X	X								X	X			
BUFFERS (1)											X												X									
BUFFERS (2)							X				X											X										
LEAVE																												X	X		X	
NOLABEL																												X	X		X	
VERIFY																												X	X	X		
MEDIUM																																
Card reader+punch					X	X	X	X			X	X				X	X		X	X	X			X	X	X			X	X	X	X
Printer			X		X		X	X			X	X				X	X		X	X	X			X	X	X		X		X	X	X
Tape					X		X	X								X	X									X		X	X		X	X
Disk												X												X	X			X	X	X		
KEYLENGTH																																

Figure 2. Conflicting File Attributes and Options

FSPE -- LE

Secondary entry points: FSPE02, FSPE03

This subroutine performs the syntactical scan of the options that must be followed by an integer enclosed in parentheses, e.g., KEYLENGTH(10). The integer is converted to binary and returned in general register R3.

If the option is not followed by a left parenthesis and a decimal digit, the routine returns false to (LINK), otherwise true to 4(LINK).

FINT -- LF

Input parameter:
PIN: points to the first digit of the decimal integer to be converted to binary.

Output parameters:
R3: converted integer.
PIN: points to the first byte following the integer.

This subroutine converts a decimal integer to binary. If the integer consists of more than 9 decimal digits, R3 is set to 32,768 = maximum blocklength + 1.

FBUF -- LG

This is part of the main routine. It scans the BUFFERS option and OR-es the bit strings of BUFFERS(1) or BUFFERS(2) into R4 and R5.

FMED -- LK

This is part of the main routine. It scans the MEDIUM option and inserts the number of the logical device and the key for the physical device type into the file table.

FSYS -- LL

Input parameters:
PIN : points to the 3-byte key the logical device name has been replaced by.
RSTNAM: number of a name table record that has already been read into storage (initialized with 0).

Output parameters:
PIN : = PIN+3.
R1: points to the name-table entry of the logical device name.

This subroutine retrieves the logical device name from the name table.

FPDT -- LM

This subroutine tests the number specified for the physical device type and inserts the respective device code into the file table. It OR-es the corresponding bit strings into R4 and R5.

FBLO -- LN

Secondary entry point: FBLO02
This subroutine checks whether the blocksize specification is greater than 32,767 and inserts it into the file table if it is less or equal.

FFIX -- LO

This is part of the main routine. It scans the F option and OR-es the corresponding bit strings into R4 and R5 if blocksize and recordsize are specified.

FUVN -- LP

This is part of the main routine. It calls FBLO to test the blocksize specification of the U or V option.

FREG -- LQ

This is part of the main routine. It scans the REGIONAL option and OR-es the corresponding bit strings into R4 and R5 if REGIONAL(1) or REGIONAL(3) is specified.

FKEL -- LR

This is part of the main routine. It checks whether the KEYLENGTH specification is greater than 255 and inserts it into the file table if it is less or equal. It inserts 255 if it is high and notes error message 194.

FNOP -- LS

Input parameter:
OLP : number of open left parentheses.

This is part of the main routine. It searches for right parenthesis (if OLP ≠ 0) to transfer control to FENV10.

If the end-of-statement key is found before a right parenthesis is detected, control is transferred to FFIT.

FFIT -- LT, LU, LV, LW, LX

This is part of the main routine. It adds default attributes or options, if necessary, and builds up bytes 2-4 of the file table. It tests for:

1. conflicting attributes or options by forming the logical product of R4 and R5;

2. missing attributes or options;

3. conflicts that cannot be detected by the general method;

4. unpermitted combinations of function attributes or physical devices with logical system units;

5. blocksize specifications that are outside of device depending limits or incompatible to the rules concerning division by recordsize or 8, respectively.

At the end of the routine, FEOS is called.

FERR - LY

Input parameter:
R0 : error number

This subroutine inserts the error number into the error table. After seven numbers have been inserted, error 215 with the severity code T is noted as 8th error. The end of statement is searched for, and control is transferred to FEOS.

FEOS -- LZ

This is part of the main routine. It inserts the file name into the file table and writes the table on SYS001. If errors of the severity code T have been detected, the file name is set to 0. A NOP key is moved into the output buffer for the file declaration. Control is then transferred to FSCN35 to continue the general scan.

JEOS -- L1

This subroutine positions the contents of input buffers 1-4 so that the currently scanned EOS is in input buffer 1 (this is done by moving and by reading in new records). It puts out the EOS and the error codes attached to it. If additional error codes have been generated, they are also put out.

JTRN -- L2

Input parameters:
PIN : pointer of source text.
POUT : pointer of output buffer.
BYZ : number of bytes to be moved.

Output parameters:
PIN : = PIN + BYZ.
POUT: address of next free byte within the output buffer.

If not all the bytes to be moved fit into the output buffer or if they do exactly fit, the buffer is filled with the first part of the text to be moved. The buffer contents are written on a work file and the remaining bytes, if any, are moved to the begin of the buffer.

PHASE PL/IB30 (SYMBOL TABLE CONSTRUCTION II) -- MA

This phase checks the symbol table constructed by phase B20. Each variable in the symbol table is tested for multideclaration.

Secondary entries in function procedures are tested to determine if they have the same attributes for return values as the main entry.

If the attribute CONTROLLED or DEFINED is given, the internal representation of the pointer variable or base variable, respectively, is set into the symbol table.

DESCRIPTION OF ROUTINES

Note: The routines JTRNA1 and MOVEA1 are described in phase A50. The corresponding flow charts are F3 and F0.

Initialization -- MB

Phase B20 constructs the scope table SCOTAB (see phase B20).

If the source program has no file declarations, i.e., if phase B25 is skipped, phase B20 leaves the scope table in the buffer area IJKMBS. Otherwise, the scope table is written onto a work file.

JRSTA1 -- MC

Secondary entry point: JRSTD2
Phase B20 has constructed the first version of the symbol table. If no symbol table overflow occurred, the symbol table is still in storage and the routine only initializes the pointer PST with the start address of that part of the symbol table that belongs to the block to be processed. Otherwise, this routine reads in part of the symbol table and loads the start address into PST.

Entry parameters:
BLZ   = number of block to be processed
SCOTAB = scope table (see phase B20)

Return parameter:
PST   = symbol table start address

JCSTA1 -- ME

This routine checks the symbol table. If a variable has the attribute CONTROLLED or DEFINED, the internal representation of the pointer variable or base variable, respectively, is moved into the symbol table.

For testing multi-declaration, each entry of the symbol table is compared with all other entries belonging to one block of the source text. Multi-declaration is given if two entries have the same name. In this way the internal representation of the pointer or base variable is set into the entry of the CONTROLLED variable and/or DEFINED variable, if both entries are in the table area at the same time.

This is done in the following manner: Assume the variable compared with all others is named A. The other is named B. If B has the attribute CONTROLLED, it is determined if A is the corresponding pointer. This is possible if the pointer is declared in the same block earlier than the controlled variable and the part of the symbol table belonging to this block is not longer than the table area. In this case, the internal representation is moved in and the movement is marked by a special bit. In the other case, if A has the attribute CONTROLLED and the entry is not marked, the pointer is searched by reading the symbol table for the current and the embracing blocks successively in a special area (in routine JSPOA1).

Entry parameters:
PST = address of the actual entry in the symbol table.
HR4 = number of records in the table area which have not yet been read.

JNSTA1 -- MF

This routine reads the next record of the symbol table if the whole table belonging to one block of the source text is not in the table area.

Entry parameters:
HR1 = address of the actual variable B (see Routine JCSTA1, Phase B30).
GRADR = limiting address of the area containing the symbol table.
NOTES = note information for the first record not yet read.
HR4 = number of records not yet read.

Return parameter:
HR1 = address of the next variable B.

Subroutine JSPOA1 -- MG

This subroutine searches for the pointer variable or base variable if they are not declared in the same block and earlier than the CONTROLLED or DEFINED variable, or if a table overflow occurs due to the number of declarations.

Entry parameter:
PST = address of variable A (see Routine JCST).

JMDCA1 -- MH

This routine checks for multi-declaration. This is given if two or more identical names appear in one block. An exception from this rule is qualified names. It is possible for a name to refer to more than one variable or data aggregate if the identically named items are parts of different structures. In order to avoid any ambiguity in referring to these identically named items, it is necessary to create a unique name. This is done by forming a qualified name. This means that the name common to more than one item is preceded by the name of the structure in which it is contained. This, in turn, can be preceded by the name of the structure in which it is declared, and so on. Multiple declaration for qualified names is given if they have identical qualifications. The qualification for the first name compared is made in routine JQULA1 and stored in area QUALF1. For the second name, the qualification is stored in QUALF2.

Entry parameters:
PST = address of the first name compared.
HR; = address of the second name.

JCHEA1 -- MI

Secondary entry point: JCHED1

This routine checks the ENTRY attribute. The first entry name in the outermost procedure has the block level 0. All secondary entry names have level 1.

JQULA1 -- MJ

This routine assigns qualifications to structure items (see Routine JMDCA1).

A test is performed to determine if all bit string data contained in the data aggregates, i.e., arrays or structures, have the attribute ALIGNED.

Entry parameter:
PST = address of the name to be qualified.

Return parameter:
QUALF1 = qualification.

JCCBA1 -- MK

Secondary entry point: JCCBB2

This routine checks the base identifier and changes the name of the base identifier or pointer into the internal representation. The program has two entries:

Entry parameters:

Main entry:
    HR1 = address of the defined identifier
    PST = address of the base identifier

Secondary entry:
    PST = address of the defined identifier
    HR1 = address of the base identifier

Return parameters:
    HR1 = unchanged. PST = unchanged.

PHASE PL/IB40 (STRUCTURE MAPPING) -- MZ

This phase calculates the storage requirements of structures. This calculation is referred to as structure mapping.

A structure is a data aggregate containing items of different types that are grouped in a given order and in such a way that the overall storage requirement is a minimum. The individual structure items have different and independent requirements of length and positioning with respect to hardware boundaries.

Each element of a structure has three mapping parameters: the alignment A, the length L, and the lefthang H. The values of the parameters depend on the declaration of the structure as shown in Figure 1. The alignment is identical to the hardware boundary requirement of the respective structure element. In DOS/TOS PL/I, there are three possible alignments levels: 1 = byte boundary, 4 = word boundary, and 8 = double-word boundary. The length is the length in bytes of the element. Data items are stored right-adjusted to their boundary. This implies the use of a third property: the lefthang. The lefthang is the number of bytes of an element (or a combination of elements) that are to the left of the alignment point of that element.

Data type	A	L	H
Numeric field	1	n	0
Float decimal short	4	4	0
Float decimal long	8	8	0
Float binary short	4	4	0
Float binary long	8	8	0
Fixed binary	4	4	0
Fixed decimal numeric Field	1	n	0
Fixed decimal (p, q)	1	Floor ((p+2)/2)	0
Bit string	1	Ceil (n/8)	0
Character string	1	n	0
Pointer	4	3	3
Label variable	4	8	0

Figure 1. A, L, and H for Structure Items

Assume the following structure:

```
1 S ,
 2 S1 CHARACTER (5) ,
 2 S2 FLOAT (16) ,
 2 S3 CHARACTER (2) ;
```

Figure 2 then shows the relationship between A, L, and H after the structure has been mapped.

```
|                                |
|<------------L--------------->|
|           |                    |
|<----H---->|
|           |
|           |
+-----------+-------------+---+---
|    S1     |     S2      |S3 |
+-----------+-------------+---+---    |
            |<----A------>|<---A---->|
            |             |          |
       alignment     alignment    alignment
         point         point        point
```

Figure 2. Relationship between A, L, and H after Mapping of a Structure

Figure 2 shows that L is independent of A and H. The value of A has two meanings:

1. The actual storage address of the byte immediately to the right of an alignment point (boundary) must be divisible by A;

2. The number of bytes between two alignments points (a boundary interval) is equal to A.

The value of H is made unambiguous by the condition

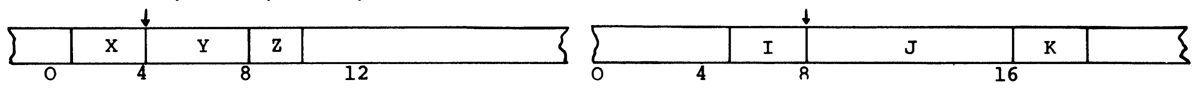
$$0 \leq H < A$$

To completely map a structure, all minor structures, if any, that contain only elementary items or arrays must be mapped first. (Refer also to the discussion of structure mapping in the DOS/TOS PL/I Programmer's Guide.) The mapping begins with the first (leftmost) element, whose mapping parameters are taken from Figure 1. The next element is appended to the right. Assume that the mapping parameters of the left and the right element are A_1, L_1, H_1 and A_2, L_2, H_2, respectively (see Figure 3, step 1). Different situations will then occur depending on the relationship between the two sets of mapping parameters, and a resulting set of parameters A_3, L_3, H_3 is generated that describes the mapping of the two elements as one compound item.

```
DECLARE 1 M,
          2 V,
            3 X POINTER,
            3 Y BINARY FIXED (31)
            3 Z BIT (16)
          2 W,
            3 I CHARACTER (3),
            3 J DECIMAL FLOAT (16),
            3 K CHARACTER (3);
```

Step 1. Mapping of the individual minor structures V and W results in:

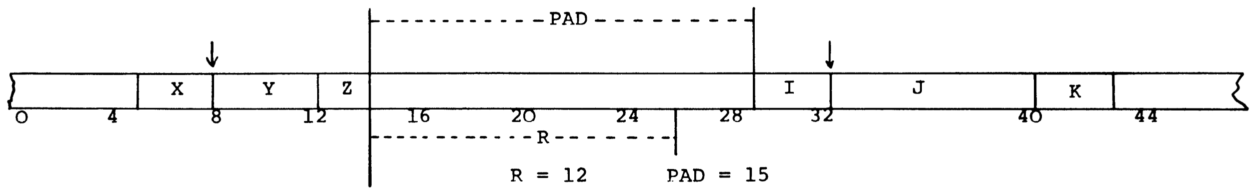
$H_V=3, \quad A_V=4, \quad L_V=9$ $\qquad$ $H_W=3, \quad A_W=8, \quad L_W=14$



The new alignment requirement of M (V and W mapped together) is:

$A_m=MAX(A_V,A_W)=MAX(4,8)=8$ $\qquad A_m = 8$

Step 2. To map V and W, V is put to the left of W at $A_m = 8$. Since only the alignment boundary and not the actual storage position is examined, the actual location in storage is of no interest as long as the boundary requirements are observed. In the following example, A_V is assumed to be at byte 8 and A_W at byte 32.



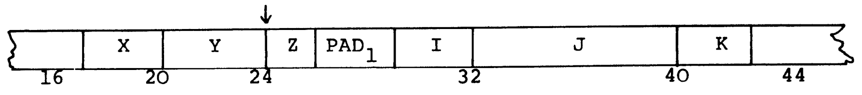
R = 12 $\qquad$ PAD = 15

$H_m = H_V - R \text{ modulo } A_m = 3 - 12 + 2 8 = 7$ $\qquad H_m = 7$

$PAD_1 = PAD - R = 15 - 12 = 3$

$L_m = L_V + PAD_1 + L_W = 9 + 14 + 3 = 26$ $\qquad L_m = 26$

Step 3. This results in the following structure map for M:



with the parameter set:

$A_m = 8$

$H_m = 7$

$L_m = 26$

Figure 3. Structure Mapping Example

1. Since items with lower boundary requirements can also be aligned at a higher boundary, but not vice versa, the following formula applies:

   $A_3 = MAX(A_1,A_2)$

2. It may happen that the two items so mapped are not contiguous, e.g.,

   ```
   1 X,
     2 B BINARY FIXED,
     2 C POINTER;
   ```

   In this case, there is a slack byte S between B and C (see Figure 4). The area occupied by the slack byte must then be added to the resulting length L_3:

   $L_3 = L_1 + L_2 + PAD$

   where PAD is in the region

   $0 \leq PAD < A_3$

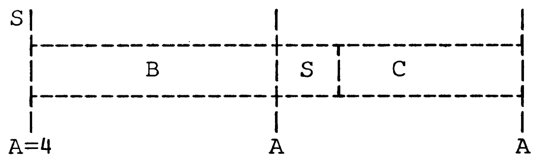


Figure 4. Inclusion of a Slack Byte S

3. If padding, i.e., inclusion of slack bytes, becomes necessary and A_1 is less than A_2, padding can possibly be minimized by moving the left element to the right as close as possible to the right element. After the shift, boundary requirement A_1 must still be satisfied for the left element.

This process can be described as follows: R is the amount of the right shift. Before shifting, the left element can be assumed to be on boundary A_3 with its (unmodified) lefthang H_1 (see Figure 3, stept 2). If the left element is then shifted R bytes to the right, the lefthang becomes:

(1) $H_3 = H_1 - R$

If H_1 is less than R, one boundary interval (A_3 bytes) on the left becomes unused and may now be disregarded. The lefthang is computed instead from the next boundary to the right by increasing H_1 by A_3. For $H_1 < R$, the new lefthang is:

(2) $H_3 = H_1 - R + A_3$

Formulas (1) and (2) can be combined to

(3) $H_3 = H_1 - R + n*A_3$

where n is 1 if H_1 is less than R; otherwise n is zero.

To have the left element adjusted at its proper boundary, R must fulfill the requirement:

(4) $\mathrm{MOD}(R,A_1) = 0$

The next formula gives the resulting padding reduction:

(5) $0 \le PAD_1 = PAD - R < A_1$

where PAD is the originally required padding (as described under item 2 above) and PAD_1 is the (reduced) padding after the right shift. The formula for L_3 then changes to

$$L_3 = L_1 + L2 + PAD_1$$

This is illustrated in step 3 of Figure 3.

The amount of padding (PAD or PAD_1) can also be formalized. The offset O_1 of the leftmost byte of the left element to the nearest boundary A_3 is

$$O_1 = L_1 - H_1 - n_1*A_3$$

where n_1 must be suitably chosen to satisfy

$$0 \le O_1 < A_3$$

(Multiples of A_3 are of no interest because of the minimum condition $PAD < A_3$.)

The padding PAD is then the difference between $A_3 - O_1$ (the number of unused bytes up to the next boundary A_3) and H_2. If H_2 is larger than $A_3 - O_1$, PAD becomes negative, i. e., there is not sufficient space to start the right element in the same boundary interval so that it must start in the same relative position in the next boundary interval to the right. This means that A_3 is added to PAD.

The multiples of A_3 can be extracted by using modulo arithmetic. This results in

$$PAD = -O_1 - H_2 \pmod{A_3}$$
$$= -L_1 + H_1 - H_2 \pmod{A_3} \text{ or}$$

(6) $PAD + L_1 - H_1 + H_2 = 0 \pmod{A_3}$

From formula (5) above we obtain

(7) $PAD_1 + L_1 - H_1 + H_2 = 0 \pmod{A_1}$

The value R defined by formula (3) can also be explained in modulo arithmetic. For convenience, its complement

(8) $T = H_3 - H_1$

is developed here. Starting from formulas (5) and (6) we obtain

$$PAD_1+R+L_1-H_1+H_2=0 \pmod{A_3}$$

or, by applying formulas (3) and (8),

(9) $PAD_1-T+L_1-H_1+H_2=0 \pmod{A_3}$

Since A_3 is divisible by A_1, comparison of (9) and (7) yields

$$-T = 0 \pmod{A_1}$$

which is equivalent to the auxiliary condition (4).

The next element to the right can now be mapped by taking the previously mapped compound item as the left element and so forth until all elements of the containing minor (or major) structure have been mapped. The structure itself is thereby reduced to a compound item. When all minor structures of the lowest level have so been reduced to compound items, mapping of the next-higher-level structure (which now contains elements and compound items only) can be started. This procedure is continued until the major structure has been mapped.

Arrays are handled in a special way. If an array is not of type POINTER, A is as shown in Figure 1, H is zero, and L taken from Figure 1 is multiplied by the number of array elements. The array is then mapped in one single step like an elementary item.

POINTER arrays differ due to their lefthang. Each element of a POINTER array except the first one must be preceded by a slack byte to satisfy the proper boundary requirements. This results in A = 4, H = 3, and L = 4 *K-1, where K is the number of array elements.

Structure mapping starts with elementary items and arrays and proceeds upwards to the major structure. Structure declarations, however, are organized in the reverse direction, starting with the major structure and going down to its elements. For this reason, the structure mapping algorithm described in Figure 5 must also start at the major structure. If the declaration to be processed is not an elementary item or an array, the routine MAPP is called recursively to handle the next lower level (blocks B3 and B2 of Figure 5). On return from this recursive call, the appropriate structure has been reduced to a compound item. The routine MAPP has one input and four return parameters. The input parameter is a pointer S to the structure (major or minor) to be mapped. The return parameters are A, L, H, and the number of items N at any level contained in this structure.

With each call of MAPP, initial values for A, L, and H are generated for accumulation during the mapping process (blocks A2, H1, and H4). This initialization allows to program the mapping algorithm as an iterative process. It is equivalent to adding a dummy element with length zero, lefthang zero, and minimum boundary requirements to the left of each structure (minor or major).

When the routine is called recursively, the old values A, L, H, and N are stacked. They are available again (unchanged) after return from the recursive call. A, L, and H serve for the sets A_1, L_1, H_1 and A_3, L_3, H_3 in the above description of the process, while AA, LL, HH work as right-side element sets A_2, L_2, H_2. A global variable LV is used in this process; it contains the level at which mapping is momentarily being performed. One variable PAD is used for both PAD and PAD_1. The distinction between PAD and PAD_1 is made by a branch in block F2.

Besides A, L, and H, the mapping algorithm must provide the symbol table with the origin of each minor structure, array, and element relative to the beginning of the structure (block H2). Since all minor structures at the lowest level have been mapped independently, the relative origin of each such minor structure starts at zero. The relative origins must therefore be adjusted when minor structures are mapped as compound items (block J3). NN in block J4 is equal to the number of all items contained in a structure.

DESCRIPTION OF ROUTINES

Note: Subroutine MOVEA 1 is described in phase A50.

JRSYA1 -- NC

This routine updates the symbol table. The respective entry is pointed to by PST. If the entry is a single item, the length of the item, i.e., the number of bytes occupied at object time, is entered into the symbol table. If the entry is a structure, it is mapped.

JPRSA1 -- ND

This routine checks whether the entire structure is in storage. If required, it reads in the remaining part. After calling JMAPA1 which performs the actual structure mapping, A, L, and H are entered into the symbol table. All symbol table entries pertaining to the structure are put out.

JPOSA1 -- NE

Secondary entry point: JPOSA5

This subroutine controls two output buffers. If the buffers are full, they are written out in overlapped mode.

The secondary entry is used if a block end in the symbol table is reached. In this case, the buffer contents are written regardless of whether or not the buffer is full.

The NOTE information of the first record of a block is entered into the scone table if the main entry is used for the first time and after each block end.

Entry parameters:

PST	= start address of symbol table to be written
BYZ	= number of bytes to be written
POUT	= output area pointer
BUFST1	= start address of first buffer
BUFST2	= start address of second buffer
BUFLIM	= limiting address of buffer currently used
STRECL	= symbol table record length

Return parameter:
PST new = PST old + BYZ

JMAPA1 -- NF

This routine calculates the mapping of structures. It may be called recursively.

An internal buffer is used for storing and returning parameters. It consists of four 32-byte sections referred to as PUSH1 -PUSH4. Each buffer entry has a length of four bytes. The eight entries per buffer section represent the levels of the structure. Thus, each structure level has an entry in each of the four buffer sections. The entries have the following format:

<u>PUSH1</u>

byte 1	alignment (A)
bytes 2-4	return address (LINK)

<u>PUSH2</u>

byte 1	lefthang (H)
bytes 2-4	start address of the structure (S) being processed

<u>PUSH3</u>

bytes 1-2	length (L) of the item being processed
bytes 3-4	number of items (N) contained in the item being processed

<u>PUSH4</u>

bytes 1-2	number of the item (I) being processed relative to the embracing structure
bytes 3-4	reserved

Level counter LV is used for addressing the internal buffer.

Entry parameter:
PST = start address of structure to be mapped

Return parameters:

PUSH1 (4*LV)	=	alignment (A)
PUSH2 (4*LV)	=	lefthang (H)
PUSH3 (4*LV)	=	length of structure (L)
PUSH3 (4*LV+2)	=	number of items (N) contained in item being processed

<u>JPADA1 -- NG</u>

This routine calculates the padding and the lefthang of a structure. The padding PAD is defined as

$0 \leq PAD < A$ and

$(PAD+HH+L-H) \text{ modulo } A = 0$

If HH+L-H = X, PAD can be defined as follows:

$(PAD+X)/A=CEIL(X/A)$
$PAD=A*CEIL(X/A)-X$
$PAD=A*FLOOR((X+A-1)/A)-X$

The increment T of the lefthang H is defined as

$0 \leq T < AA$ and

$(PAD+HH+L-H-T) \text{ modulo } AA = 0$

If Y = PAD+HH+L-H = PAD+X, T can be defined as follows:

$(Y-T)/AA=FLOOR(Y/AA)$

$T=Y/AA*FLOOR(Y/AA)$

Entry parameters:

PUSH3(BYZ)	=	L	=	length of embracing structure
PUSH2+4(BYZ)	=	HH	=	lefthang of item being processed
PUSH2(BYZ)	=	H	=	lefthang of embracing structure
PUSH1+4(BYZ)	=	AA	=	alignment of item being processed
PUSH1(BYZ)	=	A	=	alignment of embracing structure
BYZ	=			(LV-1)*4

<u>JCANA1 -- NH</u>

This routine calculates the number of items contained in a structure at any level. The symbol table entries for all structure items are assumed to be stored in the table area.

Entry parameter:
PST = address of structure to be mapped

Return parameters:
PST = unchanged
N = number of items (bytes 3-4 in PUSH3)

<u>JALHA1 -- NI</u>

This subroutine calculates A, L, and H.

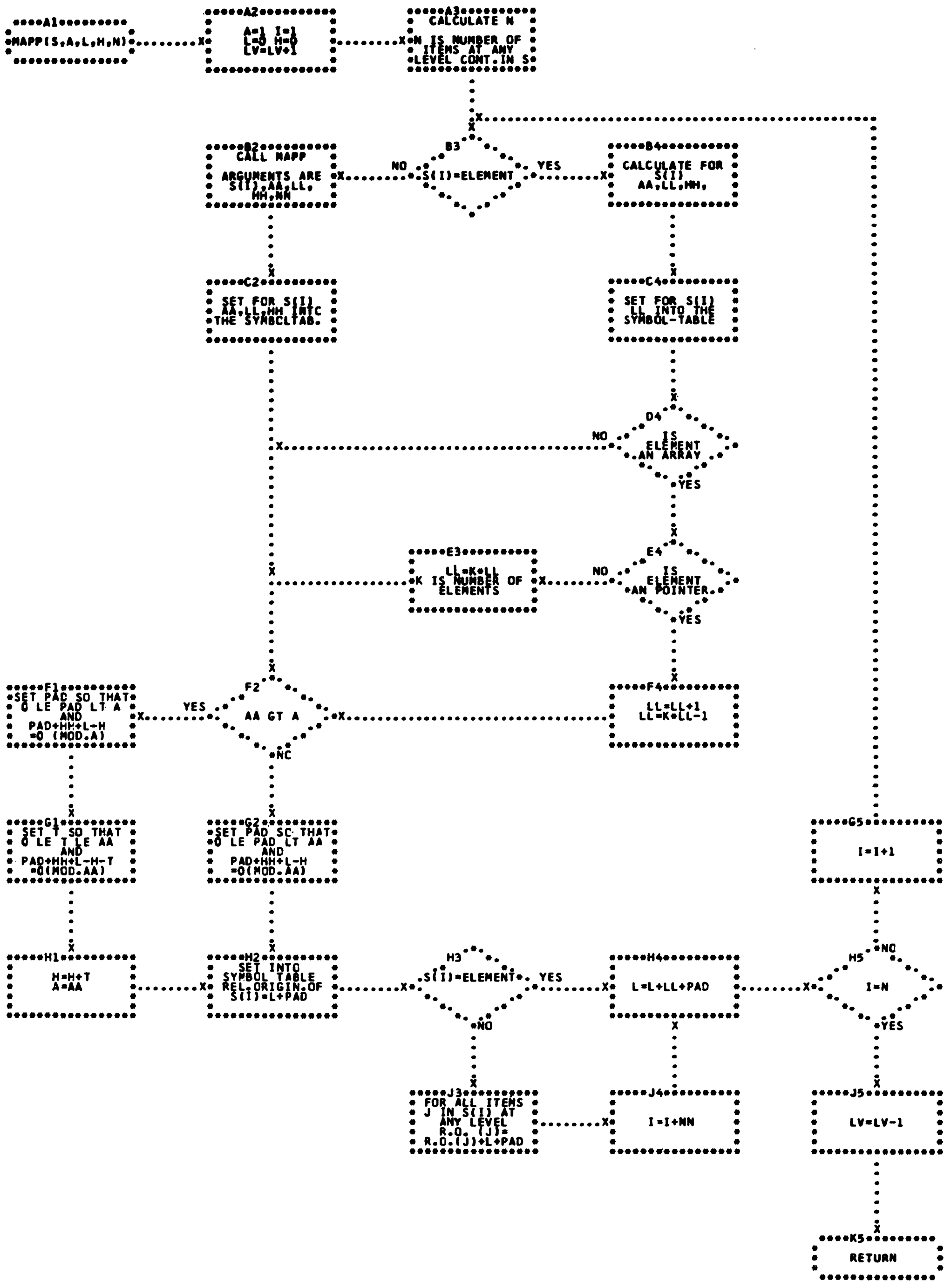

A1
MAPP(S,A,L,H,N)
A2
A=1 I=1
L=0 H=0
LV=LV+1
A3
CALCULATE N
N IS NUMBER OF ITEMS AT ANY LEVEL CONT.IN S
B2
CALL MAPP
ARGUMENTS ARE S(I),AA,LL,HH,NN
B3
S(I)=ELEMENT
NO
YES
B4
CALCULATE FOR S(I) AA,LL,HH,
C2
SET FOR S(I) AA,LL,HH INTC THE SYMBCLTAB.
C4
SET FOR S(I) LL INTO THE SYMBOL-TABLE
D4
IS ELEMENT AN ARRAY
NO
YES
E3
LL=K*LL
K IS NUMBER OF ELEMENTS
E4
IS ELEMENT AN POINTER.
NO
YES
F1
SET PAD SO THAT 0 LE PAD LT A AND PAD+HH+L-H =0 (MOD.A)
F2
AA GT A
YES
NC
F4
LL=LL+1
LL=K*LL-1
G1
SET T SO THAT 0 LE T LE AA AND PAD+HH+L-H-T =0(MOD.AA)
G2
SET PAD SC THAT 0 LE PAD LT AA AND PAD+HH+L-H =0(MOD.AA)
G5
I=I+1
H1
H=H+T
A=AA
H2
SET INTO SYMBOL TABLE REL.ORIGIN.OF S(I)=L+PAD
H3
S(I)=ELEMENT
YES
NO
H4
L=L+LL+PAD
H5
I=N
NO
YES
J3
FOR ALL ITEMS J IN S(I) AT ANY LEVEL R.O. (J)= R.O.(J)+L+PAD
J4
I=I+NN
J5
LV=LV-1
K5
RETURN

Figure 5. Structure Mapping Algorithm

PHASE PL/IB70 (CONTEXTUAL DECLARATIONS) -- OA

Phase B70 adds all contextually declared identifiers to the symbol table SYMTAB. All identifiers that either occur in a CALL statement or precede a PROCEDURE statement, an ENTRY statement, or a parenthesized list are replaced in the text string by their internal representation.

All identifiers that are built-in functions with arguments are replaced with the internal representation of the built-in functions in the text string.

Phase Input

1. Text string on TXTIN. All identifiers are identified by an E1-key.
2. Symbol table SYMTAB on SYS001. For each explicitly declared identifier, SYMTAB contains an entry with the declarations of the identifier and its internal representation.

Phase Output

1. The text string on TXTIN contains all identifiers that occur in a CALL statement, or precede a PROCEDURE statement, an ENTRY statement, or a parenthesized list characterized by an EE-key and replaced by its internal representation. All built-in functions with arguments are characterized by an EC-key and replaced by their internal representation. All remaining identifiers are characterized by an E1-key.
2. For each contextually declared identifier, block n+1 of SYMTAB in storage and/or on SYS001 contains an entry with the declaration of the identifier.

COMMUNICATION WITH OTHER PHASES

Scope Table

The scope table SCOTAB (built and described in phase B20) contains an entry for each block of the symbol table. The format of this entry is as follows:

Byte 0	Number of records of the block.
Bytes 1-3	Note key of the block on SYS001.
Bytes 4-5	Address of the block in storage relative to the beginning of the table space. If the block is not in storage, bytes 4-5 are zero.

IJKMIP

If bit 0 of IJKMIP is on, all blocks of SYMTAB are in storage. If bit 1 of IJKMIP is on, some blocks of SYMTAB are in storage. These blocks are in storage from the beginning of phase B70.

WSLIST

WSLIST is a list with an entry for each possible block level (3 entries). If block X with level number N is read, entry N of WSLIST contains:

Byte 0	Number of records of block X which are in storage.
Byte 1	Number of records of block X which are not in storage.
Bytes 2- 3	Entry of block X in the scope table.
Bytes 4- 7	Begin address of block X in storage.
Bytes 8-11	End address of block X in storage.
Bytes 12-15	Note key of the part of block X which is not in storage. If the entire block X is in storage, bytes 12-15 are zero.

The scope table and WSLIST contain the information on the location of the blocks of SYMTAB. If either one of the first two bits of IJKMIP is on, information is retrieved from the scope table only. As soon as a block that is not in storage is required in phase B70 or B80, bit 1 of IJKMIP is reset and the control of blocks in storage passes to the entries in WSLIST.

Classifying of Table Space

At the beginning of phase B70, the table space is classified for storing blocks of SYMTAB in phases B70, B75, and B80.

The table space is divided into three sections. The first section is used for storing blocks of SYMTAB. The number of records of SYMTAB that can be stored here is called M0.

The second section (starting with AN1) is used to build up block n+1. Its length is equal to the record length of SYMTAB if not all blocks of SYMTAB (except block n+1) are in storage. If all blocks (except block n+1) are in storage, the free table space is used to build up block n+1.

The third section (starting with ABS1) is called BS. This area consists of two buffers called BS1 and BS2. The length of each buffer is equal to record length of SYMTAB. BS is used for reading and scanning records of SYMTAB if a block, or part of a block, cannot be stored in the first M0 buffers of the table space without destroying other blocks that are also required for scanning. If the entire SYMTAB (except block n+1) is in storage, BS is also used to build up block n+1.

The following terms are used for classifying table space:

M0	: Number of records of SYMTAB that can be stored in the first section of the table space.
K	: Number of buffers in the table space that are used to build up block n+1. Normally, K = 1.
AN1	: Address of the area in which block n+1 is built up.
ABS1	: Address of BS and BS1.
ABS2	: Begin address of BS2.
AEBS2	: End address of BS2.
PSE	: Points to the location where the next entry of block n+1 is stored.
A0	: Address of table space.

WBSEN

Byte WBSEN contains the number of a block which is completely stored, or of which the last records are stored, in BS. If byte WBSEN is zero, no records are stored in BS.

Bit 20 of IJKMJT

Bit 20 of IJKMJT is set if a built-in function is detected in this phase.

Error Code X'45'

If an incorrectly declared identifier is found in this phase, the error code X'45' is inserted into the text string after the statement in which the incorrectly declared identifier is found.

WCTAB and Switch B75

Table WCTAB is used to indicate built-in functions coded in the text string as built-in functions, but declared by the user. If such a function is found, its matching bit in WCTAB and switch B75 are set, i.e., phase B75 will not be skipped.

Internal Pointers, Switches, and Tables

The following pointers and switches are used:

PIN	: points to the element in the input buffer which is scanned.
POUT	: indicates the address in the output buffer to which the next output will be moved.
PSY	: points to the entry of SYMTAB which is scanned.

Switch MS = Bit 0 of WSWIMS. Switch MS is set if an entry of the identifier is found in SYMTAB and the identifier is declared in this entry as a minor structure or as an element of a structure. If the identifier is declared as an array, the internal representation of it is stored. Scanning of the same block is continued, but embracing blocks are not scanned.

LVLPT: points to the WSLIST entry for the required SYMTAB block.

Scope pointer: points to the SCOTAB entry for the required SYMTAB block.

The following tables are used:

WBTAB is used to indicate the appearance of not explicitly declared built-in functions. If a not explicitly declared built-in function is found in the text string, the corresponding bit in WBTAB is set.

WTAB contains the masks for setting bits in WBTAB and WCTAB.

WNRNR contains the number of the block and the number of the embracing block of the statement being tested.

Input/Output of Text String

Three contiguous buffers are used for reading and writing of the text string. The first buffer is used as output buffer. The second buffer is the first input buffer; its address is contained in BUFB1. The third buffer is the second input buffer; its address is contained in BUFB2. The end address of the second input buffer is contained in BUFEND.

Output is performed under control of the output pointer POUT by the output routine JTRNA1 as described in phase A50.

The input pointer PIN points to the text string element to be scanned. After scanning, PIN is increased by the length of the element. If PIN points to an element not contained in the first input buffer, output of the first input buffer is performed by JTRNA1. The contents of the second input buffer are moved to the first input buffer. PIN is decreased by the buffer length and the next record is read into the second input buffer. If PIN points to an element in the first input buffer, scanning is continued.

Functional Description

The following cases are checked in this phase by scanning the text string:

1. An identifier precedes a PROCEDURE or ENTRY statement: This identifier is declared explicitly. Its entry is retrieved from SYMTAB and its internal representation is inserted into the text string.

2. An identifier occurs in a CALL statement: SYMTAB is searched for an entry of this identifier. If an entry is found, the identifier must be declared as an entry name, and its internal representation is inserted into the text string. If it is not declared as an entry name, X'EE0000' and an error message are inserted into the text string.

   If the identifier is not declared, it will be declared as an external entry name in block n+1 of SYMTAB, and its internal representation is inserted into the text string. If the name of such an identifier is equal to a built-in function, and this built-in function is noted in WBTAB, i.e., the built-in function was previously used in the text string, it is also noted in WCTAB and switch B75 is set, i.e., phase B75 will not be skipped.

3. An identifier followed by a parenthesized list occurs. SYMTAB is searched for an entry of this identifier. If an entry is found, the identifier must be declared as entry name, array, or built-in function, and its internal representation is inserted into the text string. If the declaration is not of this type,X'EE0000' and an error message are inserted into the text string. If the identifier is not declared, it is checked whether or not it is a built-in function. If it is, the identifier is replaced in the text string by the internal representation of the built-in function, and its appearance is noted in WBTAB. If the identifier is not a built-in function, it will be declared as an external entry name in block n+1 of SYMTAB, and its internal representation is inserted into the text string.

DESCRIPTION OF ROUTINES

Note: The following subroutines are used in this phase but are described elsewhere:

JERRA1 and JTRNA1 are described in phase A50. All of the remaining routines are described in phase B80:

WBSOC1	WELST3
WCAM1	WGT21
WCLEAR	WGT22
WELST1	WSETSP

Initialization -- OB

Output pointer POUT is set to the beginning of the output buffer. Input pointer PIN is set to the beginning of the first input buffer. The first two records of the text string are read into the input buffers. The begin addresses of the first and second input buffers and the end address of the second input buffer are stored in BUFB1, BUFB2, and BUFEND.

The table space is classified as described in the section Classifying of Table Space. M0, AN1, ABS1, ABS2, and AEBS2 are stored in WM0, WAN1, WABS1, WABS2, and WAEBS2, respectively.

If there are blocks in storage from previous phases, bit 1 of IJKMIP is set. If there are blocks in storage that exceed the first M0 buffers of the table space, the addresses of these blocks are cleared in the scope table. It is tested whether all blocks of SYMTAB are in storage. If they are not, K = 1. If all blocks are in storage, the address of the free table space is equal to AN1, K is equal to the number of free buffers, and ABS1 is equal to the address of the end of the table space. Pointer PSE is set to the beginning of the area used to build up block n+1.

Search for Identifier in Source Text--OC

The text string is scanned for begin of statement (statement identifier), identifier, and end of statement.

If a begin of statement is found, the number of the block and the number of the embracing block of this statement are stored in WNRNR. If the statement is a CALL statement, and no OVERLAY or a DYNDUMP is called, switch CALL is set. If OVERLAY or DYNDUMP is called, this is indicated in the statement identifier and OVERLAY or DYNDUMP is deleted in the text string.

Identifier in Source Text -- OD

PIN points to the E1-key of an identifier. If this identifier is not part of a qualified name, it is checked whether:

1. The identifier precedes a PROCEDURE or ENTRY statement, or

2. switch CALL is on, or

3. The identifier is followed by a parenthesized list.

The actions performed in these cases are described in the section Functional Description.

If the internal representation of an identifier is zero, error code X'45' is inserted into the text string by the error routine JERRA1.

Entry in SYMTAB -- OE

PIN points to the identifier an entry of which is made in block n+1 of SYMTAB as external entry name. PSE points to the beginning of the entry. The internal representation of the identifier is equal to the present value of the variable counter.

If the first character of the user-defined name of the identifier is I through N, the attributes FIXED BINARY and the length 15 are set into the entry of the identifier. Otherwise, the attributes FLOAT DECIMAL and the length 6 are set into the identifier entry.

If this entry is the last possible entry in the buffer(s) used to build up block n+1, all entries of block n+1 which are in storage are written onto SYS001, and PSE is reset. If the first bit of IJKMIP is on, it is reset; the second bit of IJKMIP is set, and K is decreased by 2, i.e., BS is now used to accommodate the part of block n+1 which is not in storage.

End of Statement or Phase -- OF

PIN points to the EA-key of End of Statement. WR4 contains the begin address of the area which is moved into the output buffer. This area including End of Statement and possible error messages from previous phases are written by the output routine JTRNA1. If an incorrectly declared identifier was found in this statement, the error bit is set on and the new error message(s) is (are) added to the possible old one. The number of error messages after a statement is limited to 8.

It is tested whether PIN points to the end of the text string. If it does, the part of the text string which is not yet on TXTOUT is written and TXTOUT and TXTIN are rewound and exchanged. If switch B75 is on, phase B75 is called; otherwise, phase B80 is called.

Check for Built-in Function -- OG

Entry : WBUIN1

Input parameter:
PIN points to the identifier which is checked if it is a built-in function.

WBTAB1 is a table which contains the second bytes of the internal representation of all built-in functions with arguments the names of which are declared in the first record of the name table NAMTAB. The second bytes are in the order of the compressed names of the built-in functions. WBTAB2 is a table which contains the second bytes of both the compressed name and the internal representations of all built-in function with arguments the names of which are declared in the second record of NAMTAB.

It is tested if the identifier is a keyword. If it is a keyword and it is declared in the first record of NAMTAB , it is checked if the keyword matches a built-in function in WBTAB1. If it does, pointers are set to the entries of this function in WTAB, WBTAB, and WCTAB, and exit YES occurs. If the keyword is declared in the second record of NAMTAB, WBTAB2 is scanned for this keyword. If it is found, pointers are set to the entries of this function in WTAB, WBTAB, and WCTAB, and exit YES occurs.

Output parameters (for exit YES):
R0 contains the internal representation of the built-in function.
R1 points to WTAB entry, R2 points to WBTAB entry, and R3 points to WCTAB entry of the built-in function.

Search for Identifier in SYMTAB -- OH-OL

Entry point: WSI0

Input parameter:
PIN points to the identifier, an entry of which is searched for in SYMTAB.

If an identifier is declared in SYMTAB, its internal representation is retrieved therefrom by WSI0. In addition, this subroutine attempts to keep blocks in storage as long as possible and reads only those blocks into storage that are required to scan for an entry for the identifier. Scanning is started in block X, i.e., in the block that contains the statement which, in turn, contains the identifier searched for. If the searched entry is not found in block X, scanning is continued in the embracing block of block X, etc. The outermost block is block n+1.

If block X of the identifier is not in storage, all blocks in storage that are not embracing blocks of block X are cleared. If there are not enough contiguous free buffers in the table space to accommodate block X, embracing blocks of block X are cleared starting with block level 1.

If the number MX of records of block X is not greater than M0, block X is stored and scanned in the table space; otherwise, the first M0 records of block X are stored and scanned in M0 buffers of the table space. The remaining records of block X are read and scanned in BS.

If scanning is continued in an embracing block of block X, block X remains in storage. If the embracing block is not in storage and the maximum of contiguous free buffers in the table space is M1, the embracing block is stored in the M1 buffers unless the number of records of the embracing block is greater than M1. Otherwise, the first M1 records of the embracing block are stored and scanned in the M1 buffers of the table space. The remaining records are read and scanned in BS.

If an entry of the identifier is found and the identifier is declared in this entry as an array in a structure, the internal representation of this entry is stored and scanning of the block is continued. If no other entry of the identifier is found in the same block, the internal representation of the array is retrieved. If another entry of the identifier is found, the internal representation of the new entry is retrieved if the identifier is not declared in this entry as a minor structure or as an element of a structure.

If an entry of the identifier is found and the identifier is declared as a minor structure or as an element of a structure, but not as an array, scanning of the block is also continued. If no other entry is found in the same block, the error routine is initialized.

If the identifier is not declared in SYMTAB, exit NOT of WSIO occurs. If it is declared and its internal representation is not zero, exit DECL occurs. If its internal representation is zero, the error routine is initialized.

Output parameters:
PIN points to the identifier in the text string.
PSY points to the entry of the identifier in SYMTAB if exit DECL occurs.

<u>Read and Scan Block X -- OM-ON</u>

Entry : WRBX1

Input parameters:

R1	contains the address A1 of the area in which block X can be stored.
R3	contains the number M1 of records which can be stored in A1.
R7	points to the entry of block X in the scope table.
LVLPT	points to the entry of block X in WSLIST.
WMX	contains the number MX of records of block X.

If MX is not greater than M1, the entire block X is stored and scanned in the table space. Otherwise, the first M1 records of block X are stored and scanned in the table space, and the remaining records of block X are read and scanned in BS.

If an entry of the identifier is found and the identifier is not declared as a minor structure or as an element of a structure, the internal representation is tested for zero.

If an entry of the identifier is found and the identifier is declared as an array in a structure, the internal representation of the array is stored. Scanning is continued.

If no entry or only entries for identifiers declared as a minor structure or as an element of a structure are found, this subroutine is left via its normal exit.

Output parameter:
PSY points to the entry of the identifier in SYMTAB if the identifier is declared but not as a minor structure or as an element of a structure.

<u>Search for Identifier in BS -- OO</u>

Entry: WREAD1

Input parameter:
R3 contains the number of records of block X to be read and scanned in BS.

Records are read and scanned in overlapped mode, i.e., while a new record is read into one buffer of BS, the record in the other buffer is scanned for an entry of the searched identifier. PSY points to the entry of SYMTAB which is scanned.

If no entry of the searched identifier is found that is not declared as a minor structure or as an element of a structure, the routine is left via its normal exit.

If an entry is found and the identifier is declared as a minor structure or as an element of a structure, switch MS is set. If the minor structure or element of a structure is an array, the internal representation of the array is stored. Scanning is continued.

Output parameter:
PSY points to the entry of the searched identifier unless this routine is left via its normal exit.

Clear Addresses in Scope Table -- OP

Entry: WCSCO1

If switch WCSCO2 is off, bytes 4 and 5 of all entries in the scope table are cleared. If switch WCSCO2 is on, bytes 4 and 5 of all entries of the scope table are tested for zero. If a nonzero entry is found, it is tested whether the end address of this block is equal to or higher than AN1. If it is higher or equal, the begin address of this block in the scope table is cleared. If it is lower and the end address of this block is higher than the highest end address of previously found blocks in storage, the end address of this block is stored.

PHASE PL/IB75 (EXTERNAL ENTRY NAMES FOR IMPROPERLY GENERATED BUILT-IN FUNCTIONS) -- OR

If a subroutine reference in the text string is identical to the name of a built-in function, the name of the identical function must be declared as an external entry name, provided the function has not been declared explicitly as a built-in function. This phase replaces such a subroutine reference (which has been declared as a built-in function in phase B70) by its correct representation as an external entry name. If there is no such subroutine reference, phase B75 is skipped.

Phase Input

1. The text string from TXTIN which contains function references that have been incorrectly declared as built-in functions.

2. Block n+1 of SYMTAB which contains all contextually declarations.

Phase Output

In the text string, all subroutine references that were improperly declared are now replaced by their proper internal representation and are characterized by an EE-key.

Communication with Other Phases

Scope table
IJKMIP
WSLIST
Table Space as classified in phase B70
WBSEN
Bit 20 of IJKMJT
WCTAB

The above areas, tables, and switches, as well as I/O handling are the same as described in phase B70.

Internal Pointers, Switches, and Tables

PIN	points to the element being scanned in the input buffer.
POUT	indicates the output buffer address to which the next output is to be moved.
PSY	points to the entry of block n+1 in SYMTAB, which is to be scanned.
EPSY	points to the end of the area that contains the entries of block n+1 which are to be scanned.
Switch 1	is on if records of block n+1 are stored in BS, but not in the first M0 buffers of table space.
Switch 2	is on if records of block n+1 are stored in the first M0 buffers of table space and in BS.
WECLIST	contains entries which consist of the compressed user name and the WCTAB entry for the built-in function. These entries are in ascending order by the internal representations of the built-in functions.

DESCRIPTION OF ROUTINES

Note: The subroutines listed below are used by phase B75, but described elsewhere. For a description of these subroutines refer to the sections indicated.

ITRNA1	phase A50
WBSOC	phase B80
WCLEAR	phase B80
WCSC01	phase B80
WSETSP	phase B80

Initialization -- OS

The pointer POUT is set to point to the beginning of the output buffer. The pointer PIN is set to point to the beginning of the first input buffer. The first two records of text string are read into the input buffers.

The bit used to indicate the presence of built-in functions in the current compilation is reset.

If the table space contains only a portion of that part of block n+1 which was built up during phase B70, the head of block n+1 is retrieved from SYS001. If the number (MX) of records of block n+1 on SYS001 is not greater than M0+2, MX records of block n+1 are read into the table space; otherwise, the first M0 records of block n+1 are stored in the first M0 buffers of the table space and the begin of the remaining records on SYS001 is noted.

Scan Source Text -- OT

The text string is scanned for an End-of-Statement indication and for built-in functions not explicitly declared.

If a built-in function is found which is not explicitly declared, the WCTAB area is tested to determine whether the function has been declared by the user. If not, a bit is set to indicate that the current compilation includes built-in functions.

If the function has been declared by the user, the compressed name of this built-in function is obtained and the internal representation of the user's function is picked up in block n+1. The built-in function in the text string is replaced by the internal representation of the user's function and the key 'EE'.

Pick up Internal Representation of User Function -- OU

Block n+1 is scanned for the entry of the user's function. Scanning starts with those entries of block n+1 which have been in storage at the beginning of this phase. Scanning continues with those entries of block n+1 which have been stored in the table space during initialization of this phase. If not all records of block n+1 are in storage, the head of the remaining records of block n+1 is picked up and the entries of these records are read and scanned in BS.

End of Statement or Phase -- OV

Pointer PIN points to key 'EA' of the end of statement. WR4 contains the address of the area whose contents are moved into the output buffer. The processed text string, including end-of-statement and error messages (if any) from previous phases are written on TXTOUT by the output routine JTRNA1.

If PIN points to the end of the text string, that part of the text string which is not yet on TXTOUT is written, TXTOUT and TXTIN are rewound and exchanged. The phase is terminated by calling phase B80.

Search for Identifier Subroutine -- OW

Entry point: WSEAR

Input parameters:

PSY	points to the area that contains the entries of block n+1 that are to be scanned.
EPSY	points to the end of the same area.
R11	contains the compressed user's name of the function, the entry of which is to be searched for.

PSY points to the entry of block n+1 that is to be searched for the identifier. If the desired entry is found, the routine is left to replace the built-in function by the internal representation of the user's function in the text string.

If PSY points to an address that is equal to or greater than the value of EPSY, this subroutine is left via the NO exit.

Output parameter:

PSY	points to the desired entry of block n+1 if not left via the NO exit.

PHASE PL/IB80 (IMPLICIT DECLARATIONS) -- PA

This phase performs the following functions:

1. All implicitly declared identifiers are added to the symbol table SYMTAB.

2. Identifiers in the text string that have an E1-key are replaced by their internal representations.

3. Identifiers in the text string, which are built-in functions without arguments, are replaced by the internal representation of the built-in functions.

Phase Input and Output

The input consists of the following:

1. The text string from TXTIN containing all identifiers which are not replaced by the appropriate internal representation with an E1-key.

2. Symbol table SYMTAB on SYS001 or in storage which contains an entry for each explicitly or contextually declared identifier.

As output, the phase produces:

1. The text string on TXTIN which contains
   a. the appropriate internal representation with an EE-key for all identifiers and
   b. the appropriate internal representation with an EC-key for all built-in functions.

2. Block n+1 of SYMTAB on SYS001. This block contains one entry for each contextually or implicitly declared identifier.

Communication with Other Phases

Scope Table
IJKMIP
WSLIST
Table Space as classified in phase B70
WBSEN
Bit 20 of IJKMJT
Error code X'45'
Error code X'44'

Areas, tables, and switches under 1 through 7, above are as described in phase B70.

Error code X'44'. If an identifier of a qualified name is not declared in SYMTAB, the error code X'44' is inserted into the text string after the statement in which the not declared qualified name was found.

Internal Pointers, Switches, and Tables

PIN	points to the element in the input buffer which is being scanned.
POUT	indicates the output buffer address to which the next output is to be moved.
PSY	points to the SYMTAB entry to be scanned.
LVLPT	points to the entry in WSLIST for the required SYMTAB block.
Scope pointer	points to the entry in the scope table for the required SYMTAB block.
WSWIMS	switch MS is on if bit 0 of WSWIMS is 1. It is set whenever an entry of an identifier is found in SYMTAB and this identifier 1. does not occur in a qualified name and 2. is declared as a minor structure or an element of a structure in this entry. The internal representation contained in this entry is stored, and scanning of the block is continued; however, embracing blocks are not scanned.
WQUALS	if WQUALS is X'01', switch QUAL is on; this indicates that a qualified name is to be tested. If WQUALS is X'81', switches QUAL and MINOR STRUCT are on; this indicates that scanning for entries for the identifiers of a qualified name was started.
WNRNR	contains 1. the number of the block containing the statement being tested and 2. the number of the embracing block.

The level lists WLEVL, WQUANT, and WNTLL are used if scanning for a qualified name is performed. The lists WQUANT and WNTLL are only used when the records of SYMTAB, which are being scanned for the qualified name, are in BS. The contents of these tables follow:

WLEVL

Bytes 0-3:	Pointer PSY of the identifier entry with smallest level number found while scanning for an identifier of a qualified name was performed.
Byte 4:	LNR1 = level number + 1 of the identifier of a qualified name which precedes the identifier being scanned for. (= 1 if the first identifier is scanned.)
Byte 5:	LNR2 = level number of the entry which is referred to in bytes 0 through 3.

WOUANT

Bytes 0- 7:	NOTE information for the record currently in BS1.
Bytes 8-11:	Not used.
Bytes 12-15:	Number of records + 1 of block X that follow the record currently in BS1.
WNTLL	Contents of WQUANT for the identifier referred to in WLEVL (bytes 0 through 3).

Functional Description

If an identifier with an E1-key is found in the text string, subroutine WSIO is called to search SYMTAB for an entry for this identifier. The identifier may or may not occur in a qualified name.

1. The identifier does not occur in a qualified name.

   If the identifier is declared, its internal representation including the key 'EE' is inserted into the text string. If the identifier is not declared, it is determined whether or not it is a built-in function without arguments.

   In case of a built-in function without arguments, the internal representation of the built-in function including the key 'EC' is inserted into the text string.

   If it is not a built-in function, the identifier is declared as arithmetic in block n+1 of SYMTAB, and its internal representation including the key 'EE' is inserted into the text string.

2. The identifier occurs in a qualified name.

   The proper entry for each identifier contained in the qualified name is looked up in the block in which the qualified name is declared. All but the last identifier of the qualified name are deleted in the text string. When the correct entry of the last identifier is found in SYMTAB, the internal representation of this identifier including the key 'EE' is inserted into the text string. Otherwise, 'EE0000' and the error code X'44' (= qualified name not declared) are inserted in the text string.

DESCRIPTION OF ROUTINES

Note: Subroutines JERRA1 (error) and JTRNA1 (output) are used by phase B80, but described in phase A50.

Initialization -- PB

Output pointer POUT is set to the beginning of the output buffer. Input pointer PIN is set to the beginning of the first input buffer. The first two records of the text string are read into the input buffers.

Pointer PSE is set to the first available byte position in block n+1 following the last entry made in phase B70.

Search for Identifier in Source Text -- PC

The text string is scanned for

1. the beginning of statements (statement identifiers),

2. identifiers with an E1-key,

3. end of statements and

4. explicitly declared built-in functions.

Input and output of the text string is as described in phase B70.

If the beginning of a statement is found, the numbers of the block containing the statement and of the embracing block are stored in WNRNR.

If an identifier with an E1-key is found, WSIO is called to search SYMTAB for an entry for this identifier. If the identifier is declared, its internal representation with the key 'EE' is inserted into the text string.

If the identifier is the first one of a qualified name, the text string is written out up to the beginning of the qualified name and switch QUAL is set before WSIO is called.

If an explicitly declared built-in function is found, the appropriate representation of the function (first byte of internal representation = 0) is inserted into the text string. The bit that indicates built-in functions in the current compilation is set.

Identifier not Declared -- PD

If an undeclared identifier is a built-in function without arguments, the internal representation of this function, including key 'EC', is inserted into the text string. The bit that indicates built-in functions in the current compilation is set.

If the internal representation of the identifier is 0, X'EE0000' is inserted into the text string. In addition, the error code X'45' is inserted by calling JERRA1.

Entry in SYMTAB -- PE

PIN points to the identifier for which an entry with arithmetic attributes is built. This entry is then moved into block n+1 of SYMTAB. PSE points to the address of the entry.

The internal representation of the identifier equals the present value of the variable counter.

If one of the characters I through N is used as the first letter of the user-defined name of the identifier, attribute FIXED BINARY with a length of 15 is set into the entry for the identifier. If the first letter is a character other than I through N, FLOAT DECIMAL with a length of 6 is set into the identifier entry.

If this entry is the last possible entry in the buffer(s) used to build up block n+1, all entries of block n+1 that are in storage are written on SYS001. PSE is reset. If the first bit of IJKMIP is 1, this bit is reset, the second bit of IJKMIP is set, and K is decreased by 2, i.e., BS is used to retrieve that part of block n+1 which is not yet in storage.

End of Statement -- PF

PIN points to key 'EA' of End of Statement. WR4 contains the address of the area whose contents are to be moved into the output buffer by calling JTRNA1.

If an improperly declared identifier or qualified name is found in this statement, the error bit is set and the new error message(s) are added to the old one. The number of error messages following a statement is limited to eight.

End of Phase -- PG

That part of the text string which has not yet been written onto TXTOUT is written out. End of block n+1 is set and that part of block n+1 which is in storage is now written on SYS001.

The number of records of, and the note information for, block n+1 are inserted into scope table entry 0. If all of block n+1 is in storage, its address in relation to the table space is also set into the scope table.

TXTIN and TXTOUT are rewound and exchanged. The phase is terminated by calling phase B90.

Scanning of Qualified Name -- PH

This routine retrieves the proper entry in SYMTAB for the last identifier of a qualified name. This is done under control of switch WQUALS and level lists WLEVL, WNTLL, and WQUANT. (WNTLL and WOUANT are used only when scanning of SYMTAB is done in BS.)

In the text string, all but the last identifier of a qualified name are deleted. When the proper entry for the last identifier is found, the internal representation of this identifier, including key 'EE', is inserted into the text string; otherwise 'EE0000' and the error code '44' (= qualified name not declared) are inserted into the text string.

All identifiers of a qualified name are declared in the same block. Each identifier of a qualified name has a level number which is greater than the level number of the preceding identifier.

PIN points to the first identifier of the qualified name in the text string and PSY points to an entry of this identifier in block X when this routine is entered at WOUAL8. The entry that contains the lowest level number is searched for the identifier pointed to by PIN. Information about the entry which presently contains the lowest level number is stored in the level lists.

Scanning for entries for the first identifier of the qualified name starts at the beginning of block X and stops when either an entry for this identifier with level number 1 or the end of the block is found. Scanning for the other identifiers starts at the entry that follows the entry with the lowest level number of the preceding identifier and stops when an entry with a level number is found that is either

1. equal to the level number plus 1 or

2. not higher than the level number of the preceding identifier.

When the scanning is stopped, WLEVL contains PSY of the entry with the lowest level number of the identifier pointed to by PIN.

If WLEVL is blank, i.e., an entry for the identifier was not found, the embracing block is scanned for the entry with the lowest level number of the first identifier of the qualified name, and so on. If the embracing block is block n+1, scanning stops. In this case, the qualified name has not been declared.

SUBROUTINES

Search for Identifier in SYMTAB -- PI - PM

Entry point: WSI0

Input Parameters:

1. No qualified name: PIN points to the identifier for which an entry in SYMTAB is searched.

2. Qualified name: PIN points to the beginning of the qualified name. Bit 7 of switch WQUALS is on.

If an identifier is declared in SYMTAB, WSI0 retrieves the internal representation of the identifier from SYMTAB. In addition, this subroutine attempts to keep blocks in storage as long as possible and reads into storage only those blocks that are required to scan for an entry for the identifier.

If block X (the block that contains the statement which, in turn, contains the identifier searched for) is not in storage, all blocks in storage that are not embracing blocks of block X are cleared. If there are not enough contiguous free buffers in the table space to accommodate block X, embracing blocks of block X are cleared starting with block level 1. If the number of records of block X is greater than M0, all blocks are cleared in storage, the first M0 records of block X are stored in the M0 buffers of the table space, and these records are scanned. The remaining part of block X is then read into, and scanned in, BS.

If scanning is continued in an embracing block of block X, block X remains in storage. If the embracing block is not in storage and the maximum number of connected buffers in the table space is equal to M1, the embracing block is stored in the M1 buffers, provided the number of records of this block is not greater than M1; otherwise, the first M1 records of the embracing block are stored and scanned in the M1 buffers of the table space and the remaining records are read into, and scanned in, BS.

The remaining functions of the subroutine vary according to the type of identifier (qualified name or no qualified name).

<u>No Qualified Name.</u> Scanning is started in block X of the identifier. If the searched entry is not found in block X, scanning is continued in the embracing block of block X, etc. Block n+1 is the outermost embracing block of all blocks.

If an entry for the identifier is found and a minor structure or an element of a structure is declared in this entry, the internal representation in this entry is stored and scanning of the block is continued. If no other entry for the identifier is found in the same block, the stored internal representation is used for the identifier; otherwise, the internal representation of the new entry is fetched, provided no minor structure or element of a structure has been declared in this entry.

If the identifier is declared and its internal representation is not 0, the routine is left via the exit DECL. If the internal representation is 0, the routine is left via the error exit (to initialize the error routine). If the identifier is not declared, the routine is left via exit NOT.

<u>Qualified Name.</u> Scanning is done as described under <u>Scanning of Qualified Name -- PH</u>. If the qualified name is not declared, the subroutine is left to initialize the error routine.

The subroutine is left via the exit DECL if the internal representation in the entry with the smallest level number of the last identifier of the qualified name is not 0. If the internal representation is 0, the routine is also left to initialize the error routine.

Output Parameters:

PIN points to the identifier (last identifier of qualified name) in the text string.

PSY points to the entry of the identifier in SYMTAB if the routine is left via the exit DECL.

Read and Scan Block X -- PN, PO

Entry points: WRBX1, WSCX1, WSCX6

Input Parameters:

R1	contains the address A1 of the area into which block X (block to be scanned) can be stored.
R3	contains the number M1 of records which can be stored in A1.
R7	points to the entry for block X in the scope table.
LVLPT	points to the entry for block X in WSLIST.
WMX	contains the number MX of records of block X.
WQUALS	indicates that scanning of a qualified name was started if bit 0 is 1.

If MX is not greater than M1, all of block X is stored in the table space and scanned; otherwise, the first M1 records of block X are stored and scanned in the table space and the remaining records of block X are read into, and scanned in, BS.

The remaining functions of the subroutine vary according to the type of identifier (qualified name or no qualified name).

No Qualified Name. If an entry for the identifier is found and the identifier is declared in this entry as a minor structure or an element of a structure, the subroutine

1. stores the internal representation in the entry,
2. sets switch MS, and
3. continues the scanning of the block.

If no entry for the identifier is found, the subroutine is left via its normal exit.

If an entry is found and the identifier in this entry is not declared as a minor structure or as an element of a structure, the subroutine is left to determine whether the internal representation is 0.

Qualified Name. If the scanned entry is not an entry for the searched identifier, the routine determines whether or not searching for entries for the identifier of the qualified name has to be continued. If not, control is transferred to continue scanning qualified names. If no entry for the identifier is found, this subroutine is left via its normal exit.

Output Parameters:

1. No qualified name:
   PIN points to the entry for the identifier in SYMTAB if the identifier is declared as other than a minor structure or as an element of a structure.
2. Qualified name:
   PIN points to an entry for the identifier if an entry was found.

Search for Identifier in BS -- PP

Entry points: WREAD1, WREAD5

Input Parameters:
R3 contains the number of records of block X to be read into BS and scanned.
LVLPT points to the entry of block X in WSLIST.

The functions of this routine vary according to the type of identifier being searched for (part or not part of a qualified name).

1. The identifier is not part of a qualified name:

   Reading records into BS and scanning is done in overlapped mode, i.e., while a new record is read into one buffer of BS, the record in the other buffer is scanned for an entry of the identifier. PSY points to the SYMTAB entry being scanned. If no entry of the identifier is found, the routine is left via its normal exit. If an entry is found and it has been declared as a minor structure or as an element of a structure, switch MS is set and the internal representation in this entry is stored. Scanning of the block is continued.

2. The identifier is part of a qualified name:

   Two records are read into BS and the beginning of the first record is noted in level list WQUANT. Then, the two records are scanned. If the scanned entry is not an entry for the identifier, a check is performed to determine if searching for entries for the identifier must be terminated. If no entry for the identifier is found, the routine gets the next two records of block X and starts scanning these records for entries for the identifier. If PSY points to the end of block, the routine is left via its normal exit.

Output Parameter:
PSY points to an entry for the identifier if this routine is not left via its normal exit.

Get Records to BS -- PQ

Entry points: WGT21, WGT22

Input Parameters:

WMX	contains the number of records of block X not yet in storage;
R7	points to the entry for block X in the scope table;
LVLPT	(only used if the routine is entered via WGT21) points to the entry for block X in WSLIST.

If entry WGT21 is used, this routine tests whether all embracing blocks of block X are in storage. If they are not, the routine is left via its NO exit.

Otherwise, and if entry WGT22 is used, the routine determines whether the number of records of block X not yet in storage is less than or equal to two. If there are more than two records, the routine is left via its NO exit. If the number of records of block X not yet in storage is two or less than two, these records are read into BS and the presence of block X in BS is noted in WBSEN and, if the entire block X is in BS, in the scope table. The routine is left via its YES-exit.

Calculate M1 -- PR

Entry point: WCAM1

M1 is the maximum number of records of SYMTAB that can be stored contiguously in the table space without destroying other records of SYMTAB that are already in storage and noted in WSLIST. This subroutine calculates M1 using the addresses of the blocks in storage as contained in WSLIST.

The list below shows the meaning of the names used in this subroutine.

ML = length of longest contiguous area not used by other blocks.

If only one block is stored:

AY = Address of the block.
AZ = Address of end of the block.
ML = Max(AY-A0,AN1-AZ)

If two blocks are in storage:

AU = Address of 1st block.
AV = Address of end of 1st block.
AX = Address of 2nd block.
AY = Address of end of 2nd block.
ML = Max(A0-AU,A-AV,AN1-AY)

If no block is in storage, M1 = M0; otherwise: M1 = ML divided by record length of SYMTAB.

Output Parameters:
R1 contains the address A1 to which a block can be read.
R3 contains the number M1 of records that can be read to A1.

Clear WSLIST Entry Y -- PS

Entry point: WCLEAR

Input Parameter:
R2 points to the WSLIST entry Y.

WSLIST entry Y contains information about the level-Y block, which is in storage. If this block is no longer needed for the searching of identifiers, the indication of the block (for being in storage) is cleared in both the scope table (bytes 4 and 5) and in WSLIST (byte 0). If the block or its end is stored in BS, the indication of the block is also cleared in WBSEN.

Set Scope Pointer -- PT

Entry point: WSETSP

Input Parameter:
R2 contains the number of the block.

The scope pointer R4 is set to the scope table entry for the block indicated by R2.

Output Parameter:
R4 points to the entry for the block in the scope table.

Clear BS -- PU

Entry point: WBSQC1

If (1) a block or the remainder of a block not yet in storage consists of no more than two records and (2) the first M0 buffers of the table area are filled with other blocks which are still needed for scanning, the block or its remainder is stored in BS and WBSEN contains the number of the block.

This subroutine clears WBSEN if a complete block or the end of a block is stored in BS. If a complete block is stored in BS, the subroutine also clears the address of this block in the scope table.

Entries in WSLIST and Scope Table -- PV

Entry points:	WELST1	used if the complete block is or will be in storage.
	WELST3	used if the end of the block is not in storage.

Input parameters:

WELST1:	R1	points to the address of the block in storage.
	R4	points to the entry for the block in the scope table.
	R5	points to the entry for the block in WSLIST.
	PSY	contains the number of records of the block.
WELST3:	R0	contains the number of records of the block in storage.
	R1	contains the address of the block in storage.
	R4	points to the entry for the block in the scope table.
	R5	points to the entry for the block in WSLIST.
	PSY	contains the address of the end of the block in storage.

When a new block is read into storage, this subroutine performs the necessary housekeeping functions in WSLIST and bytes 4 and 5 of the scope table. If entry WELST3 is used, housekeeping in byte 1 and bytes 13 through 16 of WSLIST must be done before this subroutine is called.

Clear Addresses in the Scope Table -- PW

Entry point: WCSC01

Bytes 4 and 5 of all entries in the scope table are cleared.

In this and the next phase, a statement attribute table is generated in front of each statement. This table contains all attributes for each variable that occurs in the actual statement. If the statement is a PROCEDURE statement, the first entry of the attribute table contains the attribute belonging to the entry name of the procedure.

For the construction of the attribute table see phase B20.

DESCRIPTION OF ROUTINES

Note: The following routines are described elsewhere as follows:

JSLCA1	B10	HX
JEOSA1	B10	HY
JERRA1	A50	FZ
MOVEA1	A50	F0

JMACA1 -- QB

If a statement is preceded by a label, this routine generates a label macro. The generated macro has the following format:

byte 0	X'F2'
bytes 1-2	X'0007'
byte 3	X'72'
bytes 4-6	internal representation of the label identifier

The label identifier in the source text is replaced by the label macro. The colon after the label identifier is deleted.

If the statement is a PROCEDURE statement, the symbol table entry for the entry name is set into the source text.

Entry parameter:
PIN = start address of the statement to be processed

Return parameters:
ACBLO = current block number
EMBLO = embracing block number

(For BLOT1 refer to routine JBLT).

JSID -- QC

This routine scans the source text and searches for identifiers.

Entry parameter:
PIN = start address of statement body

JSAR Routines -- QD - QG

Main entry point: JSAR

Entry parameter:
PIN = address of the name for which the symbol table entry is searched.

The routine searches for an entry in the symbol table and sets it into the output area. The entries are ordered by their structure levels, i.e., first all entries with structure level 0 (no structures or elements of structures) are written out, followed by all entries with structure level 1 (major structures), level 2, level 3, and so on.

If the current entry is a major or minor structure, all entries belonging to the items of a given structure are inserted in the output area.

If the actual entry is a minor structure or an element of a structure, the entry of the major structure is inserted in the output area immediately before the current entry.

If the actual entry has the attribute CONTROLLED or DEFINED, the entry for the pointer or base identifier is set into the output area immediately before the current entry. For this reason, the routine JSAR may be called recursively.

JRPS Routines -- QH, QI

Main entry point: JRPS
Secondary entry point: JCET

Entry parameters:

HR3	=	entry in scope table for block to be read
TABEND	=	end address of used part of table area
TBREC1	=	begin address of buffer area 1 for reading the symbol table in overlapped mode.
TBREC2	=	begin address of buffer area 2

The program reads a part of the symbol table.

JBLT -- QJ

This routine builds up the block table BLOT, which consists of 4-byte entries referred to as BLOT0 - BLOT3.

BLOT0 is associated with the part of the symbol table that contains all declarations given either implicitly or contextually. (For a description of the symbol table refer to phase B20.)

BLOT1 is associated with the part that contains all declarations given explicitly at block level 1.

BLOT2 and BLOT3 are associated with all parts that contain declarations at block levels 2 and 3.

The number of the current block is entered corresponding to the level indicated in the first byte of each entry in BLOT. The next three bytes contain the end address of the corresponding part of the symbol table. (The start address is contained in SCOTAB; see phase B20.)

Entry parameters:

EMBLO	=	number of embracing block.
REBLO	=	number of block to be processed.
TABEND	=	end address of entire symbol table in storage.

JSCC Routines -- QK, QL

Entry points: JSCC and JSC1

Entry parameters:

TABEND	=	address of last valid entry in the symbol table contained in the table storage
SAVES1	=	length of part of symbol table to be read
IJKMBC	=	number of blocks, i.e., number of entries

The routine controls the scope table and erases all invalid entries.

JTRA -- QM

Entry parameters:

HR1	=	start address of information to be written
BYZ	=	length of the information
POUT	=	next free address in the output buffer

Return parameter:

POUT	=	next free address in the output buffer.

In this routine, information is moved into the output buffer and the pointer for this buffer is controlled. If a buffer overflow occurs, the contents of the buffer are written out.

This phase compresses the attribute table constructed in the previous phase.

In phase B90, full-length 20-byte entries were made into the attribute table. If an identifier occurs more than once in one statement, more than one identical entry has been generated for this identifier in phase B90. Phase B92 deletes all identical entries except the first one and eliminates the redundant bytes of each entry.

The internal representation of the variable in the statement body is changed into a table lookup for the attribute table.

Statement Attribute Table

An attribute table is assigned to each statement. It contains the attributes for all variables. This table is located in front of the statement in the source text. The internal representation of the variable is changed into an offset.

The entries of the attribute table are of variable length depending on the attributes contained in these entries. If the variable has the attribute PICTURE, the entry is 18 bytes long. If the variable has one of the attributes DEFINED or CONTROLLED, the entry is 14 bytes long. If the variable is a minor structure or an element of a structure, the entry is 14 bytes long. If the variable has the attribute ARRAY or STRUCTURE, the entry is 12 bytes long. If the variable has the attribute FILE, the entry is 10 bytes long. For all other variables, the entry is 8 bytes long.

The construction of the attribute table is the same as that of the symbol table, except that the first two bytes of the symbol table are not entered in the attribute table.

A statement has the following format after it has been processed by all syntax phases:

A	= statement identifier key	(1 byte)
a	= specification of statement identifier	(2 bytes)
B	= prefixes	(1 byte)
b	= statement flag bits	(2 bytes)
C	= key for attribute table	(1 byte)
1c	= length of attribute table	(2 bytes)
D	= attribute table of declared variables	(1c bytes)
E	= key for constant table	(1 byte)
1e	= length of the constant table	(2 bytes)
F	= constant table of declared constants	(1e bytes)
G	= statement body	
H	= endkey of statement	(1 byte)
I	= byte for error flags	(1 byte)
K	= level number	(1 byte)
L	= block number	(1 byte)
M	= statement number	(2 bytes)
N	= error key, if error	(1 byte)
O	= error number if any error	(1 byte)

DESCRIPTION OF ROUTINES

Note: The following routines are described elsewhere as follows:

JTRAA1	B90	QM
MOVEA1	A50	FO
JEOSA1	B10	HY
JERRA1	A50	FZ

JCATA1 -- RB, RC

This subroutine generates the attribute table.

JCIR -- RD

This routine changes the internal representation of the variables into a table lookup for the attribute table. Note that the identifier key X'E1' is not changed.

Entry parameter:
PIN = start address of the statement body

JCES -- RE

Entry parameters:
PIN = begin address of the entry to be compressed
PST = address of table area into which the entry is moved

Return parameters:
HR2 = PST old. If a table overflow occurs, HR2 = 0. If the current entry is ignored, HR2 contains the address of the previous one.
PST = next free address of table area
HR1 = 0 if the actual entry is ignored; otherwise HR1 = 0.

The entry of the symbol table is compressed and set into the table area. If the same entry was made previously, the current one is ignored.

JGOF -- RF

Entry parameters:
PIN = begin address of symbol table entry of the identifier
HR2 = begin address of attribute table entry of the identifier
PAT = next free entry in the offset table
HR1 = 0 if the current entry of the attribute table is ignored

Return parameters:
PIN = unchanged
HR2 = unchanged
PAT new = next free entry in the offset table.

An offset table (OFFTAB) is generated. An entry of this table has a length of 4 bytes and contains the following information:

Bytes 0-1: internal representation of the identifier
Bytes 2-3: begin address (relative to IJKMTS) of the entry in which the attributes given to the identifier are stored.

If an offset table overflow occurs, this table is written onto SYS001.

JLEN -- RG

Entry parameter:
HR2 = begin address of attribute table entry

Return parameters:
HR2 = unchanged
HR1 = length of the entry

The length of an entry contained in the attribute table is calculated.

JCPI -- RH

Entry parameter:
PIN = input pointer

Return parameter:
PIN new = PIN old + 20.

The input pointer PIN is controlled. If, after an increase, PIN is outside the first buffer, the remainder is moved to the left and a new record is read into the last buffer.

JBIPA1 - RI

This routine changes the pointer or base identifier.

This phase constructs the array table ARYTAB. (The phase is skipped if the source program contains no arrays.) For each programmer-defined array, a 12-byte entry is incorporated in the table. ARYTAB is written on SYS001 at the end of the phase.

Up to three dimensions may be specified for an array. The format of the 12-byte ARYTAB entry for 1-, 2-, and 3-dimensional arrays is shown in Figure 1 together with the corresponding declarations. The information required for these entries is retrieved from the array statements built up in phase B15.

The array statement consists of two parts that have the following format:

Part 1 (variable length, depending on attributes)

Byte(s)	Contents
0- 5	Array statement key (X'E00044...')
6	X'F4'
7- 8	Length of attributes
9-10	Internal name
11	Rightmost four bits contain the length of one element unless it is a character string
12-14	Not used
15	Length of one element if it is a character string
16-17	Not used
18	Current array number
19-20	Number of elements
21- n	Other attributes

Part 2 (21 bytes)

Byte(s)	Contents
0	X'E1'
1- 2	Offset to attribute table
3	X'E9'
4- 5	Current array number
6	X'E9'
7- 8	Bound 1
9	X'E9'
10-11	Bound 2
12	X'E9'
13-14	Bound 3
15-20	EOS (X'EA...')

When an entry is made in the array table, the required information is retrieved from the array statement and the latter is deleted in the source text. Some bounds may be missing if the array statement was detected to be erroneous in phase B15. In this case, the entry for the array is set to zero.

Phase Input and Output

The source text is read from TXTIN. The text output is written on TXTOUT. It consists of the source text without the array statements. ARYTAB is written on SYS001, at the end of the phase, the functions of TXTIN and TXTOUT are exchanged.

DESCRIPTION OF ROUTINES

Symbols Used in Flow Charts:

C(CP)	:	contents of location pointed to by CP
C(CP+6), LENGTH 2	:	contents (length 2) of location pointed to by CP+6.

Initialization -- SB

The array table is cleared and the entry for the array table is made in TABTAB, i.e., the buffer length is set to 384 and the transfer bit is set to zero. BUFL is set to 3*IJKMBL and ENDTS is set to the end address of the array table area. The address of the output buffer is loaded into B0. The addresses of work buffer 1, 2, and 3 are loaded into B1, B2, and B3. The begin and end address of the input buffer is loaded into B4 and B5, respectively. The output pointer OP1 is set to the begin address of the output buffer and input pointer CP is set to the beginning of the input buffer.

Main Routine -- SC

The input is scanned. If a normal F key is found, i.e., no end-of-program key, LENGTH is set to the value contained in the two bytes following the F key. If an EA key is found, LENGTH is set to 6. If an EB key is found, LENGTH is set to two. UPRO is called after LENGTH has been set.

If an E0 key for an array statement is found, the array handling routine is called; otherwise, LENGTH is set to 6 and UPRO is called. If the EOP key is detected, this key is written out. The last, not yet filled-up record is also written out, if required, and the array table that was built in the table area is written on SYS001.

Byte(s)	One Dimension DECLARE A(i)	Two Dimensions DECLARE A(i,j)	Three Dimensions DECLARE A(i,j,k)
0-1	Internal name	Internal name	Internal name
2-3	Number of elements	Number of elements	Number of elements
4-5	Length of one element	Length of one element	Length of one element
6-7	X'0000'	j	k
8-9	X'0000'	X'0000'	j
10-11	Negative value of length of one element	Negative value of (length of one element + length of one element*j)	Negative value of (length of one element + length of one element*k + length of one element*k*j)

Figure 1. Format of 12-Byte Entries in ARYTAB

Array Handling -- SD - SF

Entry point: C2B2

This routine is called if an array statement is detected in the main routine, and the corresponding entry in the array table is built. The array statement is deleted in the source text and, if the first bound is zero, it is replaced by an error message. The entries in the array table are generated as described in Figure 1.

UPRO Input/Output Handling -- SG

Entry point: C6B2

At the beginning of this routine, a test is performed to determine whether the string to be written is contained in its full length in the work buffers. If it is not, LENGTH1 is set to the number of bytes not yet contained and LENGTH is set to the number of bytes that is contained in the work buffers. The move and the read routines are called, and LENGTH is set to LENGTH1. If the full string is contained in the work buffers, the move and read routines are called immediately.

Move Routine -- SH

Entry point: C7B2

This routine moves the number of bytes specified in LENGTH from the buffer address pointed to by CP to the output buffer address pointed to by OP1. If the output buffer is full, the write routine is called.

Write Routine -- SI

Entry point: C8B2

This routine checks whether the output buffer is full. If it is, the information is written on TXTOUT. The output pointer OP1 is reset to the beginning of the output area.

Read Routine -- SJ

Entry point: C9B2

If the input pointer CP is greater than the contents of B2, the contents of the last two work buffers and the input buffer are moved to the beginning of the work buffers. The input buffer is filled with the next record from TXTIN. CP is decreased by the buffer length and tested again. Processing of this routine is repeated until CP is lower than or equal to B2.

This phase constructs the external name table EXTTAB. This is done in two passes:

1. A pretable PRETAB of the external name table is built up. All information to construct PRETAB is retrieved from entries in the symbol table SYMTAB that contain the attribute EXTERNAL. Each 20-byte entry of the pretable contains the following:

Byte 0	: number of record in the name table that contains the user-defined name of this identifier
Byte 1	: entry number in this record (the first bit must be ignored)
Byte 2	: FF
Byte 3- 7	: blanks
Byte 8-19	: see external name table

2. EXTTAB is constructed by replacing the first eight bytes of PRETAB by the user-defined name retrieved from the name table NAMTAB.

The entry of NAMTAB pointed to by bytes 0 and 1 of PRETAB is searched, and the user-defined name is translated into the external code and inserted into bytes 0-7. If the name is shorter than 8 bytes, the remainder is filled with blanks. If the name is longer than 6 bytes, a warning message is generated. If the name is longer than 8 bytes, the rest is ignored and an error message is generated. These messages are inserted behind the first PROCEDURE statement of the source text.

For a description of the information contained in the external name table, refer to phase G55.

Input of the Phase: Symbol table SYMTAB on SYS001, name table NAMTAB on SYS001, and source text on TXTIN.

Output of the Phase: External name table EXTTAB on SYS001 and source text on TXTIN.

I/O Handling of the Phase

The symbol table is read from SYS001 into buffers B3 and B4. The table space and the buffers B0 to B2 are divided into sections that have the length of a buffer. The entries built up for the pretable are moved into these sections. If an overflow occurs, the pretable is written onto TXTOUT. When the last symbol-table record is processed, two cases are to be distinguished:

1. The pretable is still in storage and
   a. the remaining storage is equal to or greater than 2048 bytes. (The name table has the record length of 1024 bytes). The name table is read in overlapped mode from SYS001.
   b. the remaining storage is equal to or greater than 1024, but less than 2048 bytes. The name table is also read in from SYS001, but not in the overlapped mode.

The external name table is built in storage and, at the end of the phase, the table is written onto SYS001.

2. A table overflow has occurred and the pretable is written onto TXTOUT. In this case, the records of the pretable are read from TXTOUT and the records of the name table, pointed to by byte 0 of the entries of the pretable, are read from SYS001. Each processed record of the pretable is written onto SYS001.

If an external name longer than six bytes is detected, the source text is read from TXTIN, a warning or error message is inserted, and the text is written onto TXTOUT.

DESCRIPTION OF ROUTINES

Text in flow charts:
C(GCP) := Contents in location GCP points to

Symbols used in Flow Charts

GBUFCOUN	- contains number of records that fit into the table area.
GCOUNTER	- contains number of entries of one symbol-table record.
GCOUNT1	- see GCOUNTER.
GCP	- input pointer
GEND	- contains end address of pretable area.
GLEN	- contains number of table-area bytes available for use.
GLENGTH	- contains length of respective name in name table.
GNUMELE	- contains address in name table where the name to be searched can be found.
GPTCOUN	- contains number of buffers used for pretable area.
GPUTCOUN	- contains number of records written onto TXTOUT.
GREADR	- contains number of records to of records read.

GREST1/2	- contains remainder.
GTABLEA	- contains the begin address of pretable area.
GTP	- table-area pointer, (i.e., points to the next available location).
GBn	- contains address of buffer n.
GWEI	- if switch GWEI is on, input is done in non-overlapped mode.
GWEICHE	- if switch GWEICHE is on, input is read into the buffer pointed to by GB4; otherwise, into the buffer pointed to by GB3.

Initialization -- SN

In this routine, some counters, switches, and buffers for values, addresses, and input/output handling are defined and set to their initial values.

PRETAB SYMTAB Routine -- SO

The symbol table is read and the entries with the attribute EXTERNAL are stored in the pretable. If an overflow of the pretable occurs, pretable is written onto TXTOUT.

Pretable-in-Storage Routine -- SP and SQ

This routine is called if all entries of the pretable are in storage. The first record of the name table is read. The pretable is scanned for entries pointing (by the first byte) to the current record of the name table. If an entry is found, the user-defined name (pointed to by the second byte of the pretable entry) is moved from the name table into the pretable entry and translated into the external code. If the end of the pretable is reached, the next record of the name table is read and the pretable is scanned once more. This process is repeated until the end of the name table is reached. When the end of the name table is reached, the pretable area contains the complete external name table, which is written onto SYS001.

Pretable-not-in-Storage Routine -- SR

This routine is called if a pretable overflow has occurred in the PRETAB SYMTAB routine. The pretable is read from TXTOUT; the name table is read from SYS001. The record and the entry of the name table pointed to by the first two bytes of each pretable entry are searched and the contents (the user-defined name) are inserted into the pretable and translated into the external code. If one record of the pretable is processed, the record is moved into the output buffer and written onto SYS001.

SUBROUTINES

The following subroutines used in this phase are described elsewhere as follows:

Entry Point	Name	Phase
C6B2	UPRO (Input/Output Handling)	B95
C9B2	Read Routine	B95

GWORK Routine -- SS

Entry point: GB8A2

If the record of the name table pointed to by the first byte of the current pretable entry is found, this routine is called. It scans the name table for the user-defined name pointed to by the second byte of the current pretable entry. When found, its length is tested. If it is greater than 8, the ERROR switch is set on and the length of the name is set to eight. If it is greater than 6, a warning message is prepared. The user-defined name is moved into the current pretable entry and the remaining bytes are filled with blanks.

PUT Routine -- ST

Entry point: GB9A2

This routine writes the external name table onto SYS001.

Move Routine -- SU

Entry point: GBAB2

This routine moves the contents of the symbol table used for the external name table into the corresponding pretable entry.

Write Routine -- SV

Entry point: GB3A2

This routine is called by the PRETAB SYMTAB routine

1. when the area reserved for the pretable is filled and one more record must be moved into the pretable area and

2. when the end of the symbol table is reached and a pretable overflow has occurred.

GBUFCOUN contains the number of records of PRETAB, which are written by this routine onto TXTOUT.

Read Routine -- SW

Entry points: GB4A2, GB4A5

This routine is used in the PRETAB SYMTAB routine to read the symbol table and in the PRETAB-in-Storage routine to get the records of the name table.

Entry point: GB4A2 is used (1) to read the first two records of a table while the following records are read in overlapped mode, and (2) to read all records of a table that are not to be read in an overlapped mode.

Entry point: GB4A5 is used to read the third and all following records in overlapped mode. The input buffers are B3 and B4, and the buffer handling is controlled by the switch GWEICHE.

End-of-Phase Routine -- SX

If the error or warning switch is on, the error handling routine is called. Phase C00 is called and the text files are exchanged. Otherwise, one of the phases C00, C25 or C30 is called.

Error Handling -- SY, SZ

Entry point: C0B2

This routine is called if an error or warning message is to be generated. The source text is scanned for the first EA-key, (i.e., the EA-key of the PROCEDURE statement). The warning or error message is inserted behind this key. All other text remains unchanged.

PHASE PL/IC00 (SYMBOL TABLE LISTING) -- TM

This phase prints the symbol table, which contains all identifiers with their explicitly, contextually, and implicitly declared attributes. The listing is arranged according to the block numbers.

This phase is skipped if the Job Control SYM option is not active. However, it is not skipped in that case if (1) an incorrectly declared variable is detected, (2) a qualified name is not declared, or (3) an external name is longer than 6 characters. At the end of this phase, phase C25 is called if the source program contains IF statements. If no IF statements are to be processed, phase C30 is called.

All messages to be printed (except the user-defined name) are retrieved from SYMTAB. The user-defined name is retrieved from NAMTAB according to the compressed name in SYMTAB (bytes 0-1). If the internal representation of a name is zero, if a name is longer than 31 characters, or if an external name is longer than 8 characters, only the user-defined name and an error message are printed.

The number of NAMTAB records that can be stored in the work area (see the section Initialization -- TN) is referred to as K. If NAMTAB does not contain more than K records, each block of SYMTAB has to be scanned only once.

If NAMTAB has more than K records, SYMTAB is first written onto TXTOUT. The beginning of each SYMTAB block is noted simultaneously. When scanning the identifiers of one SYMTAB block, all parts of NAMTAB must be successively moved into the work area until all entries of the block have been listed.

Phase Input and Output

The input used by this phase consists of the tables SYMTAB and NAMTAB (contained on SYS001). The format of the symbol table listing is described in detail in the PL/I Programmer's Guide.

Buffers and Switches

Buffers 1 - 3	are used as the last part of the work area.
Buffer 4	= buffer A
Buffer 5	= buffer B and print buffer
Buffer 6	= buffer C
Switch NAMIN	is set if the entire NAMTAB can be stored in the work area.

DESCRIPTION OF ROUTINES

Initialization -- TN

The work area is used to store NAMTAB or parts thereof. Space S accommodates parts of the phase (beginning with WBEG1), the table space, and the first three buffers.

If NAMTAB can be entirely stored in space S, the work area is equal to space S. If not, the note information on the beginning of each SYMTAB block is stored in the beginning of space S. The remaining space of space S is used as work area.

Note Blocks of SYMTAB -- TO

SYMTAB is read into buffers A and B and written onto TXTOUT. The beginning of each SYMTAB block on TXTOUT is noted.

Store K Records of NAMTAB in Turn -- TP

Up to K NAMTAB records are read into the work area each time. The smallest and greatest number of NAMTAB records in storage are noted in MIN and MAX. Scanning of a SYMTAB block starts with MIN=1 and MAX=K. The records of a SYMTAB block are read into buffer A and scanned for entries the names of which are in the part of NAMTAB that is in the work area.

If the buffer pointer points to the end of the SYMTAB block and all entries of the block have been listed, scanning of the next SYMTAB block is started. Otherwise, MAX and MIN are increased by K. The next records of NAMTAB are read into the work area, and scanning of the same SYMTAB block starts again.

Store Entire NAMTAB -- TQ

Switch NAMIN is set and the entire NAMTAB is read into the work area. SYMTAB is successively read from SYS001 into buffers A and C.

User-Defined Name, Error Message -- TR

The entry of a user-defined name is retrieved from the work area. The name is moved into the print buffer and translated from internal code into EBCDIC. If the name is longer than 31 characters or if its internal representation is zero, the name is printed with an error message.

Start Fetching Attributes -- TS

The entry of the identifier in SYMTAB is scanned for attributes. If the identifier is an external name of more than 8 characters in length, the name and an error message are printed. If the identifier is a built-in function, only the name as well as the block and level number are printed. In all other cases, the internal representation and the block and level number of the identifier are moved into the print buffer. It is tested whether the identifier is an array, a structure, or an entry name.

Arithmetic and String -- TT

Base, scale, and precision of an arithmetic identifier are moved into the print buffer. Types and length of a string identifier are also moved into the print buffer.

End of Fetching Attributes -- TU

Fetching of attributes of the identifier from its entry in SYMTAB is terminated.

Subroutines -- TV

Work up precision w or length l
Entry point: WSRB01

Input parameters:
R3 points to the entry of the identifier in SYMTAB. R11 points to the print buffer.

Precision w or length l is retrieved from the entry of the identifier in SYMTAB, converted to its decimal value, and moved unpacked into the print buffer.

Suppress leading zeros
Entry point: WSRB02

Input parameter:
R2 points to the number to be checked.

Leading zeros of the number to be checked are replaced by blanks.

PHASE PL/IC25 (IF STATEMENT) -- TZ

This phase is called if the source program contains IF statements. Phase C25

- analyzes all IF nests,
- replaces all IF statements with IFFALSE statements,
- generates certain macros,
- detects any incorrect IF nesting or any incorrect use of ELSE.

Phase Input and Output

The input is a string of unambiguous 3-byte elements and elements of variable length (see output of phases A60/A65). During phases A60/A65, IF statements were made non-recursive by replacing each THEN by an EOS (End of Statement) and by placing an EOS after each ELSE, thus making ELSE a "statement."

The output is similar to the input except that few additional types of statements and/or macros have been added or substituted.

STATEMENTS AND MACROS PUT OUT BY C25

The IFFALSE Statement

Meaning of the IFFALSE statement:

If expression yields FALSE, go to nL.

This statement is substituted for each IF statement and is of the following format:

IFFALSE	statement attr. table	expression	EOS

where IFFALSE is the statement identifier, identical to the statement identifier IF. nL is a generated label of the following format:

Byte(s)	Contents
1	key X'EE'
2-3	number of the generated label. It is obtained by adding 1 to counter IJKMVC each time the label generating routine is called.
4	key X'EE'
5-6	X'0069' (indicates that the generated label is a label constant)

expression is the original expression transformed into 3-byte elements and/or elements of variable length.

The DEFINE LABEL Macro

The definition-point of a generated label is indicated by a DEFINE LABEL macro. The format of the DEFINE LABEL macro is as follows:

Byte(s)	Contents
1	macro key X'F2'
2-3	length of the macro
4	key X'72' indicating that this macro is of the type DEFINE LABEL
5	key X'BB' (in DEFINE LABEL macros, generated label constants have the key X'BB' instead of X'EE')
6-7	number of the generated label

The BRANCH Macro

Meaning of the BRANCH macro: branch to the generated label specified in bytes 6 to 8 of the macro.

The format of the BRANCH macro is as follows:

Byte(s)	Contents
1	macro key X'F2'
2-3	length of the macro
4	key X'70' indicating that this macro is of the type BRANCH
5	X'0F' (code for unconditional branch)
6	key X'EE'
7-8	number of the generated label
9-11	modifiers (here always 0)

Sample Input and Output of Phase C25

- Statements have statement identifiers consisting of capital letters (for instance: IFFALSE, SET, READ, etc.)
- Macros are identified by lower case letters (for instance: define label, branch).
- Generated labels are written like 1L, 2L, 3L etc.

  Note that the input and the output actually consists of a string of 3-byte elements and/or elements of variable length.

```
     Input                  Output

1    IF ex1;                IFFALSE 1L ex1;
        SET A=B;            SET A=B;
     ELSE;                  branch 2L
        SET C=D;            define label 1L
                            NOP;
                            SET C=D;
                            define label 2L

2    IF ex1;                IFFALSE 1L ex1
        SET A=B;            SET A=B;
     SET C=D;               define label 1L
                            NOP;
                            SET C=D;

3    IF ex1;                IFFALSE 1L ex1;
        IF ex2;             IFFALSE 2L ex2;
           IF ex3;          IFFALSE 3L ex3;
     SET C=D;               SET C=D;
                            define label 3L
                            NOP;
                            define label 2L
                            NOP;
                            define label 1L
                            NOP;

4    IF ex1;                IFFALSE 1L ex1;
        IF ex2;             IFFALSE 2L ex2;
           SET A=B;         SET A=B;
     ELSE;                  branch 3L
           SET C=D;         define label 2L
                            NOP ;
                            define label 1L
                            SET C=D ;
                            define label 3L

5    IF ex1 ;               IFFALSE 1L ex1 ;
        BEGIN ;             BEGIN ;
        alpha               alpha
        END ;               END ;
     SET A=B ;              define label 1L
                            SET A=B ;

6    DO ;                   DO
     IF ex1 ;               IFFALSE 1L ex1 ;
        BEGIN ;             BEGIN ;
        alpha               alpha
        END ;               END ;
     ELSE ;                 branch 2L
        BEGIN ;             define label 1L
        beta                NOP ;
                            BEGIN ;
        END ;               beta
                            END ;
     gamma                  define label 2L
     END ;                  gamma
                            END ;
```

Phase Performance:

Each encountered statement is tested to determine whether it

- is an "End of Unit 1", or
- is immediately following an "End of Unit 1", or
- is an "End of Unit 2".

If the statement is of the "End of Unit 1"-type, the last entry in the symbol stack (presumably IFPH4, standing for "IF") will be replaced by IFPH3 (standing for "End of Unit 1"). If there are several consecutive IFPH4 entries in the stack, each of them will be replaced by an IFPH3. Then the statement will be put out.

If the statement immediately follows an "End of Unit 1", as many macros, define labels, and FALSE labels are put out as there are consecutive IFPH3 entries in the symbol stack. The FALSE labels will be taken from the label stack. Then the statement will be put out, or new statements are generated for IF and ELSE.

If the statement is of the "End of Unit 2"-type, the statement will be put out, now followed by a macro, a define label, and an EXIT label. The EXIT label is taken from the label stack.

Processing of the Input Stream

If a DO or BEGIN statement is encountered, the corresponding one-byte symbol IFPH1 or IFPH2 is entered into the symbol stack. Then the statement is tested and processed as described in Phase Performance.

If an END of group or END of block is encountered, the corresponding symbol IFPH1 or IFPH2 is eliminated from the symbol stack. Then the statement is tested and processed as described in Phase Performance.

If an IF statement is encountered, the symbol IFPH4 is entered into the symbol stack and the statement is then processed as described in Phase Performance. A label KL is generated, entered into the label stack, and the statement --IFFALSE KL expression;-- is put out.

If an ELSE statement is encountered, the last entry in the symbol stack is replaced by IFPH5, a label nL is generated, a --branch nL;-- is put out, the last entry in the label stack mL is used to put out define label mL, and the generated label nL is entered into the label stack.

An END of procedure is subject to a specific test, for it may never be used as a "Unit 1" or "Unit 2" in an IF statement. In this case an error message is given.

Tables and Pointers

Two push down stacks are used: a symbol stack and a label stack.

The symbol stack IFPH86 consists of 100 one-byte elements. IFPH86 is used to store the following symbols:

IFPH1	for	DO
IFPH2		BEGIN
IFPH3		"End of Unit 1"
IFPH4		IF
IFPH5		ELSE

The pointer to IFPH86 is the symbolic register R7.

The label stack IFPH87 consists of 100 half-word elements, and is used to store generated labels (FALSE labels as well as EXIT labels). The pointer to IFPH87 is the symbolic register R6.

DESCRIPTION OF ROUTINES

Note: A routine is called 'open' if it gets control by a B instruction. A routine is called 'closed' if it gets control via a BAL instruction, and if control is returned by a BR instruction.

IFPH -- UA

This is the "master program" of phase C25. IFPH initializes pointers, registers, etc. and reads the first 4 records into input buffers 1 to 4.

IFPH scans the input until a statement identifier is found. Upon this, the Define Label macros (which may precede the statement), the statement identifier, and the statement attribute table are put out. Depending on the encountered statement, one of the following routines is called:

Statement:	Called routine:
IF	IFIF
BEGIN	BEBE
DO	DODO
END (of BEGIN block)	BLBL then EOST
END (of DO group)	GRGR
ELSE	ELEL
Any other statement:	VIRGO, then NSNS, then EOST.

After return of control to IFPH, the scan is continued. If the end of program is reached, TEPHA is called.

BEBE, DODO -- UN (Closed)

BEBE puts IFPH2 (symbol for "BEGIN") into the symbol stack. If this is the first entry into the symbol stack, the statement body is put out and the program returns. Otherwise, the preceding entry in the symbol stack is tested. If this is IFPH3 (symbol for "end of unit 1"), FOUT is called to put out a "Define Label" macro. The operand of this macro is the last entry in the label stack. Then IFPH3 is replaced by IFPH2. The stack pointer R7 is decremented by 1, the statement body is put out, and the symbol stack entry currently selected by R7 is tested as described.

If the tested entry in the symbol stack is not IFPH3, the statement body is put out and the program returns.

DODO performs the same as BEBE but uses IFPH1 instead of IFPH2.

BLBL -- UO (Closed)

The last entry in the symbol stack is tested. If this entry is IFPH3 (symbol for "end of unit 1"), a Define Label macro is generated. The operand of this macro is the label stack entry selected by label stack pointer R6. Then R7 is decremented by 1 and R6 is decremented by 2. Then the symbol stack entry currently selected by pointer R7 is tested as described.

If the tested entry in the symbol stack is IFPH4 (symbol for "IF"), it is replaced by IFPH3. R7 is decremented by 1. Then the symbol stack entry currently selected by pointer R7 is tested as described.

If the tested entry in the symbol stack is IFPH5 (symbol for "ELSE"), the statement body and a DEFINE LABEL are put out. The operand of the DEFINE LABEL is the label stack entry selected by R6.

BSAC (Closed) -- UF

The routine initiates output of the statement attribute table for the currently processed statement.

BYPA (Closed) -- UD

The routine puts out either the one part of the statement attribute table that contains attributes of variables, or the other part which contains attributes of constants. Then the statement body is positioned to start in input buffer 1.

DPDS (Closed) -- UJ

DPDS compares the symbol stack entry currently selected by R7 with the argument in R4. R4 contains a symbol (either IFPH1 for

"DO" or IFPH2 for "BEGIN"). If the symbol stack entry matches the argument in R4, the entry is deleted and pointer R7 is decremented by 1.

If entry and argument do not match, the search continues until a matching entry is encountered. Then the matching entry is deleted. All symbol stack entries at a higher level than the matching entry are moved down one position. Pointer R7 is decremented by 1.

ELEL -- UK (Closed)

ELEL generates a label and puts out a Branch macro with the generated label as operand. The generated label is stored in ELEL2. Then a Define Label macro with the last entry in the label stack as operand, followed by NOP, is put out. The latest Label Stack entry is replaced by the label stored in ELEL2. IFPH5 (symbol for "ELSE") is entered into the symbol stack. Finally, a NOP statement is put out.

ERROR, JERRA1 (Closed) -- US

This routine is described in phase A35.

EOST, JEOSA1 (Closed) -- UR

The routine arranges the contents of the input buffers 1 to 4 so that the currently scanned EOS is in input buffer 1. This is done by moving and reading new records. It puts out the EOS and the attached error codes. Any additionally generated error codes are also put out.

FOUT (Closed) -- UB

The routine puts out a Define Label macro. The operand of this macro is the last entry of the label stack. Stack pointer R6 is decremented by 2.

GEOS (Closed) -- UG

The routine moves the input pointer PIN until an EOS is encountered. The address of the byte preceding this EOS is stored in IFPH96.

GRGR -- UO (Closed)

Entry point to BLBL.

GSN (Closed) -- UH

GSN moves the statement identifier of the current statement into GSN4. It returns to 4(0,LINK) if the statement is correct. Otherwise, it returns to (LINK).

IFIF -- UI (Closed)

IFIF tests the symbol stack entry currently selected by R7. If this entry is IFPH3 (symbol for "end of unit 1"), a Define Label macro is generated. The operand of the Define Label macro is the label stack entry currently selected by R5. Then R7 is decremented by 1 and R5 is decremented by 2. The symbol stack entry currently selected by R7 is tested as described. Otherwise, a label is generated, stored in the label stack, and put out followed by the statement body (see description of IFFALSE statement).

IPDS (Closed) -- UL

IPDS increments stack pointer R7 by 1 and enters the rightmost byte in R0 into the symbol stack.

JTRNA1 (Closed) -- UQ

This is the output routine. Register BYZ contains the number of bytes to be put out; register PIN contains the start address. One output buffer is used.

If the remaining portion of the string to be put out is smaller than the remaining unoccupied space of the output buffer, the string is moved into the buffer. BYZ is added to POUT to update the output pointer.

If the string to be put out exceeds the unoccupied space, an appropriate portion of the string is moved to fill the output buffer to its capacity. Then the contents of the buffer are written onto the output medium. POUT is reset to the start address of the buffer. BYZ is decremented by the number of bytes moved into the buffer, and PIN is incremented by that number. Then JTRNA1 is repeated until output is completed.

LGEN (Closed) -- UC

LGEN generates a label and enters it right-justified into register R1. The format of the generated label is shwon in Figure 1.

MAMA (Closed) -- UT

This routine puts out

1. Label list
2. Statement identifier
3. Statement attribute table

NSNS -- UO (Closed)

Entry point to BLBL.

POB (Closed) -- UF

When POB is called, R2 contains the start address and R1 the end address of a string to be put out. POB is an "interface" to the routine JTRNA1 which requires the start address and the length of a string to be put out. POB performs the necessary transformations.

STEP (Closed) -- UE

STEP tests the high-order 4 bits of the byte selected by PIN. If these bits are set to X'E', PIN is incremented by 3. If these bits are set to X'F', PIN is incremented by the contents of the two bytes following the byte selected by PIN. If these bits are set to any other value, a compiler error occurred and a dump is initiated.

VIRGO -- UM (Closed)

The symbol stack entry currently selected by R7 is tested. If this entry is IFPH3 (symbol for "end of unit 1"), a Define Label macro is generated. The operand of this macro is the label stack entry currently selected by R6. R7 is decremented by 1 and R6 is decremented by 2. Then the symbol stack entry currently selected by R7 is tested as described. If the selected entry is not IFPH3, VIRGO returns to either 4 (0,LINK), if only one test has been performed, or (LINK) if more than one test has been performed.

PHASE PL/IC30 (PROCESSING CONSTANTS I) -- WA

This phase performs the following functions:

1. It scans all constants for acceptable precision.

2. It replaces the external format of the constants by an intermediate one.

3. It builds up the constant tables as part of the statement attribute tables.

Note: The character strings have already been processed in phase A45.

If a constant is preceded by a prefix plus or minus, this sign is removed from the source text, and a corresponding sign-bit is set in the constant table.

Phase Input and Output

The text input consists of a sequence of statements terminated by the end-of-program key. Each statement is composed of the following elements:

1. The statement identifier key (6 bytes) which may be preceeded by one or more label macros.

2. The symbol table, if there are any variables in the statement.

3. The statement body.

4. The end-of-statement key (6 bytes) which may be followed by one or more error-keys (2 bytes).

The statement body consists of elements which formally may be distinguished by E-keys (3 bytes) and F-keys (variable length). The constants are interspersed within the statement body and contain the following information:

- one of the six constant keys, the difference depending on the type of constant:

  X'F7' = decimal fixed-point constant
  X'F8' = decimal floating-point constant
  X'F9' = binary fixed-point constant
  X'FA' = binary floating-point constant
  X'FB' = bit-string constant
  X'FC' = sterling constant

- the length of the constant (2 bytes), and

- the constant.

The character strings have already been processed in phase A45 and are collected in the character-string table on SYS001. Within the statement body they are replaced by a reference key that consists of the following:

- Key 'character string' = X'E3'

- Offset, relative to the start of the character string table (2 bytes)

- Key 'character string' = X'E3'

- Error-byte

  X'00' if no error
  bit 0 set to 1 if error 55
  bit 1 set to 1 if error 56 (see phase A45)
  bit 2 set to 1 if error 67

- length of the character string (1 byte)

Like the input, the output consists of a sequence of statements, terminated by the end-of-program key. Each statement is composed of the following elements:

1. The statement identifier key (6 bytes) which may be preceded by one or more label macros.

2. The symbol table, if there are any variables in the statement.

3. The constant table, if there are any constants in the statement.

4. The statement body.

5. The end-of-statement key (6 bytes) which may be followed by error-keys (2 bytes).

The constant table consists of the following:

- Constant-table key = X'F3' (1 byte),

- Length of the constant table (2 bytes), and

- one or more constant entries.

Each entry of the constant table contains the following:

- Internal name of the constant (2 bytes) (N = IJKMVC, which is increased by 1 for every constant).

- Attributes of the constant (inserted and used by following phases, here initialized with X'10') (1 byte).

- Type of the constant (1 byte)

  X'60' = binary float
  X'61' = binary fixed
  X'62' = decimal float
  X'63' = decimal fixed
  X'67' = bit string

  Note: Sterling constants are stored as decimal fixed-point pence.

- Precision of the constant (1 byte)

  if binary float: P (0<P≤53)
  if binary fixed: P (0<P≤31)
  if decimal float: P< (0<P≤17)
  if bit string: P (0<L≤64)
  if decimal fixed, bits 0-3: P (0<P≤15)
  bits 4-7: Q (0<Q≤P)

- Three bytes containing zeros, used by following phases for "new type and precision." The first bit is set to 1, if the constant is preceded by a prefix minus.

- Length of the intermediate representation of the constant (2 bytes).

- Intermediate representation of the constant, depending on the type of constant:

  binary float

  binary integer contained in a field of 4 bytes (if P≤21) or 8 bytes (if P>21), followed by a binary integer (2 bytes) representing the binary exponent.

  decimal float

  decimal integer in packed decimal format (length of field = FLOOR (P+2/2)), followed by a binary integer (2 bytes) representing the decimal exponent.

  binary fixed

  32-bit binary format (see _IBM System/360, Principles of Operation_, Form A22-6821)

  decimal fixed

  packed decimal format (see _IBM System/360, Principles of Operation_, Form A22-6821). Length of field = FLOOR (P+2/2).

  Note: The position of the decimal point is recorded by the scale factor Q.

  bit string

  byte-aligned, one binary digit per bit.

Within the statement body, the constant has been replaced by the reference key. This key consists of the following:

- Key 'constant reference' = X'E9' (1 byte).

- Internal name of the constant (see constant table) (2 bytes)

The character strings are referenced by a key containing the following:

- Key 'character string' = X'E3' (1 byte).

- Offset relative to the start of the character string table (2 bytes).

- Key 'character string' = X'E3' (1 byte).

- Length of the character string (2 bytes).

DESCRIPTION OF ROUTINES

Initialization -- WB

This is the beginning of the main routine. It initializes pointers, switches etc. Then it reads in four buffers of input text.

FCSC -- WC, WD

This is part of the main routine. It performs a general scan over the source text. The labels, the statement identifier, and the attribute table are moved into the output buffer. The length of the attribute table is saved in STABL; the begin address of the statement body is saved in PINS. The statement body is scanned for constants, which are processed in FCON. If errors are detected in the character strings, the corresponding error codes are moved into the error table. When the end-of-statement key has been reached, control is transferred to FEST.

FEST -- WE

Input parameter:
PINS = address of the beginning of the statement body.

This is part of the main routine. It moves the constant table (if constants exist in this statement) into the output buffer. The constant table is followed by the statement body in which the constants are replaced by 3-byte reference keys.

Routine FCON -- WG

Input parameters:
RLEN = (register) length of external representation of constant.
PIN = (register) address of constant key.
PTAB = (register) constant table pointer.
TABL = (half-word) length of constant table + length of symbol table.

Output parameters:
PIN = PIN + length of constant key.
PTAB = points to the next available byte in the constant table.
TABL = TABL + length of the last entry processed.

By means of one of the called routines, the constant is scanned for acceptable precision. Type and precision are entered in the constant table with the constant itself in its intermediate representation. The constant table entry is completed by entering the internal name, the attribute byte, the three 0-bytes, and the length of the intermediate representation.

If IJKMVC is greater than 2^{32}-2, it is reset to 0, and an error message is produced. The same error message is generated in the case of a table-space overflow (TABL must not be greater than the table space), furthermore, PTAB and TABL are not increased.

Finally the constant key is replaced by the constant reference key and as many blanks as are needed to overlay the constant in its external format. These blanks are eliminated in FEST. If the constant is preceded by a prefix plus or minus, the minus sign is taken into account by setting the first bit of the 3 zero-bytes to 1. The prefix signs are then removed from the source text by overlaying them with the constant key and replacing all bytes of the constant by blanks.

Routine FBFL -- BI

Input parameters:
R1 = address of the constant in its external format
RLEN = length of the external format
PTAB = pointer to the constant table

Output parameters:
PTAB = unchanged
RLEN = length of the intermediate representation of the constant

This routine processes the binary floating-point constants. The precision of the constant is determined in FPFL. If there are more than 53 binary digits (error number 58), the constant is truncated on the right, the exponent is increased accordingly, and JERR is called. The exponent of the intermediate representation is obtained by subtracting the number of digits specified after the binary point from the exponent specified by the programmer. The binary digits of the external format (each digit occupying one byte) are condensed to a bit string (each digit occupying one bit) in FBIN. The constant is stored in the constant table.

Routine FBFI -- WM

Parameters: same as in FBFL.

This routine processes the binary fixed-point constants. If there are more than 31 digits (error number 62), the constant is truncated on the right, and JERR is called. The binary digits of the external format (each digit occupying one byte) are condensed to a bit string (each digit occupying one bit) by means of FBIN.

Routine FDFL -- WK

Parameters: same as in FBFL.

This routine processes the decimal floating-point constants. The constant is stored as a decimal integer followed by an exponent. This exponent is obtained by reducing the exponent specified by the programmer by the number of digits after the decimal point. If there are more than 16 digits (error number 58), the number is truncated on the right and the exponent is increased by the number of digits being truncated.

Routine FDFI -- WN

Parameters: same as in FBFL.

This routine processes the decimal fixed-point constants. If there are more than 15 digits (error number 63), the constant is truncated on the right.

Routine FBST -- WL

Parameters: same as in FBFL.

This routine processes the bit string constants. If a replication factor greater than 1 has been specified, the bit string is expanded accordingly. Bits exceeding 64 are truncated (error number 56).

Routine FSTL -- WJ

Parameters: same as in FBFL.

This routine processes sterling constants. The constant is converted to and stored as decimal fixed-point pence. The conversion is done by the instructions ADD and MULTIPLY DECIMAL; if, however, the decimal feature is not available, these instructions must be simulated. The precision of the constant is taken from the converted number; leading zeros are ignored. If more than 15 significant digits have been obtained (error number 61), the decimal fixed-point pence number is truncated on the right.

Routine FPFL -- WI

Input parameters:
R1 = address of the constant in its external format
RLEN = length of the external format

Output parameters:
R1 = unchanged
REXP = exponent specified by the programmer
RLEN = number of digits specified for fixed-point portion of constant (P)
RQ = number of digits specified after decimal (binary) point (Q)

This routine scans the precision and the exponent of a floating-point constant. If the specified exponent exceeds 3 digits (error number 57), the remaining digits are truncated.

Routine FPFI -- WI

This is a secondary entry point of FPFL.

Input parameters:
R1 = address of the constant in its external format
RLEN = length of the external format (of a fixed-point constant)

Output parameters:
R1 = unchanged
RLEN = number of digits of decimal fixed-point constant (P)
RQ = number of digits specified after decimal (binary) point (Q)

This routine scans the precision of a fixed-point constant.

Routine FREP -- WL

Input parameters:
R1 = address of the constant in its external format
RLEN = length of the external format

Output parameters:
R1 = address of the basic string
RLEN = length of basic string
REPL = replication factor

Converts the replication factor of a bit-string constant to binary. If no replication factor is specified, REPL is set to 1. A zero replication factor is ignored (error number 55) and REPL is set to 1.

Routine FBIN -- WL

Input parameters:
R1 = address of the first digit of the constant
RLEN = number of digits
Output paramters:
(R4, R5) resulting bit string (binary number), right-aligned

This routine condenses a character string of zeros and ones to a bit string.

Routine JEOS -- WP

This routine positions the contents of input buffers 1-4 so that the currently scanned EOS is in input buffer 1 (this is done by calling JMIB). The EOS and the error codes attached to it are written on the text output file. If additional error codes are generated, they are also put out.

Routine JMIB -- WQ

This routine moves input text to the left and reads in new records.

Routine JSLC -- WR

This routine determines if a statement is too long (i.e., if its EOS key is in the first 4 input buffers). If so, the statement body is deleted so that the statement consists only of the statement identifier and the EOS (with error codes). The following statement is positioned so that it begins in input buffer 1. If the statement is not too long, this routine returns to the calling routine.

Routine JTRN -- WO

Input parameters:
PIN = pointer for source text
POUT = pointer for output buffer
BYZ = number of bytes to be moved

Output parameters:
PIN = PIN+BYZ
POUT = address of next available byte within the output buffer

If all the bytes to be moved do not fit into the output buffer (or if it is completely filled), the buffer is filled by the first part of the text to be moved and then written on the text output work file. The remaining bytes, if any, are moved to the beginning of the buffer.

Routine JERR -- WQ

This routine checks whether the error table is full and returns in that case. If the error table is not full, the number of errors is increased by one and the corresponding error key is inserted.

PHASE PL/IC35 (BLOCK SORTING) -- VA, VB

The main task of this phase is to sort the blocks arising in the source program. The input is on TXTIN. Three different categories are to be distinguished.

1. The source program consists of only one block. Therefore, only one scan of the input string is required. The block with the level number zero is written onto TXTOUT.

2. The source program consists of blocks with the level numbers zero and one. This requires the input string to be scanned twice:

   a. The block with the level number zero is extracted and written onto TXTOUT. The blocks with the level number one are written onto SYS001.

   b. The input is on SYS001 and written unchanged onto TXTOUT.

3. The source program consists of blocks with the level numbers zero, one, and two. This requires the input string to be scanned three times:

   a. The block with the level number zero is extracted and written onto TXTOUT. The blocks with the level numbers one and two are written onto SYS001.

   b. The input is on SYS001. The blocks with the level number one are extracted and written onto TXTOUT. The blocks with the level number two are written onto TXTIN.

   c. The input is on TXTIN and written unchanged onto TXTOUT.

If, in the nth scan, a BEGIN statement is found that opens a block with level number n, a label is generated in front and the block with the label is written onto SYS001 or TXTOUT. Instead of the BEGIN block a CALL macro containing the new label is generated. If the BEGIN statement is in an embracing BEGIN block, the statement is additionally changed to NEW BEGIN.

If a PROCEDURE statement is found, either the entire attribute table or the first 18 bytes of the attribute table are stacked depending on the length of the table.

If an END (procedure) statement is found, the last entry in the stack is cleared.

If a RETURN statement is found, the contents of the last entry in the stack are inserted after the first 8 bytes of the RETURN statement.

The contents of the attribute table of each statement are translated into a new form by the translate subroutine.

I/O Handling (Buffers)

Six buffers are used: five buffers, i.e., output buffer 1, three work buffers, and one input buffer, are in the I/O area, and output buffer 2 is defined in the table area.

DESCRIPTION OF ROUTINES

Symbols used in flow charts:

C(B)	contents of location pointed to by B
C(B+1)	contents (length 2 bytes) of location pointed to by B+1
A(B)	address of B

Initialization -- VC

The following items are defined and set to their initial values:

ALEVEL	actual level (0)
CLEVEL	current level (-1)
MAXCL	maximum value of CLEVEL (-1)
ZAEHL	I/O record counter (0)
ZAEHL2	I/O record counter (0)
COUNTER	I/O record counter (0)
COUNTER2	I/O record counter (0)
ML	move-instruction length (4)
HISPEI	intermediate storage for address
BEGINBIT	switch
NEWBEGIN	key of changed BEGIN (X'17')
NBEGIN	key of changed BEGIN (X'16')
OUTPUT	temporary buffer
TATTRIB	attribute table stack
ATTRIB	address of TATTRIB+3*21

Pointers:

B6	start address of output buffer 2
B0	IJKMBS = start address of output buffer 1
B1	start address of work buffer
B2	B1 + buffer length
B3	B2 + buffer length
B4	start address of input buffer
B5	B4 + buffer length

Registers:

OP2	pointer for output buffer 2 is set to B6
OP1	pointer for output buffer 1 is set to B0

CP	current pointer in work buffer is set to B5
LENGTH	counter used for text output

Main Routine -- VD

The main routine scans the current input string for some special keys and calls the appropriate subroutines.

Procedure Handling -- VE and VF

Entry point: VEB2

This routine is called when a label macro is found. The label macro is written out if it is not followed by a PROCEDURE statement.

If it is followed by a PROCEDURE statement, the subroutine CLEVMAX increases the current level CLEVEL, and if CLEVEL is not zero, a library bit is inserted. Then the label and the beginning of the PROCEDURE statement are written out.

If the PROCEDURE statement is detected during the first scan through the input string, some additional actions are required, e.g., the attribute table must be stacked, translated, and written out.

Begin Handling -- VG

Entry point: VGB2

This routine is called when a BEGIN statement is found in the input string. At first the current level CLEVEL is increased by one in the routine CLEVMAX and is then compared to the actual level ALEVEL.

If the current level is equal to the actual level and the actual level is one, the switch BEGINBIT is set to one.

If the current level is not equal to the actual level, the difference between the actual level and the current level is tested. If the difference is one, some pointers are changed to move the label macro into the OP2 buffer.

The label macro is 7 bytes long and contains the following information:

Byte (s)	Contents
1	F2 - macro key
2	00
3	07
4	72 : label
5	E1
6-7	variable counter

Switch BEGINBIT is tested and if BEGINBIT = 1, the BEGIN key is replaced by the contents of NEWBEGIN.

The prestatement is moved into the OP2 buffer. Thereafter, any existing labels located behind the prestatement are moved into the OP1 buffer followed by the CALL statement.

The call macro is 19 bytes long and contains the following information:

Byte(s)	Contents
1	E0
2	FF
3	09
4-6	refer to the BEGIN statement
7	macro key
8	00
9	07
10	A0
11	E1
12-13	variable counter IJKMVC
14	EA
15-19	refer to BEGIN statement.

The variable counter IJKMVC is increased by one. When the value X'FFFF' is reached, an error message is generated.

Return Handling -- VH and VI

Entry point: VHB2

This routine is called when the beginning of a RETURN statement is found. The pointer LENGTH is increased by six and the input/output subroutine is called. Unless this routine is not called during the first scan through the input stream, (ALEVEL = 0) the end of the routine is reached. Otherwise, the attribute table is processed. If an F4-key is found, the translate routine is called and the translated attributes are moved into the output buffer. Thereafter, the constant table, if there is one, is moved and the last entry of the attribute stack (made by detecting the last PROCEDURE statement and pointed to by ATTRIB) is inserted. Then, RETKON is moved into the output area. RETKON is 6 bytes long and contains the following information:

Byte(s)	Contents
1	E1
2-3	RETURNL
4	E2
5	00
6	EB

Figure 1 shows the format of the RETURN statement at the end of this routine.

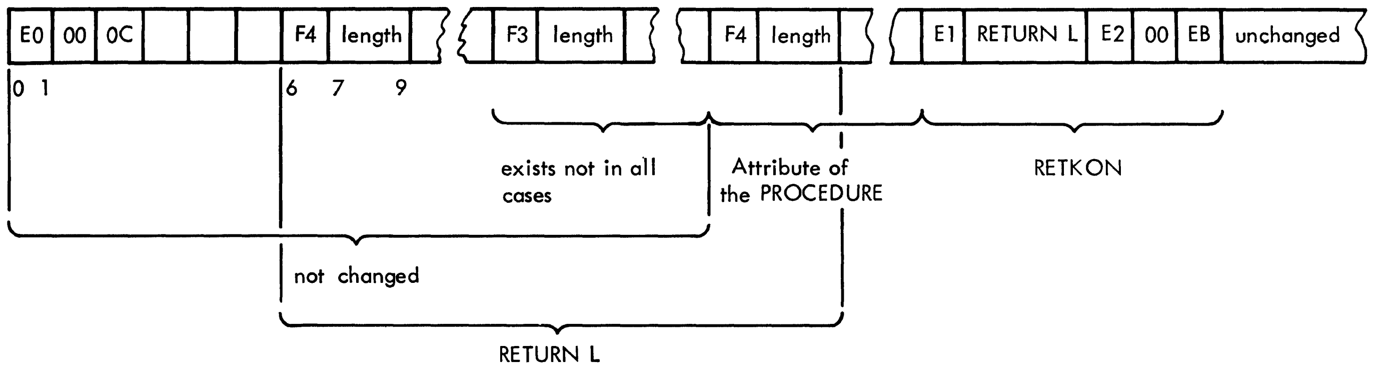


Figure 1. Format of RETURN Statement

End (Procedure) Handling -- VK

Entry point: VKB2

This routine is called when an END (procedure) statement is found. First, ALEVEL is tested. If it is zero, the last entry of the attribute-table stack is cleared.

The following actions are also performed by the END (begin) Handling routine:

The beginning of the statement is written out, the routine Label Handling is called, the statement end with error keys, if any, is written out, and CLEVEL is decreased by one before the end of the routine is reached.

END (Begin) Handling -- VL

Entry point: VLB2

This routine is called when an END (begin) statement in the source text is found. If ALEVEL and CLEVEL contain one, the BEGINBIT is set to zero before branching to the END (procedure) Handling routine.

Label Handling -- VM

Entry point: VMB2

This routine is called when generated labels are found in the prestatement. These labels are moved into the output buffer and the counter CLEVEL is increased by one.

End program -- VN and VO

Entry point: VNB2

This routine is called if the end-of-program key is detected.

If IJKMBC contains one, the end of the source text is written out and the next phase is called. Otherwise, the next scan through the input stream is started. Depending on MAXCL and ALEVEL, the text is processed. If MAXCL equals ALEVEL, the Endlevel routine is called. The output (text unchanged) of the last scan is moved onto TXTOUT and the end of the phase is reached, otherwise, the routine returns to the initialization routine of this phase, and a new scan begins.

Translate Routine -- VP - VS

Entry points:
VPB2 (translate routine 1)

TRANSLAT (translate routine 2).

The subroutine Translate translates the attributes in the variable entry into the following 1-byte form:

Bit 0 :	0 =	not controlled,
	1 =	controlled
Bits 1-3 :	000 =	Scalar variable without picture
	001 =	Scalar variable with picture
	010 =	Array without picture
	011 =	Array with picture
	100 =	ENTRY name or function name without picture
	101 =	Function name with picture
	110 =	Constant
Bits 4-7 :	0000 =	Binary float
	0001 =	Binary fixed
	0010 =	Decimal float
	0011 =	Decimal fixed
	0100 =	Zoned decimal
	0101 =	Zoned decimal (T)
	0110 =	Character string
	0111 =	Bit string
	1000 =	Sterling
	1001 =	Label
	1010 =	Pointer
	1100 =	Major Structure
	1101 =	Minor Structure
	1110 =	Others
	1111 =	File

Input parameter:
RPOI (4 bytes) contains the address of the variable entry in the prestatement to be translated.

Output parameter:
ELENG (1 byte) contains the entry length of the declared variable.

Endlevel Routine -- VT

Entry point: VTB2

This routine reads the output of the last scan of the text from SYS001 or TXTIN and writes it unchanged on TXTOUT.

CLEVMAX Routine -- VU

Entry point: VUB2

The current level CLEVEL is increased by one and compared with MAXCL. If CLEVEL is greater than MAXCL, MAXCL is set to CLEVEL.

I/O Handling -- VW

Entry point: VWB1

This routine controls the buffer handling. It is called when a string of the source program is to be written out. At first, a test is performed to determine whether the string is contained in its full length in the buffer area.

If it is, the move routine is called where the string is moved into the output area. Then, the read routine is called where the input buffer is filled, if required, and this routine returns.

If the string is not contained in its full length in the buffer area, the section of the string contained in the buffer area is moved, and the input and work buffers are filled. Then the remainder of the string is moved and written out, and the routine returns.

Read Routine -- VX

Entry point: VXB1

The input and work buffers are filled if required. Therefore, CP is compared to B2. If CP is lower than B2, the routine returns. Otherwise, the contents of the buffers B2, B3, and B4 are moved into the buffers B1, B2, and B3, respectively. The buffer B4 is filled with the next record from TXTIN or SYS001.

Move Routine -- VY

Entry point: VYB2

This routine is called if a string has to be moved into the output area. Depending on the contents of ALEVEL and CLEVEL, buffer B0 or B6 is used. The entire string or part of it is moved into the buffer area depending on the length of the string and the number of free bytes in the buffers. A full buffer is written out by the write routine, if required.

Write Routine -- VZ

Entry point: VZB2

Depending on the contents of ALEVEL and CLEVEL, OP1 or OP2 are compared to B1 or B0. If the result of this comparison is not equal, the routine returns without further actions. Otherwise, the contents of the buffers B0 or B6 are written onto TXTOUT, TXTIN, or SYS001 depending on ALEVEL.

The I/O scan is performed in this phase and the phases C55, C60, and C65. These phases process all I/O statements. The functions of these phases are:

1. to check the statements for errors that are not detected by phases A60 and A65.

2. to prepare the statements for processing in later phases:

    a. to generate DO statements for the repetitive specifications in the data lists, and

    b. to arrange the statements and include information required to permit sequential processing of the statements in later phases when the appropriate I/O macros are to be generated.

3. to generate assignment and expression statements for expressions contained in some options and lists.

Phase C50 is used to perform part of the processing required for GET and PUT statements. The options FILE, STRING, PAGE, LINE, and SKIP are checked for errors. Assignment statements are generated as required.

Repetitive specifications in the data lists are checked for number and nesting depth. A DO and an END statement are generated for each repetitive specification.

Phase Input and Output

The input for the phase is the program text on TXTIN and the file table on SYS001.

Phases C55, C60, and C65 also process the program text from TXTIN and C60 and C65 the aforeementioned file table.

Program Text on TXTIN. Each syntactical element of the text string begins with an 'E'-or an 'F'-key. Elements with an E-key have a fixed length. Elements with an F-key are of variable length. Bytes 2 and 3 of an F-element indicate the length of the element.

This phase and the other I/O-scan phases (C55, C60, and C65) process elements with one of the following keys:

E0	Statement identifier
E1	Reference to declared variable
E2	Delimiter
E3	Reference to character-string constant
E4	Reference to generated variable
E9	Reference to constants other than character-string
EA	End of statement
EB	Error
EC	Reference to library names
ED	I/O-intermediate key
EF	Keyword
F0	Generated variable table
F2	Macro
F3	Constant table
F4	Declared variable table
FE	Format integer constants
FF	End of program

The text string consists of statements that appear in the same order as in the source program, except that nesting blocks are resolved, i.e., the blocks are now ordered serially.

Each statement begins with a statement-identifier key which is followed by the declared variable and/or constant table. These tables contain the attributes of the variables and the attributes and values of the constants, respectively. (See phases B90 and B92 for the format of a variable table and phase C30 for the format of a constant table.) The syntax of the statement body which follows the table(s) is described in phases A60 and A65. The end-of-statement key terminates the statement, the end-of-program key the entire text string. A statement may have one or more labels which are in the form of internal macros. Such labels may precede either the statement identifier or the statement body.

The declared variables in the text string do not appear with their actual internal name, but with their offset in the declared-variable table.

File Table on Table File. There is one record in the file table for each file name and file name parameter. The file number is identical with the record number in the file table. Each record contains the attributes and options for the appropriate file name or file-name parameter. For the format of a file-table record refer to phase B25.

Output. This phase and the other I/O scan phases cause

1. the I/O statments from TXTIN to be processed and

2. additional (generated) statements to be inserted into the input statements. The end of the I/O statement is signalled by setting bit seven in the second byte of the last end-of-statement key to 1.

To optimize the object code that is generated on account of the I/O statements, it is necessary that the inserted statements do not destroy the contents of specific registers. Preserving the contents of these registers is ensured by setting the appropriate bits in byte four of the end-of-statement key. Bits 0 to 7 correspond to registers fourteen to five.

If an error is detected during one of the I/O scan phases, bit 1 in IJKMJT is set to 1.

Initialization, Scan (General) -- XA

General control, initialization of the phase, and scanning for the I/O statements is the same for all I/O scan phases.

Text input/output is performed in overlapped mode under control of the interface (see phase A00). Five buffers are used by the I/O scan phases: one buffer as output buffer and four contiguous buffers as input buffers. During the processing of an I/O statement, the input string is always adjusted in such a manner that the next end-of-statement key is fully in the input buffer area. Thus, no further control for reaching end-of-buffer is required.

The variable and constant tables required during processing of the I/O statements are read into the table space.

After the input buffers have been filled, the statement key is checked to determine if it is an I/O key. If it is not, the statement concerned is skipped and the input buffers are filled again, if necessary.

When an I/O key is found, the statement identifier key is saved and the statement attribute table is placed into the table space, and the statement is scanned for the end-of-statement key.

Only correct statements (without T- or S-type errors) are processed. When an incorrect end-of-statement key is found, the statement is deleted from the text string except for the end-of-statement key.

Correct statements are written out by the appropriate I/O routine. The end-of-statement routine causes the errors, if any, to be indicated in the end-of-statement key and the error number to be written out.

Interface with Other Phases

The second byte of the statement-identifier key is used to pass on information about the type of the statement (used in the phases D75 and D80).

The bits are set to 1 to indicate the following:

Bit 0	PAGE option
1	SKIP option
2	LINE option
3	Statement refers to a PRINT file
4	LIST option
5	STRING option
6	GET statement
7	PUT statement

If a data list contains a repetitive specification, bit 29 in IJKMJT is set.

If standard input file is assumed, bit 54 in IJKMLB is set; if standard output file is assumed, bit 55 in IJKMLB is set.

DESCRIPTION OF ROUTINES

Note: The symbols R0, RA through RM, BASE and RETURN are references to general registers.

Symbols used in flow charts:

AT2BUTE	Position of string bits in entry of attribute table
BGRTBPO	Address of repetitive specification table
BGSTPO	Address of parenthesis stack
BIFIMSK	Mask to test for binary fixed
BLBYTE	Position of block byte in EOS key
BLPOS	Position of block byte in GENVAD
CATBYTE	Work byte to build up the GENVAR attribute byte
CONBG	Address of declared-constant table
COUNT	Count register
CSTRMSK	Test-mask for character string
CUBL	Current block number
DASPBG	Address of data specification
DO	DO delimiter element
EDATTA	End of declared-variable table
EDRTBPO	End of repetitive specification table
EDSTPO	End of parenthesis stack
ENDBUF	End of input buffers +1
EOP	End-of-program key
EOS	End-of-statement key
ERFBT	Mask for setting the error bit in IJKMJT
ERFLAG	Error flag byte
ERRTAB	Error table

FIBL	File-block area
FIDEMSK	Mask to test for fixed decimal
FILMSK	Mask for testing the file bit
FLBYTE	Position of flag byte in end-of-statement key
FNBYTE	Relative position of file-number byte in attribute table
GENVAD	Generated variable definition
GENVAR	Generated variable reference
GEPMSK	Mask to set register-preserve bits for GET/PUT
GETMSK	Mask for setting the GET bit
GEVAR	Generated-variable reference
IJKMBL	Buffer length (entry in communication area)
IJKMBS	Address of buffer area (entry in communication area)
IJKMJT	Job information bytes (in communication area)
IJKMTS	Address of table space (entry in communication area)
IJKMTT	Address of TABTAB (entry in communication area)
IJKMVC	Variable counter in interface
INBUF	Address of input buffers
INPO	Input pointer
IOMSK	Mask to set I/O bit in END-of-statement key
ISTKEY	Internal equal sign for assignment statement
KELEN	Length of key
LC85	Offset for the C85 skip bit
LEBYTE	Address of precision byte in entry of the variable table
LEDS	Length of generated variable
LEEL	Length of one E-key element
LEGEOS	Length of end-of-statement key
LEIN	Length of internal name
LELPAR	Left list parenthesis
LENGTH	Length of area to be written out
LENGTH1	Current length
LENGTHV	Maximum length of GENVAD
LEPAR	Left parenthesis
LERR	Length of error key
LETEL	Length of two E-key elements
LEVPOS	Position of level byte in EOS
LE1BYTE	Address of length byte for character string data in attribute table entry
LE2BYTE	Address of length byte for non-character string data in attribute table entry
LIBG	Begin of LINE option
LIBUF1	Address of second input buffer
LIBUF2	Address of third input buffer
LIBUF3	Address of fourth input buffer
LINMSK	Mask for setting the LINE bit
LISMSK	Mask for setting the LIST bit
MOMAC	Move-macro area
N	Counter
NATBYT	Position of scalar/array bits in entry of attribute table
NATBYTE	Address of attribute byte in entry of variable table
OUBUF	Address of output buffer
OUPO	Output pointer
PABG	Begin of PAGE option
PALMSK	Mask for setting the PAGE bit
PRINMSK	PRINT mask
PRINTMSK	Mask for setting the PRINT bit
PSTMSK	Mask to test for PUT STRING statement
PUSTMSK	Mask to test for PUT STRING statement
PUTMSK	Mask for setting the PUT bit
REGBYTE	Position of register-preserve byte in statement key
RILPAR	Right list parenthesis
RIPAR	Right parenthesis
RTBPO	Pointer for repetitive specifications
SATBYTE	Position of statement attribute byte in statement key
SAVIO	Area for saving statement identifier
SAVMSK	Mask for preserving registers
SBC85	Mask for setting the C85 skip bit
SCAMSK	Mask for testing on scalars
SEOS	Area for end-of-statement key
SKIBG	Begin of SKIP option
SKIMSK	Mask for setting the SKIP bit
SRIMSK	Mask for setting the STRING bit
STABYTE	Statement-attribute byte
STAID	Position of statement identification in statement key
STAKEY	Area used to build the statement key
STIMSK	Mask for setting the standard input-file bit
STIMMSK	Stream-input mask
STKELE	Length of statement key
STOMSK	Mask for setting the standard output-file bit
STOUMSK	Stream-output mask
STPO	Pointer for parenthesis stack
STRXBYTE	Position of structure byte in attribute-table entry
STRSAV1 (2,3)	Save area 1 (2, 3) for GEASS parameter
T	Table file
TBPO	Pointer in table space
TEBYTE	Test byte to indicate current-statement file declarations
TINPO1 (2,3)	Temporary input pointer 1 (2, 3)
TTMSK	Mask to zero bit 2 in TABTAB entry
VARBG	Address of declared-variable table
ZTAB2	Relative entry in TABTAB of file table

The statements are processed in two passes over the text string. The operations performed in each pass are described separately.

The file or the string option, if present, is the first option. If there is no file or string option, a standard system file is assumed.

If the string option is present, the option identifier is examined to determine if this is a character string. In this case, an assignment statement is generated for a subscripted variable. This assignment statement is placed into the string option for a GET statement or behind the original I/O statement for a PUT. The subscripted variable in the string option is replaced by a generated variable, which also constitutes the left or right side in the assignment statement for a GET or PUT, respectively.

If the file option is present, the appropriate file block is fetched from the table file and the file options are checked to determine if they are consistent with the type of statement.

For a file-name parameter, an internal move macro is generated. The object code generated by this macro causes the file-name argument to be inserted into the parameter list at object time.

In a PUT statement, the PAGE and/or LINE, or SKIP options may appear ahead of the data specification.

For the expression in the LINE or SKIP option, an assignment statement to a binary fixed generated variable is generated and placed ahead of the PUT statement. The expression in the option is replaced by the generated variable.

The data list is scanned for repetitive specifications. Each left-list parenthesis, except the data list parenthesis itself, indicates the beginning of a repetitive specification. The pointer value for this parenthesis is placed into a parentheses stack. On a DO following a list element, the updated input pointer, the pointer to the end of the repetitive specification (next right-list parenthesis), and the last entry in the parenthesis stack are placed into the repetitive specification table and the last entry in the parenthesis stack is cleared.

Format of a repetitive-specification table entry:

Begin of Rep.Spec.	Address of element after DO	End of Rep.Spec.
1	5	9

<u>Pass 2</u>

The input statement is written onto TXTOUT in this order: statement key, statement attribute table, internal macro generated for a file-name parameter, and PAGE (SKIP or LINE) option, if present.

The data specifications are written out with the following changes: The format list or lists in the data specification are written ahead of the data lists. For each left list parenthesis of a data list, except the first one, a DO statement is generated. The pointer to the appropriate text after the DO is fetched from the repetitive-specification table and is found by comparing the input pointer with the stacked pointer. (First address in each entry of the repetitive-specification table).

When a DO is encountered, an END is generated to close the DO group.

Each generated statement has the same format as described above. The statement attribute table is the same as that for the GET/PUT statement.

The generated variables are defined as automatic. Whenever possible, the same definition is used for successive variables. If a new definition is required (for example, if a new block is reached), the old definition is written out. This is also done at the end of the program text when, in addition, an end-of-statement key is written out following the variable definition.

<u>Initialization and Scan -- XD, XE</u>

The functions of this routine are explained under <u>Initialization, Scan (General)</u>. The following registers are used: R0, RA through RI, RL and RETURN. RF contains the pointer for the table space. RG contains the length of the declared variables or the constant table. RH contains a temporary input pointer in number of bytes:

<u>INIGP -- XF</u>

The main purpose of this routine is to test for the presence of the GET (or PUT) options and to set the bit configuration of the second byte of the statement- identifier key (see <u>Interface with other Phases</u>.) In the CONTEB subroutine, which is called by this routine, the bit configuration of the test byte TEBYTE is set. The meaning of the bits of this byte is shown below.

<u>Bit No</u>.	<u>Meaning</u>
0-2	0 = PRINT file
	1 = file other than PRINT
3-4	not used
5	0 = file option contains no file name parameter
	1 = file option contains a file name parameter
6	0 = stream input file
	1 = file other than stream input
7	0 = stream output file
	1 = file other than stream output

SCADAL1 -- XG

This routine is used to build up the repetitive specification table.

CHECKST -- XH, XI

The routine processes the GET (or PUT) options and causes them to be written out.

SCADAL2 -- XJ

This routine causes the DO statements to be rearranged and completed. Also, END statements are generated. These statements are written out on the text output file.

FEFIBL -- XK

When the subroutine is entered, INPO points to the internal file-name in the text string.

The subroutine loads the nth file block of the file table into storage (n = file number of the current file). If the name in the file option is a file-name parameter, an internal move macro is generated and the file-name parameter in the file option is replaced by an 'ED'-key with a new internal name. If the name in the file option is not a file-name parameter, the offset in the variable table of the file-name is replaced by the actual internal file-name in the text.

The internal file-name is checked for validity. If the file-name is invalid (0 as internal name), the subroutine is left via the error exit.

CONTEB -- XL

This subroutine causes the proper bit configuration to be set in TEBYTE, which is used in the CHECKST routine to determine if the current statement is consistent with the file declaration.

INPUT -- XM

This subroutine causes the input buffers to be filled as determined by the current value of the input pointer. When this subroutine is left, at least three input buffers are filled. The input pointer is adjusted.

The subroutine uses register RL as return register.

OUTPUT -- XN

Input parameters:
Register RC contains the address of the area to be written out.
Register RD indicates the length of the area.

This subroutine causes

1. the contents of an area of arbitrary length to be moved into the output buffer and

2. the data in the output buffer to be written on the text output file when the buffer is filled. The pointer in the output buffer is updated and the register pointing at the area whose contents are to be written out is increased by the length of the area.

The secondary entry points OUTPUT1, OUTPUT2, and OUTPUT3 are used to set parameters: OUTPUT1 is used if the end of the area to be written out is determined by INPO; OUTPUT2 is used if an area of the length of one 'E'-key element is to be written out; OUTPUT3 is used if an area of the length of two 'E'-key elements is to be written.

Output parameter:
Register RC contains the input address plus the length of the area that has been written.

SKISTA -- XO

This subroutine increases the input pointer until either

1. an EOS is found or

2. the sum of the input pointer and the length of the next element exceeds the upper limit of the input buffers.

SKILI -- XP

This subroutine increments the input pointer until it points to the position immediately after the list. A list is a number of elements enclosed by a pair of normal or list parentheses. When the subroutine is entered, the input pointer points to the left parenthesis of the pair.

SKIEX -- XQ

Input parameter:
The input pointer INPO points to the beginning of the current expression.

Output parameters:
```
INPO points to end of expression plus 1:
RA contains 0 if end is a comma,
            4 if end is a right-list paren-
              thesis,
            8 if end is DO.
```

This subroutine increments the input pointer until it points to the position immediately after the current expression. For the purpose of this subroutine, an expression is a string of 'E'-key elements delimited by a comma, a DO key, or a right list parenthesis which are not enclosed in a pair of parentheses.

COMOMA0 -- XR

This subroutine generates an internal move macro and inserts an 'ED'key element with a new internal name into the text string. This new name is used as the name of the constant in the parameter list into which the argument for the file-name parameter is moved at object time.

Format of the move macro:

F2	Length = 16	X'76'	Operand 1	Operand 2

Input parameter:
Register RA points to the entry for the file-name parameter in the declared variable table.

EOST -- XS

The subroutine is called when an end-of-statement is reached or a skip to the next EOS is required.

The subroutine causes the EOS to be written on the text output file and error keys to be added if any errors have been encountered during the processing of the statement. The subroutine causes the input buffers to be refilled and INPO to be reset to point to the beginning of a new statement.

If an end-of-program key is found to follow the EOS, the end-of-phase routine is called.

EOPH -- XS

This subroutine is called when an end-of-program key is reached.

The subroutine causes the definitions of the last generated variables to be written out. These definitions are followed by an end-of-statement key and the end-of-program key.

The contents of the output buffer are written out on text output file and phase C55 is fetched using the interface routine IJKAPH (calling macro IJKPH).

ERROR -- XT

Input parameter:
Register RA contains the special error key.

The errors detected during the processing of a statement are placed into an error table.

The error table is 10 bytes long. Its format is as follows:

Byte(s)	Contents
1	Error key
2	Number of errors in current statement
3-10	Special error keys

A maximum of eight errors per statement (one special error key per error) is placed into the error table for a statement.

The subroutine causes the error bit to be set in the communication routine.

GEASS, GEDOST, GEEND -- XU

Input parameters:
Register RF contains one of the following:

X'0E'	for an assignment statement
X'12'	for a DO statement
X'13'	for an END (DO) statement.

This subroutine which is referenced in the various flow charts to phase C50 by either of the names GEASS, GEDOST, and GEEND, generates either an assignment, or a DO, or an END (DO) statement as determined by the contents of RF. The prefix mask of the statement key is taken from the statement key inside which the statement is generated. The bits used to preserve registers are set by means of the bit configuration of the SAVMSK byte.

The statement attribute table for the input statement is used for all statements to be generated and inserted in the input statement, except the END statement.

The statement body for the assignment statement consists of a generated variable contained in GENVAR, an IST-key, and the expression pointed to by TINPO (begin) and INPO (end + 1). For the DO statement, the statement body consists of the repetitive specification, the beginning and end of which is found in the repetitive specification table (INPO points to the left list parenthesis of the repetitive element). No statement body is needed for the END statement.

At the beginning and end of the statement, an EOS is generated which is that of the processed (proper) statement.

The generated variable (reference) is written out whenever an assignment statement is generated. The variable is written out ahead of the assignment statement for a GET and following the assignment statement for a PUT.

OUTTAB -- XV

This subroutine causes the declared-variable and constant tables to be written out.

GEVA0 -- XW

Input parameter:
Register RG contains:

4 when the subroutine is to use the attributes of the element to which INPO is pointing,

8 when the desired attributes are binary fixed and precision 8.

This subroutine builds up both generated variable definitions (GEVAD) and generated variable references (GEVAR).

When the subroutine is entered the first time for a program, the value in the variable counter in the communication region is inserted in GEVAD and GEVAR as an internal value. The block and level numbers are obtained from the EOS of the current statement and also inserted in GEVAD.

When a new block is reached, GEVAD is written out (by calling OUTGEV) before the above described operations are performed for the new block.

When the subroutine is entered for the second and subsequent times and a new block has not yet been reached, the same internal name is used and, therefore, the same storage is used for these variables. The length count is updated to maintain a count of the highest value of all variables. The attributes of GEVAD are character string and automatic. The attributes of GEVAR are indicated by register RG.

Output parameters:
A generated variable of the following format is constructed:

E4	Intern Name	E4	Attri-butes	E4	Preci-sion	E4	Pict.name or 0

OUTGEV -- XX

This subroutine causes the generated variable definition (GEVAD) to be written out. CEIL(N/8) variables of length eight are made available for output (= maximum length required). The first variable definition is assigned the previously processed internal name, all other variable definitions are assigned an internal name of 0.

A generated variable definition made available for output has the following format:

F0	Length	Intern. Name	Zero	X'10'	X'08'	Block Level

The portion of the text string beginning with 'internal name' is repeated as often as necessary with an internal name of 0.

PHASE PL/IC55 (I/O SCAN II) -- YA

This phase processes the data lists of GET/PUT statements. Each data list item is examined for structure and validity. Assignment and expression statements are generated as required.

In the output text, each data list item is preceded by a characteristic. Some of this information is used in phases D75 and D80.

The GET/PUT statements are scanned for their data lists. The list items are scanned for the information listed below, and this information, if present, is evaluated by this phase.

1. The item is scalar, array, or structure.

2. The item either contains an operator, a subscript, a function call, or a pseudo variable, or the item requires no calculation.

3. The item contains a numeric field other than floating point or a variable with zoned decimal attributes.

4. The subsequent item is the first item of or following an iteration group. This item is referred to as "special item."

The information is summarized in the FLAG byte with the following format:

Bit No.			Meaning
0	1	=	An expression or an assignment statement must be generated.
1	1	=	Presence of an operator or a constant not allowed for input.
2	0	=	Scalar.
	1	=	Array.
3	1	=	Special item.
4	1	=	Structure.
5	1	=	Numeric field other than floating point or zoned decimal.
6	1	=	SUBSTR or UNSPEC.
7	1	=	Wrong element; neither arithmetic nor string type.

The bit configuration of the above flag byte is evaluated when the output information is processed. Invalid data items, variables not of type string or arithmetic, array and structure expressions, pseudo arrays and pseudo structures are not written out on the text output file. Structures are reduced to their basic elements and each element is written out in the same way as other single data items.

Each data item is preceded by a characteristic which has the following format:

ED	not used	Flag

An expression statement is generated whenever there is an operator in the data item or a function in a PUT statement.

An assignment statement is generated for pseudo variables in input lists, scalar numeric fields (other than float), scalar zoned-decimal variables and subscripted variables. An assignment statement for a GET is generated with the data item to the left of the equal sign; for a PUT, the data item is on the right side of the equal sign. On the other side of the equal sign, there is always a generated variable, the attributes of which are derived from the attributes of the pseudo variable or the subscripted variable. The generated variables are decimal fixed for numeric fields and zoned decimal variables.

The generated variables are written out ahead of the assignment statement in case of a GET and following the assignment statement in case of a PUT.

For a numeric field array (except float) and for a zoned decimal array, the assignment statement is preceded by a loop-begin and followed by a loop-end macro. The loop-begin and loop-end macros generate an object time loop which causes each array element to be read in or written out.

The abovementioned assignment statement consists of

1. a generated decimal-fixed variable as for a scalar numeric field and

2. a generated pointer variable with data attributes of the array that points consecutively to each array element in the loop. All other single variables and character string constants are written out unchanged.

A constant reference is expanded from three bytes to twelve bytes. The E9-key is repeated for each 3-byte group. The additional bytes have the same contents as those of the generated variables. The constant definition for the appropriate constant is also written out.

DESCRIPTION OF ROUTINES

Note: The following routines used in this phase are described in phase C50:

INPUT	EOST
OUTPUT	ERROR
SKISTA	OUTTAB
SKILI	OUTGEV
SKIEX	

Symbols used in flow charts:

ALBYTE	Length position in LOEDM
ANBYTE	Position of number of elements in attribute table
ARAMSK	Mask to test for array
ARBIT	Mask to test for array bit
ARRMSK	Mask to test for array expressions
ATBYTE	Position of attribute byte
ATPO	Pointer for attribute table
AT2BYTE	Position of string-attribute byte in attribute table
BISTMSK	Mask to set bit string in GENVAR
BLBYTE	Block byte in end-of-statement key
BLPOS	Block byte in GENVAD
CACOKEY	Key for character-constant string
CATBYTE	Byte to generate the GENVAR attribute byte
CHAR	Area to build characteristic of element
CHKMSK	Mask to check data-list item
CONAR1	Address of constant definition with table key
CONAR2	Address of constant definition without table key
CONATT	Area to build attributes and precisions for the constant reference
CONBG	Address of declared-constant table
CONLE	Length of constant element (fixed)
CONLEN1	Length of constant definition with table key and without length bytes
CONLEN2	Length of constant definition with table key and length bytes
COPO	Pointer for constant-attributes table
CORKEY	Constant reference key
CSTRMSK	Mask to test for character string
CUBL	Current block number
DARMSK	Mask to reset array bit in CATBYTE
DBEMSK	Mask to set the special-item bit in SPEMSK
DEFIMSK	Mask to set decimal fixed in GENVAR
DELMSK	Mask to zero the delete bit
EDATTA	End of variable-attributes table
ENDBUF	End of input buffer +1
EOS	End-of-statement key
EXPMSK	Mask to test whether an expression statement must be generated
FEADD	Address of entry in variable-attributes table
FLAG	Expression flag byte indicating characteristics
FLAG1	Flag byte to control output
GENVAD	Generated-variable definition
GENVAR	Generated-variable reference
GEPMSK	Mask to test for pseudo variables
GEPOR	Area for generated pointer reference
GEVAR	Generated-variable reference
IJKMBL	Buffer length (entry in communication area)
IJKMBS	Address of buffer area (entry in communication area)
IJKMTS	Address of table space (entry in communication area)
IJKMVC	Variable counter in communication area
ILBYTE	Internal-length position in attribute table
IL2BYTE	Internal-length position in character string
INBUF	Address of input buffers
INPMSK	Mask to set CHKMSK for GET
INPO	Input pointer
IOMSK	Mask to test and set I/O bit in EOS
ISTKEY	Equal sign element in assignment statement
KELEN	Length of key
LABBYTE	Position of label in LOBGM
LAEBYTE	Position of label in LOEDM
LEBYTE	Position of length byte in constant
LEEL	Length of E-key element
LEGEOS	Length of EOS key
LEIN	Length of internal name
LENGTHV1	Intermediate length of GENVAD
LENGTV	Maximum length of GENVAD
LEPAR	Left parenthesis
LETEL	Length of two E-key elements
LEV	Level of tested structure
LEVPOS	Level byte in end-of-statement key
LE1BYTE	Character-string length-byte in attribute-table entry
LE2BYTE	Length-byte in attribute-table entry for non-character-string data
LIBUF1	Address of second input buffer
LIBUF2	Address of third input buffer
LIBUF3	Address of fourth input buffer
LISMSK	Mask to test for the LIST bit
LOBGM	Area to generate loop-begin macro
LOEDM	Area to generate loop-end macro
N	Counter
NAMBYTE	Name position in LOBGM
NATBYTE	Position of attribute byte in variable-attributes table
NEBYTE	Position of number of elements in LOBGM

OUBUF	Address of output buffer
OUPMSK	Mask to set CHKMSK for PUT
OUPO	Output pointer
PICBIT	Mask to test for picture bit
PICTMSK	Mask to test for the picture bit
POBYTE	Pointer position in LOBGM
POIMSK	Mask to reset pointer in GENVAD
POINAM	Pointer-name position of entry in attribute table
PSAMSK	Mask to test for pseudo arrays
PSSMSK	Mask to test for pseudo structures
PUTMSK	Mask to test for PUT bit
RECLEV	Recursion switch
REGBYTE	Position of register-preserve byte in statement key
RILPAR	Right list parenthesis
RIPAR	Right parenthesis
SATBYTE	Position of flag byte in statement identifier
SAVIO	Statement-identifier save area
SAVMSK	Register-preserve mask
SEOS	Save area for EOS
SPEMSK	Mask to set the special item bit in the FLAG byte
SPVMSK	Mask to test for subscripted variables
SRCDMSK	Mask to set the register-preserve bits for RC and RD
SRIMSK	Mask to test for STRING
STABYTE	Save area for statement flag-byte
STAID	Position of statement identification in statement key
STAKEY	Area to build the statement-identifier key
STKELE	Length of statement key
STRMSK	Mask to test for structure expressions
STUMSK	Mask to test for structure variables
TBPO	Pointer in table space area
TINPOI(II)	Temporary input pointer I (II)
TINPO	Temporary input pointer
TINPO1-3)	Temporary input pointer 1 (2, 3)
VARBG	Address of declared-variable table

<u>INISC2 -- YB</u>

This routine initializes the phase and controls the scanning of the input text.

<u>GEPUII -- YC, YD</u>

This subroutine causes the data list items of GET/PUT statements to be evaluated and either processed and written out or written out unchanged. Erroneous data list items are skipped.

<u>SKISTAT -- YE</u>

Input parameter:
INPO points to the preceding EOS (if entry SKISTAT1 is used) or to the statement identifier key (if entry SKISTAT2 is used).

This subroutine causes a statement to be skipped and written on the text output file. Entry point SKISTAT1 is used if the preceding and the following EOS are written out, too. Entry point SKISTAT2 is used if the statement is to be written out without the EOS's.

Output parameter:
INPO points to the position following EOS (if entry SKISTAT1 is used) or to the EOS of the current statement (if entry SKISTAT2 is used).

<u>EOPH -- YF</u>

This routine is called when an end-of-program key is encountered.

Those definitions of generated variables which are not yet written out are now written out on the text output file followed by an end-of-statement key and the end-of-program key. Phase C60 is fetched by calling the interface routine IJKAPH using the calling macro IJKPH.

<u>IDEXPR -- YG</u>

Input parameter:
INPO points to the beginning of a data list item.

This subroutine tests a data list item for specific characteristics and causes appropriate indicator bits to be set in the FLAG byte. All bits of the FLAG byte, except bit 3 are set to 1 if the appropriate characteristics are encountered.

Output parameters:
INPO points behind data list item;
TINPO points to the beginning of data list item;
FEADD contains entry from variable-attributes table if first element of data list item is a variable;
FLAG contains a bit configuration that reflects the processing performed in this subroutine.

<u>TEEL -- YH</u>

Input parameter:
ATPO points to the entry in the attribute table for the variable to be tested.

This subroutine sets bits 5 and 7 of the FLAG byte to 1 if the appropriate data-item characteristics are encountered. For pointer variables, bit seven of this byte is not set to 1 because these variables may appear in a valid expression of a PUT statement.

Output parameter:
FLAG contains a bit configuration that reflects the processing performed in this subroutine.

TESTR -- YI

Input parameter:
ATPO points to a structure or to an element of a structure in the variable-attributes table.

This routine updates ATPO. The secondary entry point TESTR1 is used if, when the routine is entered, ATPO points to an element of a structure.

Output parameters:
ATPO points to the next element or a structure or to the position following the last element of the structure.
RH contains 0 if no further element of the structure is present.
RH contains 4 if the next element is found.
GEEXP1,GEASS1 -- YJ

Input parameters:
RF contains X'0F' if an expression statement is to be generated and X'0E' if an assignment statement is to be generated. This routine generates an expression or an assignment statement as determined by the contents of RF. The prefix mask of the statement key is taken from the statement key of the input statement into which the generated statement is inserted. The register-preserve bits in the statement flag byte are set by means of the SAVMSK byte.

To generate the insertion statement, the routine uses the statement-attribute table for the input statement into which the expression or assignment statement is inserted.

The statement body for the assignment statement consists of the generated variable in GEVAR and the expression pointed to by TINPO (begin) and INPO (end+1). For an expresion statement, the statement body consists of the expression only. At the beginning and end of the assignment (expression) statement, an EOS is generated. This is the EOS of the currently processed statement. If an assignment statement is generated, the generated-variable reference is written out. It is written ahead of the assignment statement for a GET and following the assignment statement for a PUT.

OUTSVC -- YK

This subroutine causes a single variable or a single constant in a data list to be written on the text output file. The address of the item to be written out is contained in TINPO; for a variable, the entry in the variable-attribute table is contained in FEADD.

In case of a pointer-type variable, the routine returns to an error call in GEPUII. Otherwise, the variable is written out as it appears in the input stream. The same is done for a character-string constant.

For a constant other than character string, the constant definition is written out as it appears in the constant-attributes table, but with the DELETE bit set off. The constant reference is written out in a format as follows:

E9	Intern. name	E9	Attri- butes	E9	Pre- cision	E9	0

OUTCOND -- YK

Input parameter:
RC contains the value of FEADD.

This subroutine generates a constant definition in the form of a constant-attribute table entry. This definition is generated for a constant, the address of which is contained in RC. The delete bit is set to 0 and the definition is written out on the text output file.

OUTSTR -- YL

Input parameter:
FEADD contains the address of the entry in the structure-attributes table.

This subroutine causes a structure to be written out element by element. Each element is tested for validity, and for each element a characteristic is built up and written out. The subroutine returns control to GEPUII for an error call if an invalid element is found.

OUTPCT -- YM

This subroutine writes out a single numeric field with attributes other than float. The address of the numeric field is contained in FEADD. The subroutine generates a variable with the attributes decimal fixed and an assignment statement.

If the numeric field is an array, a pointer is generated in addition. This pointer replaces the numeric field in the assignment statement. One each internal loop macro is generated to precede and follow the assignment statement when this is written out. Those bits in SAVMSK that have been set prior to the generation of the assignment statement are reset before control is returned from this subroutine.

OUTSBST and OUTSPV -- YN

When entered via entry point OUTSBST, the subroutine writes out pseudo variables that appear in a data list of a GET statement. For pseudo variables in a PUT statement, an expression statement is generated and control is returned immediately to the appropriate point in GEPUII.

For SUBSTR, a variable is generated with the attributes of the first argument in the substring variable. For UNSPEC, a length of 64 bits is generated. Then, an assignment statement is generated for both.

The subroutine is entered via OUTSPV for subscripted variables. A variable and an assignment statement are generated.

CONLBE -- YO

This routine generates the internal loop-begin and loop-end macros for an array, the address of which is contained in FEADD. The bits (in SAVMSK) to preserve registers RC and RD are set to 1. The generated loop-begin macro is written out on the text output file.

GEVA/GEPOI -- YP

This routine builds up both the generated variable definition (GENVAD) and the generated variable reference (GENVAR).

When this subroutine is entered the first time for a program, the value contained in the variable counter in the communication region is used as internal name and inserted in GENVAD and GENVAR. The block and level numbers are obtained from the EOS of the current statement and inserted in GENVAD.

When a new block is reached or when specified by RG, GENVAD is written out (see OUTGEV) before the above described operations are performed.

In all other cases, the same internal name is used and, hence, the same storage area is used for these variables. The length is updated to indicate the length value of the longest variable so far processed. GENVAD has the attributes CHARACTER and AUTOMATIC. The attributes of GENVAR are indicated by register RG.

If the entry point GEPOI is used, a pointer variable is generated.

Input parameter:
Register RG contains:

4	to indicate that the subroutine is to use the attribute of the element whose address is in FEADD;
8	if a pointer variable is to be generated a new GENVAD is to be used;
12	to indicate that the desired attributes are coded decimal (length must be derived from the variable whose address is in FEADD) ; and
16	to indicate a bit string of a length of 64 bits.

PHASE PL/IC60 (I/O SCAN III) -- YS, YT

In this phase, the format list of the GET/PUT statement, the FORMAT, OPEN and CLOSE statements are processed.

The format list items are checked for validity. For a remote format item, a move macro may be generated.

The OPEN/CLOSE options are checked for errors and in some cases, assignment statements are generated.

The second byte of the statement identifier key is set to X'4F' in a CLOSE and to X'8F' in an OPEN statement. This information is used by the phases D75 and D80.

The GET/PUT statement is scanned for the format list, the preceding part of the statement is written out and a checking routine is entered. After the check, a test is performed to determine if the length of the list is not greater than three buffer lengths. In that case, an error message is given and a dummy format list is written out. Otherwise, the format list is written out without further changes. The process is repeated if further format lists are present.

There is a difference between programmer-specified labels (format labels) and generated labels in the FORMAT statement. The generated labels are written out as they appear in the text input; the format labels are handled as described below.

First, the list of the FORMAT statement is tested in the same routine as the GET/PUT format list. Then, a test is performed to determine if the statement is preceded by at least one format label and if the list is not greater than three buffer lengths. If an error occurs, a message is written and the statement is deleted from the text string. Otherwise, the statement is written out until the end-of-statement key excluding the labels. The format labels are written out behind the list in the form:

byte 0	key X'ED'
bytes 1-2	internal name

The names are also summarized in the format label table which is written out with record length 32 (16 names). A test is performed to determine if no more than 127 format labels appear in a program.

The format lists of a GET/PUT and of a FORMAT statement are processed by the same checking routine. As the validity of some format items is dependent on the type of the file to which the statement belongs, an actual check is possible only for the GET/PUT statement. The statement flag byte which is used for this check is set in such a way that the check is always right for a FORMAT statement.

For control format items (except X) the statement is tested if it refers to a PRINT file.

The nesting depth of iteration groups, (i.e., an iteration factor followed by enclosed format items or by a single format item) is examined to determine if it is not greater than five or two for GET/PUT or FORMAT statements. The depth of a group containing a remote format item is examined to determine if it is not greater than two.

If A and B format items appear in a GET statement, a test is performed to determine if a field-length constant follows.

If a remote format item appears in a FORMAT statement, the processing of the statement is terminated with an error call, and the statement is deleted from the text string. In a GET/PUT statement, the label designator in a remote format item is examined first to determine if it is a constant. In this case, the label is examined if it is internal to the same block as the GET/PUT statement. For each remote format item with label variable, an internal name is reserved which replaces the name of the label variable in the remote format item. Also, a move macro is generated and written out which will effect the storing of the label variable value in a generated constant with the reserved name which is generated by the macro generation phase for a remote format item.

In the OPEN/CLOSE statement, the name in the file option is examined first to determine if it is a file name or file-name parameter.

For a file name or file-name parameter, the appropriate file block is fetched from the table file, and a test byte and a flag byte of the following format are constructed:

<u>Test byte.</u> The bits are set to 1 to indicate the following or are not used.

Bit	0-4	always zero
	5	1 if PRINT is specified
	6	zero
	7	always 1

Flag byte. The bits are set to 1 to indicate the following or are not used.

Bit	0-3	always zero
	4	1 if neither INPUT, nor OUTPUT, nor UPDATE and BACKWARDS
	5	1 if PRINT
	6	zero
	7	1 if INPUT, OUTPUT, or UPDATE is specified

If the INPUT option is present, bit 4 of the flag byte is reset. If INPUT or OUTPUT is present, bit 7 of the flag byte is converted. If PAGESIZE is present, bit 5 of the flag byte is set.

After all bits have been set, the flag and test bytes must contain the same bit configuration. Otherwise, an error message is written out. For the PAGESIZE expression, an assignment statement is generated with a generated binary-fixed variable as left side. This is inserted into the PAGESIZE option and written out followed by the generated variable. This process is repeated for each file group. The statements are written out in the sequence the input became available with assignment statements possibly included.

DESCRIPTION OF ROUTINES

Note: For the description of SKISTAT refer to phase C55. For the descriptions of the following routines, refer to phase C50:

INPUT	SKILI	OUTTAB
OUTPUT	EOST	GEVA0
SKISTA	ERROR	OUTGEV

Symbols used in flow charts:

ATBYTE	Position of attribute byte in variable table entry
AT2BYTE	Position of attribute byte in variable table entry
BLBYTE	Position of block number in variable table entry
CHAR	ED-KEY field with label name
COMOMA	Routine for constructing move macro
CONBG	Begin of declared constants table
CONTB	Routine for constructing test byte
COUNT	Counter for format labels
COVAMS	Mask for testing label constant
CUBL	Current block number
EDATTA	End of declared variables table
EDFLTA	End of format label table
ENDBUF	End of input buffers+1
EOS	End-of-statement key
EOST	End-of-statement routine
ERROR	Error handling routine
FEFIBL	Routine for reading in appropriate file block
FIBL	File block area
FILMSK	Mask for testing file name
FLBYTE	Flag byte position in EOS
FOLATA	Format label table
FORMAT	Routine for processing format statements
FORLI	Dummy format list
FORMSK	Mask for setting flag byte for format statement
GEASS3	Routine for generating assignment statements
GEPUF	Routine for processing GET/PUT statement
GEPUMS	Mask for testing GET/PUT
GEVA	Routine for generating variables
GEVAR	Generated variable (reference)
IIOKEY	ED-key
IJKMBS	Begin of buffer area; entry in communication area
IJKMBL	Buffer length, entry in communication area
IJKMTT	Begin of TABTAB, entry in communication area
IJKMTS	Begin of table space, entry in communication area
IJKMN	Interface move routine
IJKMJT	Job information bits (communication area)
IJKMVC	Variable counter, entry in communication area
IJKPO	Interface output routine
IJKPH	Interface routine for fetching a new phase
INBUF	Begin of input buffers
INPO	Input pointer
INPUT	Routine for filling input buffers
INPUT	Input Routine
IOMSK	Mask for testing I/O bit
ISTKEY	Internal equal sign for assignment statement
LABMSK	Mask for testing label variable
LABYTE	Position of attribute byte in variable table entry
LC65,LC85	Position of C65 and of C85, skip bit in communication area
LEDLM	Length of label macro
LEEL	Length of E-key element
LEGEOS	Length of end-of-statement key
LEIN	Length of internal name
LELPAR	Left list paranthesis
LETEL	Length of two E-key elements
LIBUF1	Begin of second input buffer
LIBUF2	Begin of third input buffer
LIBUF3	Begin of fourth input buffer
MOMAC	Move macro area

NAMBYTE	Position of internal name in label macro
NEWPH	Parameter for IJKPH
OPCLO	Routine for processing OPEN/CLOSE statements
OUBUF	Address of output buffer
OUPO	Output pointer
OUTPUT,1	Output routine
OUTTAB	Routine for writing out variable and constant table
OUTGEV	Routine that writes out generated variable definitions
POINAM	Position of pointer name in variable table entry
PRINTMSK	Mask for testing print files
PUTMSK	Mask for testing PUT
RILPAR	Right list parenthesis
SATBYTE	Position of statement flag byte in statement identifier key
SAVIO	Statement identifier save area
SBC65,SBC85	Skip bit for IJXC65 and for IJXC85 in communication area
SCAFO	Routine for scanning format list
SEOS	End-of-statement key save area
SKISTA	Routine for skipping to EOS or end of buffer
SKISTAT1,2	Routine for skipping and writing out statements
SKIDLI	Routine for skipping data list
SKILI	Routine for skipping lists
SKIEX3	Routine for skipping expressions
SKISTAT1	Routine for skipping statements
STAKEY	Area for building the entire statement key
STBYTE	Statement flag byte save area
STKELE	Length of statement identifier key
STRBYTE	Position of attribute byte in variable table entry
TBPO	Pointer in table space
TBYTE1	Test byte
TINPO,1,2,3,4	Temporary input pointers
TINPO1	Temporary input pointer pointing to begin of format labels
TINPO2	Temporary input pointer pointing to end of statement
TINPO3	Temporary input pointer pointing to begin of statement
TINPO4	Temporary input pointer pointing to end of format labels
VARBG	Begin of declared variable table
ZTAB18	Entry in TABTAB for format label table
ZTOUT	Routine that writes onto table file (interface macro)

Initialization and Scan -- YU - YV

The functions of this routine are the same as in phase C50.

GEPUF -- YX

This routine causes the processing of GET and PUT statements.

FORMAT -- YY

This subroutine processes FORMAT statements.

SCAFO -- YZ

This routine checks the format lists of FORMAT and GET/PUT statements. In case of an error, an error message is written out and the scan proceeds. If an R format item is detected in a FORMAT statement, the routine returns to the initialization and scan routine, and the wrong FORMAT statement is deleted from the text string.

OPCLO -- ZA

This subroutine processes OPEN and CLOSE statements.

CONTB -- ZB

The bits for INPUT, OUTPUT, UPDATE, BACKWARDS, PRINT are set in the flag and test bytes according to the declaration.

COMOM -- ZC

This routine causes a move macro for a file name parameter and a label variable in an R format item to be constructed, and an ED-key element with a new internal name in the text string to be inserted. The macro is written out onto the text output file. The new name is used as the name of the constant in the parameter list into which the argument -- corresponding to the file name parameter or current value of the label variable -- is moved at object time.

The move macro has the following format:

F2	Length= 16	X'76'	Operand 1	Operand 2

Input Parameter:
RG points to the entry of the file name parameter or label variable in the delared variables table.

SKIDLI -- ZD

A data list which may contain entire statements is skipped.

SKIEX3 -- ZE

The input pointer is set to the address behind an expression. An expression means a string of E- and F-key elements delimited by an end-of-statement key or by a right list parenthesis.

Input Parameter:
INPO points to the begin address of an expression.

Output Parameters:
INPO points to the end of an expression;
RA - zero if end is EOS, or four if end is right list parenthesis.

EOPH -- ZF

This routine is called after reaching an end-of-program key. First, the last generated variables (their definitions) are written out followed by an end-of-statement key and the end-of-program key. If format labels appeared in the program, the last record of the format-label table is written out on the table file. The output buffer is written out on the text output file. Phase C65, C85, or D00 is called via the interface routine IJKPH depending on the skip bits in IJKMJT.

GEASS3 -- ZG

In this routine, an assignment statement is generated. The statement identifier key is provided by RF; the prefix mask of the statement key is taken from the input statement in which the statement is generated. The statement attribute table is that of the statement in which the statement is built. The statement body is a generated variable (GEVAR), and IST-key, and the expression pointed to by TINPO (begin) and INPO (end + 1).

The generated variable is also written out after the end-of-statement key of the assignment statement.

Input Parameter:
RF - X'0E', key for assignment statement.

PHASE PL/IC65 (I/O SCAN IV) -- $A, $B

In this phase, all record-oriented I/O statements, i.e., READ, WRITE, LOCATE, and REWRITE and the DISPLAY statement are processed.

In the second byte of the statement identifier key, information about the special type of the statement is made available for the phases D75 and D80. The rightmost four bits of the byte contain the following:

X'00'	if	READ SET
X'01'	if	READ SET KEY
X'03'	if	READ INTO
X'04'	if	READ INTO KEY
X'06'	if	REWRITE
X'07'	if	REWRITE FROM
X'08'	if	REWRITE FROM KEY
X'09'	if	WRITE FROM
X'0A'	if	WRITE FROM KEYFROM
X'0B'	if	LOCATE SET
X'0F'	if	DISPLAY

The leftmost four bits contain the following:

0 for consecutive buffered files,
1 for consecutive unbuffered files,
2 for regional files.

Record I/O Statements -- $A

All record-oriented I/O statements are processed in one routine. For the same option appearing in any statement, the same action is taken.

The record variable is tested for validity. If the variable in the SET option is subscripted, an assignment statement is generated.

For the KEY/KEYFROM expression, an assignment statement is also generated. The statements are written out with the options in a fixed order and with included assignment statements, if any.

During processing, a FLAG and a TEST byte are constructed depending on the file declaration and the format of the statement.

The TEST byte contains the following:

```
bit 0 - 1  if READ
    2 - 1  if KEY/KEYFROM
    3 - 1  if WRITE
    4 - 1  if LOCATE
    5 - 1  if REWRITE
    6 - 1  if DIRECT
    7 - 1  if DIRECT
```

The FLAG byte contains the following:

```
bit 0 - 1  if INPUT or UPDATE or UNBUFFERED
              without INPUT, OUTPUT, UPDATE
    2 - 1  if DIRECT
    3 - 1  if OUTPUT or DIRECT/UPDATE or
              UNBUFFERED without
              INPUT/OUTPUT/UPDATE
    4 - 1  if BUFFERED OUTPUT
    5 - 1  if UPDATE
    6 - 1  if KEY/KEYFROM
    7 - 1  if FROM/INTO
```

The condition code resulting from the instruction TM FLAG, TEST must be 1.

Display (General) -- $B

In the DISPLAY statement, for an expression other than a single, unsubscripted variable that needs no conversion or other than constant, an assignment statement is generated. In addition, for a subscriped name in the REPLY option, if present, an assignment statement is generated.

The statements are written out in the sequence the input became available, i.e., with assignment statements included.

DESCRIPTION OF ROUTINES

Note: The routine SKISTAT is described in phase C55; the following routines are described in phase C50:

SKIEX	SKILI	ERROR
INPUT	EOST	OUTGEV
OUTPUT	COMOMA0	OUTTAB
SKISTA		

Symmbols used in flow charts:

ATBYTE	Relative position of attribute byte in variable table entry
AT2BYTE	Relative position of second attribute byte in variable table entry
CKREVA	Routine for checking record variable for validity
CONBG	Begin of declared constant table
CONTBR	Routine for setting test and flag bits appropriate to the file
COUNT	Field for constructing second half of statement identifier flag
CSTRMSK	Mask for testing character string
CSTMSK	Mask for constructing character string

EDATTA	End of declared variable table
ENDBUF	End of input buffers+1
EOST	End-of-statement routine
EOS	End-of-statement key
ERROR	Error handling routine
FEFIBL	Routine for reading file block from table file
FIBL	File block area
FILMSK	Mask for testing file
FLAG,2	Flag byte
GEASSR,1,2	Routine for generating assignment statements
GENVAR	Generated variable
GEVARE	Routine for generating variables
GEVAR	Generated variable reference
IJKMBS	Begin of buffer area, entry in communication area
IJKMBL	Buffer length, entry in communication area
IJKMTS	Begin of table area, entry in communication area
IJKMN	Move routine interface
IJKMVC	Variable counter in communication region
IJKPO	Interface routine for writing onto the text output file
IJKMJT	Job control bits, communication area
IJKPH	Interface routine for fetching a new phase
INBUF	Begin of input buffer
INFRAD	Holds begin of INTO/FROM
INPO	Input pointer
INPUT	Input routine
ISTKEY	Internal equal sign for assignment statement
KELEN	Length of E-key
KEYAD	Holds begin of KEY/KEYFROM option
KEYTAD	Holds begin of KEYTO option
LC85	Position of C85, skip bit in IJKMJT
LEEL	Length of E-key element
LEGEOS	Length of end-of-statement key
LENGTHV1	Intermediate length of GENVAD
LENGTHV	Maximum length of GENVAD
LEPAR	Left parenthesis
LETEL	Length of two E-key elements
LIBUF1	Begin of second input buffer
LIBUF2	Begin of third input buffer
LIBUF3	Begin of fourth input buffer
MOMAC	Area for constructing move macro
N	New internal name
NATBYTE	Relative position of attribute byte in variable table entry
NEWPH	Parameter for IJKPH
OTSEKT	Routine for writing out SET and KEYTO option
OUBUF	Address of output buffer
OUPO	Output pointer
OUTGEV	Routine for writing out generated variable definitions
OUTPUT,1,2,3	Entries in output routines
OUTTAB	Routine for writing out declared variable and constant table
PICBIT	Mask for testing numeric fields
POIMSK	Mask for constructing pointer
PTMSK	Mask for testing pointer
REW	Parameter for IJKPH
RILPAR	Right list parenthesis element
RIPAR	Right parenthesis element
SAVIO	Statement-identifier key save area
SBC85	Skip bit for C85
SEOS	EOS save area
SETAD	Holds begin of SET
SETID	Area for constructing the assignment statement key
SKISTAT2	Routine for skipping statements
SKISTA	Routine for skipping to EOS or end of buffer
SKILI	Routine for skipping lists
SKIEX	Routine for skipping expressions
STKELE	Length of statement identifier key
STRSAV1-6	Save areas for generated variable and expression for constructing assignment statement
STRBYTE	Offset of file identification byte in variable table entry
SWITCH	Parameter for IJKPH
TBPO	Declared variable and constant table pointer
TEBYTE	Test byte
TINPO,1,2,3,5	Temporary input pointers
VARBG	Begin of declared variable table

<u>Initialization, Scan -- $C , $D</u>

The functions of this routine are the same as in phase C50.

<u>Record-Oriented I/O Routine -- $E - $G</u>

See section <u>Record I/O Statements</u>.

<u>CKREVA -- $H</u>

This routine tests the record variable for validity. The record variable must not have the attribute DEFINED, and must not be a parameter, nor an entry name; the record variable must be a level 1 variable. Structures must begin on double-word boundaries. The length of the record variable must not be greater than the block size of the appropriate file. If the appropriate file has variable length records, the length of the record variable divided by 8 must yield a remainder of four.

If the appropriate file has fixed-length records, the length of the record variable must be equal to record size.

Input Parameters:
INPO must point to the record variable.
FIBL contains the appropriate file block.

OTSEKT -- $I

In this routine, the SET and the KEYTO option parameter are written out.

If the variable is subscripted, a variable with the same attributes is generated except the storage class which is static. This generated variable is written out as option parameter. The position of the original variable and the generated variable are saved and used to construct an assignment statement at the end of the statement.

Input Parameter:
RG: if four, routine entered from SET option;
if eight, routine entered from KEYTO option.

CONTBR -- $J

In this routine, all bits in the TEST and FLAG byte and in the first half of the statement-identifier flag byte are set according to the appropriate file declaration.

Input Parameter:
FIBL contains the appropriate file block.

Output Parameters:
Some bits in TEST, FLAG, and SAVIO.

DISPLAY -- $K

See section Display (General).

GEVARE -- $L

In this routine, both the generated variable definition (GENVAD) and the generated variable reference (GENVAR) are created.

If the name field in GENVAD is zero, an internal name is fetched from the variable counter in the communication region and inserted into GENVAD and GENVAR. If RG is zero, the variable created earlier is written out at the beginning of the routine and the variable just being created is written out at the end of the routine. Otherwise, the same internal name is used. Thus, the same amount of storage will be used for these variables. The length is updated to hold the greatest value of all variables.

GENVAD has the attributes character string and static. The attributes of GENVAR are specified by RG.

Input Parameter:
RG if zero or eight, a character string is generated, the length of which is specified in KEYLEN
if four, a pointer is generated.

The output parameters GENVAR and GENVAD have the following format:

E4	Int. Name	E4	Attr.	E4	Prec.	E4	Zero

GENVAR

F0	Length	Int. Name	00	80	00	10	08	00

GENVAD

The part of GENVAD beginning with Internal Name is repeated as often as needed with the internal name = zero.

GEASSR, GEASSR1 and 2 -- $M

In this routine, an assignment statement is generated. The prefix mask of this statement key is taken from the input statement in which the assignment statement is generated. The statement attribute table is taken from the input statement in which the assignment statement is built.

The statement body for GEASSR consists of a generated variable contained in GENVAR, an IST-key, and an expression pointed to by TINPO and INPO. In GEASSR2, the order of the generated variable and the expression is changed.

In GEASSR 1, the statement body consists of an expression pointed to by either STRSAV3 and STRSAV2, or STRSAV6 and STRAV5, respectively, an IST-key, and a generated variable, located in either STRSAV1 or STRSAV4.

At the beginning and at the end of the statement, an EOS is generated which is that of the processed statement and which must be correct. The first EOS written out by GEASSR1 has the I/O bit inserted.

EOPH -- $N

This routine is called when an end-of-program key is reached. First, the last generated variables, i.e., their definitions, are written out followed by an end-of-statement key and the end-of-program key. The output buffer is written out on the text output file.

Phase C85 or C95 is called via the interface routine IJKPH as determined by IJKMJT.

This phase is called if the source program contains DO, GET, or PUT statements. It replaces certain DO statements and END (of group) statements with macros and other statements, which are then processed in a subsequent phase.

The DO statements processed in this phase are of the following form:

1. DO scalar = C1 TO C2;
   DO scalar = C1 TO C2 WHILE (expression4);
   DO scalar = C1 TO C2 BY C3;
   DO scalar = C1 TO C2 BY C3 WHILE (expression4);
   DO scalar = C1 BY C3 TO C2;
   DO scalar = C1 BY C3 TO C2 WHILE (expression4);

   C1, C2, C3 must be constants (either binary fixed or decimal fixed with scale factor 0). The scalar must be binary fixed. The decimal fixed constants are converted to binary fixed to minimize object time conversions.

2. DO scalar = expression 1 TO expression 2 BY C3;
   DO scalar = expression 1 TO expression 2 BY C3 WHILE (expression4);

   C3 must be an (optionally signed) arithmetic constant.

The END (of group) statements processed in this phase are those associated to DO statements listed above.

Phase Input and Output

The input is a string of 3-byte elements and/or elements of variable length.

The output is similar to the input, except that macros and new statements replace DO statements and END (of group) statements. Substituted statements and macros are:

- The IFFALSE statement generated when a TO and/or WHILE occurs in a specification. (See description of phase C25 for more details.)

- The Begin of DO-Head statement, and the End of DO Head statement. These two statements indicate that all statements and macros included by them belong to one DO-Head. Since the only function of these statements is to indicate the beginning or ending of a DO Head, no statement body is required. The format of these two statements is as follows:

Byte(s)	Contents
1	statement identifier key X'E0'
2-3	X'0001' for Begin of DO Head X'0101' for End of DO Head
4-6	not relevant
7	EOS key X'EA'
8-12	not relevant

- The Define Label macro,

- The Branch macro, and

- Generated label constants. (For details, see description of phase C25.)

Generated temporaries (generated variables with unknown attributes) are used to hold the 'frozen' values of expression 1 and expression 2 (see PL/I Language Specifications).

These temporaries are not defined by macros like generated labels and generated label variables, but solely by their occurence in a statement referencing them. Storage for these temporaries is assigned in subsequent phases. The format of generated temporaries is as follows:

Byte(s)	Contents
1	key X'E8' for generated temporaries
2-3	X'0000' for expression 1 (1T in examples) X'000C' for expression 2 (2T in examples)

Examples for Input/Output of Phase C85

Legend to the examples:

Statements (as opposed to macros) have statement identifiers consisting of capital letters (for instance: IFFALSE, SET, READ etc.)

Macros are identified by lower case letters (for instance: define label, branch etc.)

Generated labels are written like 1L, 2L, 3L etc.

Note: The input as well as the output is a string of 3-byte elements and/or elements of variable length.

```
1.  DO I=C1 TO C2;          BEGIN OF DO-HEAD;
    .                       SET I=C1;
    .                       branch 1L
    alpha                   define label 2L
    .                       SET I=I+1;
    .                       define label 1L
    .                       IFFALSE 3L I<=C2;
    END;                    END OF DO-HEAD;
                            .
                            alpha
                            .
                            branch 2L
                            define label 3L

2.  DO I=C1 TO C2           BEGIN OF DO-HEAD;
    WHILE ex4;              SET I=C1;
    .                       branch 1L
    .                       define label 2L
    alpha                   SET I=I+1;
    .                       define label 1L
    .                       IFFALSE 3L I<=C2;
    .                       IFFALSE 3L ex4;
    END;                    END OF DO-HEAD;
                            .
                            alpha
                            .
                            branch 2L
                            define label 3L

3.  DO I=C1 TO C2           BEGIN OF DO-HEAD;
    BY C3;                  SET I=C1;
    .                       branch 1L
    alpha                   define label 2L
    .                       SET I=I+C3;
    .                       define label 1L
    END;                    IFFALSE 3L I<=C2;*
                            END OF DO-HEAD;
                            .
                            alpha
                            .
                            branch 2L
                            define label 3L

4.  DO I=C1 TO C2           BEGIN OF DO-HEAD;
    BY C3                   SET I=C1
    WHILE ex4;              branch 1L
    .                       define label 2L
    alpha                   SET I=I+C3;
    .                       define label 1L
    .
    END;                    IFFALSE 3L ex4;
                            END OF DO-HEAD;
                            .
                            alpha
                            .
                            branch 2L
                            define label 3L

5.  DO I=ex1 TO ex2         BEGIN OF DO-HEAD;
    BY C3;                  SET 1T=ex1;
    .                       SET 2T=ex2;
    alpha                   SET I=1T;
    .                       branch 1L
    .                       define label 2L
    END;                    SET I=I+C3;
                            define label 1L
                            IFFALSE 3L I<=2T;*
                            END OF DO-HEAD;
                            .
                            alpha
                            .
                            branch 2L
                            define label 3L

6.  DO I=ex1 TO ex2         BEGIN OF DO-HEAD;
    BY C3                   SET 1T=ex1;
    WHILE (ex4);            SET 2T=ex2;
    .                       SET I=1T;
    alpha                   branch 1L
    .                       define label 2L
    .                       SET I=I+C3;
    END;                    define label 1L
                            IFFALSE 3L I<=2T;*
                            IFFALSE 3L (ex4);
                            END OF DO-HEAD;
                            .
                            alpha
                            .
                            branch 2L
                            define label 3L
```

*If C3 is positive, I <= C2 is used.
If C3 is negative, I >= C2 is used.

Phase Performance

The input stream is scanned for DO statements and END (of group) statements. All other statements are bypassed and put out unchanged. If a DO statement is encountered, the type of the statement is tested. If the statement is to be processed by this phase, a 1 is entered into STACK and the statement is processed. If the statement is to be bypassed, a 0 is entered into STACK. The pointer to STACK is incremented by 1.

If an END (of group) statement is encountered, the pointer to STACK is decremented by 1, and the last entry in STACK is tested. If this entry is a 1, the statement is processed. If this entry is a 0, the statement is bypassed.

The statement attribute tables of the statements processed in this phase are stored in the table space for later use. The appropriate table stored in the table space will be attached at the beginning of each statement. Note that the macros generated in this phase are _not_ prefixed by a table.

Tables and Pointers

STACK (with pointer STAP) consists of 15 elements, each 1 byte long. A=1 in such an element means that the associated DO statement has been processed and that the current END (of group) statement must be processed. A=0 means: bypass this statement.

KELLER (with pointer KEP) consists of 15 elements, each 1 word long. The first half-word of each element contains a 'start label', the second half-word contains an 'exit label'. (A "start label" is the generated label 2L in I/O examples 1 to 4; an "exit label" is the generated label 3L in I/O examples 1 to 4.) The information stored in this stack is used when processing END (of group) statements.

PIN, IPOINT, POINT are input pointers
POUT, OPOINT are output pointers

DESCRIPTION OF ROUTINES

(Open)	A routine is called open if control is transferred to it by 1. a simple B instruction, in which case control is also returned by a B instruction, or 2. some in-line coding that requires a separate description.
(Closed)	A routine is called closed if control is transferred to it by a BAL instruction. Control is returned by a BR instruction in this case.

R0, R1,.. Rn: symbolic registers.

<u>DOPH -- $P</u>

This is the "master" routine of this phase. DOPH sets pointers, loads registers, etc. and reads the first 4 records into input buffers 1 to 4.

DOPH scans and puts out the input stream until a specific DO statement or an associated END statement is encountered. In this case, the statement attribute table is stored in the table area. If DOPH encountered a DO statement, ANDOST is called. ANEND is called if DOPH encountered an END (of group) statement. Before ANEND is called, the statement identifier END (of group) is replaced by the statement identifier NOP. The scan continues until the end-of-program key is encountered.

<u>ANDOST (Open) -- $Q - $S</u>

This routine processes specific DO statements. The code put out by this routine is described in the I/O examples.

If a DO statement contains errors or if the table space is not large enough to hold additional entries to the attribute table, the statement is passed on unchanged to the next phase.

<u>ANEND (Open) -- $T</u>

ANEND is called only if the associated DO statement has been processed in this phase. It decrements KEP by 4 and puts out the macros

```
    branch        'Start Label'
    define label  'Exit Label'
```

'Start Label' is taken from 0(KEP).
'Exit Label' is taken from 2(KEP).

<u>BSAC (Open) -- $T</u>

The routine stores the statement attribute table for variables and constants in the table area.

<u>BYPA (Closed) -- $U</u>

The routine stores either the statement attribute table for variables or the statement attribute table for constants in the table area. Upon exit, BSAC7 contains the address of the next unoccupied byte in the table area, the statement body begins in input buffer 1, and BSAC6 contains the address of the first byte of the attribute table for constants in the table space.

<u>COSC (Closed) -- $V</u>

The routine determines whether an expression in a DO statement consists of a single, optionally signed BINARY FIXED constant or of a DECIMAL FIXED constant with a scale factor of 0. If the expression is of any other type, COSC branches to UNSUC. If one of the above specified expressions is encountered, all prefix operators (+ and -) are reduced to one. Example: + --- + results in -.

If the expression is a BINARY FIXED constant, the corresponding attribute table entry is stored in ENTRI, PIN is incremented by 3, and the program returns.

If the expression is a DECIMAL FIXED constant with a scale factor of 0, the expression is tested for being greater than 2147483647 (= 2**31 - 1). If yes, the program branches to UNSUC. If no, OLD is set to 1, the DECIMAL FIXED constant is converted to BINARY FIXED, a new attribute table entry is created in ENTRI, PIN is increased by 1, and the program returns. Upon return, PIN points to the next byte following the constant.

<u>ENDX (Closed) -- $W</u>

Upon entry, R1 contains the start address of an expression. Upon return, R1 contains the end address of an expression.

EOST, JEOSA1 (Closed) -- $Y

The routine arranges the contents of input buffers 1 to 4 so that the currently scanned EOS is in input buffer 1 (this is done by moving and by reading in new records). It puts out the EOS and the error codes attached to it. Any additionally generated error codes are also put out.

ERROR, JERRA1 (Closed) -- $X

This routine is described in phase A35.

GEOS (Closed) -- $W

The routine moves the input pointer PIN until the last byte of the statement body is reached. It stores the value of PIN in IFPH96.

GSN (Open) -- $W

This routine checks for error-free statements. If the bit checked is on, the statement contains an error and the routine returns without any further action. If the bit is off, the end-of-statement delimiter is stored in GSN4 and the routine returns.

JTRNA1 (Closed) -- $Z

This is the output routine. Register BYZ contains the number of bytes to be put out; register PIN contains the start address of these bytes. One output buffer is used. If the string to be put out fits into the remaining unoccupied space of the output buffer, the string is moved into the buffer. BYZ is added to POUT to update the output pointer.

If the string to be put out it too big, an appropriate part of the string is moved to fill the output buffer to its capacity. Then the contents of the buffer are written onto the output medium. POUT is reset to the start address of the buffer. BYZ is decremented by the number of bytes moved into the buffer. PIN is incremented by that number. Then JTRNA1 is repeated until output is completely accomplished.

LGEN (Closed) -- $X

LGEN generates a label constant of the following format:

byte 0	key for generated label constant X'EE'
bytes 1-2	number of the constant

The number in bytes 1 and 2 of the label constant is obtained by adding 1 to the counter IJKMVC each time LGEN is called. Upon exit, the generated label constant is stored right justified in R1.

IJKMVC is the "variable and constant counter" of the compiler. If its value exceeds 65534, an error is indicated.

STEP (Closed) -- $X

STEP tests the high-order 4 bits of the byte selected by PIN. If these 4 bits are set to X'E', PIN is incremented by 3. If these bits are set to X'F', PIN is incremented by the contents of the two bytes following the byte to which PIN is pointing. If these bits are set to any other value, a compiler error occurrs and a dump is initiated.

This phase is called if the source program contains at least one DO, GET, or PUT statement. All DO and END (of group) statements, bypassed by the first DO phase, are now processed.

Phase C86 performs the following functions:

- It analyzes all DO nests,
- replaces all DO statements and END (of group) statements by macros and other statements which are then processed by subsequent phases, and
- checks whether the restrictions on the nesting of DO statements and on the number of repetitive specifications are obeyed.

Phase Input and Output

The input is a string of 3-byte elements and/or elements of variable length. The complete DO or END (of group) statement body must be available in the input buffers.

The output is similar to the input, except that macros and new statements are substituted for DO and END (of group) statements. The following new statements may be substituted:

- The IFFALSE statement.
  This statement is generated whenever a TO or WHILE occurs in a specification. IFFALSE is discussed in a subsequent section of this publication.
- The "begin of DO head" statement.
- The "end of DO head" statement.
  These two statements indicate that all statements and macros included by them are associated and thus belong to one "DO head". These statements require no statement body, because they only signal the beginning or ending of a "DO head". The format of these statements is as described in phase C85.
- Assignment statement with special operands.
  If a DO statement contains more than 1 repetitive specification, an assignment statement as shown below is generated. The only difference between a label assignment written by a programmer and one generated by the DO phase is that in the generated label assignment two operands are followed by the 3-byte element X'EE0009' or X'EE0069'. The format of the assignment statement is as follows:

Byte(s)	Contents
1-3	statement identifier X'E0000E'
4-6	not relevant
7	key X'EE' for generated label
8-9	name of generated label
10-12	X'EE0009' indicating a generated label variable
13-15	3-byte element "=": X'E200FB'
16	key X'EE' for generated label
17-18	name of generated label constant
19-20	X'EE0069' indicating a generated label constant
22-27	EOS delimiter X'EA....'

- The 'Define Label' macro
- The 'Branch' macro
- The 'DO-Branch' macro
  This macro is generated if a DO statement contains more than one repetitive specification. The 'DO-branch' macro initiates a branch to the address contained in the generated label variable nV. The format of this macro is as follows:

Byte(s)	Contents
1	macro key X'F2'
2- 3	length of macro
4	X'81' for DO branch
5	X'E4' for generated label variab
6- 7	name of generated label variable
8-10	not relevant

- Generated label constants.
- Generated label variables.
  The label variables are used to hold the values of generated label constants. To allocate storage (8 bytes for 1 generated label variable), the macro 'Define Generated Variable' is generated as follows:

Byte(s)	Contents
1	key X'F0' for "define generated label variable"
2- 3	length (overall)
4- 5	name of generated label variable
6-10	attributes of the generated label variable (arranged as in the statement attribute table)
11	bits 0-1: level number bits 2-7: block number

- Generated temporaries (i.e. generated variables with unknown attributes). These temporaries are used to hold the "frozen" values of expression 1, expression 2 and expression 3 (see PL/I Language Specifications, Form C28-6809. Generated temporaries are not defined by special macros like generated labels and generated label variables. They are defined only by their occurrence in a statement referencing them. Storage for generated temporaries is assigned in later phases. The format of the generated temporaries is as follows:

Byte(s)	Contents
1	key X'E8'
2	X'00'
3	X'00' for expression 1 (1T in examples) X'0C' for expression 2 (2T in examples) X'18' for expression 3 (3T in examples)

- Specification separator. Specification separators are used to separate 2 repetitive specifications and consist of 2 macros as shown below:

X'FD0004E5'	macro 1
X'FD0004D5'	macro 2

Examples for the Input to the DO Phase and the Produced Output

Legend to the examples:

- Statements have statement identifiers consisting of capital letters ,e.g., IFFALSE, SET, READ etc.
- Macros are identified by lower case letters, e.g., define label, branch, etc.
- Generated labels are of the form 1L, 2L, 3L, etc.
- Generated label variables are written like 1V, 2V, etc.
- Generated temporaries (variables with unknown attributes) are of the form 1T, 2T, 3T.

Note that in reality the input as well as the output is a string of 3-byte elements and/or elements of variable length.

```
    Input            Output

 1. DO;              BEGIN OF DO-HEAD;
    .                NOP;
    .                END OF DO-HEAD
    alpha            .
                     .
    .                alpha
    END;

 2. DO I=exl;        BEGIN OF DO-HEAD
    .                SET I=exl;
    .                END OF DO-HEAD
    alpha            .
    .                alpha
    .                .
    END;

 3. DO WHILE ex4;    BEGIN OF DO-HEAD
    .                define label 1L
    alpha            IFFALSE 2L ex4;
    .                END OF DO-HEAD;
    END;             alpha
                     branch 1L
                     define label 2L

 4. DO I=ex1         BEGIN OF DO-HEAD;
    WHILE ex4;       SET I=ex1;
    .
    .
    alpha            IFFALSE 1L ex4;
    .                END OF DO-HEAD;
    END;             .
                     alpha
                     .
                     define label 1L

 5. DO I=ex1         BEGIN OF DO-HEAD;
    BY ex3;          SET 1T=ex1;
    .                SET 3T=ex3;
    alpha            SET I=1T;
                     branch 1L
                     define label 2L
    END;             SET I=I+3T;
                     define label 1L
                     END OF DO-HEAD;
                     .
                     alpha
                     .
                     branch 2L

 6. DO I=ex1         BEGIN OF DO-HEAD;
    BY ex3           SET 1T=ex1;
    WHILE ex4;       SET 3T=ex3;
    .                SET I=1T;
    .                branch 1L
    alpha            define label 2L
    .                SET I=I+3T;
    .                define label 1L
    END;             IFFALSE 3L ex4;
                     END OF DO-HEAD;
                     .
                     alpha
                     .
                     branch 2L
                     define label 3L

 7. DO I=ex1         BEGIN OF DO-HEAD;
    TO ex2;          SET 1T=ex1;
    .                SET 2T=ex2;
    alpha            SET I=1T;
    .                branch 1L
    .                define label 2L
    .                SET I=I+1;
    END;             define label 1L
```

```
                  IFFALSE 3L I<=2T;
                  END OF DO-HEAD;
                  .
                  alpha
                  .
                  branch 2L
                  define label 3L

 8. DO I=ex1      BEGIN OF DO-HEAD;
    TO ex2        SET 1T=ex1;
    WHILE ex4;    SET 2T=ex2;
    .             SET I=1T;
    .             branch 1L
    alpha         define label 2L
    .             SET I=I+1;
    .             define label 1L
    END;          IFFALSE 3L I<=2T;
                  IFFALSE 3L ex4;
                  END OF DO-HEAD;
                  .
                  alpha
                  .
                  branch 2L
                  define label 3L

 9. DO I=ex1      BEGIN OF DO-HEAD;
    TO ex2        SET 1T=ex1;
    BY ex3        SET 2T=ex2;
    .             SET 3T=ex3;
    alpha         SET I=1T;
    .             branch 1L
    .             define label 2L
    END;          SET I=I+3T;
                  define label 1L
                  IFFALSE 3L (3T>=0) &
                       (I<=2T) | (3T<0) &
                       (I>=2T) ;
                  END OF DO-HEAD;
                  .
                  alpha
                  .
                  branch 2L
                  define label 3L

10. DO I=ex1      BEGIN OF DO-HEAD;
    TO ex2        SET 1T=ex1;
    BY ex3        SET 2T=ex2;
    WHILE ex4;    SET 3T=ex3;
    .             SET I=1T;
    .             branch 1L
    alpha         define label 2L
    .             SET I=I+3T;
    END;          define label 1L
                  IFFALSE 3L (3T>=0) &
                       (I<=2T) | (3T<0) &
                       (I>=2T) ;
                  IFFALSE 3L ex4;
                  END OF DO-HEAD;
                  .
                  alpha
                  .
                  branch 2L
                  define label 3L

11. DO I=ex1,     BEGIN OF DO-HEAD;
    ex2, ex3;     define 1V
    .             SET 1V=1L;
    .             SET I=ex1;
    alpha         branch 2L
    .             define label 1L
    .             specification separator
    END;          SET 1V=3L;
                  SET I=ex2;
                  branch 4L
                  define label 3L
                  specification separator
                  SET 1V=5L;
                  SET I=ex3;
                  branch 6L
                  define label 6L
                  define label 4L
                  define label 2L
                  END OF DO-HEAD;
                  .
                  alpha
                  .
                  DO-branch 1V
                  define label 5L

12. DO I=ex1      BEGIN OF DO-HEAD;
    TO ex2 BY ex3 define 1V
    ex4 TO ex5    SET 1V=1L;
    BY ex6;       SET 1T=ex1;
    .             SET 2T=ex2;
    .             SET 3T=ex3;
    alpha         SET I=1T;
    .             branch 3L
    .             define label 1L
    END;          SET 1=I+3T;
                  define label 3L
                  IFFALSE 2L (3T>=0) &
                       (I<=2T) | (3T<0) &
                       (I>=2T) ;
                  branch 4L
                  define label 2L
                  specification separator
                  SET 1V=5L;
                  SET 1T=ex4;
                  SET 2T=ex5;
                  SET 3T=ex6;
                  SET I=1T;
                  branch 7L
                  define label 5L
                  SET I=I+3T
                  define label 7L
                  IFFALSE 6L (3T>=0) &
                       (I<=2T) | (3T<0) &
                       (I>=2T) ;
                  branch 8L
                  define label 8L
                  define label 4L
                  END OF DO-HEAD;
                  .
                  alpha
                  .
                  DO-branch 1V
                  define label 6L
```

FUNCTIONAL DESCRIPTION

The input stream is scanned for DO and END (of DO group) statements. All other statements are bypassed and put out unchanged. If a DO statement is encountered, its statement attribute table and its constant table is stored in the table storage for later use. The DO statement is analyzed and statements and/or macros are put out, depending on the structure of the DO statement (see examples). The attribute and constant tables are added to each generated statement. Macros generated in this phase are not prefixed by these tables.

If an END (of DO group) statement is encountered, the type of code being generated depends on the structure of the corresponding DO statement. The required information is stored in 5 push-down stacks. The two stacks AND05 and AND06 have a capacity of one element per level. The remaining three (AND01, AND02, and AND03) have a capacity of more than one element per level. In each push-down stack, the element size is one half-word. The stack pointers are the symbolic registers R6 and R7.

AND01
(stack pointer R7) contains 'start labels'. (In examples 5 to 10, '2L' is a start label).

AND02
(stack pointer R7) contains 'exit labels'. An exit label is the generated label of a statement to which control is transferred after the execution of the DO group has been terminated. (In examples 6 to 10, '3L' is an exit label.)

AND03
(stack pointer R7) contains "G-labels". G-labels are generated only if the DO statement contains more than one specification. A G-label is a generated label to which a branch is directed when the statements representing one specification have been executed. (In example 12, '4L' and '8L' are G-labels.)

AND05
(stack pointer R6) contains generated label variables. (In examples 11 and 12, "1V" is a generated label variable.)

AND06
(stack pointer R6) is used to store the number of specifications per DO statement.

PIN, IPOINT, POINT are input pointers (symbolic registers); POUT, OPOINT are output pointers (symbolic registers).

DESCRIPTION OF ROUTINES

(Open)	A routine is called open if control is transferred to it by 1. a simple B instruction, in which case control is also returned by a B instruction, or 2. some in-line coding that requires a separate description.
(Closed)	A routine is called closed if control is transferred to it by a BAL instruction. Control is returned by a BR instruction in this case.
R1, R2,...	are symbolic registers.

<u>DOPH -- AA</u>

This is the "master program" of this phase. DOPH initializes pointers, registers, etc. and reads the first 4 records into input buffers 1, 2, 3, and 4. It scans and puts out the input stream. If a DO or END (of group) statement is encountered, the statement attribute table is stored in the Table Area. ANDO is called if a DO statement has been encountered, or ANEN if an END (of group) statement occurred. (See description of <u>ANDO</u> and <u>ANEN</u>.) Before ANDO is called, a 'Begin of DO head' statement is put out. Before ANEN is called, the statement identifier END (of group) is replaced by the statement identifier NOP.

The scan is continued until the end-of-program key is encountered.

<u>ANDO -- AB, AC, AD (Open)</u>

Analyzes and processes DO statements. If the DO statement contains errors, a NOP statement is put out, EOST is called, and control returns to DOPH which continues the scan.

If the DO statement is correct, a series of macros and statements is generated (see examples). The attribute table stored in the table area is attached to each generated statement.

Error messages are produced if the DO nest is deeper than 12, and if there are more than 50 specifications in one DO nest.

<u>ANEN -- AE (Open)</u>

Puts out macros and statements, depending on the structure of the corresponding DO statement. (See examples.)

<u>BSAC -- AG (Open)</u>

This routine stores the statement attribute table and the statement constant table in the table area.

BYBY -- AO (Closed)

This routine is called only if the specification contains a BY. It generates the following:

```
  SET 3T = expression 3;
  SET SCALAR = 1T;
  branch NL
  define label ML
  SET SCALAR = SCALAR + 3T;
  define label NL
```

BYPA -- AH (Closed)

Stores either the statement attribute table for variables or the statement attribute table for constants in the Table Area. Upon exit, BSAC7 contains the address of the next free byte in the Table Area, and the statement body begins in input buffer 1.

CON -- AK (Closed)

CON may be considered as an entry to LGEN.

Generates a 'name' for constant 0 or 1 and puts it into R1. The 'name' is a 3-byte element. The first byte of this element contains X'E9'; the subsequent two bytes contain the number of the constant.

Retrievement of the constant number is discussed in the description of LGEN.

ERROR, JERRA1 -- AS (Closed)

This routine is described in phase A35.

EOST, JEOSA1 -- AR (Closed)

Arranges the contents of input buffers 1 to 4. The currently scanned EOS is moved to input buffer 1. This is accomplished by moving and by reading in new records. Puts out EOS and the error codes attached to it. Any additionally generated error codes are also put out.

GEOS -- AI (Closed)

Moves the input pointer PIN until the last byte of the statement body is reached. Stores the value of PIN in IFPH96.

GSN -- AJ (Open)

This routine is described in phase C85.

INC1 -- AL (Closed)

INC1 is called only if the specification contains a TO clause but not a BY clause. It generates the following:

```
  SET scalar = 1T;
  branch nL
  define label mL
  SET scalar = Scalar + 1;
  define label nL
  IFFALSE oL scalar < = 2T;
```

Generates an entry for the constant 1 in the statement attribute table for constants.

JTRNA1 -- AQ (Closed)

Output routine. Register BYZ contains the number of bytes to be put out, register PIN the starting address. One output buffer is used.

If the (remaining portion of the) string to be put out fits into the remaining unoccupied space of the output buffer, the string is moved into this space. BYZ is added to POUT to update the output pointer.

If the string to be put out is too big, the output buffer is filled to capacity by a part of the string, and the contents of the buffer are written onto the output medium. POUT is reset to the start address of the buffer. BYZ is reduced by the number of bytes moved into the buffer. PIN is incremented by that number. Then JTRNA1 is repeated until the output is completely accomplished.

KRAFT -- AM (Closed)

```
Puts out:  SET Scalar = 1T;
           branch nL
           define label mL
```

LGEN -- AK (Closed)

Generates a label constant of the following format:

byte 0	X'EE', key for generated label constant
bytes 1-2	number

The number in bytes 2 and 3 of the constant is obtained by adding 1 to the counter IJKMVC each time the routine is called. Upon exit, the generated label constant is stored right-justified in R1. IJKMVC is the "variable and constant counter" of the compiler. If the value of IJKMVC exceeds 65534, an error is indicated.

LVGE -- AK (Closed)

LVGE may be considered as an entry to LGEN.

The routine generates a label variable of the following format:

byte 0	X'E4', key for generated label variable
bytes 1-2	number

The number in bytes 2 and 3 of the label variable is obtained by adding 1 to the counter IJKMVC each time the routine is called. Upon exit, the generated label variable is stored right-justified in R1. IJKMVC is the "variable and constant counter" of the compiler. If the value of IJKMVC exceeds 65534, an error is indicated.

STEP -- AF (Closed)

The high-order 4 bits of the byte pointed to by PIN are tested. If these bits contain X'E', PIN is incremented by 3. If they contain X'F', PIN is incremented by the contents of the two bytes following next. If these bits contain any other value, a compiler error has occurred and a dump is initiated.

TTS1 -- AP (Closed)

Puts an entry for the decimal fixed constant 1 into the statement attribute table for constants. If no available space is found in this table, an error message is given and the processing of the current statement is terminated.

WHY -- AN (Closed)

Called only if a WHILE appears in a specification. The routine puts out IFFALSE mL expression4;

PHASE PL/IC95 (NEW INTERFACE) -- AV

This phase initiates the new interface used by the phases D00, D05, and D10. This interface has only some of the capabilities of the main interface (see phase A00) and, therefore, is shorter. The storage saved is used by the phases using this interface.

Both the old interface, which is located after the register save area and pointed to by register 12, and the LIOCS table for SYS001 are written on SYS001 and replaced by the new interface.

All items of the communication region which are used by phases D00, D05, and D10 are saved in the communication region of the new interface.

The new interface uses an automatic end-of-file branch on the text work files. Therefore, the address of the end-of-file routine of the new interface is inserted into the LIOCS tables.

Phase D00 is called by the EOPH routine of the new interface. An end-of-file indicator is written on the current text output work file and the functions of the input and output work files are switched.

Symbols used in flow charts:

IJKMJT	: Job communication bytes (old interface)
TEXTIN	: Contains address of text input work file table
TEXTOUT	: Contains address of text output work file table
KSYS001	: Contains information on IJSYS01,
KSYS002	: IJSYS02, and
KSYS003	: IJSYS03 in old interface
ZTOUT	: Subroutine for reading table records
TASAVA	: Area for saving the SYS001 table
ZTAB07	: TABTAB entry for DS table (not used at this point)
IJKWT	: Wait routine in old interface
T	: SYS001
IJKMTT	: Begin of TABTAB, entry in old communication region
TABLEM	: Contains address of SYS001 table
NOTEF	: Area for building NOTE/POINT information
POINTW	: LIOCS macro instruction
BEGINT	: Begin of old interface
WRITEI	: Begin of LIOCS write macro instruction
ENDINT	: End of old interface
TABLEN	: Length of LIOCS table
INTTABEN:	Begin of communication region in new interface
AR	: Begin and
LE	: Length of area written
NEOFAD	: Address of new end-of-file routine
EOFADD	: Relative address of end-of-file entry in LIOCS table for disk
EOFADT	: and for tape work files
DUMPSAVE:	Save area for old end-of-file address
IJKMVC	: Variable counter, entry in old communication region
IJKMNN	: Name of address constant for origin of compilation, entry in old communication region
IJKMBL	: Block length on text work files, entry in old communication region
IJKMN	: Move routine of old interface
NINTL	: Length of new interface
INTBEG	: Begin of new interface
TABLE	: Begin of communication region in new interface
EOPH	: End-of-phase routine in new interface.

NEW INTERFACE

Only three routines are provided by the new interface. A fourth entry is used internally for handling the end-of-file condition.

As in the old interface, the interphase linkage is established by a DSECT in the phases and with register 12 as base register.

Read/Write -- AW

Only a non-overlapped input/output on the text work files is provided, the same macro area is used for both. Therefore, the command code must be stored in the READ/WRITE macro instruction. Register 1 is set to text input or text output before entering the common part. Prior to executing the READ macro instruction, the READ routine checks whether the end-of-file was reached and returns if this condition occurs. The length of the area to be read is inserted into the READ/WRITE macro instruction in the READ routine only because a READ always precedes a WRITE. It must be inserted because the EOPH routine may modify this parameter. After the read/write macro a check macro is given for the respective medium.

The calling sequence is:

```
LA   1,area
BAL  14,READ/WRITE
```

where area is the input or output area.

EOPH -- AW

The end-of-phase routine writes an end-of-file indicator on the current output work file. This is done for tape work files by giving a control macro instruction and for disk work files by using the write routine with the length parameter zero. The end-of-file indicator is set to zero, the text work files are reset to their beginning by POINTS macro instructions, and the functions of the text input and output work files are switched. At the end, a load macro instruction is given with the new phase name and a branch to register 1.

The calling sequence is:

```
L    1,=C'DXXb'
BAL  14,EOPH
```

where DXX is the suffix of the phase name to be called.

EOF -- AW

This routine sets the end-of-file indicator on. It is automatically entered when an end-of-file indicator on the input work file is detected.

Communication Region

The entries IJKMVC, IJKMNN, IJKMBL, and IJKMWC are the same as in the old interface.

IJKMJT has a length of only two bytes. The first 12 bits have the same meaning as in the old interface, bits 12-15 have the following meaning:

```
bit 12 = 0: tape work files
       = 1: disk work files
bit 13 = 1: GOTO library routine must be
            called
bit 14 = 1: built-in functions in current
            compilation
```

ADLIBI is one word of the library usage bytes matching bytes 5 through 8 of IJKMLB.

Symbols used in flow charts:

EOFIND	:	End-of-file indicator
CHECK	:	LIOCS macro
TEXTI	:	Holds address of text input work file table
REWR	:	Area of read/write macro
INTTAB	:	Begin of communication region
READ/ WRITE	:	LIOCS macros
SAV01	:	Register 1 save area
TEXTOU	:	Holds address of text output work file table
PHASEN	:	Phase name area
CNTRL	:	LIOCS control macro
CCWOFF	:	Offset in module where CCW chain bit is set into table
POINTS	:	LIOCS macro
LOAD	:	DOS/TOS macro

GENERAL DESCRIPTION OF PL/I PHASES D00 - D11

These phases process the following statements:

PROCEDURE	GOTO	EXPRESSION
BEGIN	ENTRY	IF
END (PROCEDURE)	RETURN	CALL OVERLAY
END (BEGIN)	NOP	CALL DYNDUMP
CALL	SET	

If conversion is required, the appropriate macro instructions are generated. The subscripted variables, fixed- and floating-point registers, and the working storage required during execution are determined and optimized. Note that DO loops have been replaced by assignment and IF statements during the phases C85 and C86. The compound statement IF was expanded to simple statements in phase C25.

An expression statement is generated during the I/O scan 1 (phases C50 - C65) to allow the evaluation of expressions contained in these statements.

The phases D00 - D11 use similar main, error handling, initialization, and data manipulation routines and the same I/O concept.

INPUT

Phases D00 - D11 are fetched after the first I/O scan (phases C50 - C65) and the phases decomposing the DO and IF statements. The input is the program string, which consists of statement bodies preceded by the statement key and the attributes of the variables and constants contained therein.

The statement body is followed by the corresponding EOS key and flags for the errors detected during previous phases. Label macro instructions and generated variable definitions may also appear in the program string but not inside the statement body.

The program string input or output consists of syntactical units that can be identified according to the preceding key, which may be X'Fn', XEn', X'0n', or X'1n', where n is a hexadecimal digit in the range from 0 to F. The keys and their meaning in the input string for the phases D00 - D11 are shown in Figures 1 through 3.

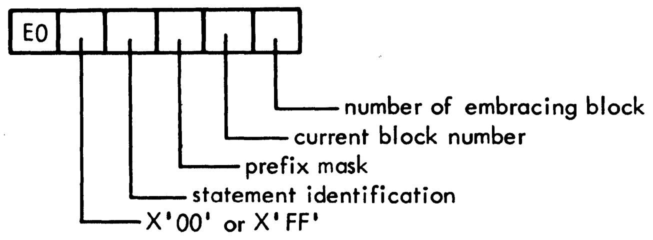
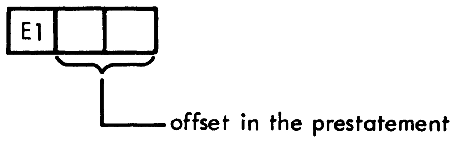
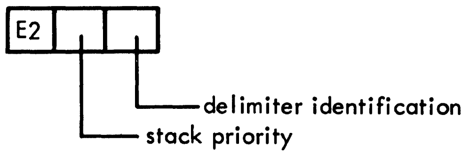
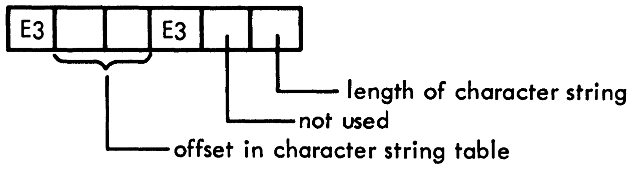
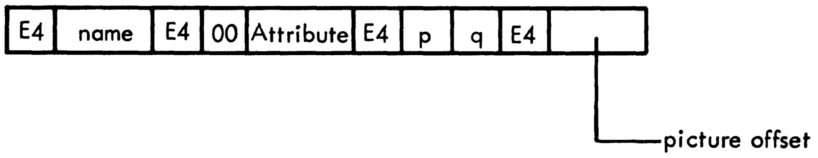
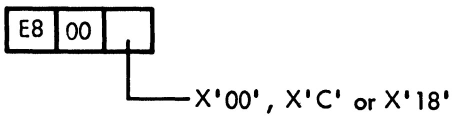
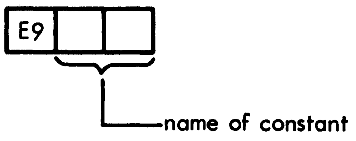
 E0 number of embracing block current block number prefix mask statement identification X'00' or X'FF'	statement identifier
	reference to declared variables
	delimiters
 E3 E3 length of character string not used offset in character string table	reference to character string constant
	reference to generated variables
E5,E6, and E7 are not available	
	reference to variables with unknown attributes
	reference to constants included in the prestatement

Figure 1. Input for Phase D00 (Part 1 of 2)

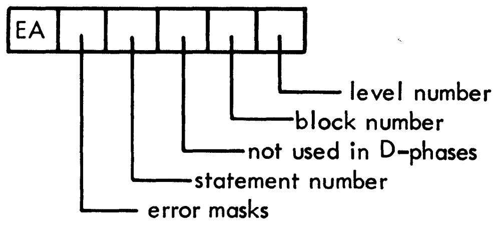
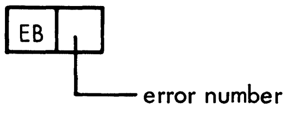
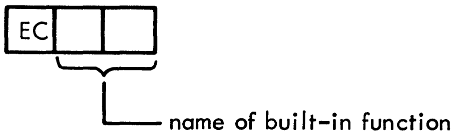
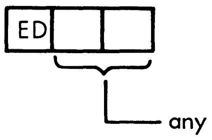
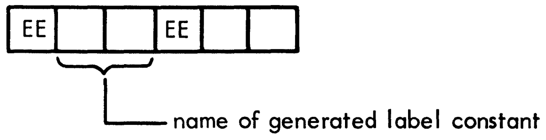
 EA level number block number not used in D-phases statement number error masks	end of statement
	error messages
	reference to built-in function
	I/O intermediate key
	reference to generated label constant
EF	not available
F0	generated variable declaration
F1	not available
F2	macro
F3	table of constants
F4	table of declared variables
F5 to FD	not available
FE	constant in format statement
FF	e nd of program string.

Figure 1. Input for Phase D00 (Part 2 of 2)

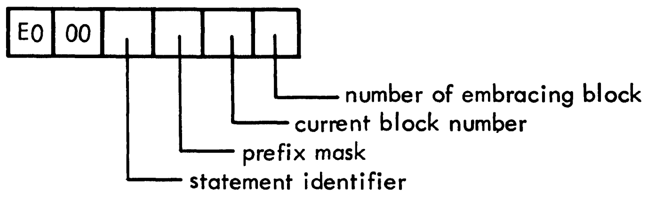
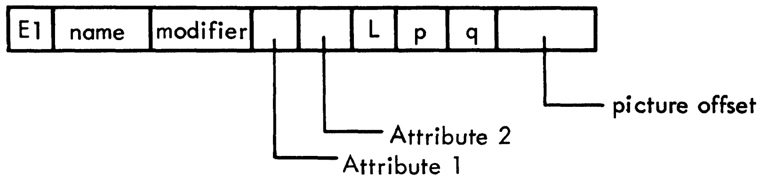
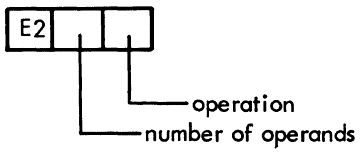
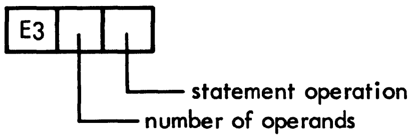
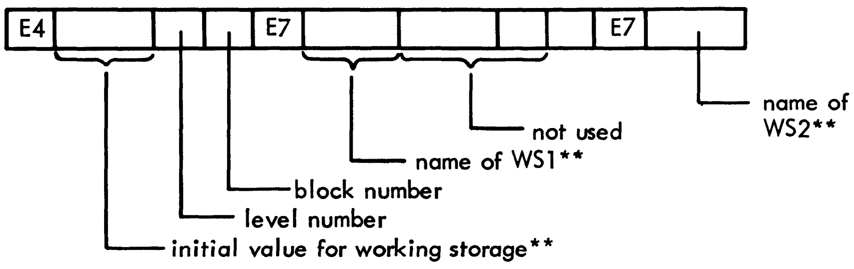
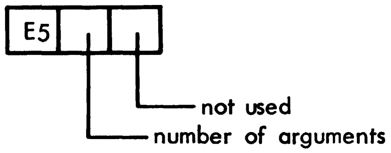
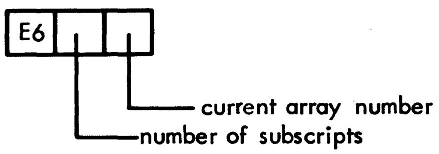
 E0 00 number of embracing block current block number prefix mask statement identifier	statement identifier
 E1 name modifier L p q picture offset Attribute 2 Attribute 1	reference to declared variables
	operations
	statements operations
 E4 E7 E7 name of WS2** not used name of WS1** block number level number initial value for working storage** * on byte boundary ** on double-word boundary	define initial value for working storage of block
	function call
	evaluate subscript

Figure 2. Input for Phase D05 (Part 1 of 4)

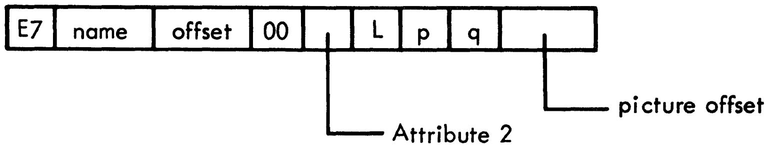
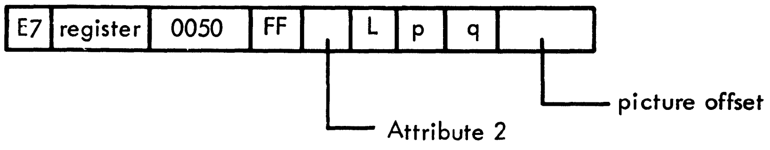
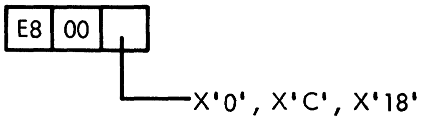
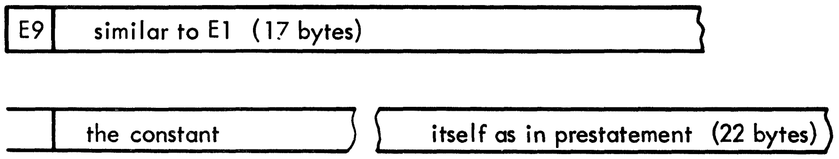
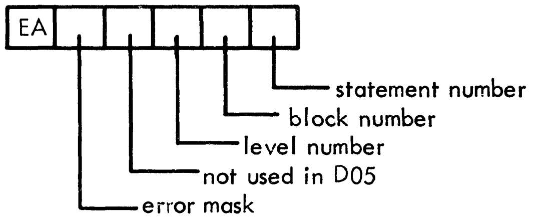
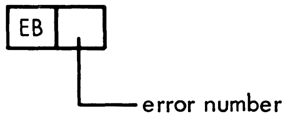
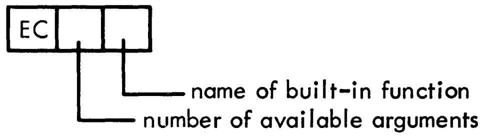
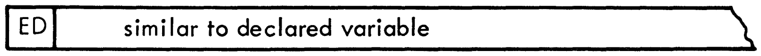
 E7 name offset 00 L p q picture offset Attribute 2	indirect reference (array assignment)
 E7 register 0050 FF L p q picture offset Attribute 2	reference to returned value
	reference to variable with unkown attributes
 E9 similar to E1 (17 bytes) the constant itself as in prestatement (22 bytes)	
 EA statement number block number level number not used in D05 error mask	end of statement
	error message
 EC name of built-in function number of available arguments	reference to built-in functions
	reference to character string constant
EE and EF are not available.	

Figure 2. Input for Phase D05 (Part 2 of 4)

F0	generated variable declaration
| F1 | 00 | 03 |	UNSPEC function
F2	macro instruction
| F3 | 00 | 03 |	string function
| F4 | 00 | 03 |	UNSPEC pseudo variable
| F5 | 00 | 03 |	string pseudo variable
F6	assembler macro instruction
F7	not interesting program string
| F8 | 00 | 03 |	array function argument (ALL ,ANY PROD, SUM)
F9	not available
| FA | 00 | 03 |	array function argument (target) (PROD,SUM)
| FB | 00 | 03 |	array function argument (target) (ALL,ANY)
FC	function reference (array function)
FD	DO separator
FE	not available
FF	end of program
00 to 1F	is array name and have the same format as X'E1'.

Figure 2. Input for Phase D05 (Part 3 of 4)

STATEMENT IDENTIFIER OPERATIONS	
E3 nn 03	dyndump
E3 01 04	overlay
⟵ prologue macro L ⟶ E3 nn 05 F2 L	procedure and entry
⟵ prologue macro L ⟶ E3 00 06 F2 L	begin
E3 00 07	end procedure
E3 00 08	end begin
E3 nn 09	call
E3 01 0A	GOTO
E3 nn 0B F2 L	entry (prologue) (similar to procedure)
E3 02 0E	SET
E3 01 0F	expression
E3 01 10	IF
Where nn is the number of operands.	

Figure 2. Input for Phase D05 (Part 4 of 4)

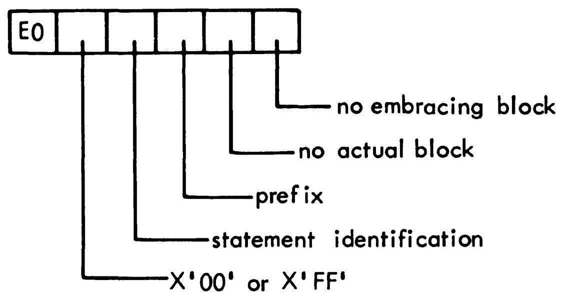
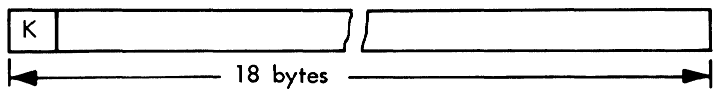
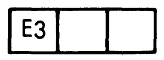
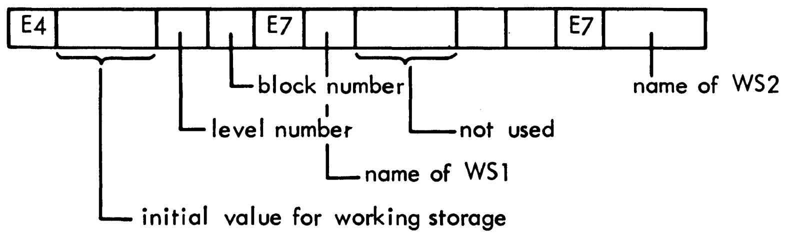
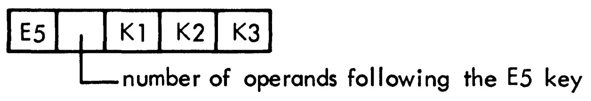
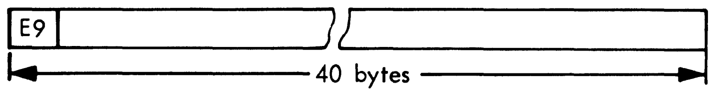
 E0 no embracing block no actual block prefix statement identification X'00' or X'FF'	statement identifier
 K may be: X'E1' X'E7' X'E8' X'EF', for detail see Input for the Routine PREMAC in phase D10	operand of macros and functions
	end of block
 E4 E7 E7 block number name of WS2 level number not used name of WS1 initial value for working storage WS1 is the name for the working storage on double word boundary and WS2 is the name for the working storage on byte boundary.	define initial value for working storage of block
 E5 K1 K2 K3 number of operands following the E5 key	premacros
 for details see Input for the Routine PREMAC in phase D10	constants

Figure 3. Input for Phase D10 (Part 1 of 2)

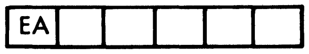
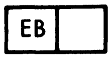
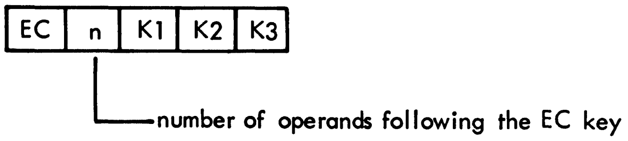
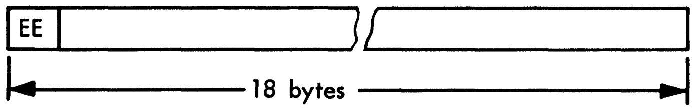
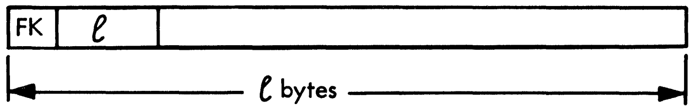
	end of statement
	error indicator
 EC n K1 K2 K3 number of operands following the EC key	function call
 for details see Input for the Routine PREMAC in phase D10	'operands(s) to be requested
 any key F'X' except X'FF'	F-keys
	end of text string

Figure 3. Input for Phase D10 (Part 2 of 2)

The attributes of variables and constants are packed into one byte as X'mn'. The meanings of m and n are shown in Figure 4.

m	Attributes*
0	Scalar variable without picture
1	Scalar variable with picture
2	Array without picture
3	Array with picture
4	ENTRY name or function name without picture
5	Function name with picture
6	Constant
7	Format label
*The high-order bit of the half-byte m is 1 in case of controlled variables. Otherwise it is 0.	

n	Attributes
0	Binary float
1	Binary fixed
2	decimal float
3	Decimal fixed
4	Zoned decimal
5	Zoned decimal (T)
6	CS aligned
7	BS aligned
8	Sterling
9	Label
A	Pointer
B	BS packed
C	Major structure
D	Minor structure
E	Free
F	File

Figure 4. Format of Attribute Byte

Phases D00 - D11 fetch the input element by element and call the appropriate action according to the switch table EFACTION (32 bytes). The actions are numbered from 0 - 31. ACTIONn refers to the key X'En' (or X'F(n-16)' if n greater than 16. The switch table contains the displacement of the actions 0 - 31 relative to the origin action divided by 2. Division by 2 is performed in order to make each displacement fit into one byte. Each action corresponds to a routine to be activated if the respective key is detected in the input stream. The FETCH routine fetches the address of the action to be taken and stores it in R10.

Since the routines may be recursively activated, the return addresses are dynamically saved and restored. The principle of push-down store is extended by coupling it with the chain and list-processing technique to facilitate optimizing the use of registers and working storage.

COMMON SERVICE ROUTINES

The service routines shared by phases D00-D11 are briefly described in the following.

Saver -- DA

This routine dynamically saves the relative return address of a subroutine in an automatic save area. Base register BASE1 always points to this save routine. The link register LINK is assumed to contain the return address to be saved when the routine is entered. Upon return from the routine, LINK contains the computed relative address that was saved. The routine is called as follows:

```
    BALR  R4,SAVER
```

SAVER, which always contains the return address (entry of XXXX), is equated with BASE1.

RETURN -- DB

This routine is used to dynamically return to the calling routine. It fetches the return address from the automatic save area and modifies the corresponding pointer by decreasing it by 2. The return routine is called as follows:

```
    BCR  CON,RETURNER
```

where CON is any condition code. RETURNER, which always contains the address of the entry point of the RETURN routine, is equated with BASE2.

ERROR -- DE

The error routine is activated by:

```
    BAL  LINK,ERROR
    DC   X'mmnn'
```

where nn is the error number and mm is the severity code. The routine skips the statement in error up to the end of the statement and inserts the error into the text after the statement end.

ADMVC -- DF

This routine generates names used during compilation. The routine fetches the current name from the communication area IJKMVC and loads it into register 0. An error is indicated if the name fetched is 0, i.e., if the number of names is greater than 32K-1. If the name fetching is successful, IJKMVC is incremented by one. No arguments are required. The routine is called as follows:

```
    BAL  LINK,ADMVC
```

Data Manipulation Routines -- A3

A general move routine MOVE is provided to move data between storage, work files, and table area. This routine is activated by one of six routines that pass the required parameters for source and target. The names and functions of these routines are given in Figure 5. New routines may be provided for additional requirements. The general instruction sequence of the routines listed in Figure 5 is as follows:

```
BALR   R4,SAVER       saves return address
BAL    RZ,MOVE        calls MOVE routine
DC     AL2(LS-Z)
DC     AL2(LT-Z)
DC     X'1RS2RTRS1RT2'
DC     AL1(AX-A1)
DC     AL1(AY-A1)
```

From / To	Input	Output	Table	Storage
Input	MOVEII	-	-	-
Output	MOVEIO	-	MOVETO	MOVESO
Table	MOVEIT	-	-	-
Storage	MOVEIS	-	-	-

Figure 5. Names and Functions of Data Manipulation Routines

Definition of parameters:

Z	name of parameter list
LS	currently available source length
LT	currently available target length
RS	register containing source address
RT	register containing target address
AX	action to be taken if source length has been exhausted (can be A1 or A4)
AY	action to be taken if length of target has been exhausted
A1	(can be A2 or A3)
A1	origin of A-action routines.

The general move routine assumes that RZ points to Z.

Parameters used in the different MOVE routines.

MOVEIT

Z is VIT	
LS is IL	currently available input length
LT is TL	currently available table length
RS is 6	register containing input address
RT is 8	register containing table address
AX is A1	read input
AY is A3	load new table pointer to RY

MOVETO

Z is VTO	
LS is TL	currently available table length
LT is OL	currently available output length
RS is 8	register containing table address
RT is 7	register containing output address
AX is A4	load new table pointer to RX
AY is A2	write output

MOVEIO

Z is VIO	
LS is IL	currently available input length
LT is OL	currently available output length
RS is 6	register containing input address
RT is 7	register containing output address
AX is A1	read input
AY is A2	write output

MOVEII

Z is VII	
LS is IL	currently available length of input
LT is IL	currently available length of input
RS is 6	register containing input address
RT is 6	register containing input address
AX is A1	read input
AY is A3	load new table pointer to RY

MOVESO

Z is VSO	
LS is TL	currently available table or storage length
LT is OL	currently available output length
RS is 1	register containing storage address
RT is 7	register containing output address
AX is A4	load new table pointer to RX
AY is A2	Write output

MOVEIS

Z is VIS	
LS is IL	currently available input length
LT is TL	currently available table or storage length
RS is 6	register containing input address
RT is 1	register containing storage address
AX is A1	read input
AY is A3	load new table pointer to RY.

Actions Taken in MOVE: A1,A2,A3,A4 -- A4

Routine A1. This routine is activated if the length available in the current input buffer is exhausted. One record is read and the length available for the input buffer as well as the corresponding input pointer PIN are initialized.

Routine A2. This routine is activated if the length of the output buffer is exhausted. The following actions are performed:

1. One record is put out;
2. The available length and the output pointer are initialized;
3. The routine waits for completion of the output operation.

Routine A3. This routine is called if the table area is full. The initial value X'7FFF' is moved into the corresponding target length.

Routine A4. This routine differs from A3 only in that it moves the initial value X'7FFF' into the source length.

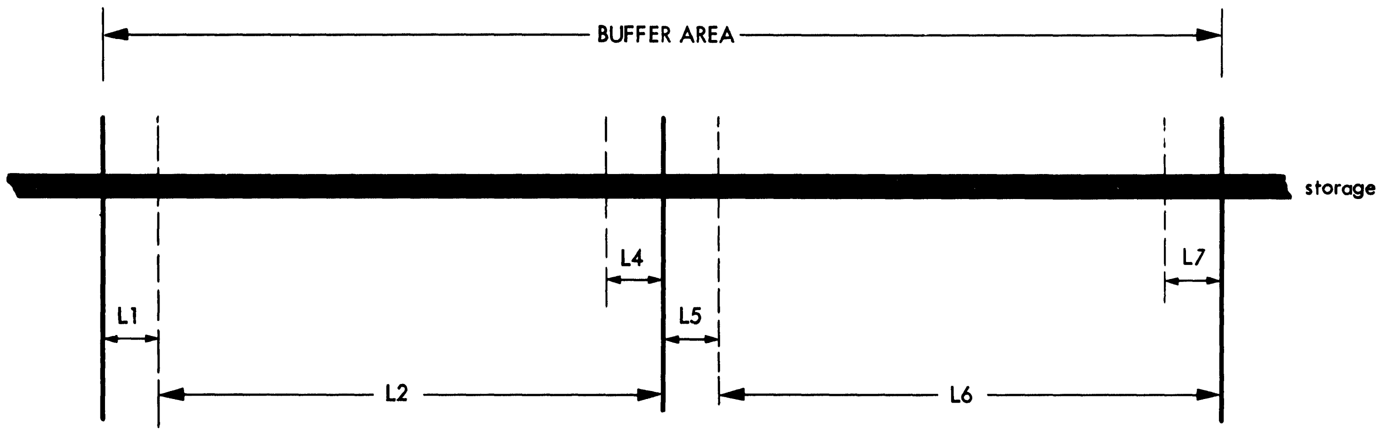
ACTION31

This routine is called if the end of the program is detected in the input stream. It terminates the current phase and calls the next phase.

BUFFER CONCEPT AND PHASE LAYOUT

The read and write buffers (one of each) are located in adjacent storage areas (see Figure 6). The first record of the string to be processed is read into L2 of the read buffer and then scanned accordingly. If the beginning of L4 is detected, the contents of L4 are moved into L1 and the next record of the string to be processed is read into L2 in non-overlapped mode. The pointer is set to the beginning of L1, and scanning is continued. This process is repeated until the entire string has been processed. For the write buffer, the procedure is the same.

This buffer concept eliminates the necessity for using the NOTE and POINT macro instructions.



L1 - secondary read buffer 1	length differs from phase to phase, but is fixed in each phase
L2 - read buffer	length depends on available storage
L4 - secondary read buffer 2	same as for L1
L5 - secondary write buffer 1	same as for L1
L6 - write buffer	same as for L2
L7 - secondary write buffer 2	same as for L1

L1 = L4 (may be zero)
L5 = L7 (may be zero)

Figure 6. Buffer Organization

This phase performs the following major functions:

1. It reorders the input stream in reverse polish notation.

2. It decomposes the array and structure assignments.

3. It generates the prologue macro instructions.

4. It processes and deletes the prestatement.

The input stream consists of statements, each statement being preceded by its corresponding prestatement and followed by the end of statement. The input stream may be considered as a continuous number of syntactical units, each unit being defined by its first byte (key). This key may be X'En' or X'Fn'. Depending on the key found, one of a group of routines (named ACTION0 through ACTION31, i.e., ACTIONn for key X'En' or ACTION (n+16) for key X'Fn') is activated.

The output is similar to the input, except that some of the syntactical units have a different format and meaning.

The table and work areas, which are preceded by the area occupied by the I/O buffers, are dynamically allocated.

Operator Priorities

The operators that may appear in a PL/I source program are ordered by relative priority, the lowest priority being zero. Within statements, operations with a higher priority are performed before operations with a lower priority preceding them. Expressions and assignment statements are evaluated from left to right. Exceptions are exponentiation, negation, prefix plus, and prefix NOT (logical NOT), which are evaluated from right to left. The operations that may appear in a PL/I source program and the corresponding priorities are listed in Figure 1.

DESCRIPTION OF ROUTINES

Note: the following routines are described in the section General Description of Phases D00 - D11.

ADMVC	MOVESO	MOVEIT
ERROR	MOVETO	SAVER

Symbols used in flow charts:

PIN	-input pointer
TP	-table pointer
SP	-stack pointer
INP	-priority of input element
STP	-priority of element on top of stack
SORG	-stack origin
STRORG	-structure origin

Fetch -- BC

The routine computes the length of the current element in the input stream and loads the address of the appropriate action into registers RL and R10, respectively. It uses the current value of the input pointer PIN as argument and the table ACTION as switch table for the individual actions. If the current element has an F-key, the length is fetched from the two bytes following the key. Otherwise, the length is computed from LETAB. The routine is activated as follows:

```
    BAL  LINK,FETCH
```

Init1 -- BB

This routine is used to initialize the stack pointer and the table pointer.

ACTION0 -- BD

This routine is activated when a statement identifier is detected in the input stream. The stack and table pointers are initialized after checking the necessity for output from the table area. If the statement is a DO header or DO trailer, it is replaced by X'F20004D5' or X'F20004E5', respectively. If the statement is a CALL statement replacing a BEGIN statement, it is put out unchanged. In all other cases, the statement is checked to determine whether it is one of the statements to be processed by the phases D00 through D20. These statements have the internal representation X'E00002' through X'E00010'. If the current statement is one of these statements, switch SW1 is set to X'FF'. Otherwise, it is set to X'00'.

ACTION1, 3, 4, 6, 8, 9, 30

This routine is called when a variable name is detected. It moves the current input element into the table area.

	Delimiter	Internal Representation	Priority in String	Type of Operation
0	-			
1	-			
2	-			
3	Built-in bracket	E200E3	-	Operand dependent
4	Entry bracket	E200E4	-	Operand dependent
5	Subscripted variable bracket	E200E5	-	Operand dependent
6	Prologue (	E200E6	-	-
7	)	E200E7	1	-
8	Comma	E200E8	1	-
10		E203EA	3	String
11		E204EB	4	Bit
12	&	E205EC	5	Bit
13	=	E207ED	7	Comparison
14	=	E207EE	7	Comparison
15	<	E207EF	7	Comparison
16	>	E207F0	7	Comparison
17	<=	E207F1	7	Comparison
18	>=	E207F2	7	Comparison
19	infix +	E208F3	8	Arithmetic
20	infix -	E208F4	8	Arithmetic
21	*	E209F5	9	Arithmetic
22	/	E209F6	9	Arithmetic
23	prefix +	E20AF7	11	Arithmetic
24	prefix -	E20AF8	11	Arithmetic
25		E20AF9	11	Bit
26	**	E20AFA	11	Arithmetic
27	statement identifier	E000mn	-	Any
28	PROCEDURE	E00005	-	-
29	BEGIN	E00006	-	-
30	END (PROCEDURE)	E00007	-	-
31	END (BEGIN)	E00008	-	-
32	CALL	E00009	-	Any
33	GOTO	E0000A	-	Label
34	ENTRY	E0000B	-	Any
35	RETURN	E0000C	-	Any
36	NOP	E0000D	-	-
37	SET	E0000E	-	Any
38	EXPRESSION	E0000F	-	Any
39	IF	E00010	-	Bit
40	OVERLAY	E00004	-	Character
41	DYNDUMP	E00003	-	Any

Figure 1. Operations and Corresponding Priorities

<u>Action5, 7, 13, 15, 17, 21-28</u>

These routines are not available because the corresponding keys cannot occur in the text string.

<u>ACTION2 -- BF</u>

This routine processes the delimiters in the input stream and sets the delimiter switch CUR to X'FF'. If the input is an equal sign, the routine returns after setting CUR to X'FF'. If the input is any delimiter other than open parenthesis, close parenthesis, or comma, the input priority is compared with the priority of the element on top of the stack. If it is higher, the current input element is stacked, the stack pointer SP is decreased by 3, and the routine is left. Otherwise, the element on top of the stack is moved to the table. Both the stack and the table pointer are then incremented by 3.

The routine continues processing by comparing the input priority with the priority of the new element on top of the stack.

If the input element is a (, the previous element is checked to determine whether it is a delimiter. If it is a delimiter (PREV switch is X'FF'), the (is stacked. If it is not (PREV switch is X'00'), a list parenthesis (X'E200E5) is stacked.

If the input element is a comma, the elements in the stack are unstacked until a (or a comma is found.

If the input element is a), the list counter is set to 1 and the delimiter in the stack is checked. The following actions are taken depending on the element found:

1. If the stacked element is a comma, the list counter is increased by 1.
2. If the stacked element is a (, both parentheses are deleted and the routine is left.
3. If the stacked element is a list parentheses, the counter value is stored in the second byte, the parenthesis in the stack is unstacked, and the routine is left.

ACTION10 -- BE

This routine is activated if the end of statement is detected in the input stream. If the statement contains error(s) with a severity code other than W, the entire statement is skipped. The end of statement is put out together with the corresponding error message(s). Otherwise, control is transferred to STATEN.

ACTION11

The error in the input stream is put out unchanged.

ACTION14

The delimiter switch CUR is set to X'FF'. Processing is as in ACTION1.

ACTION16, 18, 29

The delimiter switch CUR is set to X'FF' and the element is moved to the output medium.

ACTION19

Pointer TP is loaded with the origin of the constant table in the prestatement.

ACTION20

If the current statement is not an expression, the element is moved to the table area. Otherwise, the delimiter switch CUR is set to X'FF' and the element is moved to the table area.

CHECKOUT -- BB

This routine checks for elements in the table area that must be moved into the output buffer.

STATEN -- BG

This routine is called by ACTION10 if an end of statement is detected in the input stream. It determines whether the source program contains arrays or structures. PUTOUT is called if neither is present.

If the current statement is an assignment statement and the source program contains arrays and/or structures, the statement is checked for array or structure assignment. Depending on the type of statement, either ARROUT or STROUT is called to put out the statement.

AC1 - AC6

After the statement end has been found in routine STATEN, the information stored in the table area is scanned again, element by element. For the individual text elements (keys E0 - EF) one of the routines AC1 through AC6 is called. Table T1 (Figure 2) gives the routine called for the corresponding key.

Key	Routine
E0	AC6
E1	AC2
E2	AC1
E3	AC3
E4	AC3
E5	AC5
E6	AC5
E7	AC5
E8	AC3
E9	AC3
EA	AC4
EB	AC5
EC	AC3
ED	AC5
EE	AC3
EF	AC3

Figure 2. Format of Table T1

AC1 -- BR

This routine is called for delimiters. If the delimiter indicates an array, a structure, or a mixed array and structure expression, one of the routines ARRAY, CHAIN, or ARRCH is called.

AC2 -- BQ

This routine is called for declared variables.

AC3 -- BQ

This routine is called for operands that are not declared variables. The stack pointer is decreased by 4.

AC4 -- BQ

This routine is called if the end of statement is detected again. One of the routines DYNDUMP, ARROUT, STROUT, or PUTOUT is called.

AC5

This routine is not available since the corresponding element cannot occur.

AC6 -- BR

This routine is called for a statement identifier. It is identical with part of AC1.

CHECKSP -- BB

This routine modifies and checks the stack pointer. Error 142 is indicated if a stack overflow occurs.

ARROUT

This routine puts out the array assignment statement and the corresponding header and trailer macros. PUTOUT is called.

STROUT

This routine puts out the structure assignment statements. PUTOUT and ARROUT may be called several times to put out the statement after it has been modified.

ARRAY, ARRAYQ -- BS

This routine is called by AC1 or STROUT to process an array operand. If the operand was previously used, no action is required. The array is compared with other arrays in the statement, if any, for identical number of elements.

CHAIN --BS

This routine is called by AC1 for processing structure operands.

ARRCH

This routine is called when mixed array and structure expressions are found in AC1. The routine consists of a call of the error routine ERROR.

LENGTH -- BU

The routine computes the internal length (in bytes) of a variable or constant. In case of arrays, the length of the element is computed.

PUTOUT -- BH

When this routine is entered, the statement being checked is contained in the table area in reverse Polish notation. It is preceded by the corresponding attribute table and followed by the end-of-statement. The statement attribute table is used only to fetch the attributes of the variables and constants that appear in the source text.

The routine scans the statement body element by element and activates the appropriate action (one of the routines E0 through EF) via the switch table TAB10. Routine En refers to key En.

If a prologue is required, one of the routines E005, E006, or E00B is called. The appropriate routine is selected as described in E0.

E0 -- BI

This routine processes the "operation" statement identifier. One of the subroutines E003 - E010 is called. Routine E0nn refers to the text element E000nn, which represents the statement shown in Figure 1.

E003, E004, E00A, E00E, E00F, E010 -- BI

The library bit is set to 1, if required, and the 3-byte operation is put out.

E005, E006, E00B -- BJ

These routines generate the prologue macros. An additional branch around the prologue is generated for ENTRY statements.

E00C -- BI

If a RETURN statement returns a function value, an assignment X'3020E' and a corresponding return macro are put out. In all other cases, only the return macro is put out.

E007 -- BI

A return macro is put out for the end of block. X'E00007' is also put out.

E00D

The element is skipped.

E009 -- BI

If the CALL statement has no arguments, the corresponding CALL macro is generated. Otherwise, the number of arguments is retrieved from the previous element (function bracket) and inserted in the second byte of X'E0nn09'.

E1 -- BL

The routine constructs and puts out the 12-byte element by calling E1GEN and MOVESO.

E1GEN -- BK

In the 12-byte field O, the routine constructs the operand to be put out. TP points to the operand in the statement body. After return from E1GEN, field O contains the following information:

Bytes	Contents
0-2	name
3-4	modifier
5	storage class (attribute 1)
6	data attributes (attribute 2)
7	L = internal length in bytes
8	p
9	q
10-11	offset of picture if numeric field

E2 -- BO

The routine determines the number of operands, inserts this number in the second byte of the delimiter, and puts out X'E2nnkk'.

E3 -- BM

The routine puts out 12 bytes for character-string constants. Prior to output, the key is modified to ED.

E4 -- BN

The routine puts out 12 bytes for generated variables.

E5 -- BM

The routine puts out a 12-byte operand for the indirect target for the RETURN statement.

E6, E7, EB, ED

Since the corresponding text elements cannot occur at this point, these routines are not available.

E8 -- BM

The routine puts out X'E800xx' unchanged for variables with unknown attributes.

E9 -- BN

A 12-byte operand is put out (in a format similar to a declared variable) for constants appearing in the statement body.

The corresponding entry in the prestatement is also put out with the maximun possible length (22 bytes).

EA -- BM

The routine puts out the six bytes for the end of statement.

EC -- BM

The routine puts out the function name (except for the NULL function). In the latter case, the name for the NULL function (12 bytes) is put out instead of the function name.

EE -- BM

The routine constructs a 12-byte operand for a label constant or label variable.

EF -- BM

The routine constructs and puts out 12-byte operands for return values. If the RETURN statement refers to a main procedure, the ERROR routine is called.

TESTN -- BO

This routine is called by E2 for checking the number of arguments. ERROR is called for any number other than 3.

This phase:

1. determines the required conversions. The type of conversion depends on the operation and on the data types of the operands given.

2. determines the resulting precision after conversion.

3. determines the resulting precision of each operation.

4. determines the macro keys for the operations. The macro key depends on the operation and the data type of the operand after execution of the required conversions described under item 1. An operation may be one of the following:

   a. Built-in function
   b. Statement identifier
   c. Subscript evaluation
   d. Function call
   e. String operation
   f. Arithmetic operation

5. constructs intermediate macro instructions.

6. determines the necessity of working storage fixed-point and floating-point registers.

7. determines the type of operands (fixed-point register, floating-point register, working storage, etc.). This is machine dependent.

The tables and work areas are dynamically allocated (push-down technique is used). The I/O buffers are located in front of the dynamic area.

Phase Input and Output

The input stream is already ordered in reverse Polish notation. It consists of syntactical units that can be identified by the first byte of each element, which may be X'En' or X'Fn', where n is a hexadecimal digit from 0 to F. One of the actions 0 - 31 is activated depending on the key found.

The output is similar to the input, only that some of the syntactical units have different formats and meanings.

DESCRIPTION OF ROUTINES

MAIN -- DJ

The main routine initializes the stack and table pointers (SP and TP), activates the skip routine, fetches the program string element by element, and calls the corresponding action (ACTION0 to ACTION31). ACTION(n) refers to key X'E0'+n.

FETCH -- DD

The routine computes the length of the current element in the input stream and loads the address of the corresponding action into register R10. When the key n is detected, the address of ACTION(n-X'E0') is loaded. If the current input element undicates an array, ACTION1 is prepared to be called.

CHECKSP -- DJ

The routine fetches an 18-byte entry into the stack. The fetched entry is cleared and overlap of the table (constant stack) and the variable stack is checked. Error 142 is given in case of stack overflow.

ACTION0 (Begin of Statement) -- DK

This action is called when an E0-element is detected in the input string. The stack and table pointers are initialized, and the 6-byte input element is moved from input to output.

ACTION1, 7, 13 -- DK

The routine is called when a constant or variable is fetched. The routine stacks the input element together with the corresponding attributes. CHECKSP is called to get and clear an 18-byte stack entry.

ACTION2 (Operation) -- DK

The routine is called when an operation is detected. The routine branches to the EXPONENT routine if the detected operation is an exponentation. Otherwise, the address of the switch table (T20 - X'EA') is loaded into register R5. The output switch is set to X'E4' and the routine branches to ACTIONC0 which is common for functions and operations.

ACTION3 -- DL

This routine is called if a statement identifier is detected in the input string. Using byte 2 of the input string as a switch, this action activates the corresponding routine (E302 - E310) for each statement. Routine E3nn refers to the input element E300nn. The switch table ACTTAB3 (15 bytes) is used for this purpose. Each byte of this table contains the displacement of the corresponding routine divided by 2. Division by 2 is performed to make the displacement fit into one byte.

ACTION4 (Define Initial Value for Working Storage) -- DN

The current element is put out unchanged after saving the current block number and level number.

ACTION5 -- DM

A function call is generated. The routine moves 2 to ACT56+2 in order to allows saving of all floating-point registers, if necessary. The call switch CALLSW is set to 0 in order to pass a result argument. The current element is skipped and ACTION50 is called to process similar to the CALL statement.

ACTION50 -- DO

The routine processes a CALL statement or function call. The arguments are checked and assignments to dummy variables are generated, if necessary (e.g., constant, variable inclosed between brackets ...etc.). In case of function calls the target field in which the function value is to be returned is generated.

ACTION6 -- DN

The routine converts the subscripts to binary integer, constructs the correspond-ing inermediate macro instruction, and puts it out together with additional request 0. The result which has the same data characteristic as the type of array is stacked and the indirect bit is set in the stack.

ACTION8 (Variables with Unknown Attributes) -- DQ

The operand is moved from input to stack (3 bytes). If a value is already assigned to the operand, the corresponding attri-butes are moved from DO to the stack and the routine continues similar to ACTION1. Otherwise, the routine is completed.

ACTION9 -- DK

The action for constants is similar to the action for declared variables (ACTION1). In addition, the 22 bytes following the constant reference are stacked in the constant table area.

ACTION10 -- DK

The 6-byte end-of-statement is moved from input to output.

ACTION11 -- DK

The error message is moved from input to output.

ACTION12 -- DP

The action is activated to process built-in functions. If the built-in function is a substring pseudo-variable, the routine 3030 is called. Otherwise, R5 is set to point to the data table for built-in functions. The output switch ACTION1+1 is set to X'E3'; F1 is set to X'13', and ACTIONC0 which is common to ACTION2 and ACTION120 is called.

Action14, 25, 30

These routines are not available since the corresponding text elements cannot occur.

ACTION15 -- DK

Similar to ACTION1, but the key X'EF' is replaced by X'E5'.

ACTION16, -22, -23, -28 -- DR

The input element is moved to output.

ACTION17

The routine processes the UNSPEC function.

ACTION18, ACTION29 -- DS

The input element is moved to output. The routine checks if the element is a DO header, DO trailer or none of both. If it is a DO header, the DO stack is initial-ized by clearing it to zero. If it is a DO trailer, the required DO variables are generated. Otherwise, it is put out unchanged.

ACTION19

This routine processes the STRING func-tion. It is similar to ACTION17 except that the bit switch is set to X'06'..

ACTION20

The routine processes the UNSPEC pseudo variable.

ACTION21

This routine processes the string pseudo variable.

ACTION24 -- DR

The 3-byte element is skipped and the following 12-byte element is stacked.

ACTION26 -- DR

The operand X'E800600000' is moved to the stack as source. The current input element is moved from input to output.

ACTION27 -- DR

The operand X'E100090000' is moved to the stack as source. The current input element is moved from input to output.

ACTION31

This routine is called if the end of the program is detected. It terminates the current phase and calls the next phase.

ACTIONC0 -- DP

This routine is used for constructing premacros (key X'E5' or X'EC'). It consists of a sequence of subroutine calls. The following routines may be called:

MOVEII	to skip the current element in the input stream.
MOVEDATA	to determine N-1, N+1, M, M-18, and M+18, where N is the number of operands and M is the current value for the stack pointer.
CHECKENT	to check if any one of the operands is a function without arguments. The routine modifies the preceding key to X'61' if it is.
ACTION2C	to determine the required conversion and the precision resulting from this conversion.
DETERMIN	to determine the precision of the result.
FINDKEY	to determine the appropriate macro instruction key for the operation.
PUTOUTFC/ PUTOUTE5	to put out the intermediate macro instruction.

In case of a comparison operation, a SET TRUE macro is put out in addition.

PQ -- DZ

The routine computes the values p, q, L, and p-q and stores the results in R4, R5, R3, and R2, respectively. The routine assumes that R1 points to the operand on top of the stack.

PQ1 -- DZ

The routine computes p, q, L, and p-q for one operand on top of the stack and stores the results in P1, Q1, LL1 and LMQ1, respectively. The routine then calls PQ.

PQ2 -- DZ

The routine computes p, q, L, and p-q for the two operands on top of the stack by calling PQ1 twice. The computed values are stored in the fields P1, Q1, LL1, LMQ1, P2, Q2, LL2, and LMQ2, respectively.

FINDKEY -- EA

The routine checks whether the available key or function name is to be modified according to the data type of the result and performs the modification, if necessary. The routine assumes that the field RESULT contains the type of the result of the current operation.

MOVEDATA -- EB

The routine computes N, N-1, N+1, N*18 (N-1)*18, and (N+1)*18 and stores the results in N0, N1, N2, M0, M1, and M2, respectively. N is the number of operands for the current operation. If N>12, an error is indicated (149).

PUTOUTE5

This routine is used for putting out the output elements that refer to the operation processed. The output consists of a 5-byte element (with the key E5), followed by the operands of the operation (18 bytes for variables, 40 bytes for constants), and terminated by an 18-byte EE element giving the additional requests. For details refer to the section Input for the Routine PREMAC in phase D10.

PUTOUTFC

This routine is used for putting out the output elements that refer to the function processed. Except for the first 5 bytes, the output has the same format as in the routine PUTOUTE5.

ARITH1 -- EI

This routine moves attributes from a result of an operation or function to the RESULT field.

ARITH2 -- EI

After determining which operand represents the result of an operation, this routine moves the attributes of the result to the RESULT field.

RESLEN

This routine is used to compute the length of a result.

EXPONENT -- EN

This routine is used for exponentiation operations. After changing the operation key to a function key, the routine determines the type of exponentiation and the corresponding function name.

E302

The routine generates the intermediate macro instruction for the substring pseudo variable assignment. The data type of the substring (bit or character) is moved from the target field to the data vector.

E303 (CALL DYNDUMP) -- DU

The routine generates the intermediate macro instruction for the DYNDUMP statement. E303 branches to ACTION12.

E304 (CALL OVERLAY) -- DU

The routine calls ACTION2 to generate the intermediate macro instruction for the CALL OVERLAY statement.

E305, E306, E30B -- DU

No action is required.

E307,E308 -- DU

These routines are called when an END statement is encountered. The end-of-block element E3xxxx is put out.

E309 (CALL Statement with Arguments) -- DO

The routine calls ACTION50 after setting the CALLSWITCH to GENTAR1-GENTAR in order to suppress the generation of the target field.

E30A (GOTO Statement) -- DT

The routine checks whether the target is a label constant or label variable. If it is neither or both, a diagnostic is produced. If it is a label variable, a branch-to-label-variable intermediate macro instruction is generated. If it is a label constant and the level number of the label and the block containing the GOTO statement are identical, a simple branch macro instruction is generated. Otherwise, a label assignment is generated and a branch-to-label-variable macro instruction is generated. If a branch to 'label variable' is generated, the library GOTO bit is set to 1.

E30E -- DV

The routine generates the assignment macro instruction. If the operand on the left-hand side of the equal sign is a constant or entry name, an error message (54) is generated. If the operand on the left-hand side is a DO variable, the attributes given on the right-hand side are stacked in the DO variable stack.

E30F -- DU

This routine is called for expressions. The routine continues with routine E303 after clearing the picture byte.

E310 -- DW

The routine checks whether or not a comparison operation has been generated prior to the current IF and generates a branch-on-condition macro or a branch-if-true macro, respectively.

Example

```
a)   IF A>B THEN GOTO L;
           Compare  A,B
           BNH      L1
           B        L

     L1 : .........
     L  : .........

b)   IF A THEN      GOTO L;
           Convert  A to bit
           OC       A',A'
           BZ       L1
           B        L

     L1 : .........
     L  : .........
```

E30C, E30D

These routines are not available since the corresponding elements cannot occur.

ACTION2C

This routine is common for ACTION12 and ACTION2 which are called by ACTIONC0. The routine fetches the corresponding characteristic data for the operation or function from the T-table and stores this data in DATA+3. The routine calls the routines COMMON and CONVERT to determine the data and storage type required for each operand.

DSGEN -- DW

The routine generates DS instructions for working storage in the current block. R0 contains the length to be generated. To ensure that the generated working storage lies in the first 4K, the area is generated as a multiple of DS of length 8 (double-word). If the length is 0, no working storage is generated.

MOVECON -- EM

The routine tests whether the operand is a variable or a constant. If it is a constant integer, it is converted to binary integer in register R0 and switch CON is set to 0. Otherwise, CON is set to X'FF'. SP points to the argument in the variable stack and TP points to the constant value in the constant stack. If the sign bit is 1, the two's complement of the integer constant is loaded into R0.

PRECSION -- EF

The routine fetches the appropriate subroutine according to the matrix FROTO (see Figure 1) to compute the precision resulting from a specified conversion. R3, which contains the type available, and R5, which contains the type required, are passed as arguments. The resulting P, Q, and L are stacked in 15(R1), 16(R1), and 14(R1), respectively. The routine uses the tables FROTO, TYPER, and TYPEC and the group of routines PREC.

CHECKENT -- EE

This routine is called by ACTIONC0. It checks whether the operand is an entry name without arguments. If an operand is an entry name (function value), this is noted by replacing the E1-key by 61 to allow the generation of a dummy variable as well as the appropriate function call in phase D10.

CONVERT -- DX

The routine computes the data type for n operands required by an operation.

The conversion matrix MATRIX (see Figure 2) is used to determine the data type required for the result. The required data type is a function of the type of operation and the data type of the operands. An example of how the required data type is determined is given below.

TO/FROM	0	1	2	3	4	5
	Binary float	Binary fixed	Decimal float, float numeric field	Bit	Decimal fixed, zoned, zoned(T), decimal numeric field	Character
0 Binary float	-	PREC3	PREC6	PREC15	PREC9	-
1 Binary fixed	PREC1C	-	PREC7	PREC16	PREC10	-
2 Decimal float	PREC20	-	PREC11	-	PREC11	-
3 Bit	PREC1	PREC4	PREC7	-	PREC12	PREC1B
4 Binary integer	PREC2	PREC2	PREC2	PREC16	PREC2	-
5 Decimal fixed	PREC23	PREC26	PREC23	PREC25	PREC18 if not decimal fixed	-
6 Character	-	-	PREC17 if numeric field	PREC13	PREC17 if numeric field	-

Each of the routines PREC. computes the precision resulting from the data type conversion. For details on the rules for computing these precisions refer to the SRL publication PL/I Subset Language Specifications, Form C28-6809. The precisions not defined in the language are as follows:

Binary float to binary fixed		PREC1C
	P = min (P,31)	Q = 0
Decimal float to binary fixed		PREC7
	P = min (CEIL (P*3.32),31)	Q = 0
Binary float to decimal float		PREC20
	P = CEIL (P/3.32)	Q = 0/1
Binary float to decimal fixed		PREC23
	P = 5	Q = 0

P, Q, and L of the source data are passed as parameters in RP, RQ, and RL. The resulting values are returned in the same registers.

Figure 1. Matrix FROTO Used to Determine the Routine for Computing the Precision after Conversion

		13	12	11	10	9	8	7	6	5	4	3	2	1	0	
		Pointer	Label	Fixed decimal numeric field	Float numeric field	Binary integer	Sterling numeric field	Bit	Character	Zoned decimal (T)	Zoned decimal	Decimal fixed	Decimal float	Binary fixed	Binary float	
0	Binary float	F	F	0	0	0	0	0	F	0	0	0	0	0	0	0
1	Binary fixed	F	F	1	1	1	1	1	F	1	1	1	1	1	1	1
2	Comparison	D	F	3	2	1	3	9	8	3	3	3	2	1	0	2
3	Decimal fixed	F	F	3	3	3	3	3	F	3	3	3	3	3	3	3
4	Float	F	F	2	2	0	2	0	F	2	2	2	2	0	0	4
5	Fixed	F	F	3	3	1	3	1	F	3	3	3	3	1	1	5
6	Character	F	F	6	6	F	6	6	6	6	6	F	F	F	F	6
7	Bit	F	F	7	7	7	7	7	7	7	7	7	7	7	7	7
8	String	F	F	6	6	F	6	7	6	6	6	F	F	F	F	8
9	Binary integer	F	F	1	1	1	1	1	F	1	1	1	1	1	1	9
10	Binary	F	F	1	0	1	1	1	F	1	1	1	0	1	0	10
11	Decimal	F	F	3	2	3	3	3	F	3	3	3	2	3	2	11
12	Label	F	C	F	F	F	F	F	F	F	F	F	F	F	F	12
13	Pointer	D	F	F	F	F	F	F	F	F	F	F	F	F	F	13
14	Any	D	C	B	A	9	8	7	6	5	4	3	2	1	0	14
15	Coded arithmetic	F	F	3	2	9	3	9	F	3	3	3	2	1	0	15

Figure 2. Conversion Matrix MATRIX

Assume an addition of two operands is to be performed. The first operand is of the type bit (X'FA' which corresponds to line 7 in MATRIX), the second operand is of the type numeric field floating point (X'A' which corresponds to line 10 in MATRIX). The common data type to which both operands must be converted is then determined as follows:

The addition implies that the operation type required is coded arithmetic (column 15 in MATRIX).

For the first operand, column 15 of line 7 in MATRIX points to 9, which is binary integer. For the second operand, column 15 of line 10 points to 2, which is decimal float.

If an operation O of type to has n operands of type t_1, t_2 ....tn, the common type required (t) is:

```
t =  MATRIX(t1,t0) &
     MATRIX(t2,t0) &
         •
         •
         •
         •
     MATRIX(tn,t0).
```

In order to determine the common data type of both operands, the values found (X'09' and X'02') are anded and yield a result of zero. Thus, the data type to which both operands must be converted is 0 = binary float.

COMMON -- DY

This routine determines the data type required for each operand for a specified operation according to the information contained in DATA. The routine further determines the data type of the result and stores it in RESULT. The routine PRECSION is called n times to determine the precision and length of each operand after conversion. R3 and R5, which contain the type available and the type required for each operand, are passed as arguments.

DETERMIN -- ED

This routine is called by ACTIONCO. It selects one of a group of routines provided for computing the resulting precision of an operation of function. The actions performed for operations by the individual routines are shown in flow chart ED. A special group of routines is called for functions. These routines have the standard label P followed by the internal name of the function in decimal notation. For a list of all functions and their internal names refer to Appendix C.

The routines store the characteristics of the result in the 6-byte field RESULT, whose format is as follows:

bytes 0-1	data attribute
byte 2	length (in bytes) of the result
bytes 3-4	precision P and Q of the result
byte 5	Zero

Note: The type of the result is already determined and stored in the result field prior to calling any one of these routines. The routines may determine a new key by calling KEY4MOD and move it to DATA+2, if required.

P(NA)

This is a group of routines; the descriptions of the routines follow. P(NA) refers to the function with the internal decimal name NA. Functions other than those described below cannot occur as input for this phase. After phase D05, the set of possible function names is expanded by separating long and short float functions.

P80 (TIME)

The function length and precision 9 are moved into RESULT+2.

P81 (DATE)

The same as P80. L and P are 6.

P82, P184 (NULL and ADDRESS)

The attributes for pointer results are moved into RESULT. The pointer switch (MVCRES1+1) is set to X'20'.

P84 - P116

After conversion to float, if necessary, these functions have the same attributes as the argument.

P118 (ATAN (Radiant)) -- EK

The routine checks the number of arguments and modifies the name accordingly. If only one argument is available, ARITH1 is called. Otherwise, ARITH2 is called.

P120 (ATAN (Degree)) -- EK

The routine is similar to P118. The keys are different.

P126 (REPEAT) -- EK

The routine computes the length of the bit or character string resulting from the REPEAT function.

L = (N+1)*P

P1260 -- EK

This routine is used by routine P126 for computing and storing the length L of the result field.

P128 (INDEX)

The result is binary integer with the precision (15,0). The routine modifies the key, if necessary, to 129 if the argument is of the type character.

P130 (SUBSTR)

The length of the substring is checked and converted to binary integer. The resulting (L, P, Q) are determined by calling P1260.

P132 (BOOL)

RESULT(L, P, Q) = max (L_1, P_1, Q_1), (L_2, P_2, Q_2); For this reason, P132 calls GET-MAXPQ after modifying R1 and R2 to point to the first and second argument of the function.

P134 and P138 (MAX and MIN) -- EJ

The routines compute the precision resulting from these functions. The precision rules are given in the SRL publication PL/I Subset Language, Form C28-6809. The function name is modified according to the resulting data attribute.

P1344

The routine computes P-Q in R4 and Q in R5 for fixed-scale data. The parameter R1 points to the 18-byte argument.

P146	short float	**	integer
P147	long float	**	integer
P148	decimal fixed	**	integer
P149	binary fixed	**	integer
P150	short float	**	short float
P151	long float	**	long float

These routines compute the precision of exponentation results. For P146 and P147, P is identical with the base. The precision for P148 and P149 is previously determined in the routine EXPONENT. P150 computes the precision of the floating-point result as max (P1, P2).

P160 (ABS) -- EL

The routine determines the key for the individual ABS functions (4 are available) and computes the resulting precision. If the argument is float, the result has the same precision as the argument. If the argument is fixed, the resulting precision is (P+1,Q).

P161 (SIGN)

The routine computes the precision of the SIGN function.

P162 (FLOOR), P163 (CEIL) -- EL

These routines compute

P = max ((P-Q+1), 1) and
Q = 0

in case of fixed scale, and P in case of floating-point.

P165 (TRUNC) -- EL

After moving the initial key for TRUNC to KEY4MOD+3, processing is identical to P162 and P163.

P168 (ROUND) -- EL

The routine computes the precision of the result and modifies the function name according to the resulting data type.

P166	BIT	P171	FLOAT
P167	CHAR	P172	FIXED
P169	BINARY	P173	PRESISIO
P170	DECIMAL		

These routines are identical. They generate assignments for the above functions. The precision of the target may be specified by the programmer. If the precision is not specified, the rules for data type conversion are applied.

P174 (MOD)

The routine computes the precision of the result and modifies the function name according to the resulting data type.

P175	ADD
P176	MULTIPLY
P177	DIVIDE

The routines are identical. The calling sequence is generated either in phase D20 or in phase D17. The precisions, if specified by the programmer, will be selected. If no precision is specified, the rules for the addition, multiplication, and division are applied.

P178 (HIGH), P179 (LOW)

The two routines are identical.

P180	SUM
P181	PROD
P182	ALL
P183	ANY

These four functions are identical.

KEY4MOD

The routine modifies the function name according to the data type of the result. The initial key is moved into KEY4MOD+3, and the routine assumes that the result has already been determined in RESULT.

Note: The following routines are described in the section General Description of Phases D00 - D11.

ERROR	MOVEIO	MOVESO	MOVEIS
ADMVC	MOVEII	SAVER	

PHASE PL/ID10 (MACRO GENERATION I) -- EP

This phase is built up of three logical parts:

1. Scan (ACTION0 - ACTION31)
2. Determination of registers and working storage.
3. Generation of intermediate and conversion macros.

The communication among the three parts is as follows:

```
                 ┌────────────┐
                 V            V
┌────────┐   ┌────────┐   ┌────────┐
| Part 1 |   | Part 2 |   | Part 3 |
└────────┘   └────────┘   └────────┘
   Λ             Λ
   └─────────────┘
```

Symbols used in flow charts:

PIN	:	input pointer
OPR	:	output pointer
SP	:	stack pointer
TP	:	table pointer
INP	:	priority of input element
STP	:	priority of element on top of stack
SORG	:	stack origin
STRORG	:	structure origin
SPB	:	stack pointer (occupied stack chain)
SPA	:	= SP = stack pointer (free chain)
TABORG	:	table origin

Note: The following routines, which are used in this phase, are described in the section General Description of the Phases D00 - D20.

ADMVC	MOVESO (=MOVSTO)
SAVER	MOVEIT

PART 1 OF D10 -- EP

Part 1 scans the input, element by element, and calls the appropriate routine (ACTION(x)). ACTION(x) refers to text elements with the key X'E0'+x. There are as many actions as there are elements (marked by keys) to be processed in this phase. However, actions may be represented by the same coded routine. The subroutines ACTION(x) are the only subroutines used by part 1 of phase D10.

Operands are moved from input into a dynamic area (stack.

ACTION0

The pointers for the register table, the stack, and the table area are initialized. If the current statement is an expression statement, a store macro instruction for the registers is generated, if required, and a switch is set for generating the corresponding restore macro instruction when processing of the statement is completed.

ACTION1, -7, -8, 14

The element is moved to the table area.

ACTION3

The element is skipped and the working storage required in the current block is generated.

ACTION4

The initial values and the names of the byte-aligned and double-word-aligned working storage are retrieved from the input stream and stored. The input element is moved into the field INITIAL.

ACTION5, ACTION12 -- EQ

The 5-byte element is moved to the table area. The corresponding operands are fetched from the input stream and also moved to the table area. If an operand is an intermediate result, it is retrieved from the stack. After all operands have been fetched, PREMAC is called to generate the necessary macro instructions for conversion and for the operation, if required.

ACTION6, -13, -20, -21, -2, 17

These actions are not available since the corresponding text elements cannot occur.

ACTION9

The constant value is moved into the constant area. The constant operand is moved into the operand area. The negative offset value of the constant value in the constant area is moved into bytes 10-11 of the operand.

ACTION10

This routine is called if the end-of-statement is detected in the input stream. A load-multiple macro instruction for these I/O registers is generated if registers were saved at the beginning of the statement. The length of the working storage required in the current statement is compared with that of the current block. The greater of the two values is saved as working area of the block.

ACTION11,-16,-18,-19,-22,-24,-28

The input element is skipped.

ACTION15

This routine is called if the input element replaces an intermediate result (EF-key). The element is moved to the variable area and the corresponding stack address is evaluated and moved into bytes 2-5 of the variable.

ACTION25

Switch ACT10SW is set to and the routine proceeds similar to ACTION29.

ACTION26, ACTION27

The output is modified for array functions and the element is skipped.

ACTION29

The input element is skipped.

ACTION30

The specified registers are freed (by calling ACOMAX) if they are occupied.

ACTION31

When the end of the text string is detected, this routine terminates phase D10 and calls the interface macro IJKPH to fetch phase D11.

PART 2 OF D10 (PREMAC) -- GF

Part 2 of phase D10 (entry point PREMAC) is called if an operation is detected in the input stream. It prepares the macro instructions. This includes the processing of assignments, conversion of the operands to the required data types, moving of operands to the required storage types, conversion of floating-point operands from short float to long float, and to make requests for additional required operands. The routines called for performing the individual functions are discussed in later sections.

The general input format of this routine is as follows:

```
 (1) (1) (1) (1) (1) (18)
┌───┬───┬───┬───┬───┬────┐───
│K  │ N │K1 │K2 │K3 │OP1 │
└───┴───┴───┴───┴───┴────┘───


     (18) (1)  (1)  (2)          (2)
───┌────┬───┬────┬───┐─── ───┬────┬───┐
   │OPn │EE │M   │AR1│       │ARm │   │
───└────┴───┴────┴───┘───────┴────┴───┤
         └──────────────V──────────────┘
                     18 bytes
```

K =	X'E5' for macro instructions
	X'EC' for functions
N	Number of operands
OP	Operand (see detailed description below)
EE	X'EE'
M	Number of additional requests
AR	Additional request

The format of the operands is:

Byte(s)	Contents
1	Operand key
2-3	Name of the operand
4-5	Modifier
6	Attribute byte = byte 5 of SYMTAB entry
7	Attribute byte specifying data type of operand
8	Length of the operand
9	Precision of operand, or length in bytes (bits) for character (bit) string
10	Scale factor of operand
11-12	Offset in constant table pointing to corresponding constant if operand key E9 or 69. Offset in character string pointing to corresponding ED if byte 13 contains 8, 10, or 11. Name of corresponding DED if byte 13 contains 4 or 5.
13	Number giving available data type of operand.
14	Number giving required data type of operand.
15	Length of operand required after conversion
16	Precision of operand required after conversion
17	Scale factor of operand required after conversion
18	Required storage type

The output of the routine PREMAC may consist of the following:

1. Macro instructions

Byte(s)	Contents
1	key X'F2'
2-3	length of macro instruction.
4	key; for the format of the following bytes see phase E50 for the respective key.

2. Assembler instructions

Byte (s)	Contents
1	key X'F6'
2-3	length of instruction
4-5	key1, key2; for format see phase E50 for the respective keys.

3. Constants

Byte (s)	Contents
1	key X'FD'
2-3	length of constant; constant itself is taken unchanged from output of phase D05.

4. Premacros

Byte (s)	Contents
1	key X'F5' or X'FC'
2-3	length fo premacro; 1 = 12 (N+1) +22 C
4	number of operands
5-7	K1, K2, K3 (see input)
8-12	not used

Each of the following operands is 12 bytes long. A constant (22 bytes) may be appended to each operand.

		0	1	2	3	4	5	6	7	8	9	10	11
		BIN. FLOAT	BIN. FIXED	DEC. FLOAT	DEC. FIXED	ZONED DEC.	ZONED DEC (T)	CHAR STR.	BIT STR.	STERL. N.F.	BIN. INT.	FLOAT N.F.	FIXED N.F.
0	BIN. FLOAT	-	4/13	1/3	4/15	3/9	3/10	0/0	0/28	4/19	0/7	5/23	5/24
1	BIN. FIXED	4/12	-	1/3	4/15	3/9	3/10	0/0	0/28	4/19	1/12	5/23	5/24
2	DEC. FLOAT	1/2	4/13	-	4/15	3/9	3/10	0/0	0/28	4/19	0/7	5/23	5/24
3	DEC. FIXED	1/2	4/14	4/14	-	3/9	3/10	0/0	0/28	4/19	7/18	4/21	5/24
4	ZONED DEC.	1/2	4/13	4/14	3/5	-	1/30	0/0	0/28	4/19	7/18	5/23	5/24
5	ZONED DEC (T)	1/2	4/13	4/14	3/6	1/29	-	0/0	0/28	4/19	7/18	5/23	5/24
6	CHAR. STRING	0/0	0/0	0/0	0/0	1/1	1/1	-	4/17	1/1	0/0	1/1	1/1
7	BIT STRING	4/12	0/11	1/3	2/4	3/9	3/10	4/16	-	4/19	7/8	4/22	5/24
8	STERL. N.F.	1/2	4/13	4/14	6/27	3/9	3/10	0/0	0/28	-	7/18	5/23	5/24
9	BIN. INT.	4/12	0/11	1/3	2/4	3/9	3/10	0/0	0/28	4/19	-	4/22	5/24
10	FLOAT N.F.	1/2	4/13	6/25	4/15	3/9	3/10	0/0	0/28	4/19	0/7	-	5/24
11	FIXED N.F.	1/2	4/13	4/14	6/26	3/9	3/10	0/0	0/28	4/19	7/18	5/23	-

Figure 1. Conversion Table COTAB

NR.	CONVERSION	Type returned*	New. Attr.**	Macro Key	Routine Name	Ind. Rout. Name
1	ZONED DEC. - CHAR. STR.	6	6	-	-	
2	BIN. FL. - DEC. FL.	2	2	-	-	
3	DEC. FL. - BIN. FL.	0	0	-	-	
4	DEC. FIX - BIN. INT.	9	1	50	-	
5	DEC. FIX - ZON. DEC.	4	4	52	-	
6	DEC. FIX - ZON. DEC. (T)	5	5	51	-	
7	BIN. INT - BIN. FL.	0	0	53	-	
8	BIN. INT - BIT. STR	7	7	54	-	
9	ZON. DEC - DEC. FIX	3	3	55	-	
10	ZON. DEC T - DEC. FIX	3	3	55	-	
11	BIN. FIX - BIN. INT	9	1	06	-	
12	BIN. FL. - BIN. FIX BIN. INT. - BIN. FIX	1	1	41 06	64 -	
13	BIN. FIX - BIN. IFL	0	0	41	65	
14	DEC. FL - DEC. FIX	3	3	41	41	51
15	DEC. FIX - DEC. FL	2	2	41	42	50
16	CHAR.STR. - BIT. STR	7	7	41	45	
17	BIT. STR - CHAR. STR	6	6	41	46	
18	BIN. INT - DEC. FIX	3	3	57	-	
19	STERL. N.F. - DEC. FIX	3	3	41	67	
21	FLOAT N.F. - DEC. FIX	3	3	41	43	51
22	FLOAT N.F. - BIN. INT	9	1	41	43	49
23	FLOAT N.F. - DEC. FL.	2	2	41	43	50
24	FIX. N.F. - DEC. FIX.	3	3	42	68	-
25	DEC. FL. - FLOAT. N.F.	10	12	41	41	52
26	DEC. FIX - FIX. N.F.	11	13	42	69	-
27	DEC. FIX - STERL. NF	8	18	42	70	-
28	BIT. STR. - BIN. INT	9	1	56	-	-
29	ZON. DEC. - ZON. DEC (T)	5	5	-	-	-
30	ZON. DEC (T) - ZON. DEC	4	4	B6	-	-
0	ERROR	0	-	-	-	-

*Identical with ROUTAB. **Identical with ATAB.

Figure 2. Routine Table

CONVERT -- GG

This routine performs the conversion of operands from available to required data type. Most of the conversions are performed in single steps.

The conversion table COTAB (see Figure 1) is used to determine the conversion steps required. The elements V and W in column Y and row X in COTAB give the first conversion step for a conversion from available data type (Y) to required data type (X). V indicates the routine of this part of the phase (CACTION(V)) that handles the required conversion step. W refers to the routines that represent the individual conversion steps. The data type Y' obtained after conversion step W is given as element (w) of ROUTAB. The new attribute of the operand is given as element (W) of ATAB. ROUTAB and ATAB are identical with columns 3 and 4 of the routine table (see Figure 2).

The conversion is continued with further conversion steps until the new data type Y' equals the required data type X.

ADDPQL

This routine determines P, Q, and L of the target depending on the routine number W. The table shown in Figure 3 gives the formulas to be evaluated for the routines.

LS, PS, QS	L, P, Q of the source
LT, PT, QT	L, P, Q of the target
LC, LB	length of character or bit string (for source (S) and target (T), respectively.

The routine number specified in the lefthand column of Figure 3 is the same as the number specified in the lefthand column of Figure 2.

Rout. number	EVALUATION OF LT, PT, QT
3	no evaluation (error)
1	LCT=PS, LT=LCT

2	PT=CEIL (PS/3.32)
	if PT > 6: LT=8,QT=X'58'
	if PT ≤ 6: LT=4, QT=X'00'

3	PT=MIN (CEIL(PS*3.32),31
	if PT > 21: LT=8, QT=X'58'
	if PT ≤ 21: LT=4, QT=X'00'

4	PT=31, LT=4, QT=0
5	PT=PS, QT=QS, LT=PT
6	PT=PS, QT=QS, LT=PT
7	PT=PS, QT=0, LT=4
8	LBT=PS, LT=CEIL(LBT/8)
9	PT=PS, QT=QS, LT=FLOOR(PT+2)/2)
10	PT=PS, QT=QS, LT=FLOOR(PT+2)/2)
11	PT=31, QT=0, LT=4
12	PT=PS, QT=0, LT=4

13	PT=PS
	if PT > 21: LT = 8, QT =X'58'
	if PT ≤ 21: LT = 4, QT =X'00'

14	PT=PS, QT=0, LT=FLOOR(PT+2)/2)

15	PT=PS
	if PT > 6: LT = 8, QT = X'58'
	if PT ≤ 6: LT = 4, QT = X'00'

16	LBT=MIN(LCS*8,64), LT=MIN(LS,8)
17	LCT=LBS, LT= LCT

18	PT=CEIL (PS/3.32) + 1,
	QT=0, LT=FLOOR(PT+2)/2)

19	PT=PS, QT=QS LT=FLOOR(PT+2)/2)
20	no evaluation (no routine)
21	PT=PS, QT=0, LT=FLOOR(PT+2)/2)
22	PT=31, QT=0, LT=4

23	PT=PS
	if PT > 6: LT = 8, QT = X'58'
	if PT ≤ 6: LT = 4, OT = X'00'

24	PT=PS, QT=QS, LT=FLOOR(PT+2)/2)
25 26 27	must be assignment, PT, QT, LT are given
28	PT=31, QT=0, LT=4

Figure 3. Evaluation of L, P, and Q of Target (Part 1 of 2)

Figure 3. Evaluation of L, P, and Q of Target (Part 2 of 2)

ADASSI(0), -(2) -- GI

The routine processes assignments to binary float (X = 0) and decimal float (X = 2).

ADASSI(1) -- GJ

The routine processes assignments to binary fixed. If the source alos has the attributes binary fixed, the assignment is identical with conversion CV36.

ADASSI(3) -- GK

The routine processes assigments to decimal fixed. If the source also has the attributes decimal fixed, processing is continued by the part of the phase that prepares the input for the macro generation. The final assignment macro is then constructed in the following phase.

ADASSI(4), -5, -8, -11 -- GL

This routine processes the assignments to zoned decimal (X = 4), zoned decimal T (X = 5), sterling numeric field (X = 8), and fixed numeric field (X = 11). The assignment is performed in two steps: first to decimal fixed and then final assignment to the required data type.

If the initial assignment was

```
where ATT = 0, 1, ...  11
      ATS = 4, 5, 8, or 11
```

it is separated into

```
      GW0(3,LGW0,PT,QT)=S(ATS,LS,PS,QS);
      T(ATT,LT,PT,QT)=GW0(3,LGW0,PT,QT)
```

where LGW0 = FLOOR((P+2)/2)

The switch ADASCO separates the first assignment (GW0 = S) and the second assignment (T = GW0). The initial value of the switch is zero. Thus, the NO-branch refers to the first and the YES-branch to the second assignment.

If the attributes of source and target are equal, the assignment is identical to a character-string assignment.

ADASSI(6),-7 - -GM

The routine processes assignments to character string (X = 6) and bit string (X = 7).

ADASSI(9), -10 -- GN

The routine processes assignments to binary integer (X = 9) and floating-point numeric field (X = 10). If source and target have the attributes floating-point numeric field, the assignment is identical to action(4) of the conversion routine after setting the parameters V and W and changing a SECTAB entry.

ADTEE6 -- GO

The subroutine is used for replacing indirectly given operands. If the macro instruction M (OP) has the operand OP, OP may be one of the following:

1. a normal operand (no action required)
2. a variable or constant that must be taken only by its value.
   The instruction

   ```
   MVC X(L), OP
   ```

   is generated and OP is replaced by X.

   ```
   M(OP)  M(X)
   ```

3. a call without parameters. The call macro

   ```
   CALL (OP,RE)
   ```

   where OP = entry name and RE = return value is generated and OP is replaced by RE.

   ```
   M(OP)  M(RE)
   ```

ADCOB3 -- GP

This routine is used for initializing pointers. Z1 points to the start address of the text element with an E5 key. Z2 points to the first operand. N specifies the number of operands.

CACTION0 -- GQ

The routine performs three conversion steps. Parameter W specifies the type of conversion. W = 0 indicates an invalid conversion. For W greater than zero see the routine table (Figure 3). ADMVC is a subroutine used for increasing and testing the variable counter IJKMVC.

CV36 -- GR

This subroutine is used for the construction of binary assignment macros. If the preceding statement identifier contained an indication that size overflow must be checked, the macro key X'05' is used. Otherwise, the key X'06' is used.

CACTION1 -- GS

This routine performs the conversion steps that require no transfer of the operand to another address. Thus, the requested target field may be freed. For the individual conversion steps refer to the routine table (see Figure 3).

CACTION2 -- GT

This routine converts from decimal fixed to binary integer.

CACTION3 -- GU

This routine performs the conversion steps between zoned decimal (T) and decimal fixed. Refer to the routine table in Figure 3.

CACTION4, -5, -6 -- GV-GX

This routine performs the conversion steps that use library routines. Two types of macro instructions may be constructed:

```
1.  (1)  (2)   (1)  (1)  (1)  (6)  (6)   (2)   (2)
    ┌──┬────┬───┬───┬───┬───┬───┬────┬────┐
    │F2│0016│41 │ I │RN │OP1│OP2│DED1│DED2│
    └──┴────┴───┴───┴───┴───┴───┴────┴────┘

2.  (1)  (2)   (1)  (1)  (1)  (6)  (6)  (2)
    ┌──┬────┬───┬───┬───┬───┬───┬───┐
    │F2│0014│42 │ I │RN │OP1│OP2│DED│
    └──┴────┴───┴───┴───┴───┴───┴───┘
```

Format 1 refers to CACTION4 and is used for calling library routines that require two DEDs.

Format 2 refers to CACTION5 and CACTION6. For CACTION5, DED must be the DED of OP1. For CACTION6, DED must be the DED of OP2. RN is the routine name.

In addition to the construction of the required macro instruction, the routine sets the bits for the routine name and for the indirectly used library routine. The routine name and the name of the indirectly used library routine are given in columns 5 and 6 of the routine table (see Figure 3).

DEROUT0-3,-6,-7 -- GY

These routines generate the DEDs. For coded arithmetic and string data the DEDs differ in their length and key (L and K) only.

CACTION7 -- GZ

This routine performs two conversion steps (see routine table in Figure 3).

ACOMA -- HA-HC

This part of the program is no subroutine but a narrative description of that part of PREMAC that constructs the premacros (the YES-tree after the decision box D3 in flow chart GF. The following actions may be required:

1. Additional request.
   Besides the operands, some macros and functions require additional registers or working storage. This additional storage must be "requested".

2. Conversion of float operands.
   Floating-point operands of float macros or float functions must have the same length. If one of the operands is long float, all other float operands that are not long float must therefore be converted to long float.

3. Storage type conversion.
   Byte 17 of the specifies the required storage type. The operand gets the required storage type. This may cause a conversion if the operand already was assigned some storage type.

4. Putting out premacros.
   The premacros are put out in the format described under Output of PREMAC.

ADCOST -- HD

This subroutine converts operands from short float to long float.

ADDEQ

This subroutine is used to determine the sign of a scale factor.

PART 3 OF D10

Part 3 of phase D10 consists of a collection of subroutines called by part 2 to determine the operands required to construct the macro instructions. It checks the storage type required for the operand(s). If required, registers are freed and working storage is generated.

The technique applied is the so-called stack technique. This technique assumes that two stack chains exist at any moment during processing: a free chain and an occupied chain. If stacking is required, the last item of the free chain is deleted and added (with the corresponding information) as the last item to the occupied chain. Unstacking is the reverse procedure of stacking.

The individual subroutines forming part 3 are discussed in the following.

SOURCE(I)

SOURCE(I) may be one of 0, 1, 2, 3, and 4. The routines are called if a source operand is to be converted to a required storage type. The routines merely provide the parameters for the routine SOURCE. The required storage type for SOURCE and the corresponding parameters for the individual routines are listed in the following table:

Routine	Required Storage Type	Parameter
SOURCE0	Any of 1, 2, 3, 4	X'0F00'
SOURCE1	Declared or working storage	X'0C00'
SOURCE2	Fixed register	X'0100'
SOURCE3	Float register	X'0207'
SOURCE4	Working storage	X'0400'

TARGET(I)

TARGET(I) may be one of 0, 1, 2, 3, and 4. The routines are called if a target operand is to be converted to a required storage type. The routines merely provide parameters for the routine TARGET. The required storage type for TARGET and the corresponding parameters for the individual routines are analogous to those of SOURCE(I).

SOURCE

The routine evaluates the operand to be used in an operation as a reference to a source field. It may be called from one of the routines SOURCE0 through SOURCE4 as follows:

```
BAL       LINK, SOURCE
DC        X'mmnn'
```

where mm and nn are the parameters required for the routine GETOP, which performs the main part of the processing.

TARGET

This routine is called by one of the routines TARGET0 through TARGET4 and is similar to SOURCE. The operands are used target fields.

The main part of the processing is performed by the called routine GETOP.

GETOP -- EX

This routine determines an operand with specified storage type (working storage, fixed-point register, or floating-point register). The routine checks if the type required and the type currently available are identical. If they are, the routine is left. Otherwise, REQUEST is called to generate an operand that has the required storage type and to store the contents of the current operand in the generated operand.

This routine is called by SOURCE as follows:

```
BAL       LINK, GETOP
DC        X'mmnn'
```

where mm may be

X'0F'	any type
X'0C'	no register type
X'01'	fixed-point register
X'02'	floating-point register
X'04'	working storage

and nn is either X'07' (floating-point register) or X'00' (all other cases).

FREE -- ET

The routine frees an operand in the occupied stack chain and adds it to the free stack chain.

FREEING

This routine is called after the macro generation for each operation. It frees all operands used as source fields during the respective operation if these operands are work areas that have entries in the occupied stack chain.

REQUEST -- ES

This routine is called to get the work area, which may be specified as working storage, fixed-point register, or floating-point register. It is called as follows:

```
BAL       LINK, REQUEST
DC        X'mmnn'
```

where mm is one of the following:

00	single fixed-point register
01	double fixed-point register
02	short floating-point register
03	long floating-point register
04	full-word
05	double-word
06	byte-aligned working storage
07	specified single fixed-point register
08	specified double fixed-point register
09	not used
0A	specified short floating-point register
0B	specified long floating-point register

nn is the specified register or the length in bytes of the required storage.

GETST -- ER

The routine deletes the last item in the free stack chain and adds it to the occupied stack chain.

FETCHA, FETCH1 -- EU

This routine is called by REQUEST to select the parameters for the current request. Selection is performed using the tables D and P. The table P consists of 9-byte entries that have the following format:

byte 0	mask for searching in REGTAB
byte 1	successful condition code during searching
byte 2	successful action displacement
byte 3	unsuccessful action displacement
byte 4	mask to reserve in REGTAB
byte 5	mask to free in REGTAB
byte 6	macro instruction key if storage change is required
bytes 7-8	request, if required.

The register and working storage table REGTAB consists of full-word entries of the following format:

byte 0	search mask (key). It may be one of the following: X'03' fixed-point register X'0C' floating-point register X'10' double-word X'0E' byte-aligned
byte 1	displacement in QTAB
bytes 2-3	either register pair or offset if storage

The table QTAB contains the relative displacement (1 byte) for entries in RTAB. The entries in RTAB have the following format:

byte 0	displacement of operand format relative to S00
byte 1	displacement (relative to R012) of routine to construct and return the operand. <u>The routine is executed</u>.
bytes 2-3	the required request parameters (if available and required type are not identical)
byte 4	macro instruction key if storing is required (if available and required type are not identical)

byte 5	available storage type X'mn', where m and n have the following meaning:
	n = 1 - fixed-point register
	n = 2 - floating-point register
	n = 4 - working storage
	n = 8 - declared variable
	m = 0 - short
	m = 1 - long
	m = 2 - byte-aligned

byte 6	X' ' for floating-point data X' ' in all other cases

The communication between the reference to the working area and the tables REGTAB, QTAB, and RTAB is as shown in Figure 2.

<u>ACTIO(R5)</u>

The selection of the routine to be used depends on the address contained in register 5. One of the four routines described in the following may be selected.

<u>SACTIONX -- EU</u>

This routine is called if the required and available types of an operand are identical (successful action, see format of the P table). In this case, the operand is a register or working storage.

<u>SACTIONZ -- EV</u>

This routine is called if an operand is required as register, but it is currently contained in storage. In this case, a register (or a pair of registers) is requested by calling REQUEST with the appropriate parameters, and a load macro instruction is generated.

<u>FACTIONY -- EU</u>

This routine is called if working storage on full-word or double-word boundary is required and this is not available in REGTAB.

<u>SACTION7 -- EW</u>

This routine is called if a specified register is required.

First Part

The first part of phase D11 restores the old interface for use by this and the following phases. The new communication region is saved. The old interface and the saved LIOCS table for SYS001 are read from SYS001. The interface is read into the appropriate storage area, whereas the LIOCS table must be moved. If tape work files are used, a back-space record command is given to synchronize the tape position with the information in the table.

All items of the old communication region that may have been changed in the new communication region by phases D00, D05, or D10 are set to the new values. The text input medium is read to the end of information, and the NOTE information on that point is set into the old communication region. The medium is reset with a POINTS macro instruction. The old end-of-file address is set into the appropriate entries of the work file tables.

Symbols used in flow charts

TABLE	: Contains address of SYS001 in new interface
TABLEM	: Save area in phase D11 for information required from new interface
INTTABEN:	Begin of saved new communication region
EOFADD	: Relative address of end-of-file entry in LIOCS table for disk
EOFADT	: and for tape work files
TEXTIN	: Contains address of input work file table
NEOFAD	: Address of end-of-file routine used in this phase
POINTR	: LIOCS macro instruction
T	' SYS001
CHECK	: LIOCS macro instruction
READ	: LIOCS macro instruction
EXCP	: PIOCS macro instruction
BSR	: Backspace record
TASAVA	: Area where SYS001 table is read in
WAIT	: PIOCS macro instruction
IJKMVC	: Variable counter, entry in old interface
IJKMLB	: Library usage bytes, entry in old interface
ADLIBIN	: Part of IJKMLB in new interface.
IJKMJT	: Job communication bytes, entry in old communication region
READ	: LIOCS macro instruction
IJKMBL	: Block on work files, entry in old communication region
NOTE	: Contains information on input work file in old interface
POINTS	: LIOCS macro instruction
DUMPSAVE:	Save area for old end-of-file routine
TEXTOUT	: Contains address of output work file table.
IJKMWC	: Length of dynamic work space used in D00-D10, entry in old interface
LREAD	: Length that can be read regardless of buffer boundaries
LWRITE	: Length that can be written regardless of buffer boundaries
PIN	: Input pointer
OPR	: Output pointer
CURBUF	: Current buffer index
K	: Key
MA1	: Area used for macro construction
AA30	: Area containing the premacro
AAP3	: Area containing P and Q of operands

Main Part

The second part of phase D11 generates macros. The input of the phase contains so-called premacros, which furnish the information required for generation of the macros. The format of the generated macros must be the same as required by the code generation phases E50 and E60. The phase also generates compiler constants.

Input

The input to phase D11 may include the following elements:

1. Statement identifier (6 bytes) with the key X'E0'.

2. I/O text (12 bytes) with the key X'E1' or X'E4'.

3. End of statement (6 bytes) with the key X'EA'.

4. Error indicator (2 bytes) with the key X'EB'.

5. Premacros

Byte(s)	Contents
1	K = key X'F5' or X'FC'. If K = X'FC' and K2 is greater than X'40', the text element represents a function and is moved unchanged into the output buffer.
2- 3	Length of premacro 1 = 12 (N+1) +22 C, where C = number of constants.

4	Number of operands (N).
5	K1
6	K2
7	K3
8-12	Not used

These 12 bytes are followed by the operands, each of which has a length of 12 bytes. Each operand may be followed by a 22-byte constant.

6. Permutable text elements

Byte(s)	Contents
1-12	Text element 1
13	Key X'FE'
14-16	Not used
17-28	Text element 2

Text element 1 is exchanged for text element 2.

7. End of text string, key X'FF'.

8. Further F keys

Byte(s)	Contents
1	F-key other than those described above.
2-3	Length to be skipped (including bytes 1-3).
4-n	Element to be skipped.

Output

The output of phase D11 contains the same number of elements as the input. However, the following additions and permutations have been performed:

1. After the statement identifier indicating an entry statement, the assembler code CNOP 0,4 has been inserted into the text string.

   Format of the code:

   X'F6000780C00004'

2. After the end of block statement, the assembler code

   END OF BLOCK   CNOP 0,4

   has been inserted into the text string. Format of the code;

   X'F6000380C6000080C00004'

3. The text element following the FE-key is exchanged with the text element preceding the FE-key. The FE-key has been changed to X'F7'. (See Input.)

4. Premacros (see Input) have been processed to macros and constants.

   Format of the macros

Byte(s)	Contents
1	X'F2'
2-3	Length to be skipped
4-n	Element to be skipped

   Format of the constants

Byte(s)	Contents
1	X'F3'
2-3	Length to be skipped
4-n	Constant to be skipped

5. Compiler constants have been inserted at the beginning of the text string.

DESCRIPTION OF ROUTINES

First, the phase evaluates the buffer addresses and the buffer lengths. Then the input buffer is filled with input text. The first output moved into the output buffer consists of compiler constants. For the scan of the text string the routine FETCH is used. The processing of the individual input elements is performed by the corresponding ACTION routines.

The routine MOVESO is discussed in the description of phase D00.

Symbols used in flow charts:

LREAD	:	Length that can be read regardless of buffer boundaries
LWRITE	:	Length that can be written regardless of buffer boundaries
PIN	:	Input pointer
OPR	:	Output pointer
CURBUF	:	Current buffer index
K	:	Key
MA1	:	Area used to construct the macro
AA30	:	Area containing the premacro
AAP3	:	Area containing P and Q of the operands

FETCH -- HN

The routine FETCH scans the text string, and determines the subscript for calling the appropriate ACTION routine for the individual input elements.

ADMVC -- HO

ADMVC increments and tests the variable counter of the compiler. If more than 641536 variables are counted, the current statement is skipped and an error message is inserted into the text string.

ACTION0 -- HP

The statement identifier is saved by STIN. If this statement identifier indicates an END statement, switch ACT10 is set to zero. If this statement identifier indicates an entry statement, CNOP 0,4 is generated.

ACTION1, -4 -- HQ

The text element is either an operand (which is moved unchanged into the output buffer) or information on I/O statements (which is required for the routine ADIOST (see MASU(X)).

ACTION2, 3, 5-9, 12-15, 25, 27

These routines are not available sincethe corresponding text elements cannot occur in the text string.

ACTION10 - HR

This routine is used when the end of a statement is detected. The text element is moved unchanged into the output buffer. If the statement was an end-of-block (ACT10=0), the assembler code

'END OF BLOCK' CNOP 0,4

is generated.

ACTION11, 16-20, 22-24, 26, 29

The text element is moved unchanged into the output buffer.

ACTION28 -- HS

The text element is either a function call (moved unchanged into the output buffer) or a decimal arithmetic macro (processed like in ACTION21)

ACTION30 -- HT

The permutable text element is exchanged as described under Input.

ACTION31 -- HU

This routine detects the end of the input text and initiates the calling of the next phase.

ACTION21 -- HV, HW

ACTION(21) performs the main objective of phase D11, the generation of macros.

HV/D3	If the macro to be generated is the decimal compare macro, the number of operands N will be set to 3. The premacro contains two operands but the macro must have three. The third operand OP1 must not contain any information.
HV/F4	The pointers are set: P1 pointer to the macro generation area, P2 pointer to the premacro (input), P3 pointer to P-Q-save area.
HV/H4	Byte 8 of the operand is extended to one word and stored in P3.
HV/J4	Byte 9 of the operand is extended to one word and stored in P3+4.
HV/J1	If SWY is unequal to 0, OP1 was followed by a constant or the macro contains more than one operand. If OP(1) is not followed by a constant, the bytes addressed by P2+7 and AA30+31 must have the same contents: because in the routine MASU(X) AA30+31 will be used for P2+7. If the contents of these bytes are not identical to each other, one byte is moved.
HW/B3	SW1=15: a decimal macro is generated.
HW/B4	Normally, the operands of the premacro are given in the sequence OP(N), OP(2), OP1. The operands of the decimal premacros are given in the sequence OP(1), OP(N) OP(2) (N = 2 or 3). Therefore, the operand in the macro generation area and P, Q in P-Q-save area must be rearranged.
HW/E3	X is determined by using table TAB1. If X exceeds 18, the appropriate element of TAB1 contains an offset pointing to routine MASU(X), which then constructs the macro. If X does not exceed 18, the appropriate element of TAB1 contains the length of the macro.

ADMACO -- HX

Routine ADMACO moves the constants following the operands of a premacro into the output buffer.

MASU(X)

The routines MASU(X) refer to several macro keys. The relation between the macro key and the routine name is given by the table TAB1 shown in Figure 1.

Macro Key	MASU (X)	MACRO
0	DC AL1 (ADBASC-MASU)	BINARY ADDITION
1	DC AL1 (ADBASC-MASU)	BINARY SUBTRACTION
2	DC AL1 (16)	BINARY MULTIPLICATION WITH O.C.
3	DC AL1 (ADBOIV-MASU)	BINARY DIVISION
4	DC AL1 (10)	BINARY NEGATION
5	DC AL1 (ADASSI-MASU)	ASSIGNMENTS
6	DC AL1 (4)	BINARY ASSIGNMENT WITHOUT O.C.
7	DC AL1 (4)	BINARY EXPONENTIATION
8	DC AL1 (ADBASC-MASU)	BINARY COMPARISON
9	DC AL1 (ADMULI-MASU)	BINARY MULTIPLICATION WITHOUT O.C.
10	DC AL1 (4)	FREE
11	DC AL1 (4)	FREE
12	DC AL1 (ADIOST-MASU)	
13	DC AL1 (4)	FREE
14	DC AL1 (4)	FREE
15	DC AL1 (4)	FREE
16	DC AL1 (ADDARI-MASU)	DECIMAL ADDITION
17	DC AL1 (ADDARI-MASU)	DECIMAL SUBTRACTION
18	DC AL1 (ADDMUU-MASU)	DECIMAL MULTIPLICATION
19	DC AL1 (ADDIV-MASU)	DECIMAL DIVISION
20	DC AL1 (ADDNEG-MASU)	DECIMAL NEGATION ONE OP.
21	DC AL1 (ADDSHI-MASU)	DECIMAL ASSIGNMENT
22	DC AL1 (ADNEG1-MASU)	DECIMAL NEGATION,TWO OP.
23	DC AL1 (4)	DECIMAL EXPONENTIATION
24	DC AL1 (ADDARI-MASU)	DECIMAL COMPARISON
25	DC AL1 (4)	FREE
26	DC AL1 (4)	FREE
27	DC AL1 (4)	FREE
28	DC AL1 (4)	FREE
29	DC AL1 (4)	FREE
30	DC AL1 (4)	FREE
31	DC AL1 (ADUNA-MASU)	UNARY PLUS
32	DC AL1 (16)	SHORT FL.ADDITION
33	DC AL1 (16)	SHORT FL.SUBTRACTION
34	DC AL1 (16)	SHORT FL.MULTIPLICATION
35	DC AL1 (16)	SHORT FL.DIVISION
36	DC AL1 (16)	SHORT FL.NEGATION,2OP.
37	DC AL1 (10)	SHORT FL.NEGATION,1 OP.
38	DC AL1 (16)	SHORT FL.ASSIGNMENT
39	DC AL1 (4)	SHORT FL.EXPONENTIATION
40	DC AL1 (16)	SHORT FL.COMPARISON
41	DC AL1 (4)	SHORT FL.GENERAL EXPONENTIATION
42	DC AL1 (4)	FREE
43	DC AL1 (4)	FREE
44	DC AL1 (4)	FREE
45	DC AL1 (4)	FREE
46	DC AL1 (4)	FREE
47	DC AL1 (4)	FREE
48	DC AL1 (16)	LONG FL.ADDITION
49	DC AL1 (16)	LONG FL.SUBTRACTION
50	DC AL1 (16)	LONG FL.MULTIPLICATION
51	DC AL1 (16)	LONG FL.DIVISION
52	DC AL1 (16)	LONG FL.NEGATION,2 OP.
53	DC AL1 (10)	LONG FL.NEGATION 1 OP.
54	DC AL1 (16)	LONG FL.ASSIGNMENT
55	DC AL1 (4)	LONG FL.EXPONENTIATION
56	DC AL1 (16)	LONG FL.COMPARISON
57	DC AL1 (4)	LONG FL.GENERAL EXPONENTIATION
58	DC AL1 (4)	FREE
59	DC AL1 (4)	FREE
60	DC AL1 (4)	FREE
61	DC AL1 (4)	FREE
62	DC AL1 (4)	FREE

63	DC	AL1 (4)	FREE
64	DC	AL1 (ADCHOP-MASU)	CHAR.STR.CONCATENATION
65	DC	AL1 (4)	CONVERSION,ACTION 4
66	DC	AL1 (4)	CONVERSION,ACTION 5
67	DC	AL1 (4)	CONVERSION, ACTION 6.
68	DC	AL1 (4)	FREE
69	DC	AL1 (4)	FREE
70	DC	AL1 (4)	FREE
71	DC	AL1 (4)	FREE
72	DC	AL1 (4)	FREE
73	DC	AL1 (4)	FREE
74	DC	AL1 (4)	FREE
75	DC	AL1 (4)	FREE
76	DC	AL1 (4)	FREE
77	DC	AL1 (4)	FREE
78	DC	AL1 (4)	FREE
79	DC	AL1 (4)	FREE
80	DC	AL1 (4)	CONVERSION 0
81	DC	AL1 (4)	CONVERSION 1
82	DC	AL1 (4)	CONVERSION 2
83	DC	AL1 (4)	CONVERSION 3
84	DC	AL1 (4)	CONVERSION 4
85	DC	AL1 (4)	CONVERSION 5
86	DC	AL1 (4)	CONVERSION 6
87	DC	AL1 (4)	CONVERSION 7
88	DC	AL1 (ADCHAP-MASU	CHAR,STR,COMPARISON
89	DC	AL1 (16)	SHIFT AR.SINGLE RIGHT
90	DC	AL1 (16)	SHIFT AR.SINGLE LEFT
91	DC	AL1 (16)	SHIFT AR.DOUBLE RIGHT
92	DC	AL1 (16)	SHIFT AR. DOUBLE LEFT
93	DC	AL1 (4)	
94	DC	AL1 (4)	FREE
95	DC	AL1 (4)	FREE
96	DC	AL1 (4)	FREE
97	DC	AL1 (4)	FREE
98	DC	AL1 (4)	FREE
99	DC	AL1 (ADBSIP-MASU)	BIT STR.NOT,2 OP.
100	DC	AL1 (ADDING-MASU)	BIT STR.NOT,1 OP.
101	DC	AL1 (ADBISA-MASU)	BIT STR.ASSIGNMENT
102	DC	AL1 (ADBSOP-MASU)	BIT STR.AND
103	DC	AL1 (ADBSOP-MASU)	BIT STR.OR
104	DC	AL1 (ADBASA-MASU)	BIT STR.COMPARISON
105	DC	AL1 (16)	SHIFT LO.SINGLE RIGHT
106	DC	AL1 (16)	SHIFT LO.SINGLE LEFT
107	DC	AL1 (16)	SHIFT LO.DOUBLE RIGHT
108	DC	AL1 (16)	SHIFT LO.DOUBLE LEFT
109	DC	AL1 (4)	FREE
110	DC	AL1 (4)	FREE
111	DC	AL1 (4)	FREE
112	DC	AL1 (4)	BRANCH ON CONDITION
113	DC	AL1 (ADRTLC-MASU)	RETURN TO LABEL CONSTANT
114	DC	AL1 (7)	DEFINE LABEL
115	DC	AL1 (4)	FREE
116	DC	AL1 (4)	FREE
117	DC	AL1 (ADRTLC-MASU)	ASSIGN LABEL CONST.
118	DC	AL1 (16)	POINTER ASSIGNMENT
119	DC	AL1 (4)	FREE
120	DC	AL1 (16)	POINTER COMPARISON
121	DC	AL1 (4)	FREE
122	DC	AL1 (ADIF-MASU)	IF-MACRO
123	DC	AL1 (4)	FREE
124	DC	AL1 (4)	FREE
125	DC	AL1 (ADUNA-MASU)	UNARY PLUS
126	DC	AL1 (4)	FREE
127	DC	AL1 (4)	FREE
128	DC	AL1 (11)	RETURN TO 1.VARIABLE

129	DC	AL1(10)	RETURN TO 1.VARIABLE
130	DC	AL1(4)	FREE
131	DC	AL1(16)	FREE
132	DC	AL1(16)	FREE
133	DC	AL1(11)	ASSIGN LABEL /.
134	DC	AL1(ADBASA-MASU)	ASSIGN CHAR.STR.
135	DC	AL1(4)	FREE
136	DC	AL1(4)	FREE
137	DC	AL1(4)	FREE
138	DC	AL1(4)	FREE
139	DC	AL1(4)	FREE
140	DC	AL1(4)	INITIAL
141	DC	AL1(4)	FORMAT
142	DC	AL1(4)	LOAD TRANSMIT
143	DC	AL1(4)	FREE
144	DC	AL1(ADCALL-MASU)	CALL
145	DC	AL1(4)	RETURN
146	DC	AL1(4)	PROLOGUE
147	DC	AL1(16)	STM
148	DC	AL1(16)	STD
149	DC	AL1(16)	MOVE ADDRESS
150	DC	AL1(4)	DO BEGIN
151	DC	AL1(4)	DS
152	DC	AL1(4)	ARRAY EXPR.BEGIN
153	DC	AL1(4)	FREE
154	DC	AL1(4)	FREE
155	DC	AL1(4)	FREE
156	DC	AL1(4)	LOAD VARIABLE
157	DC	AL1(4)	SET BYTE
158	DC	AL1(4)	MOVE
159	DC	AL1(4)	FREE
160	DC	AL1(4)	LIBRARY CALL (1)
161	DC	AL1(4)	RETURN FUNCTION VALUE
162	DC	AL1(ADOVLA-MASU)	OVERLAY
163	DC	AL1(16)	L
164	DC	AL1(16)	LE
165	DC	AL1(11)	MVI
166	DC	AL1(4)	DO END
167	DC	AL1(4)	DC
168	DC	AL1(4)	ARRAY EXPR.END
169	DC	AL1(4)	FREE
170	DC	AL1(4)	FREE
171	DC	AL1(4)	FREE
172	DC	AL1(4)	LOAD DED
173	DC	AL1(4)	LOAD SCALAR
174	DC	AL1(4)	LOAD ARRAY
175	DC	AL1(4)	FREE
176	DC	AL1(4)	LIBRARY CALL(2)
177	DC	AL(ADSECO-MASU)	SET TRUE ON COND.
178	DC	AL1(4)	FREE
179	DC	AL1(16)	LM
180	DC	AL1(16)	LD
181	DC	AL1(4)	DEF.RESULT
182	DC	AL1(4)	FREE
183	DC	AL1(4)	FREE
184	DC	AL1(4)	FREE
185	DC	AL1(4)	FREE
186	DC	AL1(4)	FREE
187	DC	AL1(4)	FREE
188	DC	AL1(4)	LOOP BEGIN
189	DC	AL1(4)	LOOP END

Figure 1. Table TAB1 Used to Find MASU(X)

Routines Listed in TAB1

They generate the individual macros.

L	:	Length of the macro
n	:	Number of operands
R1	:	Precision of the (n-1)-th operand
R2	:	Scale factor of (n-1)-th operand
R3	:	Precision of the n-th operand
R4	:	Scale factor of the n-th operand
LABEL	:	Label taken from IJKMVC
SO	:	The location after the n operands of the macro

{R1-R2}---> SO (2 bytes)
means R1-R2 will be inserted with a length of two bytes into location SO.

ADBASC

Used for binary addition, subtraction, and comparison

L = 18
{R1-R2} ---> SO (2 bytes)

ADBOIV

Used for binary division

L = 18
{R3+1} ---> SO (2 bytes)

ADASSI

Used for assignments except the decimal fixed, the bit string, and the label assignment. No macro will be constructed. The premacro is used only for generation of constants if necessary.

ADMULI

Used for binary multiplication

a. if overflow must be checked:
   Macro key : X'02' L = 16

b. if overflow must not be checked:
   Macro key : X'09' L = 16

   if OP1 is given by a register R, the preceding register R-1 is used as the operand.

ADIOST

No macro but a text element which must be generated in connection with I/O statements. The element has the format

bytes	0	F7
	1-2	0010
	3	10
	4-5	name
	6	E4
	7-8	modifier
	9	E4
	10-11	attributes
	12	E4
	13	P
	14	Q

ADDARI

Used for decimal addition, subtraction, comparison.

L = 28
{FLOOR ((R3+2) /2)}---> SO
{FLOOR ((R1+2) /2)}---> SO+1
{Byte (3) from stmnt. identifier}---> SO+4
{N-R4} ---> SO+2
{N-R2} ---> SO+3
{LE} ---> SO+5
N is given by AAP3+20
LE is given by AA30+19.

ADDMU

Used for decimal multiplication
L = 27
{FLOOR ((R3+2) /2)}---> SO
{FLOOR ((R1+2) /2)}---> SO+1
{Byte (3) from stmnt. identifier}---> SO+4
{LE} ---> SO+2

ADDIV

Used for decimal division
L = 25
{FLOOR ((R3+2) /2)}---> SO
{FLOOR ((R1+2) /2)}---> SO+1
{Byte (3) from stmnt. identifier}---> SO+4
{15-R3} ---> SO+2

ADDNEG

Used for decimal negation with one operand, overlay, and if-macros.
L = 11.
{LE} ---> SO
LE is given by AA30+19

ADDSHI

Used for decimal assignment
L = 21
{FLOOR ((R3+2) /2)}---> SO
{FLOOR ((R1+2) /2)}---> SO+1
{Byte (3) from stmnt. identifier}---> SO+4
{R4-R2} ---> SO+2

ADNEG1

Used for decimal negation with two operands
L = 18
{FLOOR ((R3+2) /2)}---> SO
{FLOOR ((R1+2) /2)}---> SO+1

ADUNA

Used for prefix plus. No macro will be generated.

ADCHOP

Used for character string concatenation
```
L = 24
{R3}     ---> SO
{R1}     ---> SO+1
```

ADCHAP

Used for character string comparison
```
{R3}     ---> SO
{R1}     ---> SO+1

a.  if R1 = R3
            L = 18
b.  if R1 ≠ R3
            L = 24
            third operand : GW0
            {GW0} ---> SO+2   (6 bytes)
```

ADBSIP

Used for 'bit string not' with two operands
```
L = 19
{L1}     ---> SO
{M}      ---> SO+2
```
L1 is given by AA30+31
M is a mask depending on R3.

ADDNIG

Used for 'bit string not' with one operand
```
L = 13
{L1}     ---> SO
{M}      ---> SO+2
```
L1 is given by AA30+19
M is a mask depending on R1.

ADBISA

Used for bit string assignment
```
L = 19
{L2}     ---> SO+1
```
L2 is given by AA30+19
Otherwise identical to ADBSIP.

ADBSOP

Used for bit string 'and' and 'or'
```
L = 18
a.  if R1 does not exceed R3
    {L1} ---> SO
    {L2} ---> SO+1
b.  if R1 exceeds R3
    {L1} ---> SO+1
    {L2} ---> SO
    OP1 and OP2 are exchanged.
```
L1 is given by AA30+31
L2 is given by AA30+19.

ADBASA

Used for bit string comparison and character string assignment
```
L = 18
{L1}     ---> SO
{L2}     ---> SO+1
```

ADRTLC

Used for return to label constant and assign label constant.
L = 25.

ADIF

Used for IF macro.
L = 17.
Otherwise identical to ADDNEG.

ADCALL

Used for the CALL macro

1. The number of parameters N is inserted preceding the operands.

2. The macro identification =X'90' is inserted.

3. The length of the macro is calculated (L = (N+2) *6 - 1) and inserted.

4. The second operand is also used as the (n+1)-th operand.

5. If the first operand is extern, OP1 DC V(OP1) is generated.

ADSECO

Used for the set true on condition macro
```
L = 14
{LABEL}  ---> SO   (2 bytes)
```
Byte AA30+6 will be inserted preceding the operands.

This phase evaluates the subscripts. If the subscripts are constants, evaluation is optimized as far as possible. In addition, this phase generates the macro instructions required for the format label assignments. The phase is skipped if there are no arrays and no format labels in the source program.

Phase Input

The text input string may contain information marked by one of the keys listed below. The input string is skipped or processed depending on the key found (see Figure 1).

Key	Meaning	Skippable Length if not Processed
E0	Begin of statement	Implied: 6 bytes
EA	End of statement	Implied: 6 bytes
EB	Error information	Implied: 2 bytes
F2	Possible format label assignment	Processed. For string format see Figure 2.
FC	Array	Processed. For string format see Figure 3.
FFFF	End of program	
Other F-keys	Of no interest	Skippable length contained in the 2 bytes following the key.

Figure 1. Keys Scanned in the Input Text String

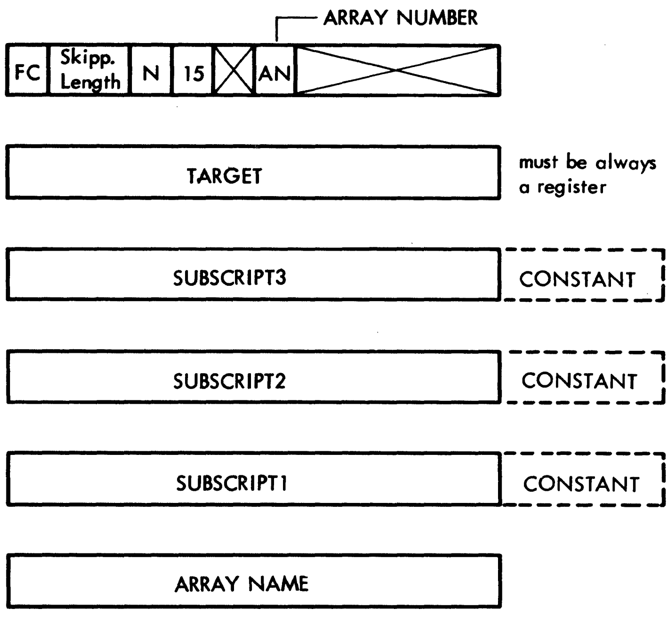


Figure 3. Format of Input String Marked by FC-Key

Phase Output

During the scan through the text, the information marked by an FC-key is checked for subscripts. If subscripts are found, the corresponding macro instructions are generated and put out. If a label assignment macro instruction is detected, it is modified to format label assignment if the label operand is a format name. This is checked by searching the corresponding name table. All other information is put out unchanged.

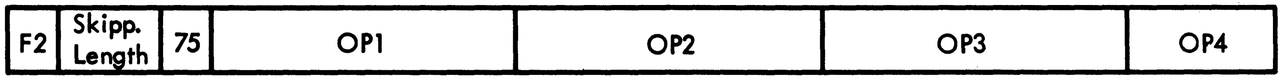


Figure 2. Format of Input String Marked by F2-Key

Generated Macro Instructions

1. Multiply Half-word -- MHMAK

F2	ØØØB	B8	OP1	R

Calling Sequence: MH R,OP1

2. Add -- AMAK

F2	ØØ12	ØØ	OP1	OP2	ØØØØ

Calling Sequence:

a. if OP2 register
```
    AR   OP1, OP2
```

b. if OP2 storage
```
    A    OP1, OP2
```

3. Load -- LMAK

F2	ØØ1Ø	A3	R	OP2

Calling Sequence:

If OP2 = register

```
        LR   R,OP2
```

if OP2 = storage

```
        L    R,OP2
```

4. Move Address -- MOVMAK

F2	ØØ1Ø	95	OP1	OP2

Calling sequence:

a. if OP1 = register
```
    LA   OP1,OP2
```

b. if OP1 = storage
```
    LA   5,OP2
    ST   5,OP1
```

5. Format label assignment macro instruction

F2	ØØ1Ø	95	OP1	OP2

For a description see the generator phases E50-E61.

Evaluation of Subscripted Variables

1. Arrays with 1 dimension:

In the input string, the subscript appears in the form shown in Figure 4.

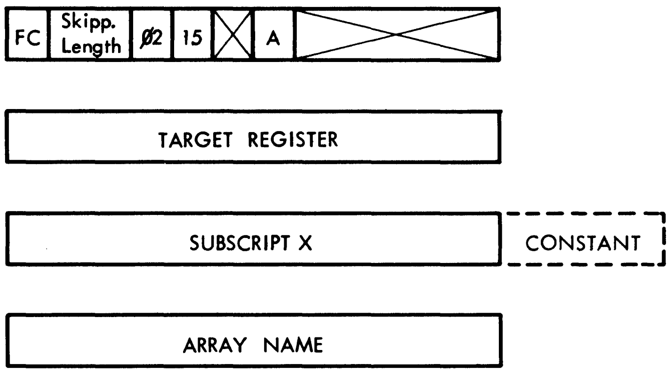


Figure 4. Input String (1-Dimensional Array)

A = array number (used to find the corresponding entry in the array table). The array table entry has the format shown in Figure 5.

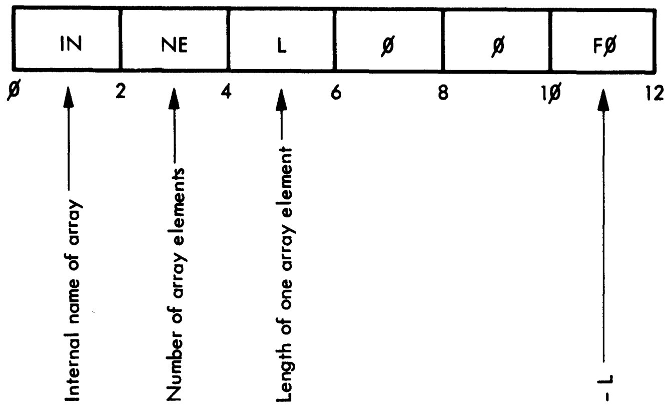


Figure 5. Format of Array Table Entry for 1-Dimensional Array

The element A(X) is determined as follows:

A(X) = A(1) + F0 + L * X

If X is a constant, L * X is computed during compilation and added to F0. In this case, the code produced is:

```
LA   REG,A(1)
A    REG,F0
```

Otherwise, the code produced is:

```
L    REG,X
MH   REG,L
A    REG,F0
LA   R5,A(1)
AR   REG,R5
```

2. Arrays with 2 dimensions:

If there are two subscripts, the input string has the format shown in Figure 6.

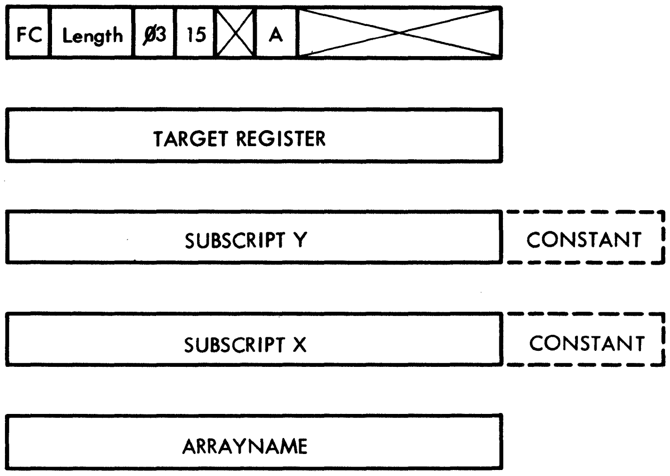


Figure 6. Input String (2-Dimensional Array)

The corresponding entry in the array table has the format shown in Figure 7.

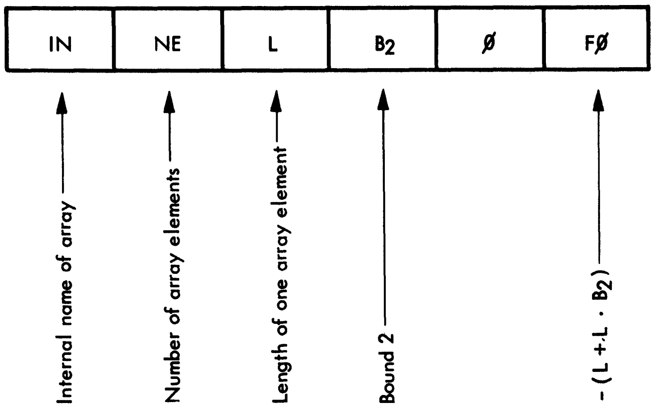


Figure 7. Format of Array Table Entry for 2-Dimensional Array

The code produced for evaluation of A(X,Y) depends upon the subscripts X and Y. The following 4 cases are possible:

a. X and Y = variables

A(X,Y)=A(1,1)+F0+L(Y+B2*X)

The code produced is:

```
L      REG,X
MH     REG,B2
A      REG,Y
MH     REG,L
A      REG,F0
LA     R5,A(1,1)
AR     REG,R5
```

b. X = constant, Y = variable

A (X,Y)=A(1,1)+F0+$\underline{\text{L*B2*X}}$+L*Y

L * B2 * X is computed during compilation and added to F0.

The code produced is:

```
L      REG,Y      LR if Y register
MH     REG,L
A      REG,F0
LA     R5,A(1,1)
AR     REG,R5
```

c. Y = constant, X = variable

A(X,Y)=A(1,1)+F0+$\underline{\text{L*Y}}$+L*B2*X

L * Y is computed during compilation and added to F0.

The code produced is:

```
L      REG,X
MH     REG,L*B2
A      REG,F0
LA     R5,A(1,1)
AR     REG,R5
```

d. X and Y = constants

A(X,Y)=A(1,1)+F0+$\underline{\text{L*Y}}$+$\underline{\text{L*B2*X}}$

L * Y and L * B2 * X are computed during compilation and added to F0.

The code produced is:

```
LA     REG,A(1,1)
A      REG,F0
```

3. Arrays with 3 dimensions:

If there are 3 subscripts, the input string has the format shown in Figure 8.

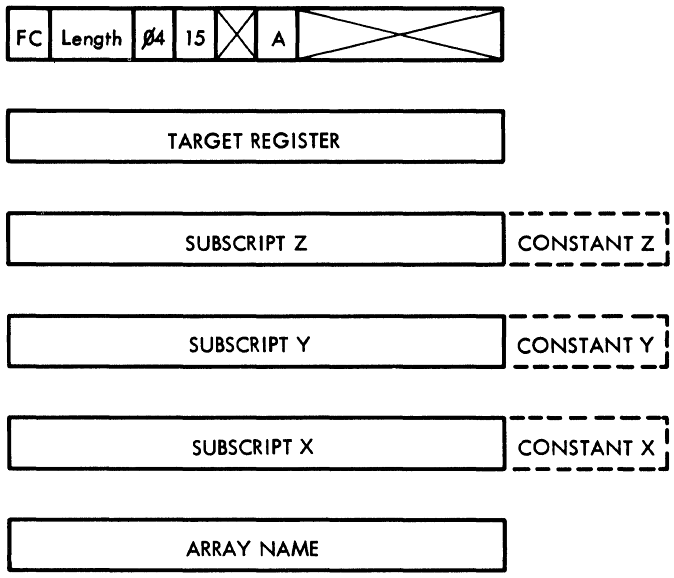


Figure 8. Input string (3-Dimensional Array)

The corresponding entry in the array table has the format shown in Figure 9.

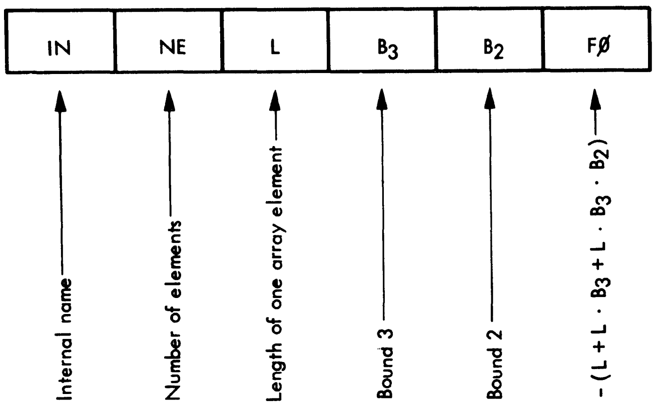


Figure 9. Format of Array Table Entry for 3-Dimensional Arrays

The code produced to evaluate the element A(X,Y,Z) is optimized as follows:

a. <u>X, Y, and Z = variables</u>

A(X,Y,Z)=A(1,1,1)+F0+L(Z+B3(Y+B2*X)

The code produced is:

```
L     REG,X
MH    REG,B2
A     REG,Y
MH    REG,B3
A     REG,Z
MH    REG,L
A     REG,F0
LA    R5,A(1,1,1)
AR    REG,R5
```

b. <u>X = constant, Y and Z = variables</u>

A(X,Y,Z)=A(1,1,1)+F0+<u>L*B3*B2*X</u>+L(Z+B3*Y)

L * B3 * B2 * X is computed during compilation and added to F0.

The code produced is:

```
L     REG,Y
MH    REG,B3
A     REG,Z
MH    REG,L
A     REG, F0
LA    R5,A(1,1,1)
AR    REG,R5
```

c. <u>Y = constant, X and Z = variables</u>

A(X,Y,Z)=A(1,1,1)+F0+
<u>L*B3*Y</u>+L(Z+B3*B2*X)

L * B3 * Y is computed during compilation and added to F0.

The code produced is:

```
L     REG,X
MH    REG,B3*B2
A     REG,Z
MH    REG,L
A     REG,F0
LA    R5,A(1,1,1)
AR    REG,R5
```

d. <u>Z = constant, X and Y = variables</u>

A(X,Y,Z)=A(1,1,1)+F0+<u>L*Z</u>+L*B3(Y+B2* X)

L * Z is computed during compilation and added to F0.

Code produced is:

```
L     REG,X
MH    REG,B2
A     REG,Y
MH    REG,L * B3
A     REG,F0
LA    R5,A(1,1,1)
AR    REG,R5
```

e. <u>X and Y = constants, Z = variable</u>

A(X,Y,Z)=A(1,1,1)+F0+<u>L*B3*Y</u>+<u>L*B3*B2*X</u>+L*Z

L * B3 * Y and L * B3 * B2 * X are computed during compilation and added to F0.

The code produced is:

```
L     REG,Z
MH    REG,L
A     REG,F0
LA    R5,A(1,1,1)
AR    REG,R5
```

f. <u>X and Z = constants, Y = variable</u>

A(X,Y,Z)=A(1,1,1)+F0+<u>L*Z</u>+<u>L*B3*B2*X</u>+L*B3*Y

L * Z and L * B3 * B2 * X are computed during compilation and added to F0.

The code produced is:

```
L      REG,Y
MH     REG,L*B3
A      REG,F0
LA     R5,A(1,1,1)
AR     REG,R5
```

g. <u>Y and Z = constants, X = variable</u>

A(X,Y,Z)=A(1,1,1)+F0+<u>L*Z</u>+<u>L*B3*Y</u>+L*B3*B2*X

L * Z and L * B3 * Y are computed during compilation and added to F0.

The code produced is:

```
L      REG,X
MH     REG,L*B3*B2
A      REG,F0
LA     R5,A(1,1,1)
AR     REG,R5
```

h. <u>X, Y, and Z = constants</u>

A(X,Y,Z)=A(1,1,1)+F0+<u>L*Z</u>+<u>L*B3*Y</u>+<u>L*B3*B2*X</u>

The underlined expressions are computed during compilation and added to F0.

The code produced is:

```
LA     REG,A(1,1,1)
A      REG,F0'
```

DESCRIPTION OF ROUTINES

The text input string is read into the 3 consecutive input buffers IBU1, IBU2, and IBU3. The array table has a maximum length of 384 bytes and is read into the area IBU1-384. The format label table has a maximum length of 256 bytes and is read into the area following the input buffer IBU3. The main routine (see general flow chart JA) controls the scan of the input string and calls the corresponding processing routine. The following symbols are used in the individual routines:

IBU1, 2, 3	:	Input buffers 1, 2, 3
OBU1, 2	:	Output buffers 1, 2
INPT	:	Input pointer
OPT	:	Output pointer
IREC	:	Switch indicating the number of records to be read
OREC	:	Switch indicating the number of records to be put out
ONSWIT	:	Switch which is set to one if phase D17 must be called
EOS	:	End of statement
BUFFL	:	Buffer length
SLENG	:	Skippable length
SAVR0, 1	:	Save buffers for registers
COLE	:	Constant length
ADBL	:	Pointer to array name
TRBYT	:	1-byte entry used to hold type of subscripted Variable
STABL	:	Contains frame for X'E9' operand
FDCO	:	Contains frame for X'FD' operand
AMAK	:	Add macro
MHMAK	:	Multiply-halfword macro

<u>INIT1 -- JB</u>

The routine tests for arrays or format labels in the current compilation. If an array or format label is found, the corresponding routine is called in order to read in the array table and/or the format label table. The addresses for the 3 input buffers and the 2 output buffers are computed and the input and output pointers are set to their initial values.

<u>E0ACT -- JC</u>

Input: INPT points to an E0-key.

The routine resets the error switch ERRSW and clears the error stack ERROSTK. The begin of statement (6 bytes) is put out and the input pointer is adjusted.

<u>EAACT -- JD</u>

Input: The input pointer INPT points to an EA-key.

The key EA indicates the end of statement. If the error switch ERRSW is not zero, a bit is set to indicate that execution is to be deleted, and both the end-of-statement and the error stack ERROST are put out. Otherwise, only the end of statement is put out.

<u>EBACT -- JE</u>

The error message marked by the EB-key is put out and the input pointer is increased.

<u>ASKIP -- JF</u>

This routine is required to move input information to the output. It increases the input pointer by means of the skip routine SKIP.

SKIP -- JG

Input parameter: Reg 0 contains the length by which the input pointer is to be increased.

The input pointer is increased by the specified length and checked as follows:

```
┌─────────┬─────────┬─────────┐
│ Buffer1 │ Buffer2 │ Buffer3 │
└─────────┴─────────┴─────────┘
^         ^         ^         ^
|         |         |         |
|         |         |         |
IBU1      IBU2      IBU3      OBU1
```

If the input pointer points to one of the buffers 2 or 3, the buffers 2 or/and 3 are shifted to the left and the input pointer is updated accordingly. Then, the next records are read into the buffers 2 or/and 3.

READIN -- JH

Input parameter: IREC (1 byte) indicates the number of records to be read into the input buffers.

This subroutine controls the reading of the input string into the input buffers.

MOV31 -- JI

The subroutine moves the contents of the input buffer IBU3 into IBU1.

MOV21 -- JI

The subroutine shifts the contents of the two input buffers by one buffer length to the left in the first input buffer.

MOVO21 -- JI

The subroutine moves the contents of the second output buffer into the first output buffer.

MOVOUT -- JJ

Input parameters: REG0 contains the length to be moved. REG1 contains the address of the field to be moved.

The routine moves the input string or the data specified in REG0 and REG1 into the output buffer and transfers control to the routine OUT.

OUT -- JK

R0 contains the length by which the output pointer OPT is to be increased. The routine updates the output pointer and, if the buffer is full, the record is written.

EOPACT -- JL

The input pointer INPT points to the end-of-program (EOP) key. The routine writes the last record onto the work file and tests whether or not the next phase D17 is to be called. Phase D40 is called if phase D17 is skipped.

IJKNN -- JH

The routine fetches a name from IJKMVC in the communication region and stores it in NAME. The name counter is increased by 1 and restored into the communication region. If the name counter becomes greater than 64K, an error message is generated.

ARRTAB -- JN

The routine reads the array table from the work file into the area following the output buffers.

FORMTAB -- JO

The format label table is read from the work file into storage.

LABELAS -- JP

The macro instruction is tested for label assignment. If it is a label assignment, the label operand is tested for format label by searching the format label table. When the label name is found in the label table, the macro instruction is modified to a format label macro instruction.

SUBSCR -- JQ

The subroutine controls the evaluation of subscripted variables. It calls the error check routines and the routines that generate the macro instructions.

INITSUB -- JR

The subroutine initializes the evaluation of subscripts. The array number is used to find the corresponding entry in the array table. (See the description of the input string.)

DIMCHK -- JS

Depending on the number of subscripts, the routine calls the appropriate routine to test the array table entry.

CHECK3, 4, 5 -- JT

The routines test the array table entry. If it contains an invalid number of dimensions, the error routine is called.

TRACT5 -- JU

The routine calls one of 8 possible actions in the case of 3 dimensions (see the section Evaluation of Subscripted Variables).

TRACT4 -- JV

Calls the corresponding action in the case of 2 dimensions (see the section Evaluation of Subscripted Variables).

TRACT3 -- JW

Calls the corresponding action in the case of 1 dimension (see the section Evaluation of Subscripted Variables).

TESCON -- JX

Tests if the subscripts X, Y, and Z are constants.

If X=constant, bit 7 in TRBYT is set to 1.
If Y=constant, bit 6 in TRBYT is set to 1.
If Z=constant, bit 5 in TRBYT is set to 1.

SACT50 -- JY

Evaluates A(X,Y,Z) where X,Y,Z = variables
A(X,Y,Z)=A(1,1,1)+F0+L(Z+K(Y+J*X)) where

L	=	length of array element
K	=	bound 3 of declared array A(I,J,K)
J	=	bound 2 of declared array A(I,J,K)
F0	=	- (C0+C1+C2)

where C0 = L
C1 = L * K
C2 = L * K * J

SACT51 -- JZ

Evaluates A(X,Y,Z) where X = constant, Y and Z = variables.

SACT52 -- JZ

Evaluates A(X,Y,Z) where Y = constant, Y and Z = variables.

SACT53 -- JZ

Evaluates A(X,Y,Z) where Z = constant, X and Y = variables.

SACT54 -- KA

Evaluates A(X,Y,Z) where X and Y = constants, Z = variable.

SACT55 -- KA

Evaluates A(X,Y,Z) where X and Z = constants, Y = variable.

SACT56 -- KA

Evaluates A(X,Y,Z) where Y and Z = constants, X = variable.

SACT57 -- KA

Evaluates A(X,Y,Z) where X, Y, and Z = constants.

MOVGEN -- KC

The routine puts out the constant specified by R4 and moves the internal name of the constant into the area MAK87.

CVFISCH -- KD

R4 points to the argument to be converted. The routine calls the conversion routine and returns the converted constant to R4.

PMAK87 -- KE

The routine puts out the constant F0 and moves its internal name into the area MAK87. (For a description of F0 see SAC 50).

MAKGEN -- KF

The routine generates the macro instructions according to the number N in the area MAK87. The contents of MAK87 are shown in Figure 10.

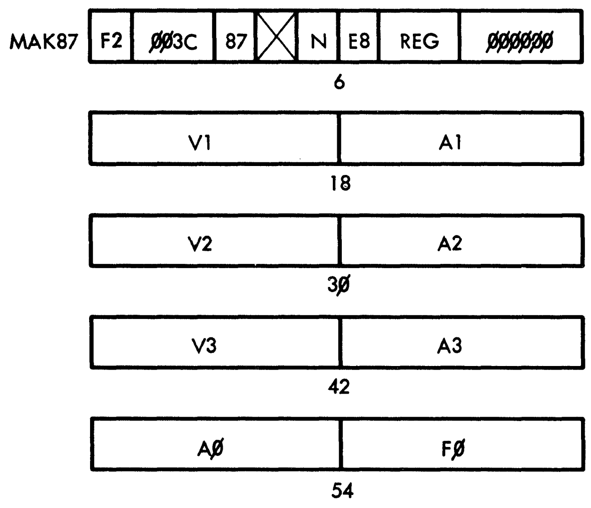


Figure 10. Contents of MAK87

The code produced is:

```
1. if N = 0:  LA   REG,A0
              A    REG,F0

2. if N = 1:  L    REG,V1   LR if V1 register
              MH   REG,A1
              A    REG,F0
              LA   R5,A0
              AR   REG,R5
```

```
3. if N = 2:  L    REG,V1   LR if V1 register
              MH   REG,A1
              A    REG,V2   AR if V2 register
              MH   REG,A2
              A    REG,F0
              LA   R5,A0
              AR   REG,R5

4. if N = 3:  L    REG,V1   LR if V1 register
              MH   REG,A1
              A    REG,V2   AR if V2 register
              MH   REG,A2
              A    REG,V3   AR if V3 register
              MH   REG,A3
              A    REG,F0
              LA   R5,A0
              AR   REG,R5
```

<u>SACT40 -- KG</u>

Evaluates A(X,Y) where X and Y = variables.
A(X,Y)=A(1,1)+F0+L(Y+J*X)

where: L = length of array element
J = bound 2 in the declared array A (I,J)
F0 = -(L + L * J)

<u>SACT41 -- KG</u>

Evaluates A(X,Y) where X = constant,
Y = variable.
A(X,Y)=A(1,1)+F0+<u>L*J*X</u>+L*Y

L*J*X is computed and added to F0.

<u>SACT42 -- KG</u>

Evaluates A(X,Y) where Y = constant,
X = variable.
A(X,Y)=A(1,1)+F0+<u>L*Y</u>+L*J*X

L * Y is computed during compilation and added to F0.

<u>SACT43 -- KG</u>

Evaluates A(X,Y) where X and Y = constants.
A(X,Y)=A(1,1)+F0+<u>L*J*X</u>+<u>L*Y</u>

L * J * X and L * Y are computed during compilation and added to F0.

<u>LJX -- KH</u>

Computes L * K * X and adds the result to F0. (For a description of L, K, X and F0 see <u>SACT40</u>).

<u>LY -- KH</u>

Computes L * Y and adds the result to F0.

<u>SACT30 -- KI</u>

Evaluates A(X) where X = variable.

A(X) = A(1) + F0 + L * X where
L = length of array element
F0 = -L

<u>SACT31 -- KI</u>

Evaluates A(X) where X = constant.
A(X) = A(1) + F0 + <u>L * X</u>

L * X is computed during compilation and added to F0.

<u>FISCH -- KJ , KL - KO</u>

The routine converts constants from their intermediate representation to binary integer. The input parameter PIN contains the address of the constant entry that contains the constant to be converted. Output parameter PIN remains unchanged. The result is stored in CONS (4 bytes).

PHASE PL/ID17 (LIBRARY CALLS FOR BUILT-IN FUNCTIONS I) -- LA

The function of this phase is to generate macro instructions for the linkage required to call the library built-in functions. However, in some cases, the functions are evaluated in line, i.e., not via the library. If the compilation contains no built-in functions, this phase is skipped.

Phase Input and Output

The text string may contain information marked by E or F keys. The formats of the various types of information are shown below. The E or F key is always contained in the first byte of the particular string of information.

1. Begin of statement (six bytes): E0
2. End of statement (six bytes): EA
   Last byte contains statement number
3. Error information (two bytes): EB
4. Information string containing a library built-in function indicated by an FC key and X'13' in byte 4. The format of the string is as follows:

byte	0	X'FC'
bytes	1-2	skippable length
byte	3	number of arguments including target
byte	4	X'13'
byte	5	internal name
bytes	6-12	not used

   The arguments may be constants, variables, or registers. Variables and registers appear in the following form:

byte	0	E-key
bytes	1-2	internal name
bytes	3-4	modifier
bytes	5-6	attributes
byte	7	length at object time
byte	8-9	precision (P, Q)
bytes	10-12	not used

   Constants are marked by an E9-key and contain the following additional information:

bytes	0-1	internal name
bytes	2-7	attributes
bytes	8-9	length of constant
bytes	10-22	intermediate form of constant (left-aligned)

5. End-of-program key (FFFF).

   Note: F-keys other than FC (indicating a library built-in function) and FF (end of program) are skipped.

All information marked by E and F keys (except FC) is written out unchanged. When a statement is marked by an FC key, the routine determines the library function and either generates an appropriate internal macro instruction or causes (1) the information to be written out unchanged and (2) phase D20 to be called. Which of the actions is performed is determined by the type of library function involved. If the input text does not contain functions that are to be processed by phase D20, that phase is skipped and phase D40 is called. The internal macros that may be generated during this phase are listed below. In addition, the format of the information string written out by this phase and the calling sequence generated are shown for each of the possible macros.

Internal macros are identified by an F2 key in the first byte and a macro identification in the fourth. The second and third bytes indicate the skippable length.

1. LIBRARY CALL:

F2	0007	BA	E9	00	LIB

   Calling sequence:
```
L     15,LIB-name
BALR  14,15
```

2. OI:

F2	000B	B6	OP1	M

   Calling sequence:
```
OI    POI,M
```

3. MVI:

F2	000B	A5	OP1	M

   Calling sequence:
```
MVI   OP1,M
```

4. MOVE MAK9E:

F2	0011	9E	OP1	OP2	L

Calling sequence:

```
MVC   OP1(L), OP2
```

5. MOVE MAK86:

F2	0012	86	OP1	OP2	L1	L2

Calling sequence:

a. if L1 ≤ L2:
```
MVC    OP1(L1),OP2
```

b. if L1 > L2:
```
MVI    OP1,X'40'
MVC    OP1+1(L1-1),OP1
MVC    OP1(L2),OP2
```

6. MOVE ADDRESS MAK0:

F2	0010	95	OP1	OP2

Calling sequence:

a. if OP1 = register
```
LA    OP1,OP2
```

b. if OP1 = storage
```
LA    5,OP2
ST    5,OP1
```

7. SUBSTR MAK88:

F2	0010	88	OP1	OP2

Calling sequence:

```
LA     5,OP1
A      5,OP2  (or AR 5,OP2)
BCTR   5,0
```

BUILT-IN FUNCTIONS PROCESSED

The table shown in Figure 1 is a listing of all built-in functions processed by this phase. In addition, this table shows the names of the modules called for the built-in function at object time, the internal representation generated for the built-in function, and the type of linkage used to transfer control to the appropriate module.

Built-in Function		Module Name	Internal Name		Link Type
			Dec	Hex	
TIME		STMM	80	50	S
DATE		SDTM	81	51	S
SQUARE ROOT	(SHORT)	QQSM	84	54	A
SQUARE ROOT	(LONG)	QQLM	85	55	A
EXP	(SHORT)	QASM	86	56	A
EXP	(LONG)	QALM	87	57	A
LOG	(SHORT)	QLSA	88	58	A
LOG	(LONG)	QLLA	89	59	A
LOG2	(SHORT)	QLSC	90	5A	A
LOG2	(LONG)	QLLC	91	5B	A
LOG10	(SHORT)	QLSB	92	5C	A
LOG10	(LONG)	QLLB	93	5D	A
SINE	(SHORT)	QSSD	94	5E	A
SINE	(LONG)	QSLD	95	5F	A
SINE-DEGREE	(SHORT)	QSSC	96	60	A
SINE-DEGREE	(LONG)	QSLC	97	61	A
COSINE	(SHORT)	QSSB	98	62	A
COSINE	(LONG)	QSLB	99	63	A
COSINE-DEGREE	(SHORT)	QSSA	100	64	A
COSINE-DEGREE	(LONG)	QSLA	101	65	A
TAN	(SHORT)	QTSB	102	66	A
TAN	(LONG)	QTLB	103	67	A
TAN-DEGREE	(SHORT)	QTSA	104	68	A
TAN-DEGREE	(LONG)	QTLA	105	69	A
SINH	(SHORT)	QCSA	106	6A	A
SINH	(LONG)	QCLA	107	6B	A
COSH	(SHORT)	QCSB	108	6C	A
COSH	(LONG)	QCLB	109	6D	A
TANH	(SHORT)	QDSA	110	6E	A

Figure 1. Table of Built-in Functions during Phase D17 (Part 1 of 2)

TANH	(LONG)	QDLA	111	6F	A
ATANH	(SHORT)	QBSA	112	70	A
ATANH	(LONG)	QBLA	113	71	A
ERF	(SHORT)	QRSB	114	72	A
ERF	(LONG)	QRLB	115	73	A
ERFC	(SHORT)	QRSA	116	74	A
ERFC	(LONG)	QRLA	117	75	A
ATAN	(SHORT)	QNSD	118	76	A
ATAN	(LONG)	QNLD	119	77	A
ATAN-DEGREE	(SHORT)	QNSC	120	78	A
ATAN-DEGREE	(LONG)	QNLC	121	79	A
ATAN- (X/Y)	(SHORT)	QNSB	122	7A	B
ATAN- (X/Y)	(LONG)	QNLB	123	7B	B
ATAN-DEGREE (X/Y)	(SHORT)	QNSA	124	7C	B
ATAN-DEGREE (X/Y)	(LONG)	QNLA	125	7D	B
REPEAT BIT		RBRB	126	7E	C
BIT CONCATENATION		RBKA	127	7F	D
INDEX BIT		RBIM	128	80	D
INDEX CHARACTER		RGIM	129	81	D
BOOL		RBBM	132	84	E
REPEAT CHARACTER		RGKM	133	85	D
MAX.	(FLOAT SHORT)	RMSX	134	86	F
MAX.	(FLOAT LONG)	RMLX	135	87	F
MAX.	(BIN FIXED)	RMBX	136	88	F
MAX.	(DEC FIXED)	RMPX	137	89	F
MIN.	(FLOAT SHORT)	RMSN	138	8A	F
MIN.	(FLOAT LONG)	RMLN	139	8B	F
MIN.	(BIN FIXED)	RMPN	140	8C	F
MIN.	(DEC FIXED)	RMPN	141	8D	F
SUBSTR BIT	(RIGHT)		142	8E	H
SUBSTR CHAR	(RIGHT)		143	8F	in-line
SUBSTR BIT	(LEFT)		144	90	H
SUBSTR CHAR	(LEFT)		145	91	in-line
EXP	(FLOAT SHORT + INTEGER)	RESM	146	92	I
EXP	(FLOAT LONG + INTEGER)	RELM	147	93	I
EXP	(DEC + INTEGER)	REPM	148	94	J
EXP	(BIN FIX + INTEGER)	REBM	149	95	K
EXP	(GENERAL SHORT)	RXSA	150	96	L
EXP	(GENERAL LONG)	RXLM	151	97	L
HIGH			178	B2	in-line
LOW			179	B3	in-line
ADDRESS			184	B8	in-line
DYNDUMP		SDMP	194	C2	M
FLOOR	(FLOAT SHORT)	RTSM	200	C8	A
FLOOR	(FLOAT LONG)	RTLM	201	C9	A
FLOOR	(BIN FIXED)	RTBM	202	CA	N
FLOOR	(DEC FIXED)	RTPM	203	CB	O
CEIL	(FLOAT SHORT)	RVSM	204	CC	A
CEIL	(FLOAT LONG)	RVLM	205	CD	A
CEIL	(BIN FIXED)	RVBM	206	CE	N
CEIL	(DEC FIXED)	RVPM	207	CF	O
MOD	(FLOAT SHORT)	RSSM	208	D0	B
MOD	(FLOAT LONG)	RSLM	209	D1	B
MOD	(BIN FIXED)	RSBM	210	D2	P
MOD	(DEC FIXED)	RSPM	211	D3	Q
ROUND	(FLOAT SHORT)	RUSM	212	D4	in-line
ROUND	(FLOAT LONG)	RULM	213	D5	in-line
ROUND	(BIN FIXED)	RUBM	214	D6	R
ROUND	(DEC FIXED)	RUPM	215	D7	R
TRUNC	(FLOAT SHORT)	RWSM	216	D8	A
TRUNC	(FLOAT LONG)	RWLM	217	D9	A
TRUNC	(BIN FIXED)	RWBM	218	DA	N
TRUNC	(DEC FIXED)	RWPM	219	DB	O

Figure 1. Table of Built-in Functions Processed during Phase D17 (Part 2 of 2)

The various types of linkages are described below.

Linkage Type A

R1 contains the address of a parameter block (PBL) as shown below.

Address of Source
Address of Target

The compiler generates one of four possible variations of a coding sequence. Which of the four variations is generated is determined by the type of storage (static or dynamic) that contains the source and target data:

1. If both SOURCE and TARGET are STATIC, the compiler generates:

```
          LA      R1,PBL
          L       R15,LIB
          BALR    R14,R15
          --------------
   PBL    DC      A(SOURCE)
          DC      A(TARGET)
```

2. If only SOURCE is STATIC, the compiler generates:

```
          LA      R1,PBL
          LA      R5,TARGET
          ST      R5,PBL+4
          L       R15,LIB
          BALR    R14,R15
          --------------
   PBL    DC      A(SOURCE)
          DC      XL4'0'
```

3. If only TARGET is STATIC, the compiler generates:

```
          LA      R1,PBL
          LA      R5,SOURCE
          ST      R5,PBL
          L       R15,LIB
          BALR    R14,R15
          --------------
   PBL    DC      XL4'0'
          DC      A(TARGET)
```

4. If neither SOURCE nor TARGET is STATIC, the compiler generates:

```
          LA      R1,GWS
          LA      R5,SOURCE
          ST      R5,GWS
          LA      R5,TARGET
          ST      R5,GWS+4
          L       R15,LIB
          BALR    R14,R15
```

where GWS = general working storage in the outermost block.

Linkage Type B

R1 contains the address of a parameter block (PBL) as shown below.

Address of X
Address of Y
Address of Target

The compiler generates a coding sequence as determined by the type of storage (static or dynamic) that contains the arguments X and Y and the target data. The process of generating the coding sequence is as for type-A linkages.

Linkage Type C
R1 contains the address of a parameter block (PBL) as shown below.

L	Address of Bit String
N	Address of Target

where L = length of bit string and
N = repetition factor.

The compiler generates a coding sequence as determined by the type of storage (static or dynamic) that contains the bit string and the target. The process of generating the coding sequence is as for type-A linkages. However, L and N must be inserted into the PBL before control is transferred. This is ensured by two MVI instructions as shown in the example below, which is a sequence for both bit string and target data in STATIC storage:

```
       LA      R1, PBL
       MVI     PBL,L
       MVI     PBL+4,N
       L       R15,LIB-name
       BALR    R14,R15
       --------------
PBL    DC      A(BITSTRING)
       DC      A(TARGET)
```

Linkage Type D

R1 points to a parameter block (PBL) as shown below.

L1	Address of String1
L2	Address of String2
Address of Target	

where L1 = length of string 1
L2 = length of string 2

The compiler generates a coding sequence. The process of generating the coding sequence follows the same rules that apply to type-A linkages. L1 and L2 must be inserted into the PBL before control is transferred. This is accomplished by MVI instructions as are used to the insert L and N into PBL for a type-C linkage.

Linkage Type E

R1 points to a parameter block (PBL) as shown below.

L1	Address of String1
L2	Address of String2
Address of Mask	
L	Address of Target

where L1 = length of string 1 (1=1,2)
      L = max (L1,L2)

The compiler generates a coding sequence. The process of generating the coding sequence follows the same rules that apply to type-A linkages. L1, L2, and L must be inserted into the PBL before control is transferred. This is accomplished by MVI instructions as are used to the insert L and N into the PBL for a type-C linkage.

Linkage Type F

R1 points to a parameter block (PBL) as shown below.

	Address of Operand1
	Address of DED (Operand1) *
	Address of Operandn
	Address of DED (Operandn) *
	Address of Target
X'80'	Address of DED (Target) *

* The format of DEDs is shown under Linkage Type J.

If all operands and the TARGET data are in STATIC storage, the compiler generates:

```
LA      R1,PBL
MVI     PBL+8n,X'80'
L       R15,LIBNAME
BALR    R14,R15
```

```
PBL  DC      A(OPERAND1)
     DC      A(DED1)
     .
     .
     DC      A(OPERANDn)
     DC      A(DEDn)
     DC      A(TARGET)
     DC      A(DED OF TARGET)
```

For each operand in DYNAMIC storage, the address is determined by means of code and stored in PBL as shown:

```
LA      R5,OPERANDi
ST      R5,PBL+4(i-1)
```

where i = operand identification.

Linkage Type H

R1 contains the address of SOURCE.
R2 contains the address of OURCE DED.*
R3 contains the address of TARGET.
R4 contains the address of TARGET DED.*

* Format of the DED:

25	J	I

where J = length,
      I = offset.

The compiler generates:

```
LA      R1,SOURCE
LA      R2,SOURCE DED
LA      R3,TARGET
LA      R4,TARGET DED
L       R15,LIBname
BALR    R14,R15
--------------
```

Linkage Type I

R1 contains the address of argument X.
R2 contains the address of exponent N.
R3 contains the address of TARGET.

The compiler generates:

```
LA      R1,X
LA      R2,N
LA      R0,TARGET
L       R15,LIBname
BALR    R14,R15
```

Linkage Type J

R1 contains the address of argument X.
R2 contains the address of DED(X).
R3 contains the address of exponent N.
R4 contains the address of TARGET.
R5 contains the address of TARGET DED.

The DEDs have the following format:

1. FIXED BINARY:

0	P	Q+128

2. DECIMAL FIXED:

8	P	Q+128

3. FLOAT:

4	P

The format used for the generation of DEDs is determined by the type of data to be described. The compiler generates:

```
LA      R1,X
LA      R2,DED (X)
LA      R3,N
LA      R4,TARGET
LA      R5,TARGET DED
L       R15,LIBname
BALR    R14,R15
```

Linkage Type K

R1 contains the address of argument X
R3 contains the address of exponent N
R4 contains the address of TARGET

The compiler generates:

```
LA      R1,X
LA      R3,N
LA      R4,TARGET
L       R15,LIBname
BALR    R14,R15
```

Linkage Type L

R1 contains the address of exponent Y
R2 contains the address of argument X
R3 contains the address of TARGET

The compiler generates:

```
LA      R1,Y
LA      R3,X
LA      R3,TARGET
L       R15,LIBname
BALR    R14,R15
```

Linkage Type M

R0 points to a parameter block (PBL1) as follows:

L1	L2		Ln	X'FFFF'

R1 points to an address block (PBL2) as follows:

Address of Argument 1
Address of Argument 2
Address of Argument n

The compiler generates:

```
LA      R,PBL1
LA      R1,PBL2
L       R15,LIBname
BALR    R14,R15
```

Linkage Type N

R1 contains the address of a parameter block (PBL) as shown below:

Q+128	Address of Argument X
Address of Target	

The compiler generates a coding sequence. The process of generating the coding sequence follows the same rules that apply to type-A linkages. The value of Q+128 must be inserted into the PBL before control is transferred. This is accomplished by an MVI instruction (refer to Linkage Type C.

Linkage Type O

R1 points to a parameter block (PBL) as shown below.

Address of Source
Address of Target
Address of DED*

* Format of the DED:

P of Source	Q+128	P of Target

The compiler generates a coding sequence. The process of generating the coding sequence follows the same rules that apply to type-A linkages.

Linkage Type P

R1 contains the address of a parameter block (PBL) as shown below:

Q1+128	Address of Argument X
Q2+128	Address of Argument Y
Address of Target	

The compiler generates a coding sequence. The process of generating the coding sequence follows the same rules that apply to type-A linkages. The value of Q+128 must be inserted into the PBL before control is transferred. This is accomplished by MVI instructions as are used to the insert L and N into the PBL for type-C linkages.

Linkage Type Q

R1 contains the address of a parameter block (PBL) as shown below.

Address of X
Address of Y
Address of Target
Address of DED *

*Format of the DED:

P of X	(Q of X)+128	P of Y	(Q of Y)+128	P of Target

The compiler generates a coding sequence. The process of generating the coding sequence follows the same rules that apply to type-A linkages.

Linkage Type R

R1 contains the address of a parameter block (PBL) as shown below.

Address of X
Address of Target
Address of DED *

* Format of the DED:

P	Q+128	N

The compiler generates a coding sequence. The process of generating the coding sequence follows the same principle rules that apply to type-A linkages.

Linkage Type S

R1 contains the address of TARGET. The compiler generates:

```
LA      R1,TARGET
L       R15,LIBname
BALR    R14,R15
```

IN-LINE FUNCTIONS

This section describes the operations performed on built-in functions that are generated in-line.

Substring Built-in Function

A substring function may appear in either of the following two formats:

1. X = SUBSTR(CS, I, J)

2. SUBSTR(CS, I, J) = X

If the function appears in the first of the two formats, the compiler generates:

```
LA      R5,CS
A       R5,I
BCTR    R5,1
MVC     X(J),0(R5)
```

otherwise, the compiler generates:

```
LA      R5,CS
A       R5,I
BCTR    R5,0
MVC     0(J,R5),X
```

However, if J exceeds the length of X, the compiler generates the following three instructions for the MVC instruction in the above sequences:

```
MVI     0(R5),X'40'
MVC     1(J-1,R5),0(R5)
MVC     0(L,R5),X
```

where L = length of X.

The following condition is tested for during compile time:
$1 \le J \le K$

where K = length of CS. In case of an error, J is replaced by K.

HIGH and LOW Built-in Functions

Code generated for T = HIGH(X)
or T = LOW(X) is:

```
1.  If X = 1 either MVI   TARGET,X'FF'
                or MVI   TARGET,X'00'

2.  If X ≠ 1 either MVI   TARGET,X'FF'
                or MVI   TARGET,X'00'
               and MVC   TARGET+1(LT-1),
                         TARGET
    where LT = length of target
```

ROUND Built-in Function for Floating Point

If argument X is FLOAT, the compiler generates:

```
      MVC      TARGET(L),SOURCE
      OI       TARGET+LI,X'01'

where L    =   length of source
      LI   =   3 if short floating point
      LI   =   7 if long floating point.
```

ADDR(X) Built-in Function

For the address function, the compiler generates the macro instruction MOVE ADDRESS. This is described in the section Phase Input and Output.

DESCRIPTION OF ROUTINES

Three input buffers are available to read the input text string. Two output buffers are used to write out the output information.

The text string is scanned and the information is written out unchanged as long as no E or F keys are encountered. When an FC key is encountered, the appropriate Macro-Generation routine is called.

The saving and restoring the link register on entering and leaving nested subroutines is done by a save and a return routine. The various routines of this phase are described in the following.

Symbols used in flow charts:

EOP	:	End of program
IBU1	:	Text input buffers
IBU2		
IBU3		
INPT	:	Input pointer
OBU1	:	Text output buffers
OBU2		
OBU3		
OPT	:	Output pointer
EOS	:	End-of-statement key
SLENG	:	Skippable length
IREC	:	Number of input records desired
OREC	:	Number of output records desired
ADBL	:	Stack containing pointers to arguments
GWS8	:	Internal name of general working storage
MAKO	:	Move-Address macro instruction
ADCO	:	Contains address for address macro
DED	:	Data element descriptor
LIBC	:	Library call macro

D17 -- LA

This is the main routine. It controls the input, the output, and the processing of the text string. The various E- and F-keys are translated and the appropriate subroutines are called.

INIT1 -- LB

This routine determines the addresses of the three input buffers IBU1, IBU2, and IBU3 and the two output buffers OBU1 and OBU2. The input pointer (INPT) and the output pointer (OPT) are set to their initial values.

E0ACT -- LC

Input parameter: INPT points to an E0-key.

The routine resets the error switch ERRSW and clears the error stack ERROSTK. The begin statement is written out and INPT is updated. In addition, the error counter ERRCT (set to indicate previously detected errors) is set to zero.

EAACT -- LD

Input parameter: INPT points to an EA-key.

An EA key indicates the end of the currently processed statement. If the error switch ERRSW is not 0, a bit is set to indicate that (1) execution is to be suppressed and (2) the end of the statement and the error stack ERROSTK are to be written out. Otherwise, only the end of statement is written out.

EBACT -- LE

The error message marked by an EB key is written out and INPT is increased.

ASKIP, SKIPF7, INCRE -- LF

This routine is used to move input information to the output area. The routine uses the routine SKIP to increase the constants of INPT.

SKIP -- LG

Input parameter:
Register 0 contains the length by which the input pointer INPT is to be increased.

INPT is increased by the specified length and checked to determine which of the buffers (1, 2, and 3) INPT points to. If INPT points to buffer 2 or 3, buffers 2 and/or 3 are shifted to the left and INPT is updated accordingly. The following records are read into buffers 2 and/or 3.

READIN -- LH

Input parameter:
IREC (1 byte) indicates the number of records to be read into the input buffers.

This routine controls the reading of the input string into the input buffers.

MOV31, MOV21, MOVO21 -- LI

Routine MOV31 moves the contents of input buffer IBU3 into IBU1. Routine MOV21 shifts the contents of input buffers 2 and 3 to the left by one buffer length. Routine MOVO21 moves the contents of the second output buffer into the first output buffer.

MOVOUT -- LJ

Input parameters:
REG0 contains the length of the data to be moved;
REG1 contains the address of the area that contains the data to be moved.

The routine moves the input string or data as determined by registers REG0 and REG1 into the output buffer. Control is then transferred to the OUT routine.

OUT -- LK

Input parameter:
R0 contains the value by which OPT is to be increased.

The routine updates OPT and, if the buffer is full, writes a record.

EOPACT -- LL

Input parameter:
INPT points to the end-of-program key (EOP).

The routine causes the last record to be written on TXTOUT and determines whether or not the next phase (D20) is to be called. If phase D20 is not to be called, phase D40 is called.

IJKMNN -- LM

This routine fetches a name from the communication region IJKMVC and stores this name in NAME. The name counter is increased by 1. If the value in the name counter exceeds 64K, an error message is passed on; otherwise, the name counter is restored.

INTREST -- LN

When an FC-key is found, the main routine calls the appropriate macro generation routine. A translate table is used to determine the proper macro generation routine.

Before the selected routine is entered, the number of arguments is checked. If an improper number of arguments is provided, control is returned to the main program.

N1, N2, N3, N4, N6 -- LO

The routine checks the number of arguments specified by the user. In case of an error, the error routine SETERR is called.

SETERR -- LP

This routine moves the error message into the error stack and increases the error pointer ERRSW.

FUNCTA -- LQ

This routine generates the parameter block for the linkage. An internal macro is generated to load the address of the parameter block into register 1.

If all arguments are in automatic storage, the parameter block is generated in the general working storage of the outermost block. Otherwise, the parameter block is generated in static storage.

If the parameter block is generated in the general working storage of the outermost block, an address is calculated and inserted for each parameter in the parameter block during object time. If the parameter block is generated in static storage, an address constant is generated for each of the parameters.

Abbreviations used in the flow chart:

ADBL	=	Stack containing the pointers to the arguments.
GWS8	=	Internal name of general working storage.
MAK0	=	Move-Address macro instruction. See the description of macro instructions in the section Phase Input and Output.

REPROUN -- LR

REPROUN is a secondary entry point of the FUNCTA routine. It exchanges the pointers to the arguments in block ADBL to permit the linkages for the REPEAT and ROUND functions to be handled in the same way as for other functions.

ARGADR -- LS

When this routine is entered, INPT points to the FC-key of a function. The routine scans the function and stores the pointers to the arguments in reverse order of their appearance in block ADBL. In addition, the arguments are checked to determine if they are in static storage.

GENADCO -- LT

Input parameter:
R5 points to the argument.

The routine generates an address constant and moves it into the output buffer. The address constant has the following format:

(1)	(2)	(2)	(1)	(2)	(2)	(2)
FD	000C	Int. Name	60	0004	Int. Name	Modif

Bytes 8 through 12 are the name and the modifier of the argument for which the address constant is to be generated. Bytes 3 and 4 contain the name of the parameter block if it is the argument in the block; otherwise, these bytes contain 0.

GENCO -- LU

Input parameter:
R5 points to the argument.

The routine generates a 4-byte constant which is not optimizable. If it is the first constant in the parameter block, the name of the constant is obtained from the name counter in the communication region. Otherwise, the name field of the constant is set to 0.

PMAKO -- LV

This routine writes out the internal macro MAKO. For the format and the calling sequence of this macro refer to the section Phase Input and Output.

PUCO -- LV

Input parameter:
R4 contains the address of a 34-byte entry for the constant.

If the argument is a constant required during object time, the skippable length is calculated and the constant is written out with an FD-key.

SPECFUN -- LW

The routine determines if the function concerned is one without standard linkage. If so, the routine calls the corresponding linkage-generating routine to generate the additional linkage information.

BOOLF -- LX

For the BOOL function the internal MVI macros are generated in addition to the standard linkage in order to store the lengths of strings in the parameter block. See the description of type-E linkages in the section Built-in Functions Processed.

MAMIFI Routine -- LY

The function key is tested to determine if it is for a MAX or a MIN function. If it is a function key for a MAX or a MIN function, the DEDs for the arguments are generated and the address constants for the DEDs are passed on.

INCHARF -- LZ

This routine generates internal MVI macros and stores the lengths of arguments in the parameter block for the INDEX and REPEAT (BIT) functions.

HIGHLOW -- MA

This routine generates the required internal macros for the functions:

```
      TARG = HIGH(X) and
      TARG = LOW(X).
```

Code produced is:

```
1.  if X = 1:
      MVI   TARGET,X'FF' or X'00'
2.  if X ≥ length of target:
      MVI   TARGET,X'FF' or X'00'
      MVC   TARGET+1(LT-1),TARGET
3.  if 1 < X < length of target:
      MVI   TARGET,X'40'
      MVC   TARGET+1(LT-1),TARGET
      MVI   TARGET,X'FF' or X'00'
      MVC   TARGET+1(X-1),TARGET
```

ADDRF -- MB

This routine generates an internal Move Address macro for the address function. For the code produced, refer to the description of the MOVE ADDRESS MAKO macro in the section Phase Input and Output.

MODF -- NA

This routine generates internal MVI macros that store the scale factor in the first byte of each of the addresses in the parameter block. Refer to the description of type-Q linkages in the section Built-in Functions Processed.

FLCEILF -- NB

This routine generates an internal MVI macro that stores the scale factor Q+128 in the first byte of the source address. Refer to the description of type-N linkages in the section Built-in Functions Processed.

FLCLDF -- NC

This routine generates the DED (Data Element Descriptor) for a function that requires a type-O linkage. Refer to the description of type-O linkages in the section Built-in Functions Processed.

GENDEDAD -- ND

Input parameter:
NAME (two bytes) contains the name of the DED.

The routine generates an address constant for the DED.

MODDF -- NE

This routine generates the DED for a function that requires a type-Q linkage. Refer to the description of type-Q linkages in the section Built- in Functions Processed.

ROUNDBF -- NF

This routine generates the DED for a function that requires a type-R linkage. Refer to the description of type-R linkages in the section Built-in Functions Processed.

CONVN -- NG

Input parameter:
R4 points to the 34th byte of argument N;
N is converted to a binary integer.

The routine is entered if the programmer has specified a decimal integer. This decimal integer is converted to binary fixed and stored in N (one byte).

REPE -- NH

This routine generates internal MVI macros for the REPEAT function.

The MVI macros are generated to store, at object time, the length of the bit string and the repetition factor N in the parameter block. Refer to the description of type-C linkages in the section Built-in Functions Processed.

EXPOA -- NI

This routine generates internal Move-Address macros to properly pass on the parameters. Refer to the description of type-I linkages in the section Built-in Functions Processed.

PARG -- NJ

This routine writes out the internal Move Address macro and determines whether or not the argument pointed to by R4 is a constant. If so, the constant is written out using the routine PUCO.

EXPOB -- NK

This routine generates the linkage for a general exponentiation. Refer to the description of type-J linkages in the section Built-in Functions Processed.

DED -- NL

This routine fetches an internal name for the DED from the communication region and generates an internal Move-Address macro in order to pass on the address of the DED.

DEDGEN -- NM

Input parameters:
R4 points to a 12-byte entry;
NAME (two bytes) contains the internal name to be given to the DED.

The routine generates the DED according to the type specified in the 12-byte entry and writes out the DED constant.

DYNDUMP -- NN

This routine generates the linkage for the DYNDUMP library module. Refer to the description of type-M linkages in the section Built-in Functions Processed.

PLENG -- NO

The length of the currently processed argument is normally fetched from byte 7 of the entry. If the argument is an array or a structure, the length is fetched from bytes 10 and 11 and then stored in the length block LBLOCK.

TIMDAT -- NP

The routine generates (1) an internal LA-macro to pass on the address of the target and (2) an internal Library-Call macro for the functions TIME and/or DATE.

FLTRO -- NQ

The routine generates the internal macros for the ROUND (float) function. Refer to the description of the ROUND built-in function for floating point in the section IN-Line Functions.

An internal MAK9E macro is generated to cause the generation of the MVC instruction. An OI macro causes the generation of the OI instruction. For formats of the macros, refer to the section Phase Input and Output.

LIBCALL -- NR

The routine generates the library macro instruction for the appropriate built-in function and calls the BITSET routine.

BITSET -- NS

The routine sets the appropriate bit in the library bit string of the communication region.

TSCSIJ -- NT

This routine evaluates: T = SUBSTR (CS,I,J) where CS = character string.

For the code produced and the format of the internal SUBSTR MAK88 macro which produces the MVC instruction to move the substring to T, refer to the description of the SUBSTR MAK88 macro in the section Phase Input and Output.

CHECKJK -- NU

The routine determines whether or not the following requirement for the SUBSTR function is met: 1 < j < k

If the requirement has not been met, an error message is produced.

SCSIJT -- NV

This routine evaluates SUBSTR(CS, I, J) = X. The generated calling sequence is:

```
LA      R5,CS
A       R5,I
BCTR    R5,0
```

For the format of the internal macro MOVE MAK88 which is generated to produce the MVC instruction(s) that moves X into the substring area, refer to the description of the MOVE MAK88 macro in the section Phase Input and Output.

TBSBST -- NW

This routine evaluates: T = SUBSTR(BS, I, J) and SUBSTR(BS, I, J) = T
where BS = bit string.

For the code produced as a calling sequence, refer to the description of type-H linkages in the section Built-in Functions Processed.

TSUBST -- NX

This routine generates the DED for the source if T = SUBSTR(BS,I,J)
or for the target if SUBSTR(BS,I,J) = X.

SUBSTT -- NY

The routine generates the DED for the target if T = SUBSTR(BS,I,J)
or for the source if SUBSTR(BS,I,J) = X.

EXCHP -- NZ

The routine exchanges the pointers to the arguments if SUBSTR(BS,I,J) = X.

This phase scans the input stream for functions not processed in phase D17. When an FC-statement key is detected, the appropriate subroutine (selected according to a special key in the FC-statement) is called for processing the function. Upon completion of processing, the FC-statement is skipped and scanning continues. All statements that do not have an FC key are either moved unchanged into the output stream or skipped.

Input and output handling of this phase is the same as that of phase D17. If no FC key was detected in phase D17, this phase is skipped.

Phase Input

The built-in functions processed in this phase have one to four source arguments and one target. The source and target arguments follow one another in inverse order of the function F=F(X1,X2,X3,X4). The standard input format is as follows:

FC statement -- 12 bytes
Target operand -- 12 bytes
Argument Xn(n4) -- 12 bytes
...
Argument X1 -- 12 bytes

Note: If one of the arguments is a constant, the 12-byte argument is immediately followed by the information on the constant.

Format of the FC Statement

byte	0	statement key
bytes	1-2	skippable length (includes the 12 bytes of the FC statement + 12 x number of arguments)
byte	3	number of arguments (including target)
byte	4	X'13' (indicates that this is a library function)
byte	5	internal library function name (00 - FF)
bytes	6-12	not used

Format of Arguments

The format of variable and register arguments is as follows:

byte	0	key: E1 refers to a declared variable; E5 refers to a register pointing to data.
bytes	1-2	internal name (number of register or internal number used for variable)
bytes	3-4	modifier: refers to internal name
byte	5	attributes 1
byte	6	attributes 2
byte	7	object-time length in bytes
byte	8	precision
byte	9	scale factor
bytes	10-11	not used

If the argument is a constant, the following information is appended to the 12-byte argument:

bytes	12-13	internal name
bytes	14-17	attributes
bytes	18-19	length of constant
bytes	20-31	intermediate form of constant (left-aligned)

The array functions SUM, PROD, ALL, and ANY use the following non-standard input:

bytes	0-11	FC statement
bytes	12-22	array statement. It has the same format as the other arguments (with the exception of the last byte - see byte 23 below). The array elements are converted to equal base and scale. The array argument is the source of conversion.
byte	23	number of array elements
bytes	24-nn	conversion macro instructions of the length xx. The F8 key of the following statement indicates the end of the conversion macro instruction and the end of the function input. The F8 statement contains the information on the target of conversion. The conversion macro instructions are moved into the output as part of the function output.

Phase Output

With the exception of the library functions processed in this phase (identified by an FC key), all statements are moved unchanged into the output stream. Constants in the argument operands cause the generation of a constant with an F3 key. For the string array functions ALL and ANY, a string variable is generated to accommodate the provisional result.

The generated output macro instructions (except for the F2 key, the skippable length, and the special macro instruction KEY) generally have a target and source operands. These operands are identical to the first six bytes of the input operand. Label, length, and precision information is added to the macro instruction as required. Length and precision are taken from the attributes of the input operands. The output formats for the individual libary functions are as follows:

ABS

```
byte       0  X'F2'
bytes   1- 2  skippable length
byte       3  key: 3B - long float
                   2B - short float
                   1B - decimal fixed
                   0B - binary fixed
bytes   4- 9  target argument
bytes 10-15  source argument
```

SIGN

```
byte       0  X'F2'
bytes   1- 2  skippable length
byte       3  key: 2A - float
                   1A - decimal fixed
                   0A - binary fixed
bytes   4- 9  target
bytes 10-15  source
bytes 16-17  label
bytes 18-19  length (inserted only if the
             source is decimal fixed
```

ADD, MULTIPLY, DIVIDE

The standard macro instructions ASSIGN, ADD, MULTIPLY, and DIVIDE are used for these functions. The number, order, and type of the macro instructions used depend on the scale and precision of the arguments and on the arithmetic operation. The ASSIGN macro instruction is used to assign one or two arguments to a register or working storage. The result of the arithmetic operation is always stored in a register. If required, the register is assigned to the target.

SUM, PROD

An example of the output format (and the generated code) for these functions is shown in Figure 1.

ALL, ANY

An example of the output format (and the generated code) for these functions is shown in Figure 2.

DESCRIPTION OF ROUTINES

Subroutines Described in D17

INT1	: Initialization routine (get buffer length, set buffer pointer)
EOACT	: Action for E0 key (reset error switches, output of statement begin).
EAACT	: Action for end-of-statement key (move out error stack, output of statement end).
EBACT	: Action for EB key (move out error message).
ASKIP	: Increase input pointer (move input into output area).
READIN	: Read records into input buffer.

ABS, SIGN -- OA

A special macro instruction (see the macro generator phases E50-E60) is provided for these functions. Thus, only base and scale must be tested to obtain the correct macro instruction. The operands are moved unchanged from the input stream into the macro instruction.

ADD, MULTIPLY, DIVIDE -- OB, OC

These functions are not used in version 1 of this compiler.

EOPACT -- OE

This subroutine terminates the phase after all input has been processed. It generates the string variables and register save areas required for the array built-in functions processed in this phase. Finally, it loads phase D40.

INTREST -- OD

This subroutine is called if the FC key indicates a built-in function to be processed in this phase. The appropriate processing routine is selected by means of the function key, which is part of the input. The output of the code macro is part of the processing routine for the respective function. On return from this routine, the input pointer is increased and the next statement is scanned.

SUM, PROD, ALL, ANY -- OF

Because of their similar output, these array functions are handled in nearly the same way. They differ in their key and in the length of the macro instructions only.

The string variable in the string array functions ALL and ANY replaces the floating-point register to hold the partial result. The initialization values for the individual functions are as follows:

SUM	- zero
PROD	- floating-point 1
ALL	- all ones
ANY	- all zeros

ASGN -- OI

Depending on the attributes of source and target, this subroutine generates the appropriate assign macro instruction. (For a detailed description of the macro instructions refer to the macro generator phases E50/E60.) The following five cases may occur:

decimal	to	decimal
binary fixed	to	binary fixed
short float	to	short float
long float	to	long float
short float	to	long float

Source and target (pointed to by registers 5 and 6, respectively) must have the same base and scale. The macro instruction key is determined by means of the source attributes (base and scale) and the precision. The field width is calculated by means of the precision and the scale factor. The generated macro instruction is moved into the output stream.

This subroutine is called as follows:

```
    BAL  LR, ASGN
```

SETPNT -- OL

This subroutine sets pointers to one or two source and one target arguments in the input stream. R5 is used to point to one source argument; R6 and R5 are used to point to the first and second source argument, respectively; R4 is used to point to the target argument.

Precision arguments are disregarded; the precision is always taken from the argument attributes. Contrary to argument constants, precision constants contained in the input stream are skipped. The two types of constants are differentiated by their appearance in the input stream. Therefore, every function calling this subroutine furnishes the parameters for the expected minimum and maximum number of arguments. An error message is produced if the actual number of arguments in the FC statement is either lower than the minimum or higher than the maximum. The subroutine is called as follows:

```
    BAL  LR, SETPNT
```

Parameters:

R0 = number of required arguments
R1 = number of actual arguments
R2 = maximum number of arguments permitted in this function.

MOVOUT

This subroutine moves a character string into the output buffer. R0 contains the length and R1 the begin address of the string.

GENLAB

This subroutine fetches a label name from the variable counter. The counter is increased each time a label is generated. If the counter overflows, an error message is produced and the counter is reset.

SETERR

This subroutine stores the error number (ERRNR) in the error stack up to a maximum of 8.

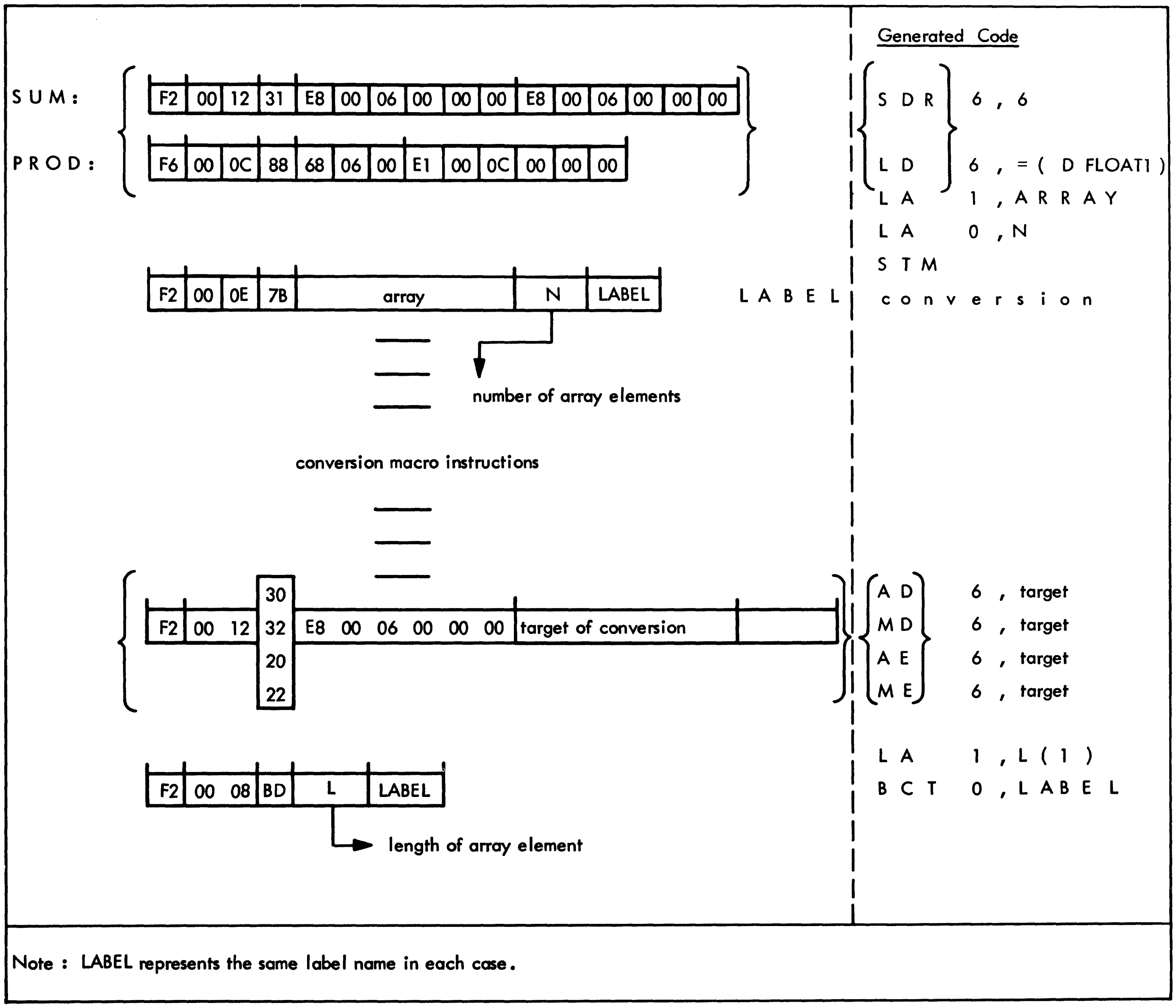
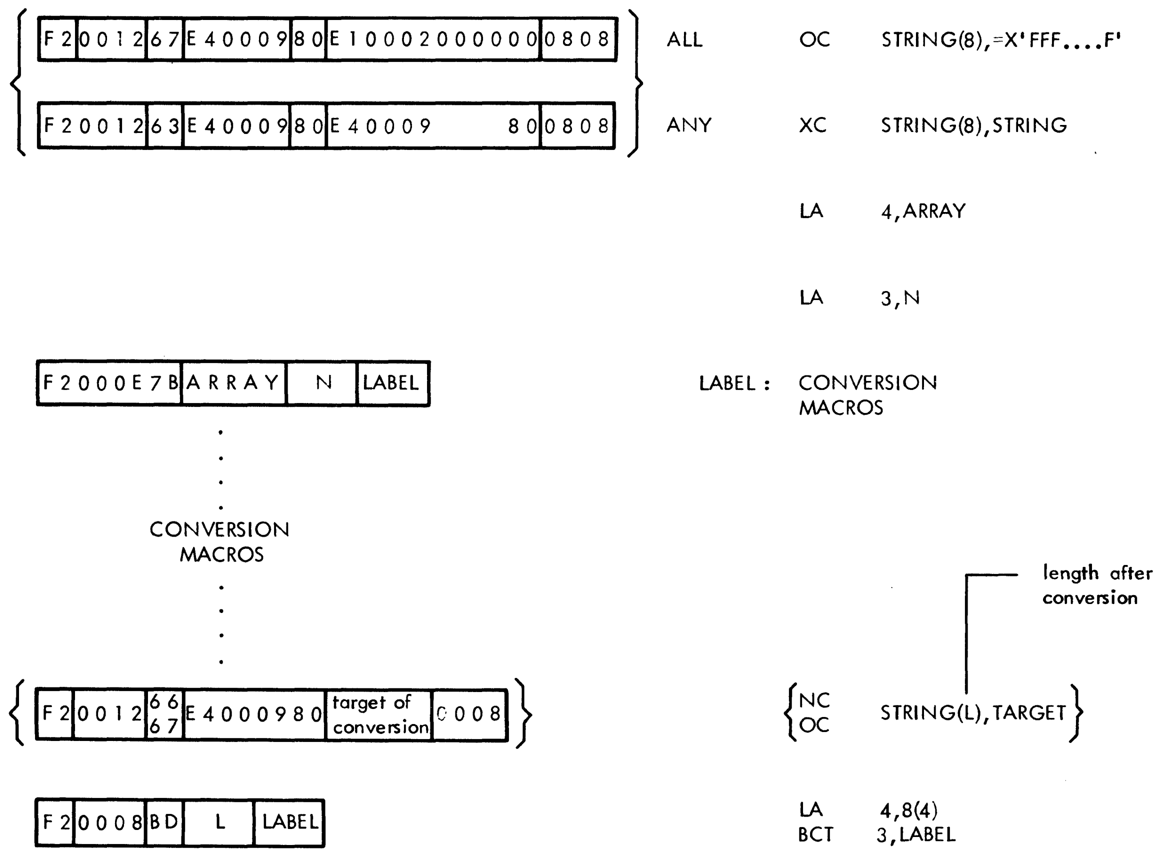

Generated Code
SUM:
F2 00 12 31 E8 00 06 00 00 00 E8 00 06 00 00 00
PROD:
F6 00 0C 88 68 06 00 E1 00 0C 00 00 00
SDR 6,6
LD 6,=(D FLOAT1)
LA 1,ARRAY
LA 0,N
STM
F2 00 0E 7B array N LABEL
LABEL conversion
number of array elements
conversion macro instructions
F2 00 12 30 32 20 22 E8 00 06 00 00 00 target of conversion
AD 6, target
MD 6, target
AE 6, target
ME 6, target
LA 1,L(1)
BCT 0,LABEL
F2 00 08 BD L LABEL
length of array element
Note : LABEL represents the same label name in each case.

Figure 1. Example of Output Format for SUM and PROD

IBM Confidential


F 2 | 0 0 1 2 | 6 7 | E 4 0 0 0 9 | 8 0 | E 1 0 0 0 2 0 0 0 0 0 0 | 0 8 0 8
F 2 0 0 1 2 | 6 3 | E 4 0 0 0 9 | 8 0 | E 4 0 0 0 9 8 0 | 0 8 0 8
ALL
ANY
F 2 0 0 0 E 7 B | A R R A Y | N | LABEL
CONVERSION MACROS
F 2 | 0 0 1 2 | 6 6 6 7 | E 4 0 0 0 9 8 0 | target of conversion | 0 0 0 8
F 2 | 0 0 0 8 | B D | L | LABEL
length after conversion

```
         OC     STRING(8),=X'FFF....F'
         XC     STRING(8),STRING
         LA     4,ARRAY
         LA     3,N
LABEL :  CONVERSION
         MACROS
         {NC
         {OC    STRING(L),TARGET}
         LA     4,8(4)
         BCT    3,LABEL
```

Figure 2. Example of Output for ALL and ANY

Input

The input text stream for this phase consists of source statements in internally coded form, and of both E-key and F-key items generated and inserted into the text by previous phases.

ON, REVERT, SIGNAL, and STOP statements are in the output format of phase A65. Some I/O statements are still in an intermediate format and are to be processed later by the subsequent compiler phases D75 and D80.

Each statement in the source program results in the following input to phase D40: statement identifier key X'E0', end-of-statement key X'EA', and, if any, error keys X'EB'; located between each of these begin- and end-of-statement keys are either E- or F-keys representing the function of the statement, or the statement in internally coded form.

Blocks in the input text are not nested. All procedure blocks or begin blocks have been filed one after the other in phase C35.

Output

The following elements have been processed:

Block description table for static storage,
Prologue macro,
Certain F-key elements,
Statement prefixes,
ON statement,
REVERT statement,
SIGNAL statement,
STOP statement

Appropriate macros have been substituted for each ON, SIGNAL, REVERT, or STOP statement. Certain entries in each prologue macro have been made. A Block Description Table for each program block has been inserted into the text.

Macros have been inserted for each statement with an individual prefix, signaling these statements during execution.

No input/output action on SYS001 occurs in this phase. The processing of the prologue macro and of the REVERT statement require two text scans in phase D40.

FUNCTIONAL DESCRIPTION

The Block Description Table generated during this phase is part of the static storage during object program execution. Each compilation block is represented in this table by a single block description in the format shown in Figure 1.

<-----------------4 bytes------------------->

OFFSET	PREFIX
A(ENTRY1)	
A(ENTRY2) . . .	
A(ENTRYn)	
ON ENTRY1	
ON ENTRY2	
. . .	
ON ENTRYn	

OFFSET	Contains offset to first ON entry, if present. Otherwise ignored.
PREFIX	Dynamic and static prefix byte as described in the library control routine.
A(ENTRY1)	Address of the first entry point of the block.
A(ENTRYn)	Address of the nth entry point of the block.
ON ENTRYn	Present for each ON condition mentioned in the block.

Figure 1. Format of Block Description Table

Each block description entry is generated during scanning of the corresponding block in the text stream.

The half-word for PREFIX consists of a dynamic and a static prefix byte.

The static prefix byte will be set up during phase D40 according to the prefix status at the beginning of the block. Whenever a single statement in the source program of the compilation is prefixed, certain macros are inserted into the text stream, providing a corresponding status of the dynamic prefix byte during execution of this statement. (See pertinent section on prefixes).

If an entry point for a block is encountered during scanning of the input text stream, an address constant is generated and added to the block description entry. This address constant contains the name of the entry in internally coded form.

Each generated block description entry is named internally. These names form a string of labeling numbers, used for designating single block descriptions. Each individual name is created by adding 1 to the current value of the communication region entry IJKMNN. Each block description entry will be keyed in such a form that a contiguous portion of the static storage in the object program is occupied by the entries.

Generation of single ON entries in the block description is discussed in the section dealing with ON statements.

Processing of the Prologue Macros. A prologue macro has been generated during preceding phases for each entry point to a begin or procedure block. The format of these prologue macros is discussed in the description of phase E50. During phase D40, three entries in each prologue macro are completed:

1. The entry name of the block in internally coded form. This entry name is contained in the last label macro preceding the prologue macro.

2. The internal name of the Block Description Table. This entry is completed during the second text scan.

3. X'03' is entered into a reserved byte of the prologue macro, if the block includes ON statements. X'01' is entered, if no ON statements are included. This entry is completed during the second text scan, after the block has been checked for ON statements during the first scan.

Eliminating Special F-key Elements. The input text includes X'F7' keys, which served to exclude statements from processing during preceding phases. These keys are eliminated. Certain types of previously excluded statements may now be processed.

Label macros of the form X'F2000772......' are eliminated during the first text scan because they are now obsolete. These label macros, located outside the associated block, specified block names.

Processing of Prefixes. Each statement contains the prefix mask assigned to it according to the prefix lists in the source program.

If a PROCEDURE or BEGIN statement is encountered, the associated prefix mask becomes the static prefix byte in the Block Description Table for the corresponding block. Certain special macros are inserted into the text stream whenever the prefix of a single statement differs from the prefix of the whole block. These macros are inserted at the beginning of a statement (following the label, if any) and at the end of the prefix scope. The macro inserted at the beginning of the statement initiates modification of the dynamic prefix byte of the Block Description Table in the static storage. The macro inserted at the end of the prefix scope initiates restoring of the block prefix mask to the preceding status. Normally, the end of a statement is also the end of a prefix scope, if the statement has its own prefix. However, GOTO, IF, CALL, and RETURN are an exception to this rule.

For GOTO, the prefix mask must be restored before the macro branch. This is done by a scan performed before branching. Similar procedures are provided for CALL and RETURN.

Into each IF statement an end-of-statement key is inserted after the evaluation of the scalar expression. The scope of the condition prefix ends at this key. However, certain branch macros inside the evaluation of the scalar expression branch to an address beyond the condition scope. For this reason, the condition is reset before branching, and set again after the branch macro with one exception: If the branch macro is followed by an end-of-statement key, the statement prefixes are not restored after the branch macro. In any case, the block prefix is reset at the end-of-statement.

Processing of ON Statements. In the PL/I source program, the ON statement defines the action to be performed if an interrupt occurs for the specified condition. To provide this action during object program execution, the following code is generated:

1. For each condition specified in an ON statement inside a block, a so-called ON-doubleword is set up in the Block Description Table for this block.

2. Specific macros are inserted into the text where the ON statement is located. These macros modify the contents of the ON-doubleword in the Block Description Table.

If an interrupt occurs during execution of the PL/I program, a library routine called 'interrupt handler' scans the static storage for the ON-doubleword corresponding to the condition in the actual block. The performed interrupt action depends on the presence and contents of this ON-doubleword.

The format of the ON-doubleword in the Block Description Table is as follows:

bytes 0-1	ON code
bytes 2-3	file address or not used
byte 4	flags
bytes 5-7	printer to address constant for label

<u>ON Condition Field</u>

X'01'	OVERFLOW
X'03'	ZERODIVIDE
X'04'	FIXEDOVERFLOW
X'05'	SIZE
X'06'	CONVERSION
X'09'	ERROR
X'0A'	ENDFILE
X'0B'	ENDPAGE
X'0C'	TRANSMIT
X'0D'	KEY
X'0E'	RECORD

<u>Flag Byte</u>

bit 0	if on: last ON entry of the block
bit 1	if off: ON statement not executed
bit 2	reserved
bit 3	if on: ON mark = system action
bit 4	if on: ON mark = GOTO statement
bits 5-7	reserved

<u>File Address Fields</u>

This second field is used exclusively for I/O conditions: ENDFILE, ENDPAGE, TRANSMIT, KEY, and RECORD. The field is reserved to contain the address of the corresponding DTF table. If an I/O condition is mentioned in an ON statement, the file name named together with this condition may designate either a file constant or a file variable. If the file name designates a file constant, a macro for the file address is generated and inserted to point to the file address constant of the form AL3(file DTF) in the double word. If the file name designates a file parameter, the file address required in the ON entry depends upon the value of this parameter. For this reason, the actual file address value is moved into the ON entry whenever an ON statement concerning an I/O condition with this file name is encountered. The macro for this move instruction is generated by this phase.

<u>Flag Byte Field</u>

Flag bit 0 is set to either 0 or 1 by this phase in generating the ON entries. The flag bits 1, 3, 4 are set during the execution of the object program: a macro for a MVI instruction is inserted for each ON statement, taking into consideration the setting of flag bit 0.

<u>Pointer Field</u>

The last field in the second word is used if the ON unit is a GOTO statement. The branch of the GOTO statement may be directed to a label variable or a label constant. If the branch is directed to a label variable, a macro for an address constant of this label variable has been generated by a preceding phase. If the branch is directed to a label constant, a macro for such an address constant is generated and inserted into the text. The format of this address constant is discussed in <u>IBM System/360, Disk and Tape Operating Systems, PL/I Subset Library Routines, Program Logic Manual</u>, Form Y33-9008. The address of the label address constant is moved into the pointer field of the ON-doubleword in the Block Description Table. If a GOTO statement is encountered, the invocation count is moved from the DSA into the second word of the label address constant. (The invocation count is discussed in the description of the library routines).

The following is generated during processing of ON statements:

The ON-doubleword (ON-code, flag bit 0, and -- if the file name is a constant -- file address).

Macros for processing during execution (if the file name is a parameter, move file address, - move flag bits. If a GOTO statement is encountered, move label constant address and invocation count).

A macro for a label variable, if a GOTO statement branches to a label constant.

<u>Processing of REVERT Statements.</u> REVERT statements are used to nullify the effect of the last executed ON statement and to cause the action specification to be restored to its status in the immediate, dynamically encompassing block. To achieve this, flag bit 1 of the corresponding ON-doubleword in the Block Description Table is set to 1.

There is a particular housekeeping for the addresses of the individual ON entries in this phase. The ON conditions in a REVERT statement which do not have a counterpart in the ON statement are ignored.

For each REVERT statement associated with an ON statement in the same block, a macro is inserted into the text stream, providing an OI instruction for setting flag bit 1 of the ON-doubleword to 1.

Processing of SIGNAL Statements. The SIGNAL statement simulates the occurrence of an ON condition. For each SIGNAL statement a macro calling the library "ON-handler", is inserted into the text stream. The called library routine requires one or two parameters. Register 1 must point to a constant of the form AL1 (ON Code, 0). For signaled I/O conditions, the address of a file address constant must be moved into the library work space IJKZWSA. If the file address refers to a file name constant, the address constant is generated and inserted into the text (for static storage). If the file address refers to a file name parameter, the address constant has already been generated.

Processing of STOP Statements. The STOP statement initiates termination of a program. A macro calling the library stop routine is inserted into the text stream.

'Data-Housekeeping'. Three work buffers are used for text input and output. Work buffer 3 is used for overlapped text output, work buffers 4 and 5 are used for overlapped text input.

At the beginning of a text scan, two records are read in overlapped mode into work buffers 4 and 5. The scan starts at work buffer 4. The text is scanned sequentially from left to right. Routine KTESCA is called to move the scan pointer to the next text element. If the scan pointer eventually reaches the 5th work buffer, the following action is initiated:

The contents of work buffer 5 are moved to work buffer 4, the scan pointer is decremented by the buffer length, and a new record is read into work buffer 5 in overlapped mode.

The scan now points again to the first text element in work buffer 4, and processing continues.

Text output is performed in both text scans by the routine KONSTOUT. This routine is called whenever an element is to be moved to the text output. KONSTOUT moves this element to the 3rd work buffer. If this work buffer is eventually completely filled, the contents of the buffer are put out in overlapped mode by the current text output medium.

Tables

The core storage area from the end of the second work buffer down to the phase end is occupied by the following three tables:

ON/REVERT Table: Adjacent to the second work buffer, a table is established during the first text scan for all ON conditions stated in ON or REVERT statements of individual blocks. Each individual table entry has a length of 4 bytes. The format of the entry is shown below.

byte	0	flag byte	
		bit 0 on	= last condition mentioned in block
		bit 1 on	= condition mentioned in REVERT statement only
		bit 2 off	= no ON statements in the current block (used in the first table entry of a block only)
byte	1	condition code	

The boundaries of blocks are indicated by the status of flag bit 0. Note that only one entry is generated for each ON condition and the associated file name, though they may occur together in several ON or REVERT statements. After the macros for the Block Description Table have been generated during the first text scan, all entries with flag bit 1 set to 0 are removed from the ON/REVERT table. The remaining entries are grouped together to be used again during the second scan. These entries now contain all conditions that occur in REVERT statements, but not in ON statements of the same block. An exception to this rule is the first entry for a block if its flag bit 2 is set to 0 and its flag bit 0 is set to 1.

During the second scan, the condensed ON/ REVERT table serves to recognize all those REVERT statements which must be treated as Null statements. The first entry of each block is preceded by a two-byte field containing the number of different ON conditions in this block. During the second scan, this number leads to the last ON-doubleword in the Block Description Table.

Entry Name Table: This table is created for the entry names of a block, beginning at the phase end. The entry names are retrieved from the label macros preceding a prologue macro. Each entry name consists of a number, 2 bytes long. If the end of an input text block is reached during the scan, this table contains all entry names. For each entry to a block, a macro is generated as part of the Block Description Table macro to obtain the address constant of this entry name. The symbolic address of this address constant is the entry name: E DC A(E)

This address constant is correct because the name will be treated differently depending on whether it is the name or the value of the address constant. (See the description of phase G40.)

The Entry Name Table is used during the first scan only, and is overwritten during the second scan.

ON-Entry Reference Table: This table is used for the calculation of ON-doubleword addresses. It is created during the second text scan, beginning at the phase end and overwriting the Entry Name Table. Each table entry is 3 bytes long. The first byte contains an ON condition code, bytes 2 and 3 contain either zero, or a file name occurring together with an ON condition in an ON or REVERT statement. This table is used to address the ON-doublewords in the Block Description Table. Calculation of ON-doubleword addresses is based on the location of the associated entry in the ON-entry Reference Table because the sequence of the entries reflects the sequence of ON-conditions as they occur in the text.

During the first scan, each Block Description Table receives as its symbolic address the current value of IJKMNN in the communication region. IJKMNN is incremented by 1 each time a new block is encountered. At the beginning of the first text scan, the current value of IJKMNN is saved in an internal buffer called KMNN. During the second scan, KMNN is incremented by 1 each time a new block is encountered, and thus contains the internally coded, symbolic address of the currently corresponding Block Description Table. The address of an individual ON-doubleword is calculated as follows:

Address = Table symbol + Contents of the 1st half-word of table + 8 L (L = sequential number of entry in ON-entry Reference Table, starting with 0).

DESCRIPTION OF ROUTINES

During the first text scan, macros are generated to create the Block Description Table. The individual routines, used only during this scan, are:

- Main part
- Processing of the prologue macro,
- Insertion of the Block Description Table,
- Processing of the ON and REVERT statements.

Certain subroutines, used in both text scans, are discussed in the section Subroutines.

Main Part of First Scan -- OQ

This routine sets certain pointers to text input and output areas and to the tables in static storage. The first two records are read in from the input text. The routine then enters a loop for scanning the text. During this scan, certain keys are selected, and the appropriate routines for processing these keys are called. A label macro outside a program block encountered is omitted in the output. Text output is accomplished by subroutine KTESCA.

Processing the Prologue Macro -- OR

The prologue macro specifies either the beginning of a new block (Main Entry) or an ENTRY statement in a block.

If the prologue macro specifies the main entry, the following processing is performed: The previous statement prefix is saved as the new block prefix and used in testing each statement of the current block for an individual condition prefix. For the second and all following blocks, the subroutine to insert the Block Description Table is called. The beginning of the next block is marked in the ON/REVERT Table. An initial length of 12 bytes is assigned to the macro for the Block Description Table. The counter for the ON conditions occurring in this block is set to zero. The name for the Block Description Table of the new block is moved from IJKMNN into the macro. IJKMNN is then incremented by 1.

All other prologue macros are processed as follows: The name of the Block Description Table is moved into the prologue macro. The ON offset in this table macro is incremented by 4, and a new entry name (2 bytes) is provided. The length of the Block Description Table macro is then incremented by 9. This macro contains items for each address constant of an entry name.

A test is performed to detect whether a secondary entry point to the block has a condition prefix different from the block prefix. In this case, a warning message is entered into the text.

<u>Insertion of Block Description Table -- OS</u>

This routine inserts the macro for the first part of the Block Description Table, including all address constants for entry names. The value of the ON counter is stored in the ON/REVERT table of the block. A macro is inserted into the text to generate an ON doubleword for each ON-condition occurring in an ON statement. The ON conditions and file names are retrieved from the ON/REVERT table in static storage. All entries for ON conditions occurring in REVERT statements only remain in the table, but are rearranged for the second scan.

<u>Processing ON and REVERT -- OT</u>

A table is assembled. It contains the format of the condition code in the input text (2 bytes) and in the ON-doubleword (1 byte) for each encountered ON condition.

Subroutine KONLOOK is called to set the appropriate pointers to this table, and to set certain pointers to the ON or REVERT statement string of the input text. (The format of this statement string is discussed in the description of the syntax phases). A test is performed to detect whether the ON/REVERT table includes an entry for this condition/file name. If not, a new entry is made into the table for this condition/file name, and the ON counter is incremented. Flag bits are set in the table entry corresponding to the ON or REVERT statement. The ON unit in an ON statement is checked for a correct GOTO, and, if applicable, an error message code is inserted into the text stream. The format of the ON/REVERT statements in the text stream is not changed by this routine.

<u>Main Part of Second Scan -- OU</u>

During the second text scan, processing of the ON, REVERT, SIGNAL, and STOP is completed. The macros for the function of these statements are inserted into the text stream and the condition prefixes of individual statements are processed. The individual routines, used only during this scan, are:

- Main part of second scan,
- Processing of the prologue macro,
- Statement selection
- Processing of the REVERT statements,
- Processing of the SIGNAL and STOP statements,
- Processing of the condition prefixes,
- Processing of the ON statement.

After the end of the input text has been reached, a final text output is performed. Then a test is made to detect whether the first or the second scan is completed.

After the second scan, phase D70 is initiated.

After the first scan, both work files are rewound and switched in their function. Certain pointers are reset and the first two records are read in overlapped mode. The scan then starts by checking for DO statements having an own condition prefix. If such a DO statement is encountered, the statement prefix is set for the scope of the DO header, i.e., the appropriate macros to set and reset the prefix code in the Static Storage Table are inserted. A statement key X'E0' and an F2-macro are tested and the corresponding routines are called. Then subroutine KTESCA is called to scan the current text input.

<u>Processing of the Prologue Macro -- OV</u>

This routine sets the flag bits in the prologue macro indicating the presence or absence of ON statements in the current block. If the prologue macro signals the beginning of a new block, the pointer for the ON/REVERT table in the upper part of the Table Area is decremented to the next block limit. KMNN is incremented by 1, enabling correct addressing of the Block Description Table. (The value of KMNN is the symbolic address of the Block Description Table). For each new block, an On-Entry reference table is created in the lower part of the Table Area during the scan. The pointer to the current table is set when a new block is encountered.

The block prefix is saved to be compared to subsequent statement prefixes of the block.

<u>Statement Selection - OW</u>

This routine distinguishes three classes of statements:

1. ON, REVERT, SIGNAL, STOP statements are processed by the appropriate routines.
2. Statements headed by a prefix macro are processed by a special routine.
3. All other statements are bypassed. No interrupt may occur during execution of these statements.

<u>Processing the REVERT Statements -- OX</u>

Subroutine KONLOOK is called to set certain pointers both in the tables and in the text stream. Then the On-Entry reference table is scanned to detect whether the condition associated to the statement already occurred in the same block. This test is based on numbering and correct addressing in the Block Description Table. If the condition did not occur previously, the ON/REVERT table is scanned for a dummy entry. If this statement has no corresponding ON statement in the same block, the key for a warning message is inserted into the text stream. If the statement has a corresponding ON statement, a new element is added to the table in the lower part of the Table Area and a macro is inserted into the text stream to set the flag bit of the ON-doubleword in the Block Description Table.

Processing of SIGNAL and STOP -- OY

If a SIGNAL statement is encountered, subroutine KONLOOK is called to set certain pointers to the text stream. Then macros are generated and inserted into the text stream to call the library interrupt handler. Some of these macros prepare parameters for the ON condition code and, if necessary, for the file name address.

If a STOP statement is encountered, a macro is inserted into the text stream to call the library STOP routine.

For each SIGNAL or STOP statement, a corresponding call bit is set in the interphase communication region to provide the appropriate library routine in a later compiler phase.

Processing of Condition Prefixes - OZ

This routine is called if a statement has an individual condition prefix and if this condition can occur during execution of the statement. One macro has already been inserted to set the statement prefix in the Block Description Table. This routine inserts the macro for resetting the prefix byte into the text string. For assignment statements, SET statements, dynamic statements, and I/O statements, the correct position to insert this macro is the end of statement key X'EA'. The condition prefix scope ends here.

For CALL, GOTO, and RETURN statements, the condition prefix is reset if a branch macro is encountered and before branching.

For IF statements, the scope of the prefix ends together with the evaluation of the IF expression. However, one or more branch macros are encountered during evaluation. The length of the text stream for the evaluation may exceed the buffer length. For this reason, a reset macro is inserted to precede each branch macro of this evaluation. Another macro to set the statement prefix is inserted after each branch macro, with one exception: if the branch macro is followed by an end-of-statement key, nothing is inserted. This procedure ensures that the block prefix is reset after expression evaluation.

Processing of ON Statements - 01, 02

Subroutine KONLOOK is called to set pointers to both tables and in the text stream. The On-Entry reference table is looked up to detect whether the condition occurred already in the same block. If the condition has not yet occurred, a new entry for this condition and file name is made in the table.

If a file parameter occurs in the ON statement, a macro is inserted into the text stream to move the file address into the corresponding ON-doubleword.

The ON unit is tested and the appropriate flag bits for the ON-doubleword are prepared. If the ON unit is a GOTO statement with a label constant, two macros concerning the label constant are inserted into the text stream. The first macro provides a label address constant, the second macro moves the address of the label address constant into the ON-doubleword. If the ON unit is a GOTO statement with a label variable, the address constant macro for this label variable has already been inserted in a previous phase. In this case, the label name designates this address constant. Only the second macro is inserted to move the address into the ON-doubleword.

The macro to set the flag bits in the ON-doubleword is inserted into the text stream. If a null statement is used in the ON unit for an ENDFILE, KEY, or CONVERSION condition, an error code is inserted into the text stream.

KTESCA -- 03, 04

This routine moves the text scan pointer and performs certain tests. If an E-key element is encountered, the pointer is modified by a fixed length depending on this E-key.

If an F-key signaling the end of text is encountered, the main part of the second scan is called (if not yet processed).

If any other F-key element is encountered, several tests are made. Certain macros are selected which require setting of a special prefix byte in the Block Description Table. The object code of these macros may cause a fixed overflow interrupt which is to be interpreted as a SIZE condition by the library interrupt handler.

To signal the SIZE condition to the library interrupt handler, a second macro is inserted preceding the macro causing the interrupt. This inserted macro sets bit 7 in the dynamic prefix byte of the Block Description Table to one. A third macro is inserted after the interrupt-causing macro. The third macro resets bit 7 to zero.

If the F-key element is of the form X'F7....', the byte containing X'F7' and the three bytes following it are eliminated.

If the F-key element does not begin with X'F7', the length for modifying the scan pointer is retrieved from bytes 2 and 3 of the element. If this length exceeds the buffer length, pointer modification is accomplished in several steps.

KCONDOUT -- 05

This subroutine checks whether the scan pointer reached the second text input area. In this case, the following steps are performed:

1. The scanned text is put out. The contents from the second text buffer are moved into the first text buffer.
2. A new record is read in overlapped mode into the second text buffer.
3. The scan pointer is decremented by the buffer length.

KONSTOUT -- 06

Before this subroutine is called, register 1 must contain the start address of the information to be added to the output text and register 2 must contain the end address + 1 of this information. Output is done in one or several steps, depending on the length of the information area and on the available space in the output area. The information to be put out is accumulated in the text output area. If this area is eventually filled to capacity, output is accomplished in overlapped mode. Another portion of information is then accumulated in the output area.

KINTER, KOTE -- 07

If the program is too big, one of the tables in the Table Area will eventually overflow. In this case, KINTER is called to enter a severity error code and to truncate the text at the current position of the scan. Compilation is terminated.

Subroutine KOTE is called when text is to be put out. Text output is accomplished in portions beginning at the current text start address and ending at the current position of the scan pointer.

KONLOOK -- 08,09

This subroutine is called whenever an ON, SIGNAL, or REVERT statement is encountered. These statements may have a corresponding prestatement table. KONLOOK sets pointers to the begin and to the end of this prestatement table. Other pointers to be set by KONLOOK are:

1. Pointer to the ON condition table of this phase,
2. pointer to I/O conditions in the ON condition table, and
3. pointer to the ON/REVERT table in the upper part of the Table Area.

If an I/O condition is posted in the statement, a test for correct file name reference is performed. If the file name is incorrect (symbol for file name does not reference a file) an error code is inserted into the text and elimination of the statement is initiated. If KONLOOK is called during the first text scan, this elimination is nullified by the calling routine.

PHASE PL/I D70 (PROCESSING CONSTANTS II) -- PK

This phase does the following:

1. It performs the conversion of constants from the base, scale, and precision specified by the programmer to the base, scale, and precision needed at object time. The "new type and precision" of the constants are determined in phase D05.
2. To get the internal representation of the decimal and binary floating-point constants, i.e., to perform the conversion from base decimal and binary to hexadecimal.

Phase Input and Output

In previous phases, the source text is converted into a stream of elements (macros, statement-identifier keys, end-of-statement keys, error-keys, and constant tables).

The constant tables consist of the following:

- X'F3' = key for constant-table,
- length of table (2 bytes),
- One or more constant entries. These entries consist of the following:

  Internal name of the constant (2 bytes).

  Attributes of the constant (1 byte):

  Bits 2-3:

00 =	optimizable
01 =	delete
10 =	constant is not optimizable but containing block is optimizable
11 =	constant should not be deleted

- Old type of constant (1 byte; see phase C35).
- Old precision of constant (1 byte; see phase C35).
- New type of constant (1 byte).

```
Bit  0-1: negation
Bits 4-7: 0= float
          1 = binary fixed
          2 = float
          3 = decimal fixed  (packed)
          4 = decimal fixed  (zoned)
          5 = decimal fixed  (zoned T)
          6 = character string
          7 = bit string
```

- New precision of constant (2 bytes)
```
Byte  0: P if arithmetic constant
         L if character string or bit
           string
Byte  1: Q if fixed point constant.  If
           floating point constant:
             0= short, ≠ 0 = long.
```
- Length of the following intermediate representation of the constant (2 bytes).
- Constant (intermediate representation).

Note: NT = 0 and NP = 0 means: NT = OP and NP = OP, i.e. no conversion takes place unless OT = binary or decimal float.

The individual entries of the constant tables prepared for output have been reduced to the following:

- Internal name of constant (2 bytes)
- Attribute of the constant (1 byte)
- Length of constant that follows

With the exception of floating-point numbers, the internal representation of the constants is the same as the intermediate one (see phase C35). The floating-point constants contain the following.

- (1 byte) :
  bit 0 : sign : 1 = neg., 0 = pos.
  bits 1-7 : hexadecimal exponent (excess 64 number)
- (variable) :hexadecimal fraction of 6 or 14 hexadecimal digits depending on whether it is a short or long fraction, respectively. (See IBM System/360 Principles of Operation, Form A22-6821).

The decimal fixed-point ZONED constants are stored as unsigned integers, each digit occupying one byte, in ZONED T constants the zone of the rightmost digit is replaced by the sign. (See IBM System/360 Principles of Operation, Form A22-6821).

Initialization -- PL

This routine initializes pointers, switches, etc., and reads in two buffers of input text.

FGSC -- PM, PN

This is part of the main routine of the phase. It scans the input stream, bypasses and moves all program elements, except end-of-program-keys and constant tables, into the output buffer. Each new constant table is built up in the table space, and, after all constants of the corresponding old table have been processed, it is moved into the output buffer. Constants with the delete-bit on and the optimizable-bit off are deleted from the table. Constants which do not have to be converted are moved unchanged into the new constant table; only the 5 bytes including types and precisions are deleted.

If the table space is filled and there are still some constants to be processed, the new constant table is written onto the text output file and a second one is built up. When the end-of-program key is found, it is moved into the output buffer, which is written onto the text output work file. Control is passed to IJKPH to call phase D75 or D80 depending on whether the job-information-bit (bit 28) is on or off.

FEOS -- PP

This routine is part of the main routine. It is called when an end-of-statement key is found during the scan of the input text. If it terminates an erroneous statement (i.e., if bits 8-11 of the EOS key are not all zeros) the job-information bit (bit 1) is set on. Before doing this, bits 8-11 of the EOS key have been OR-ed by the severity-code bits of any errors occuring during conversion of a constant belonging to this statement. The error-tail of the end-of-statement key is extended by the error key(s), and the respective error number(s) is stored in the error table. A macro is generated to signal the SIZE condition at object time.

Note: Only an end-of-statement key with bit 15 on indicates the termination of an I/O statement.

JIRN -- PO

Input parameters:
PIN = contains address of source
POUT = pointer of output buffer
BYZ = number of bytes to be moved

Output parameters:
PIN = PIN+BYZ
POUT = address of the next available byte in the output buffer.

This routine loads the output buffer. If all the bytes to be moved do not fit into the output buffer or if they do exactly fit, the buffer is filled by the first part of the text to be moved. The buffer is written onto the text output work file. The remaining bytes, if any, are moved into the buffer (left-aligned).

FERR -- PQ

This routine prepares parameters to note error number 64 (CONSTANT CONVERSION UNDEFINED DUE TO SIZE-ERROR) in the error table.

FERRUC -- PQ

This routine is branched to if a conversion to character string other than from bit string is requested. It prepares parameters to note error number 146 (ILLEGAL CONVERSION OR COMBINATION OF DATA TYPES) and deletes the constant.

JERR -- PQ

Input parameter:
R0 = error number

This routine notes up to 8 errors in the error table.

FPIN -- PR

Input parameters:
PIN = input pointer
PIN = contains the address of the first byte of the input text which is to be moved into the output buffer.

This routine is called each time the input pointer is increased. It updates the input pointer so that it points always to an address within the first of the two input buffers. When the input pointer goes beyond this range, all source text from PINS up to PIN is moved into the output buffer. If not all the text can be read in one move, several steps are used. If necessary, the input text is moved to the left so that the input pointer points to an address within the first buffer.

Secondary Entry Point: FPINX
This part of the routine moves the input text to the left and reads in a new record without moving any text into the output buffer.

FCON -- PS

Input parameter:
PIN = points to the constant-table entry of the constant to be converted.

This routine prepares the parameters for one of the conversion routines. The conversion routines convert the old type and precision of the constant to the new type and precision. The conversion routines are:

FBSL	bit string to float
FBIL	binary fixed to float
FBLL	binary float (intermediate representation) to float

FSBI	bit string to binary fixed
FBII	binary fixed (change and check for precision)
FSDI	bit string to decimal fixed
FBID	binary fixed to decimal fixed
FDDI	decimal fixed (change and check for precision)
FSCS	bit string to character string
FBLS	binary float (intermediate representation) to bit string
FBIS	binary fixed to bit string
FBSS	bit string (change and check for precision)
FNDI	binary float (intermediate representation) to decimal fixed
FLDI	float to decimal fixed
FNBI	binary float (intermediate representation) to binary fixed
FDIS	decimal fixed to bit string
FDIL	decimal fixed to float
FDFL	decimal float (intermediate representation) to float
FDLS	decimal float (intermediate representation) to bit string
FDLB	decimal float (intermediate representation) to binary fixed
FLBI	float to binary fixed
FDIB	decimal fixed to binary fixed
FDLD	decimal float (intermediate representation) to decimal fixed

<u>FBSL -- PT</u>

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the bit string left-aligned.

This routine converts the bit string to float. The bit string is first converted to binary fixed (precision (31,0). Then control is transferred to FBIL.

<u>FBIL -- PT</u>

This is a secondary entry point of FBSL.

Input parameters:
PIN = address of constant-table entry
CONS = (fullword) contains the binary fixed-point constant.
RBFI = R5 (general register) same as CONS.

The routine converts binary fixed point constants to floating point. The parameters are transformed into parameters for FBLL, i.e., the constant is interpreted as a binary floating point constant (intermediate representation) with exponent = 0. Then control is transferred to FBLL.

<u>FBLL -- PU</u>

This is a secondary entry point of FBSL.

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the binary integer (precision (53,0)
REXP = (general register) contains the binary exponent
LIND = (general register) switch to indicate whether FBLL has been called by another conversion routine (≠0) or by FCON (=0)

Output parameters:
PIN = unchanged
CONS = result of the conversion
RLEN = 4 if short fraction, 8 if long fraction.

This routine converts binary float (intermediate) to float.

After having been shifted left by (REXP-(FLOOR (REXP/4))*4) the binary integer is interpreted as the fraction of an unnormalized floating- point constant with the hexadecimal exponent (64+FLOOR(REXP/4)).

To normalize the constant, the fraction is shifted left as many hexadecimal digit positions as necessary, and the exponent is reduced accordingly.

If the exponent is less than 0 or greater than X'7F', the constant is set to 0 or to the largest hexadecimal value (entry points FBLL30 or FBLL35, respectively), and error numbers 59 or 58, respectively, are put into the error table.

Finally, if LIND ≠ 0, control is transferred to the calling conversion routine. Otherwise, if the sign is minus, it is taken into account by setting the leftmost bit of the constant to 1.

Secondary entry points: FBLL10, FBLL30, FBLL35, FBLL25

<u>FSBI -- PV</u>

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the bit string left-aligned.

This routine converts from bit string to binary fixed. The constant is loaded into a pair of general registers and right-aligned by a double shift. The result is interpreted as a binary fixed point constant (precision (31,0)), and significant binary digits exceeding 31 are truncated. Control is then transferred to FBII.

<u>FBII -- PW</u>

This is a secondary entry point of FSBI.

Input parameters:
PIN = address of constant-table entry

CONS = (fullword) contains the binary fixed-point constant
RBFI = R5 - (general register) same as CONS
LIND = switch to indicate whether FBII has been called by another conversion routine (≠0) or by FCON (=0)

Output parameters:
PIN = unchanged
CONS = result of conversion
RLEN = 4

The constant is shifted as many binary digit positions as specified by the target scale factor (note that scale of source is always 0), left or right depending on whether the scale factor is positive or negative. If there are more significant digits than specified for the precision of the target, the constant is truncated to the left, and FERR is called.

If LIND ≠ 0, control is transferred to the calling conversion routine. Otherwise, the sign specified for the constant is taken into account, i.e., the constant is complemented if the sign is minus.

Secondary entry point: FBII03

FSDI -- PX

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the bit string left-aligned.

This routine converts from bit string to decimal fixed. The bit string is first converted to binary fixed (precision (31,0)), and control is transferred to FBID.

FBID -- PX

This is a secondary entry point of FSDI.

Input parameters:
PIN = address of constant-table entry
RBFI = (gen.reg.) binary fixed-point constant

This routine converts from binary fixed to decimal fixed. The binary fixed -point constant is converted to decimal (precision(15,0)), and control is transferred to FDDI.

FDDI -- PY

This is a secondary entry point of FSDI.

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the decimal fixed point constant.

Output parameters:
PIN = unchanged
CONS = result of conversion, left-aligned
RLEN = length of result in bytes

This routine converts packed decimal fixed to decimal fixed, packed or zoned or (zoned(T) format.

Second entry point: FDDIX

Input parameters: as above
R0 = scale factor of source

Third entry point: FDDI45

The constant is shifted as many decimal digit positions as the difference between the scale factor of target and the source. It is shifted left or right depending on whether the difference is positive or negative.

If there are more significant digits than specified for the precision of the target, FERR is called and the excess 2*CETL(P+2/2)-1 or P left digits are truncated on the left, depending on whether packed or zoned format is called for.

The constant is left-aligned in CONS and supplied with the appropriate sign bits as specified by the target.

If zoned or zoned (T) format is called for, the constant is unpacked and the sign replaced by the zone or left unchanged, respectively.

FSCS -- PZ

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the bit string left-aligned
PTAB = pointer of new constant table.

Output parameters:
PIN = unchanged
PTAB = points to the next entry.

This routine converts bit strings to character strings. The character string is built one character at a time in the new constant table. If the length of the target exceeds the length of the source, the character string is expanded with blanks.

FBLS -- QA

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the binary integer (precision (53,0))
REXP = (register) contains the binary exponent.

Binary float (intermediate) is converted to bit string. The routine calls FNBI which converts the binary float constant (intermediate representation) to binary fixed (precision = min(31,P(source)). Control is then transferred to FBIS.

FBIS -- QA

This is a secondary entry point of FBLS.

Input parameters:
PIN = address of constant-table entry
RBFI = (register) contains binary fixed-point constant

This routine converts binary fixed to bit string. The binary fixed-point constant is interpreted as a bit string of length P(source), i.e., RBFI is shifted left (32-P) bit positions. Then control is transferred to FBSS.

FBSS -- QA

This is a secondary entry point of FBLS.

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the left-aligned bit string (expanded on the right by 0's)

Output parameters:
PIN = unchanged
CONS = target string, left-aligned
RLEN = length of target in bytes = CEIL (length of target in bits/8)

This routine truncates or expands bit strings according to the new precision. string. string. If the target sign is negative, the string is inverted.

FNDI -- OB

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the binary integer
REXP = (register) contains the binary exponent

This routine converts binary float (intermediate) to decimal fixed. FBLL is called to convert the binary float number (intermediate form) to hexadecimal float. Then control is tranferred to FLDI.

FLDI -- OB

This is a secondary entry point of FNDI.

Input parameters:
CONS = (double word) hexadecimal floating point number

Output parameter:
CONS = decimal integer (precision (15,0))
R0=RSCF: scale factor

This routine converts hexadecimal float to decimal fixed. If the source is zero, FDDI is called to generate a decimal zero. If the excess 64 hexadecimal exponent is 78, the hexadecimal fraction of 14 digits may be interpreted as a binary integer (precision(56,0)). This binary integer is converted to decimal (precision(17,0)). If the result consists of more than 15 significant digits, the result is truncated and shifted right one or two decimal digit positions, and the scale factor (initialized with 0) is reduced accordingly. FDDIX is called to process the decimal fixed-point numbers.

If the source exponent is not 78, the hexadecimal fraction may be interpreted as a binary integer which is to be multiplied by 16**y (y = source exponent-78).

D = I0 * 16**y

this is equivalent to

```
D/10**X=I0*16**y/10**y=I1*16**0
if 16**y/10**X=1, i.e.,
   16**y=10**X, i.e.,
      x=y* ln16/ln10
```

This means: the source is to be divided by 10**x to get an integer result which may be converted to decimal. The target result is the decimal integer and the scale factor X.

To get the hexadecimal (binary) integer, the source is divided by 10**X1 or multiplied by 10**-X1, depending on whether X1 is positive or negative. X1 = min(78, FLOOR ((y*19728+8192) 16384) (19728/16384=ln 16/ln 10 ; 8192 = rounding), and the scale factor is increased by X1. This is repeated until the exponent of the result is 78.

FNBI -- QC

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the binary integer
REXP = (register) contains the binary exponent
LIND = (register) switch to indicate whether FNBI has been called by another conversion routine (≠0) or by FCON (=0)

Output parameters:
PIN = unchanged
CONS = result of conversion
RLEN = 4

This routine converts binary float (intermediate) to binary fixed. The binary integer is shifted as many binary digit positions as the sum of the exponent and the scale factor of the target. It is shifted left or right depending on whether the sum is positive or negative.

If there are more significant digits than specified for the precision of the target, FERR is called, and the number is truncated on the left.

If LIND ≠ 0, control is passed to the calling conversion routine; if LIND = 0, the sign specified for the target is taken into account, i.e., the constant is complemented if the sign is minus.

<u>FDIS -- QD</u>

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the decimal fixed-point constant.

This routine converts decimal fixed to bit string. The decimal fixed point constant is converted to binary fixed, precision min (31, CEIL((P-Q)/3,32)). FBIS is called to perform the conversion from binary fixed to bit string.

If P-Q = 0, a string of length 1, value 0 is generated in CONS. FBSS is called to process the bit string.

<u>FDIL -- QE</u>

Input parameters:
PIN = address of constant-table entry
CONS = (double word) contains the decimal fixed-point constant.

This routine converts decimal fixed to float. The input parameters are transformed into input parameters for FDLL (REXP = -scale factor). Control is transferred to FDLL.

<u>FDLL (FDFL) -- QE</u>

Input parameters:
PIN = address of constant-table entry
FINT = (= CONS(double word) - 1): decimal integer (prec. 17)
REXP = decimal exponent

Output parameter:
CONS = hexadecimal floating-point constant.

This routine converts decimal float (intermediate) to float. The decimal integer is converted to binary (precision 56,0), after having been left-aligned and REXP having been reduced accordingly. The binary integer may be interpreted as the fraction of a normalized floating point number with the excess 64 hexadecimal exponent of 78.

If the integer is zero, the result is set to a true floating-point zero. If the integer is not zero and if REXP is greater than 59, FBLL is called to set the constant to the highest floating-point number

The normalized hexadecimal number is multiplied by 10**X (X = REXP) if REXP is positive, or divided by 10**'X' if REXP is negative. If the result is zero, i.e., in the case of an exponent underflow (the interrupt having been masked off), FBLL30 is called to set the constant to zero. FBLL10 is then called to process the floating-point constant.

<u>FDLS -- QG</u>

Input parameters:
PIN = address of constant-table entry
FINT = (= CONS (double word) -1): decimal integer (precision 17)
REXP = decimal exponent

This routine converts decimal float to bit string. By means of FDLB the source is converted to binary integer (precision = min(31,CEIL(P(Source)*3,32)). FBIS is called to perform the conversion from binary fixed to bit string.

<u>FDLB -- QH</u>

Input parameters:
PIN = address of constant-table entry
FINT = (=CONS (double word)-1): decimal integer (precision 17)
REXP = decimal exponent

This routine converts decimal float to binary fixed. By means of FDLL the source is converted to hexadecimal float, then control is transferred to FLBI.

<u>FLBI -- QH</u>

This is a secondary entry point of FDLB.

Input parameter:
CONS = (double word) floating-point number

This routine converts float to binary fixed. The fraction of the floating point numer is interpreted as a binary integer which is to be divided by 16**X(X = 78-hexadecimal exponent). The fraction is shifted left by RSHI = (78-exp)*4-scale factor of target. FBII03 is called to process the binary fixed-point number.

If RSHI is negative or greater than 56, CONS is set to 0. When RSHI is negative, FERR is also called.

FDIB -- QI

Input parameters:
PIN = address of constant-table entry
CONS = (double word) decimal integer

This routine converts decimal fixed to binary fixed. If the scale factor of the target is ≠0, by means of FDIL, the source is converted to hexadecimal float. FLBI is called to perform the conversion from float to binary fixed.

If the conversion is to binary integer, the fractional digits of the source are truncated, and the resulting decimal integer is converted to binary.

If the resulting binary integer is greater than or equal to 2**31, FERR is called and CONS is zeroized. Otherwise, FBII is called to process the binary fixed-point number.

Routine FDLD -- QJ

Input parameters:
PIN = address of constant-table entry
FINT = (= CONS (double word) - 1); decimal integer (precision 17)
REXP = decimal exponent

This routine converts decimal float to decimal fixed. By means of FDLL, the source is converted to hexadecimal float. FLDI is called to perform the conversion from float to decimal fixed.

DIFL -- QL

Performs the simulation of a floating-point division.

Input parameters:
R1 = address of dividend
R2 = address of divisor

Output parameters:
CONS = result of division
R1   = unchanged

Note: 1. Dividend and divisor are assumed to be positive, normalized, long floating-point numbers.

2. The divisor is not 0.

3. An exponent underflow cannot occur.

This routine simulates floating-point division. The exponent of the result is obtained by subtracting the exponent of the divisor from that of the dividend and adding the difference to 64. To get the result fraction, the fractions of dividend and divisor are interpreted as two binary integers, which are divided by means of the Euclidean algorithm, i.e., the subtraction method. Before this is done, the integer belonging to the divisor is multiplied by 16, and the exponent is increased by 1 if the dividend fraction is greater than the divisor fraction. If the dividend fraction is 0, the result is a true 0.

MUFL -- QL

Performs the simulation of a floating-point multiplication

Input parameters:
R1   = address of multiplicand
R2   = address of multiplier

Output parameters:
CONS = result of multiplication
R1   = unchanged

Note: 1. Both operands are assumed to be positive, normalized, long floating-point numbers.

2. An exponent-overflow or underflow cannot occur.

The sum of the exponents of the two operands -64 form the exponent of the intermediate result. To get the fraction of the intermediate result, the fractions of both operands are interpreted as two binary integers (precision 56) that are to be multiplied. To do this, both integers are split into two parts (precision of A2 and B2 = 31; precision of A1 and B1 = 25):

```
A*B= (A1*2**31+A2)*(B1*2**31+B2)
    = A1*B1*2**62+A1*B2*2**31
             + A2*B1*2**31
```

The result (precision 112) is derived from shifting the bits and adding the 4 results obtained in the 4 multiplications (A2 * B2, A2 * B1, A1 * B2, and A1 * B1), as shown in Figure 1.

The result has a maximum of 56 digits and is truncated on the right.

To normalize the intermediate result, the fraction is shifted left as many hexadecimal digit positions as necessary, and the exponent is reduced accordingly.

If one of the two operands is 0, the result is a true 0.

MUDI -- QK

This routine performs the floating-point multiplication and division. If no floating-point feature is available, the simulation routines MUFL or DIFL are called, respectively.

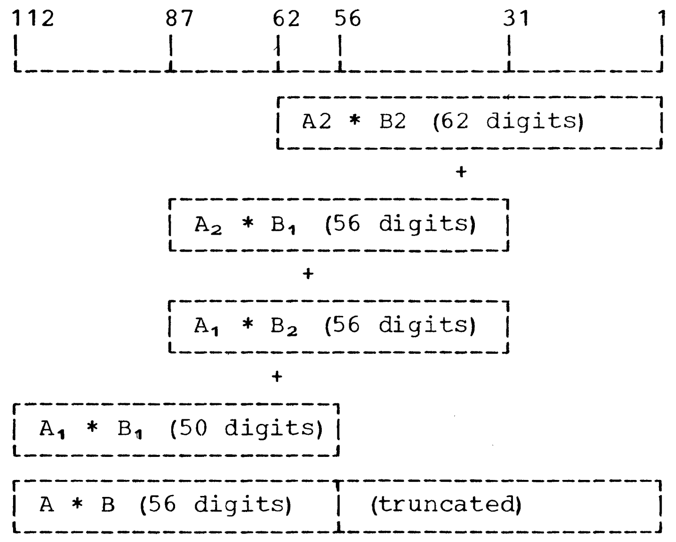

112
87
62
56
31
1
A2 * B2 (62 digits)
+
A_2 * B_1 (56 digits)
+
A_1 * B_2 (56 digits)
+
A_1 * B_1 (50 digits)
A * B (56 digits)
(truncated)

Figure 1. Sample Floating-point Multiplication

PHASE PL/ID75 (GENERATION OF I/O MACROS I) -- QP

This phase processes the first group of I/O statements, GET, PUT, and FORMAT statements. GET and PUT statements are translated into several macros, each of which will result in library calls and corresponding parameters. Both macros and parameters are generated in accordance with the elements and options of the statement processed. Also, macros and parameters that effect the insertion of values into the calling sequence at object time are produced. For FORMAT statements, only parameters (format strings) are generated, one for each label of the FORMAT statement. All other parts of the text, even if embedded in the statements processed, are skipped and remain unchanged. Error and/or warning indications are generated on variable counter overflow, incorrect data, or incorrect format item.

Phase Input and Output

The input for this phase is obtained from TXTIN. The input text consists of:

1. Program text that is already translated into the macro language.

2. I/O statements other than GET, PUT, and FORMAT as delivered by the compiler phases C50 - C65.

3. GET, PUT, and FORMAT statements as delivered by the compiler phases C50 - C65.

4. End-of-program key.

The output for items 1, 2, and 4 above appears in unchanged form in the output text. The output for statements of item 3 are transformed as described above.

Communication with Other Phases

Characterizations of statements, options, data list elements, end-of-statements belonging to I/O statements, and file parameters are obtained from phases C50 - C65. These are necessary for the sequential processing of this phase. The presence of errors is indicated in the communication region for use by the diagnostic phase. For each library routine occurring, a bit is set in the library bitstring in the communication region for use by phase D80.

Phase Description -- QP

The input text string is scanned by the search routine. All elements belonging to the groups described under items 1 and 2 of the section Phase Input and Output are written out unchanged into the output text. When an element of the group described under item 3 of the section Phase Input and Output is found, the search routine branches to the appropriate processing routine. After a GET, PUT, or FORMAT statement has been processed, control is transferred back to the search routine. After the end-of-program key has been found, the corresponding library bits of the communication region are set. The end-of-program key is edited and phase D80 is called.

SEARCH -- QR

After the error indication is cleared and the buffers (using the input routine) are filled, elements of the input string are tested successively. Only elements with EA, EB, E0, and F-keys are expected to be found. All elements, except the end-of-program keys, GET, PUT, and FORMAT statement keys, are skipped with their appropriate length and are written using the skip routine. When a not-skippable key is found, an appropriate exit out of this routine is performed.

Subroutines -- QS

STR (Step A4). If RA (register 1) points to the name of a variable in the table space, when this subroutine is entered, the name of the major structure, the pointer, or the base will be found if the variable is an element of a structure, or is controlled or defined. All possible combinations are allowed. If the variable is an element of a structure, also the offset and for variables that are controlled or defined, the attribute byte of the pointer (with special flag) or of the base is inserted.

SKIP (Step A2). In this routine, the input pointer is updated by the number contained in register RG. The contents of the input buffers passed by the input pointer are written using the output routine. Before the end of the buffer area is reached, the buffers are always filled, the input pointer is repositioned accordingly (input routine), and the procedure is continued.

COUNT (Step F3). This routine increases the variable counter by one and tests it for overflow. If an overflow has occurred, an error indication is stored in an error byte or a switch.

SETIN1, SETIN2 -- QT

This routine has two entries. When entry 1 is used, the input pointer (Register INPO) is moved to the next syntactical unit belonging to the statement itself and all embedded syntactical units with the keys F0, F2, F3, F6, EB, and also end-of-statements (key EA) are skipped if the latter are not flagged as belonging to the I/O statement processed. After each skipped unit, the input buffers are filled again if required (input routine). Using entry 2, the input pointer is first moved three bytes ahead before performing the function as described above.

GETPUT -- QU-QY

When a GET or a PUT statement is found, the search routine branches to this routine. First, the flag byte (2nd byte of the statement identifier) which indicates that the options are saved, and the statement attribute table are moved into the table space using the move routine. Macros and constants are generated for calling the initialization library routine. These will be different for file or string options and for GET or PUT statements and EDIT or LIST options. If, for a file option, a file parameter is present which is indicated by the first byte of the file name, a dummy name is inserted and the constant is labeled. Thus, provisions are made for inserting the actual file name at object time by code prepared in the phases C50 - C65. For the string option, the string variable is examined and passed to the library routine in a similar manner as for data list elements (as described below).

After the macros and constants for the initialization have been written, the flag byte is examined for the various options if file option has been found. When options of a PUT statement are found, macros and constants are generated for the necessary library calls and eventually for object code to insert values at object time. The format list, if present, is translated into a format string and written as one or more constants using the build-format-string routine. The data list is processed next; macros and constants for library calls are generated corresponding to one library call for each data element. For this purpose, the characteristic of the current data element is saved. This indicates whether it is scalar or array and whether it is the first element of a group (special item).

For each data element, a DED (data element descriptor) is constructed for library use. If more adjacent data elements have the same DED, it is constructed and loaded only once. The construction of the DED and the name part of the macro belonging to a given data element is different for character string constant, declared variable, and generated variable elements.

For declared variables, the name and characteristics of the variable are found in the variable table stored in the table space. If it is found to be an element of a structure, controlled and/or defined, the name part of the macro is modified accordingly using the structure subroutine. If the declared variable is an array, the macro is modified to produce a suitable loop. For a generated variable that is non-integer binary fixed, macros and constants are also generated for the call of a binary fixed/binary float conversion library routine.

If more format and data lists follow in the statement, the above processing of these lists is repeated as often as required. The scanning of the statement is performed by the set input pointer subroutine, which skips and edits in unchanged form all embedded elements not belonging to the statement itself.

When a variable counter overflow, incorrect data, or incorrect format items are found, these indications are stored and suitable warning and/or error codes are produced in the output after reaching the end-of-statement key. The error test is made for counter overflow by the count routine, for format items by the build format string routine.

Note: Auxiliary routines for generating initialization macros, parameters, and move macros are: GENIM, GENPAR, and GENMO. These routines belong to GETPUT, and their function is described with that routine.

BFSTR -- QZ

The elements of the format list are successively translated into elements of a format constant which is built in the input buffer. When a remote format item is reached, the partial format constant built until this point is written out and a construction of a new constant is started because of the difference in constant types (hexadecimal versus address constants). If the remote format item has a label variable rather than a label constant, provision is made for inserting the actual value at object time in a similar manner as for file parameters (see description above).

FORMAT -- RB

When a FORMAT statement is found, the search routine branches to this routine which causes the format list to be translated into a format string using the build format string subroutine. This string is written as a constant (once for each label of the statement) with the current label as the name.

INPUT -- RC

This routine causes the input buffers to be filled using the IJKAGI external routine depending on the current position of the input pointer (INPO). After control is transferred from the routine, at least three of the four input buffers are filled. The input pointer is adjusted.

OUTPUT, OPWSP -- RD

Output of text is accomplished via this routine. When the standard entry is used, register RC must be loaded with the address and register RD with the length of the output area. An additional entry (OPWSP) serves for automatic loading of register RC with the address of the work space. The routine uses one output buffer, whose pointer (register OUPO) is adjusted in course of the output function. For physical output, the IJKAPO external routine is used.

MOVE -- RE

This routine is used by the input and output routines and for moving the pre-statement into the table space. Moving of the contents of a source area into a target area is performed (these areas must not overlap). Address and length of the source area are to be loaded into registers RC and RA, respectively; the address of the target area into register RB.

OPEN, CLOSE, DISPLAY, and record-oriented I/O statements of the input text string are selected and replaced by macros that generate the required library calls and parameters belonging to these calls. Both macros and parameters are generated in accordance to the elements and options of the statement processed. Macros and parameters that effect the insertion of values into the calling sequence at object time are also produced. All other parts of the text, even if embedded in the statements processed, are skipped and remain unchanged. After having scanned the input text, the library bit string in the communication region is completed, address constants are generated from it and edited for library use. Error indication is generated when a counter overflow is detected.

Phase Input and Output

The input for this phase is obtained from TXTIN. The input text consists of:

1. Program text already translated into the macro language.

2. OPEN, CLOSE, DISPLAY, and record-oriented I/O statements as delivered by the phases C50 - C65.
3. End-of-program key.

The output for the program text and the end-of-program key (items 1 and 3 above) appears in unchanged form in the output text. The output for statements listed under item 2 above is transformed as described in the statement processing routine.

The text output is followed by the library address constants as described in the end-of-program processing routine.

Communication with Other Phases

Characterizations of statements, options, file attributes, end-of-statements belonging to I/O statements and file parameters are obtained from the phases C50 - C65. These are necessary for the sequential processing of this phase. The presence of an error is indicated in the communication region for the use of the diagnostic phase.

Bits set by previous phases and this phase in the library bit string in the communication region are used to generate address constants that cause object-time loading of the required library routines.

Phase Description -- RF

The input text string is scanned by the search routine. All elements belonging to the group described under item 1 of the section Phase Input and Output are written out unchanged into the output text. When an element of the group described under item 2 of the section Phase Input and Output is found, control is transferred to the statement processing routine.

After a statement has been processed, control is transferred back to the search routine. When an end-of-program key is found, control is transferred to the EOP processing routine. When control returns from this routine, phase E25 or E50 is called. Which one of these phases is called depends on whether or not an error has been indicated in the communication region.

SEARCH -- RF

After the buffers (using the input routine) are filled, elements of the input string are tested successively. Only elements with EA, EB, E0, and F-keys are expected to be found. All elements, except end-of-program keys of statements to be processed in this phase, are skipped with their appropriate length and are edited using the skip routine. When a not-skippable key is found, an appropriate exit out of this routine is made.

Statement Processing Routine -- RG-RK

When a statement to be processed by this phase is found, the search routine branches to the statement processing routine. First, the flagbyte (2nd byte of the statement identifier) which describes the statement and the options is saved and the statement attribute table is moved into the table space by means of the MOVE routine. For OPEN-CLOSE and record-oriented statements. the first part of the library parameter is built. For this purpose, an address constant for the file name is constructed. If a file parameter rather than a file name is present, which is indicated by the first byte of the file-name symbol, a dummy name is inserted and the constant is labeled. This way, provision is made for inserting the actual file name at object time by code prepared by the phases C50 - C65. The first byte (flag byte) of the library parameter is constructed in accordance with the stored information of statement type and options.

For record-oriented statements, the length of the record variable and various address constants belonging to the SET, FROM, INTO, KEY, and KEYFROM options are built into the library parameter. These options may appear in any order in the original statement, but the corresponding address constants and also the length of the record variable have a predetermined place in the library parameter.

For a declared variable in the options, the name and characteristics of the variable are found in the variable table stored in the table space. If it is found to be an element of a structure, controlled and/or defined, the address constant is modified accordingly using the structure subroutine. If the variable is found to have the storage class dynamic, a load-variable macro is produced which serves for object-time insertion of the actual address. Finally, an initialization macro is generated that will produce the appropriate library call. The key for the library routine is obtained from the file declarations stored from the statement flag byte, (i.e., second byte of the statement identifier).

For an OPEN or CLOSE statement, the library parameter consists of a series of combinations of a flag byte, (i.e., characterizing the statement, the options and whether it is the last element of the series), and adjacent-file-address constant. Each element of the series corresponds to an options group of the statement. When a file parameter or a PAGESIZE option is found, the series is completed and a new series is started. For each series, one single initialization macro is generated (different variant for OPEN and CLOSE) which itself will produce the appropriate library call. For a PAGESIZE option, a separate library parameter with a move macro to generate code for object-time insertion of the actual value, and an initialization macro for the library call are produced immediately after the library call relating to the corresponding file option.

The DISPLAY statement is processed in a separate section of this phase. First, a DED (data element descriptor), load-DED-macro, load-transmit macro, and load-scalar macro for the given expression are generated. These are required for the object-time call of the appropriate library routine. Building of these elements is different for character-string constants, declared or generated variables in much the same way as for the data-list elements of stream-oriented data-transmission described in phase D75. If a REPLY option is present, the same procedure is repeated for the REPLY variable with the library routine code changed accordingly.

When a variable counter overflow during the statement processing is found, the count routine sets a switch which causes the output of an error indication with the end-of-statement. For each library routine, a bit is set in the library bit string in the communication region for the use by the end-of-program processing routine in this phase. The scanning of the statement is performed by the set INPO subroutine which skips and writes out in unchanged form all embedded elements not belonging to the statement itself.

EOP -- RK, RL

When an end-of-program key is found, the search routine branches to the end-of-program processing routine. First, the library bit string in the communication region is updated. Thereafter, it is scanned bit by bit from the end to the beginning, and for each bit set, an address constant with the same name that corresponds to a library routine is generated. For the bits corresponding to the conversion routines (numbers 40-55), special address constants with a single common name are generated. Having done this, additional bits in the library bit string are set that correspond to primary entry points of routines whose secondary entry points are already incorporated. Finally, the rest of the output text in the output buffer is edited.

Note: Descriptions of the following subroutines used or called for in this phase can be found in phase D75:

STR	COUNT
SKIP	MOVE
SETIN1	INPUT
SETIN2	OUTPUT
OPWSP	

This phase is used only if one or more errors are detected in the preceding phases of the compiler. It collects and sorts the errors detected by the preceding phases, and prints these errors in a standard format. Three kinds of errors are distinguished:

1. Errors causing an interruption of the compilation;

2. Errors causing the deletion of the execution but allowing a continuation of the compilation;

3. Errors allowing a continuation of the compilation and the execution of the compiled program.

If errors causing the termination of the compilation are detected, the diagnostic phase is the last phase of the compilation.

Format of the Error Codes

Errors detected in a statement are inserted behind the statement. The end-of-statement keys, which introduce a sequence of error codes, are recognized in particular bit positions in the error-indicator byte. The format of the error codes behind the end-of-statement key is shown below.

EOS	Error Indicator	Lev.No.	B1.No.

Statement No.	ERROR	E.No.	ERROR	E.No.

Error indicator (1 byte) contains information on whether or not an error is present and information about the severity of the error.

Bit 8 = 1 : the sequence of errors contains at least one error causing the termination of the compilation.

Bit 9 = 1 : the sequence of errors contains at least one error not causing a close of the compilation, but causing the deletion of the execution. Bit 10 = 1, the sequence of errors contains at least one error not causing a close of the compilation and not causing the deletion of the execution of the compiled program. (Bit 11 to 16, free for information in the preceding phases).

Statement No. The number of the statement as described in preceding phases.

ERROR. A fixed key (EB) indicating an error. The number of EB's is equal to the number of errors in a statement.

E.NO. The number which corresponds to an error comment.

Format of the Error List

During the first diagnostic phase an error list is printed out. The list has the format shown in Figure 1.

Logical Flow

The algorithm of this phase is separated into the following parts:

1. Storage allocation for the phase.
2. Scan of the text string.
3. Storing of error comments.
4. Sorting of error comments.
5. Printing of the error list.

The error messages are printed by ascending statement-numbers. For printing the error list, the error comments must be present in storage. Since only limited storage capacity is available, only a part of all error comments can be stored. This part consists of the error comments that are needed first in order to begin printing error messages in sequence of the statement numbers.

Suppose the maximum number of error comments that can be in storage at one time is NX. It must then be determined which NX different error numbers appear first in the text string. In the second part of the phase (scan of the text string), the number of different errors N is counted.

If the scan is interrupted (i.e., when the end of the text string is reached or when N is equal to NX), the comments referring to the detected error numbers must be loaded into core storage. Before the needed error comments are loaded into core storage, all errors comments must be stored on SYS001 (storing of error comments). The needed error comments are then stored from SYS001 (sorting of error comments).

If the detected errors are printed (printing of error list), the interrupted second part of the program (scan of text string) is continued until the next NX different errors or the end of source text are detected.

```
DIAGNOSTIC MESSAGES

5E error no (1) Ib statement no (1) b error comment (1)
5E error no (2) Ib statement no (1) b error comment (2)
5E error no (3) Ib statement no (1) b error comment (3)
 •       •   •   •              •    •
 •       •   •   •              •    •
 •       •   •   •              •    •
5E error no   (i) Ib statement no  (1) b error comment   (1)
5E error no (i+1) Ib statement no  (2) b error comment (i+1)
5E error no (i+2) Ib statement no  (2) b error comment (i+2)
 •       •     •   •               •    •
 •       •     •   •               •    •
 •       •     •   •               •    •
5E error no   (m) Ib statement no  (k) b error comment    (m)

1. Words written in lower case letters in the actual list are replaced by their
   actual values.

2. The letter 'b' stands for blanks.

3. The 'error no' consists of three decimal digits.

4. The 'statement no' consits of four decimal digits.

5. The 'error comment' may consist of a maximum of 61 characters.
```

Figure 1. Format of PL/I Diagnostic Messages

DESCRIPTION OF ROUTINES

Symbols Used in Flow Charts

TSP	: Length of table space
LLENO	: Length of list LENO
LCOM	: Length of comment
PHSP	: Free space in this phase
NO	: Number of error comments that can be stored at one time
NX	: Maximum number of error comments that can be stored at one time
ERRS	: Bit in the communication region
STNO	: Statement number
ENO	: Error number
N	: Counted number of errors
A0	: Number of error comments in one phase overlay
C0	: Start address of comment storage area
B0	: Start address buffer 1
B1	: Start address buffer 2
B2	: End address buffer 2

Storage Allocation --SB

The number of different error comments that can be in storage at one time depends on the available storage. The comments may be stored in the phase storage (the available storage is then fixed and equal to 4K reduced by the program space), or they may be stored in the table storage (the availbale storage will then be the table storage reduced by the table storage reduced by space for LENO). Which storage is used for the comments must be determined in phase E25 itself, because the table storage is of variable length.

Text Scan and Error Counting -- SC, SD

Input for phase E25 may consist of:

1. End-of-statement keys with or without errors.

2. Macro instructions.

3. Statement identifiers with prefixes.

4. Declared and generated variables.

5. Constants.

The error codes, identified by their key, are searched for in the text string. The error numbers of all detected errors together with the statement numbers are entered in a list LSTNO. The errors are ordered by sequence of the statement numbers which is equal to the sequence of their occurrence. Each detected error is noted again in a second table LENO which is arranged by sequence of the error numbers; each element of the table refers to one error number.

The number of different errors N is counted. The scan of the text string is interrupted if N is equal to NX. The scan is terminated if the end-of-source-text key is detected.

Storing of Error Comments -- SE

Before phase G25 is called, the error comments are stored on SYSRES as a part of the phase (or phase overlay). The comments are called one at a time and in consecutive order.

In order to get the possibility of multiple use of the same error comment, the comments must be stored in another place. If only a small number of different errors is detected, the corresponding comments are stored in the table storage. If the number of different errors is greater than NX, all comments are stored on SYS001. This part of the phase is passed only once.

Sorting of Error Comments -- SF

This part is used only if the error comments are stored on SYS001 and not in the table storage. (Storing of comments in the table storage is compounded with sorting.) The required error comments (referring to the detected error numbers) are selected from SYS001 and stored in the phase storage. The detected error numbers are entered in the list LENO. After moving an error comment, the element of the list LENO is replaced by the new address of the comment.

Printing of the Error List -- SH

The error messages must be printed in the same sequence as the list LSTNO. The error number and the statement number given in this list must be converted from binary to decimal representation and inserted in the error message. The address of the corresponding error comment is given in the list LENO by the element referring to the error number. The error comment must also be inserted in the message.

After printing one message, the next element of the list LSTNO is taken. The interrupted scanning of the text string is continued if the messages for all detected errors have been printed and the end-of-source-text key has not been detected.

End of Text String -- SG

This part is called when the end-of-source-text key is detected. Control is passed to part 4 (sorting of error comments) unless the end of the source text occurs before NX errors are counted in the text string. If there are less than NX errors, this routine sorts the error comments from the phase overlays of phase E25 and stores them simultaneously in the table space.

HERH -- SJ, SK

The routine HERH is used for skipping within the input text with respect to the buffer boundaries. The buffers are filled with new text if a buffer boundary is passed.

LOAD

This is a supervisor macro used in this phase.

GENERAL DESCRIPTION OF THE GENERATOR PHASES (PL/IE50, PL/IE60, PL/IE61)

The objective of the generator phases is the generation of object code. Before the generator phase, the algorithm to be represented by this code is given by macros. The definition of the different macros is such that either each macro is associated with a fixed set of code or the selection of the needed code is possible only by means of the operands of the macro.

The input text for the generator phases contains macros and other information which is not used in the generator phases.

The code to be inserted for a macro is partially prepared in the model-instruction dictionary. Because the macros consist only of an identification and operands, additional information about the macros (not in the macro instruction) is given in the model-instruction dictionary. The information in the model-instruction dictionary is either a subroutine in machine language or a set of predefined instructions for frequent operations. The predefined instructions are internally defined; they are decoded by the generator phases.

The generated code consists of machine instructions and pseudo instructions for communication to the assembler. Except for the format of the operands, the machine instructions referred to are the IBM System/360 Assembler Language machine instructions.

Because it is not possible to store the complete model-instruction dictionary in the phase, multiple passes over the text string are necessary to generate the code. For each pass another part of the model-instruction dictionary is stored, and the macros referring to this part can be processed.

Format of the Input Text

The input text string consists of the following elements (the number of bytes of a part of the element is given in parentheses over the boxes; the number of bits is given under the boxes):

1. Statement identifiers with prefixes

```
 (1)
┌────┬─────────────────────────┐
│E0  │                         │
└────┴─────────────────────────┘
```

E0 = statement-identifier key. The statement identifier is 6 bytes long.

2. Declared variables

```
 (1)   (2)
┌────┬─────────┬────────────── ──────┐
│F4  │   lc    │                     │
└────┴─────────┴────────────── ──────┘
```

F4 = key for attribute table
lc = length of attribute table

3. Constants

```
 (1)   (2)
┌────┬─────────┬────────────── ──────┐
│F3  │   le    │                     │
└────┴─────────┴────────────── ──────┘
```

F3 = key for constant table
le = length of constant table

4. Generated variables

```
 (1)   (2)
┌────┬──────────────────────── ──────┐
│F0  │   lf                          │
└────┴──────────────────────── ──────┘
```

F0 = key for attribute table
lf = length of attribute table

5. End-of-statement

```
  (1)   (2)         (1)   (1)
┌───┬─────────┬───┬────┬────── ──────────┐
│EA │5 bytes  │EB │ 0  │ EB     0  EB   0│
└───┴─────────┴───┴────┴────── ──────────┘
```

EA = key for end-of-statement
EB = key for error
 0 = error number

6. Assembler code

```
    (1)      (2)
┌─────┬─────────┬──────   ──────┐
│ F6  │   lI    │               │
└─────┴─────────┴──────   ──────┘
```

F6 = key for assembler code = X'F6'
lI = length of code

7. END of program

1 byte, hexadecimal FF

8. Macros

```
 (1)  (2)  (1)   (3)    (2)   (1)

┌───┬───┬───┬──────┬─────┬─────┐─────
│F2 │lm │ C │OP(1) │M(1) │B(1) │--->
└───┴───┴───┴──────┴─────┴─────┘─────
```

```
      (3)     (2)    (1)
   ---T------T-----T-----┐
      |OP(n) |M(n) |B(n) |
   ---┴------┴-----┴-----┘
```

F2	= macro key X'F2'
lm	= length of the macro
C	= identification of the macro
OP(1)	= operand (1)
M(1)	= modifier for operand (1)
B(1)	= byte 5 from SYMTAB entry referring to operand (1)

The maximum length of the macros is 200 bytes. The format of the operands may differ from the format shown above.

MODEL-INSTRUCTION DICTIONARY

The model-instruction dictionary contains the prepared code and information about the macro which is not given in the macro instruction. Each macro refers to the particular part of the model-instruction dictionary (referred to as the macro definitions). A macro definition, or parts of macro definitions, may be common to more than one macro.

Macro Definition

The code generated from a macro instruction depends on the identification (operation code) and the parameters (operands) of the macro. For a macro definition, the generated code depends only on the information contained in operands of the macro instruction. If a macro definition is common to more than one macro instruction, the identification is treated as operand in order to determine certain instructions which must be altered to generate the code for a particular macro.

A macro definiton contains a macro definition header and one or more model-instruction sets.

The relation between the operands and the code generated is given in the first part of the macro, the macro definition header. The macro definition header indicates the model-instruction set to be used. A model-instruction set consists of code instructions and information which gives the location where the several operands of the macro must be inserted into the code.

Macro Definition Header

The macro definition header contains the information required for selecting the needed model-instruction sets. The information may be in machine language or may consist of special instuctions. Since many conditions and operations used for the selection are unique for most of the macros, these conditions and operations may be represented by special instructions. These special instructions are interpreted by the generator phases.

Conditions

The conditions represented by special instructions are:

1. Compare

```
┌--T--T-----T-----┐
|C |K | P1  | P2  |
└--┴--┴-----┴-----┘
0  4  8    16    24
```

K	= Key for compare = 0
C	= Condition code
P1,P2	= Indicate locations in the operand list of the macro.

Byte (P1) of the operand list of the macro is logically compared with byte (P2). If the result is in accordance with the condition code C, the condition is accepted.

2. Compare Immediate

```
┌--T--T-----T-----┐
|C |K |  I  |  P  |
└--┴--┴-----┴-----┘
0  4  8    16    24
```

K	= Key for compare immediate = 1
C	= Condition code
P	= Indicates a location in the operand-list of the macro
I	= Immediate data

Byte (P) of the operand list of the macro is logically compared with I. If the result is in accordance with the condition code C, the condition is accepted.

3. Test Under Mask

```
┌--T--T-----T-----┐
|C |K |  I  |  P  |
└--┴--┴-----┴-----┘
0  4  8    16    24
```

K	= Key for test under mask = 3
C	= Condition code
P	= Indicates a location in the operand list of the macro
I	= Immediate data.

Byte (P) of the operand list is tested under the mask I. If the result is in accordance with the condition code C, the condition is accepted.

The Result Word

The work done on a sequence of conditions is given in a result word W as follows:

1. The initial value for the result word is zero (W = 0).

2. The work done on a condition modifies the result word. If the condition is satisfied, the result word is multiplied by two and then increased by one.

   W = 2 * W + 1

   If the condition is not satisfied, the result word is multiplied by two.

   W = 2 * W

Operations

There are additional special instructions to indicate operations referring to the operand list of the macro or macro definition header:

1. Macro subroutine

```
┌────┬──────────┐
| K  |    A     |     K = Key 01
└────┴──────────┘     A = Address
0    8          24
```

   The treatment of the macro definition header is continued in a subroutine given by the address A.

2. Set pseudo operand

```
┌────┬────┬────┐
| K  |I   | P  |
└────┴────┴────┘
0    8    16   24
```

   K = Key 02
   I = Immediate data
   P = Indicates a location in the operand list of the macro.

   The immediate data I is stored as a pseudo operand in byte (P) of the operand list.

3. End-of-Condition Sequence

```
┌─────┬─────┬─────┬──────   ───┬─────┐
| K   | S   |A(1) |A(2)        |A(n) |
└─────┴─────┴─────┴──────   ───┴─────┘
0     8     16    24
```

   K = Key 03
   S = Address in macro definition header
   A(1) = Address of model-instruction set (1).

   The work done on the condition sequence is interrupted. The treatment of the macro is continued in a model-instruction set.

   The subscript of the model-instruction set is given by the determined result word. The address of the model-instruction set has the same subscript. If A(1) equals 0, no model-instruction set is taken. After inserting the determined model-instruction set, the treatment of the macro is continued in the macro definition header at the address S. S = 0 indicates the end of the macro definition header.

   The number of addresses A in the instruction must be equal to the maximum value possible for the result word.

4. Take saved result word (1 byte: X'04')

   The result word W is taken from the value saved at an earlier time.

5. End-of-Condition sequence.

   Save result word (1 byte: X'05')

   The work done on the condition sequence is interrupted. The result word determined in the condition sequence is saved.

6. Conditional Branch

```
┌────┬────┬────┬────┐
| 06 | WX | A  | S  |
└────┴────┴────┴────┘
0    8    16   24   32
```

   WX = Value to be compared with the result word
   A = Address of a model-instruction set
   S = Address in macro definition header.

   a) W = WX:

      The work done on the condition sequence is interrupted. The treatment of the macro is continued in the model-instruction set at the address A. If A equals 0, no model-instruction set is taken. After inserting the model-instruction set, the treatment of the macro is continued in the macro definition header at the address S. S = 0 indicates the end of the macro defitition header.

   b) W ≠ WX:

      No action is performed.

7. Unconditional Branch

```
┌────┬────┐
│ K  │ S  │
└────┴────┘
0    8    16
```

K = Key 07
S = Address in macro definition header. The treatment of the macro is continued in the macro definition header at address S.

Model-Instruction Sets

Model-instruction sets contain prepared code together with information on where to insert the several operands or pseudo operands of the macro. The information on the operands has the format:

```
┌────┬────┬────┐
│ P  │ M  │ LM │
└────┴────┴────┘
0    8    16   24
```

P = Indicates a location in the operand list of the macro.
LM = Indicates the location in the model-instruction set where the operand has to be inserted.
M = Gives a modification or length specification of the operand

M=K, K≤5: the operand has a length of K bytes.
M=6: the operand has a length of two bytes and the absolute value of the operand is taken.
M=7: the operand has a length of one byte. The operand is decreased by 1 before it is inserted.
M=19: the operand has a length of three bytes and must be inserted with indirect addressing.
M=21: the operand has a length of five bytes and must be inserted with indirect addressing.

Some values of P have the following special meaning:

1.
```
 (1)  (1)  (1)
┌────┬────┬────┐
│ 0  │ I  │LM  │
└────┴────┴────┘
```

The immediate data I is inserted at location LM.

2.
```
 (1)  (1)
┌────┬────┐
│ 1  │DLM │
└────┴────┘
```

The location counter of the model-instruction set is increased by DLM.

3.
```
  (1)    (1)    (1)    (1)
┌─────┬──────┬──────┬──────┐
│ 224 │OP.C. │ P1   │ P2   │
└─────┴──────┴──────┴──────┘
```

The RR instruction given by the operation code 'OP.C.' has to be formed. The registers are given in P1 and P2.

If 208 ≤ Pi < 224 (Pi=P1 and/or P2), the immediate data Pi-208 is inserted.

4.
```
  (1)    (1)    (1)    (1)    (1)
┌─────┬──────┬──────┬──────┬──────┐
│ 225 │OP.C. │ P1   │ P2   │ P3   │
└─────┴──────┴──────┴──────┴──────┘
```

The RX, RS, or SI instruction given by the operation code 'OP.C.' has to be formed. The operands are given in P1, P2, and P3. A test is performed on P3 to determine whether or not it must be indirectly addressed.

If 208 ≤Pi <224 (Pi =P1 and/or P2), the immediate data Pi-208 is inserted.

5.
```
  (1)    (1)    (1)    (1)    (1)    (1)
┌─────┬──────┬──────┬──────┬──────┬──────┐
│ 226 │OP.C  │ P1   │ P2   │ P3   │ P4   │
└─────┴──────┴──────┴──────┴──────┴──────┘
```

The SS instruction given by the operation code 'OP.C.' has to be constructed. The operands are given in P1, P2, P3, and P4.

The operands referring to P3 and P4 are tested to determine whether or not they must be indirectly addressed.

If 208 ≤Pi <224 (Pi = P1 or P2), the immediate data Pi-208 is inserted.

Behind the information with P ≤226, the code is given. The code has the format:

6.
```
  (1)    (2)
┌─────┬───────────┬──────┬───   ───┬──────┐
│  L  │ 11        │IN(1) │          │IN(n) │
└─────┴───────────┴──────┴───   ───┴──────┘
```

L = Key for assembler code F6
11 = length of the code
IN(1) = Instruction (1)

The format of the instruction is described on the following pages.

Not only the operands of the instructions but all parts of the model-instruction set may be changed by inserting operands or pseudo operands of the macro.

FORMAT OF THE INSTRUCTIONS

The code (the output of the generator phases) consists of machine instructions and pseudo instructions for communication with the Assembler.

Machine Instructions

Except for the format of the operands, machine instructions refer to the IBM System/360 Assembler Language machine instructions.

There are five basic machine formats:

1. RR format

(1)	(1)	(1)	(1)
88	Op.C	R1	R2

The first byte contains the key for the machine instructions, the second byte contains the operation code, and the following two bytes contain the operands.

2. RX format

(1)	(1)	(1)	(1)	(3)	(2)
88	Op.C	R1	X2	N	M

R1	=	General register containing first operand
X2	=	General register referring to second operand
N	=	Name of second operand
M	=	Modifier for second operand.

3. RS format

(1)	(1)	(1)	(1)	(3)	(2)
88	Op.C	R1	R3	N	M

R1 and R3 are general registers

N and M specify the second operand as in the RX format.

4. SI format

(1)	(1)	(1)	(1)	(3)	(2)
88	Op.C.	X'00'	12	N	M

12 is an immediate operand; N and M specify the second operand as in the RX format.

5. SS format

(1)	(1)	(1)	(1)	(3)	(2)	(3)	(2)
88	Op.C.	L1	L2	N1	M1	N2	M2

L, N and M give the lengths, names, and modifiers of the operands, respectively. The L1 field contains zeros if only one length is present.

Pseudo Instructions

1. CNOP

(1)	(1)	(1)	(1)
80	C0	b	W

The CNOP instruction allows alignment of an instruction at a specific boundary without breaking the instruction flow, should any bytes be skipped for alignment. C0 is the code for the instruction CNOP.

Operand b specifies to which byte in a word or double-word the location counter is to be set. Operand W specifies whether the byte b is in a word or a double-word.

2. DC AL3

(1)	(1)	(3)	(2)
80	C1	N	M

N = Name
M = Modifier

3. DC X

(1)	(1)	(2)	
80	C2	L	W

W - hexadecimal constant
L - Length of W

4. DS

(1)	(1)	(2)
80	C3	L

The DS instruction is used to reserve storage areas. L is the length of the storage to be reserved. The DS model instruction has a meaning different from the IBM System/360 Assembler instruction DS. The model instruction does not align on boundaries.

5. LABEL

(1)	(1)	(2)
80	C4	NAME

The LABEL instruction allows the setting of a label in the program.

6. BEGIN

(1)	(1)	(1)	(1)	(2)
80	C5	Bl.	L	N

The BEGIN instruction marks the beginning of a procedure or block.

Bl.	- Block number
L	- Level number
N	- Name of procedure or label referring to a begin block.

7. END (Procedure or Block)

(1)	(1)	(2)
80	C6	not used

8. DC 'length of block'

(1)	(1)	(1)	(1)
80	C7	Bl.	L.

This DC instruction is used to indicate that the length of the block (4 bytes) must be inserted into the text string.

Bl.	= Block number
L.	= Level number

9. OPT

(1)	(1)	(2)
80	C8	LABEL

The OPT instruction is used in connection with optimizable branch instructions (see description of phase G00). The assembler replaces the OPT instruction with a machine instruction. The LABEL given by the OPT instruction refers to the branch address in the following branch instruction.

10. DC A(STATIC)

(1)	(1)	(2)
80	C9	not used

This DC instruction is used to indicate that the start address of the static storage (4 bytes) must be inserted into the text string.

11. INDIVISIBLE CODE (L)

(1)	(1)	(1)	(1)
80	CA	L1	L2

The instruction is used to indicate that the following code cannot be divided by additional instructions. At object time the length of the code is L2 bytes. The length of the assembler code is L1 bytes.

12. USED REGISTER (R)

(1)	(1)	(2)
80	CB	R

The instruction is used to inform the Assembler which registers are used for indirect addressing.

R = 5	- register 5 is used in addition to the register used until now.
R = 6	- register 6 is used in addition to the register used until now.
R = 0	- no register is used; registers 5 and 6 are free.

Treatment of the Macros

Code cannot be generated from the macros in one pass over the text string, due to the size of the model-instruction dictionary. Therefore, the model-instruction dictionary is divided into smaller parts. Each part of the model-instruction dictionary corresponds to one pass over the text string.

Two phases are used for the generation of code from the macros. In the first generator phase, one pass is made over the text string. All further passes are made in the second generator phase.

THE MACROS AND THE GENERATED CODE

In the following, the different macros are described in detail. The description of each macro consists of the following items:

Format of the Macro. OP1, OP2... OP(I) are operands of the standard format described above. Differing operands have different names. The meaning of the operands with the use of the macros in special cases are not described. This section deals only with the meaning of the operands that determine the generated code.

The Generated Code. The generated code is shown for all possible cases. Indirect addressing is not considered. This is described under Treatment of Indirect Addressing in the description of phase E60.

The code shown also includes Assembler instructions (e.g., USED REGISTER) described under Pseudo Instructions. In some cases, the code generated for a macro contains submacros (e.g., *SHIFT*), which are described elsewhere in this section.

Fixed Binary Addition

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)
F2	0012	00	OP1	OP2	s-q

2. Generated code

	OP1=REGISTER	OP1=STORAGE	
OP2 =REG.	AR R1,R2	L 4,S1 AR 4,R2 ST 4,S1	*
OP2 =STOR.	A R1,S2	L 4,S1 A 4,S2 ST 4,S1	
OP2 =REG.	SLA R1,s-q AR R1,R2	L 4,S1 SLA 4,s-q AR 4,R2 ST 4,S1	**
OP2 =STOR.	SLA R1,s-q A R1,S2	L 4,S1 SLA 4,s-q A 4,S2 ST 4,S1	
OP2 =REG.	SLA R2,q-s AR R1,R2	L 4,S1 SLA R2,q-s AR 4,R2 ST 4,S1	***
OP2 =STOR.	L 4,S2 SLA 4,q-s AR R1,4	L 4,S2 SLA 4,q-s A 4,S1 ST 4,S1	

* s-q = 0
** s-q > 0
*** s-q < 0

OP1 and OP2 may be indirectly addressed.

Fixed Binary Subtraction

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)
F2	0012	01	OP1	OP2	s-q

2. Generated code

	OP1=REGISTER	OP1=STORAGE	
OP2 =REG.	SR R1,R2	L 4,S1 SR 4,R2 ST 4,S1	*
OP2 =STOR.	S R1,S2	L 4,S1 S 4,S2 ST 4,S1	
OP2 =REG.	SLA R1,s-q SR R1,R2	L 4,S1 SLA 4,s-q SR 4,R2 ST 4,S1	**
OP2 =STOR.	SLA R1,s-q S R1,S2	L 4,S1 SLA 4,s-q S 4,S2 ST 4,S1	
OP2 =REG.	SLA R2,q-s SR R1,R2	L 4,S1 SLA R2,q-s SR 4,R2 ST 4,S1	***
OP2 =STOR.	L 4,S2 SLA 4,q-s AR R1,4	L 4,S2 SLA 4,q-s S 4,S1 LCR 4,4 ST 4,S1	

* s-q = 0
** s-q > 0
*** s-q < 0

OP1 and OP2 may be indirectly addressed.

Fixed Binary Multiplication with Overflow Check

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	02	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LR 5,R2 USED REGISTER(0) MR 4,R1 SLDA 4,32 USED REGISTER(0) LR R1,4	L 5,S1 USED REGISTER(0) MR 4,R2 SLDA 4,32 USED REGISTER(0) ST 4,S1
OP2= STOR.	L 5,S2 USED REGISTER(0) MR 4,R1 SLDA 4,32 USED REGISTER(0) LR R1,4	L 5,S2 USED REGISTER(0) M 4,S1 SLDA 4,32 USED REGISTER(0) ST 4,S1

OP1 and OP2 may be indirectly addressed.

Fixed Binary Division

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)
F2	0012	03	OP1	OP2	P+1

2. Generated code

	OP1 = REGISTER	OP1 = STORAGE
OP2= REG.	SR 5,5 USED REGISTER(5) LR 4,R1 SRDA 4,P+1 USED REGISTER(0) DR 4,R2 USED REGISTER(0) LR R1,5	SR 5,5 USED REGISTER(5) L 4,S1 SRDA 4,P+1 USED REGISTER(0) DR 4,R2 USED REGISTER(0) ST 5,S1
OP2= STOR.	SR 5,5 USED REGISTER(5) LR 4,R1 SRDA 4,P+1 USED REGISTER(0) D 4, S2 USED REGISTER(0) LR R1,5	SR 5,5 USED REGISTER(5) L 4,S1 SRDA 4,P+1 USED REGISTER(0) D 4,S2 USED REGISTER(0) ST 5,S1

OP1 and OP2 may be indirectly addressed.

FIXED BINARY NEGATION

1. Format of the macro

(1)	(2)	(1)	(6)
F2	000A	04	OP1

2. Generated code

OP1 = REGISTER	OP1 = STORAGE
LCR R1, R1	L 4,S1 LCR 4,4 ST 4,S1

OP1 may be indirectly addressed.

Fixed Binary Assignment with Overflow Check

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)	(2)	(2)
F2	0016	05	OP1	OP2	LABEL	X	Y

2. Generated code

a. Y ≥ 0, Y ≥ X

	OP1=REGISTER	OP1=STORAGE
OP2= REG	LR R1,R2 SLA R1,Y SRA R1,X	SLA R2,Y SRA R2,X ST R2,S1
OP2= STOR	L R1,S2 SLA R1,Y SRA R1,X	L 5,S2 USED REGISTER(0) SLA 5,Y SRA 5,X ST 5,S1

All shift instructions are deleted if the number to be shifted is 0.

OP1 and OP2 may be indirectly addressed.

b. Y ≥ 0, Y < X

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LR 4,R2 LPR R1,4 SLA R1,Y SRA R1,X LTR 4,4 BC 10,LABEL LCR R1,R1 LABEL:	LPR 4,R2 SLA 4,Y SRA 4,X LTR R2,R2 BC 10,LABEL LCR 4,4 LABEL: ST 4,S1
OP2= STOR.	L 4,S2 LPR R1,4 SLA R1,Y SRA R1,X LTR 4,4 BC 10,LABEL LCR R1,R1 LABEL:	L 4,S2 LPR 5,4 SLA 5,Y SRA 5,X LTR 4,4 BC 10,LABEL LCR 5,5 LABEL: ST 5,S1

c. Y < 0

	OP1=REGISTER	OP1=STORAGE
OP2= =REG.	LR 4,R2 LPR R1,4 SRA R1,X-Y LTR 4,4 BC 10,LABEL LCR R1,R1 LABEL:	LPR 4,R2 SRA 4,X-Y LTR R2,R2 BC 10,LABEL LCR 4,4 LABEL: ST 4,S1
OP2= STOR.	L 4,S2 LPR R1,4 SRA R1,X-Y LTR 4,4 BC 10,LABEL LCR R1,R1 LABEL:	L 4,S2 LPR 5,4 SRA 5,X-Y LTR 4,4 BC 10,LABEL LCR 5,5 LABEL: ST 5,S1

Fixed Binary Assignment without Overflow Check

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)	(2)
F2	0014	06	OP1	OP2	LABEL	X

2. Generated code

a. X < 0

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LR R1,R2 SLA R1,-X	SLA R2,-X ST R2,S1
OP2= STOR.	L R1,S2 SLA R1,-X	L 5,S2 SLA 5,-X ST 5,S1

b. X = 0

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LR R1,R2	ST R2,S1
OP2= STOR.	L R1,S2	L 5,S2 ST 5,S1

c. X > 0

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LR 4,R2 LPR R1,4 SRA R1,X LTR 4,4 BC 10,LABEL LCR R1,R1 LABEL:	LPR 4,R2 SRA 4,X LTR R2,R2 BC 10,LABEL LCR 4,4 LABEL: ST 4,S1
OP2= STOR	L 4,S2 LPR R1,4 SRA R1,X LTR 4,4 BC 10,LABEL LCR R1,R1 LABEL:	L 4,S2 LPR 5,4 SRA 5,X LTR 4,4 BC 10,LABEL LCR 5,5 LABEL: ST 5,S1

OP1 and OP2 may be indirectly addressed.

Fixed Binary Comparison

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)
F2	0012	08	OP1	OP2	s-q

2. Generated code

	OP1=REGISTER	OP1=STORAGE	
OP2 REG.	CR R1,R2	L 4,S1 CR 4,R2	*
OP2 STOR	C R1,S2	L 4,S1 C 4,S2	
OP2 REG.	SLA R1,s-q CR R1,R2	L 4,S1 SLA 4,s-q CR 4,R2	**
OP2 STOR.	SLA R1,s-q C R1,S2	L 4,S1 SLA 4,s-q C 4,S2	
OP2 REG.	SLA R2,q-s CR R1,R2	L 4,S1 LSA R2,q-s CR 4,R2	***
OP2 STOR	L 4,S2 SLA 4,q-s CR R1,4	L 4,S2 SLA 4,q-s L 5,S1 CR 4,5	
*	s-q = 0		
**	s-q > 0		
***	s-q < 0		

Fixed Binary Exponentiation

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)
F2	0016	07	X	TARGET	N

2. Generated code

```
LA      1,X
LA      3,N
LA      4,TARGET
L       15,N'95'
BALR    14,15
```

X and TARGET may be indirectly addressed.

Fixed Binary Multiplication without Overflow Check

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	09	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	MR R1,R2	L 5,S1 USED REGISTER (0) MR 4,R2 USED REGISTER (0) ST 5,S1
OP2= STOR.	M R1,S2	L 5,S1 USED REGISTER (0) M 4,S2 ST 5,S1

OP1 and OP2 may be indirectly addressed.

<u>SIGN, Fixed Binary</u>

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)
F2	0012	0A	OP1	OP2	LABEL

2. Generated code

	OP1 = REGISTER	OP1 = STORAGE
OP2= REG.	LTR R1,R2 OPT LABEL BC 8,LABEL LA R1,1 OPT LABEL BC 2,LABEL LCR R1,R1 LABEL	LTR 5,R2 OPT LABEL BC 8,LABEL LA 5,1 OPT LABEL BC 2,LABEL LCR 5,5 LABEL ST 5,S1
OP2= STOR	L R1,S2 LTR R1,R1 OPT LABEL BC 8,LABEL LA R1,1 OPT LABEL BC 2,LABEL LCR R1,R1 LABEL	L 5,S2 LTR 5,5 OPT LABEL BC 8,LABEL LA 5,1 OPT LABEL BC 2,LABEL LCR 5,5 LABEL ST 5,S1

OP1 and OP2 may be indirectly addressed.

<u>ABS, Fixed Binary</u>

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	0B	OP1	OP2

2. Generated code

	OP1 = REGISTER	OP1 = STORAGE
OP2= REG	LPR R2,R1	LPR R2,R2 ST R2,S1
OP2 STOR.	L R1,S2 LPR R1,R1	L 5,S2 USED REGISTER (0) LPR 5,5 ST 5,S1

OP1 and OP2 may be indirectly addressed.

<u>Fixed Decimal Addition</u>

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)	(1)	(1)	(1)	(1)	(1)	(1)
F2	001C	10	OP1	OP2	OP3	L2	L3	A	B	S	L

2. Generated code

```
SHIFT   GW0(16),S2(L2),A *
SHIFT   GW0+16(16),S3(L3),B **
AP      GW0(16),GW0+16(16)  **
```

* if A=0, OP2,L2,A is changed to OP3,L3,B.
** if B=0, the instructions are replaced by AP GW0(16),S3(L3).

if S = X'30':

```
ZAP     S1(L),GW0(16)   else:
MVC     S1(L),GW0+16-L
```

OP1, OP2, and OP3 may be indirectly addressed. SHIFT is a submacro described below.

<u>SHIFT X(LX), Y(LY), Z</u>

1. Sequence of the operands if the submacro is called:

(6)	(6)	(1)	(1)	(1)
X	Y	LX	LY	Z

2. Generated code

T = TRUN((Z-1)/2)

a. Z = 0:

```
        ZAP     X(LX),Y(LY)
```

b. Z < 0 and odd and |T|<LY:

```
MVO     X(LX),Y(LY+T)
MVN     X+LX-1(1),Y+LY-1
```

c. Z < 0 and even and |T|<LY:

```
MVO     X(LX),Y(LY+T)
MVN     X+LX-1(1),Y+LY-1
MVO     X(LX),X(LX-1)
```

d. Z > 0 and odd and T<LX:

```
MVO     X(LX-T),Y(LY)
XC      X+LX-T-1(T+1),X+LX-T-1
MVN     X+LX-1(1),Y+LY-1
```

e. Z > 0 and even and T<LX:

```
MVO     X(LX-T),Y(LY)
XC      X+LX-T-1(T+1),X+LX-T-1
MVN     X+LX-1(1),Y+LY-1
MVO     X(LX),X(LX-1)
```

f. (z < 0 and |T|>LY) or (Z > 0 and T>LX)

```
ZAP     X(LX),=0(1)
```

Fixed Decimal Subtraction

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)	(1)	(1)	(1)	(1)	(1)	(1)
F2	001C	11	OP1	OP2	OP3	L2	L3	A	B	S	L

2. Generated code

```
SHIFT   GWO(16),S2(L2),A
SHIFT   GWO+16(16),S3(L3) B *
SP      GWO(16),GWO+16(16)  *
```

if S = X'30'

```
ZAP     S1(L),GWO(16)
```

all other cases:

```
MVC     S1(L),GWO+16-L
```

P01, P02, and OP3 may be indirectly addressed.

The submacro SHIFT is described after FIXED DECIMAL ADDITION.

* if B=0, the instructions will be replaced bytSP GWO(16),S3(L3)..

Fixed Decimal Multiplication

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)	(1)	(1)	(1)	(1)
F2	001B	12	OP1	OP2	OP3	L2	L3	L	S

2. Generated code

```
ZAP     GWO(16),S2(L2)
MP      GWO(16),S3(L3)
```

if S=X'30'

```
ZAP     S1(L),GWO(16)
```

all other cases:

```
MVC     S1(L),GWO+16-L
```

OP1, OP2, and OP3 may be indirectly addressed.

The submacro SHIFT is described after DECIMAL FIXED ADDITION.

Fixed Decimal Division

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)	(1)	(1)	(1)
F2	0019	13	OP1	OP2	OP3	L2	AL3	A

2. Generated code

```
SHIFT   GWO(L3+8),S2(L2),A
DP      GWO(L3+8),S3(3)
MVC     S1(8),GWO
```

OP1, OP2, and OP3 may be indirectly addressed.

The submacro SHIFT is described after DECIMAL FIXED ADDITION.

Fixed Decimal Negation, 1 Operand

1. Format of the macro

(1)	(2)	(1)	(6)	(1)
F2	000B	14	OP1	L

2. Generated code

```
XI      S1+L-1,X'01'
```

OP1 may be indirectly addressed.

Fixed Decimal Assignment

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)	(1)	(1)	(1)
F2	0015	15	OP1	OP2	L1	L2	A	-	S

2. Generated code

```
a. S ≠ 4 :SHIFT   S1(L1),S2(L2),A

b. S=4,A<0:
             SHIFT   GW0(16),S2(L2),A
             ZAP     S1(L1),GW0(16)

c. S=4,A>0: A is odd:
             SHIFT   GW0(16),S2(L2),A
             ZAP     S1(L1),GW0(16)

d. S=4,A>0: A is even:
             SHIFT   GW0(16),S2(L2),A+1
             ZAP     GW0+16-L1(L1),GW0(16)
             MVO     S1(L1),GW0+16-L1(L1-1)
             MVN     1+L1-1(1),GW0+15
```

OP1 and OP2 may be indirectly addressed.

Fixed Decimal Negation, 2 Operands

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)
F2	0012	16	OP1	OP2	L1	L2

2. Generated code

```
ZAP     S1(L1),S2(L2)
XI      S1+L1-1,X'01'
```

OP1 and OP2 may be indirectly addressed.

Fixed Decimal Exponentiation

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)	(2)	(2)
F2	001C	17	X	TRG	N	DED X	DED TRG

2. Generated code

```
LA      1,X
LA      3,N
LA      4,TARGET
USED    REGISTER(5)
LA      2,DED X
LA      5,DED TARGET
L       15,N'94'
BALR    14,15
```
X and TARGET may be indirectly addressed.

Fixed Decimal Comparison

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)	(1)	(1)	(1)	(1)	(1)	(1)
F2	001C	18	-	OP2	OP3	L2	L3	A	B	S	C

2. Generated code

```
SHIFT   GW0(16),S2(L2),A
SHIFT   GW0+16(16),S3(L3),B
CP      GW0(16),GW0+16(16)
```

If A=B, the following code is generated:

```
CP      S2(L2),S3(L3)
```

OP2 and OP3 may be indirectly addressed.

SIGN, Fixed Decimal

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)
F2	0012	1A	OP1	OP2	LABEL

The modifier of OP2 must be increased by L-1(L= length of OP1) if the macro is used for the SIGN function.

2. Generated code

OP1=REGISTER	OP1=STORAGE
SR R1,R1	SR 5,5
ZAP S2(1),S2(1)	ZAP S2(1),S2(1)
OPT LABEL	OPT LABEL
BC 8,LABEL	BC 8,LABEL
LA R1,1	LA 5,1
OPT LABEL	OPT LABEL
BC 2,LABEL	BC 2,LABEL
LCR R1,R1	LCR 5,5
LABEL	LABEL
	ST 5,S1

OP1 and OP2 may be indirectly addressed.

ABS, Fixed Decimal

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)
F2	0012	1B	OP1	OP2	L

L is the length of the operands.

2. Generated code

```
ZAP     OP1(L),OP2(L)
NI      OP1+L-1,X'FE'
```

OP1 and OP2 may be indirectly addressed.

Short Float Addition

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	20	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	AER R1,R2	STD 0,GW0 LE 0,S1 AER 0,R2* ST 0,S1 LD 0,GW0
OP2= STOR.	AE R1,S2	STD 0,GW0 LE 0,S1 AE 0,S2 STE 0,S1 LD 0,GW0

*R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Short Float Subtraction

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	21	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	SER R1,R2	STD 0,GW0 LE 0,S1 SER 0,R2* STE 0,S1 LD 0,GW0
OP2= STOR.	SE R1,S2	STD 0,GW0 LE 0,S1 SE 0,S2 STE 0,S1 LD 0,GW0

*R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Short Float Multiplication

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	22	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	MER R1,R2	STD 0,GW0 LE 0,S1 MER 0,R2* ST 0,S1 LD 0,GW0
OP2= STOR.	ME R1,S2	STD 0,GW0 LE 0,S1 ME 0,S2 STE 0,S1 LD 0,GW0

*R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Short Float Division

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	23	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	DER R1,R2	STD 0,GW0 LE 0,S1 DER 0,R2 STE 0,S1 LD 0,GW0
OP2= STOR.	DE R1,S2	STD 0,GW0 LE 0,S1 DE 0,S2 STE 0,S1 LD 0,GW0

* R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Short Float Negation, 2 Operands

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	24	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LCER R1,R2	STE R2,S1 XI S1,X'80'
OP2= STOR.	MVC GW0(4),S2 XI GW0,X'80' LE R1,GW0	MVC S1(4),S2 XI S1,X'80'

OP1 and OP2 may be indirectly addressed.

Short Float Negation, 1 Operand

1. Format of the macro

(1)	(2)	(1)	(6)
F2	000A	25	OP1

2. Generated code

OP1=REGISTER	OP1=STORAGE
LCER R1,R1	XI S1,X'80'

OP1 may be indirectly addressed.

Short Float Assignment

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	26	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LER R1,R2	STE R2,S1
OP2= STOR.	LE R1,S2	MVC S1(4),S2

OP1 and OP2 may be indirectly addressed.

Short Float Exponentiation (Integer)

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)
F2	0016	27	X	TARGET	1N

2. Generated code

```
LA      1,X
LA      2,N
LA      3,TARGET
L       15,N'92'
BALR    14,15
```

X and TARGET may be indirectly addressed.

Short Float Comparison

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	28	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	CER R1,R2	STD 0,GW0 LE 0,S1 CER 0,R2* LD 0,GW0
OP2= STOR.	CE R1	STD 0,GW0 LE 0,S1 CE 0,S2 LD 0,GW0

*R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Float General Exponentiation

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)
F2	0016	29	X	TARGET	Y

2. Generated code

```
LA      1,Y
LA      3,TARGET
LA      2,X
L       15,N'96'
BALR    14,15
```

SIGN, Float

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)
F2	0012	2A	OP1	OP2	LABEL

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	SR R1,R1 LTER R2,R2 OPT LABEL BC 8,LABEL LA R1,1 OPT LABEL BC 2,LABEL LCR R1,R1 LABEL	SR 5,5 LTER R2,R2 OPT LABEL BC 8,LABEL LA 5,1 OPT LABEL BC 2,LABEL LCR 5,5 LABEL ST 5,S1
OP2= STOR.	L R1,S2 LTR R1,R1 OPT LABEL BC 8,LABEL LA R1,1 OPT LABEL BC 2,LABEL LCR R1,R1 LABEL	L 5,S2 LTR 5,5 OPT LABEL BC 8,LABEL LA 5,1 OPT LABEL BC 2,LABEL LCR 5,5 LABEL ST 5,S1

OP1 and OP2 may be indirectly addressed.

ABS, Short Float

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	2B	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LPER R1,R2	STE R2,S1 NI S1,X'7F'
OP2= STOR.	LE R1,OP2 LPER R1,R1	MVC S1(4),S2 NI S1,X'7F'

OP1 and OP2 may be indirectly addressed.

Long Float Addition

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	30	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	ADR R1,R2	STD 0,GW0 LD 0,S1 ADR 0,R2* STD 0,S1 LD 0,GW0
OP2= STOR.	AD R1,S2	STD 0,GW0 LD 0,S1 AD 0,S2 STD 0,S1 LD 0,GW0

*R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Long Float Subtraction

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	31	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	SDR R1,R2	STD 0,GW0 LD 0,S1 SDR 0,R2* STD 0,S1 LD 0,GW0
OP2= STOR.	SD R1,S2	STD 0,GW0 LD 0,S1 SD 0,S2 STD 0,S1 LD 0,GW0

*R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Long Float Multiplication

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	32	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	MDR R1,R2	STD 0,GW0 LD 0,S1 MDR 0,R2* STD 0,S1 LD 0,GW0
OP2= STOR.	MD R1,S2	STD 0,GW0 LD 0,S1 MD 0,S2 STD 0,S1 LD 0,GW0

*R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Long Float Division

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	33	OP1	OP2

2. Generated code

	OP1=REGISTER	OP=STORAGE
OP2= REG.	DDR R1,R2	STD 0,GW0 LD 0,S1 DDR 0,R2* STD 0,S1 LD 0,GW0
OP2= STOR.	DD R1,S2	STD 0,GW0 LD 0,S1 DD 0,S2 STD 0,S1 LD 0,GW0

*R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Long Float Negation, 2 Operands

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	34	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LCDR R1,R2	STD R2,S1 XI S1,X'80'
OP2= STOR.	MVC GW0(8),S2 XI GW0,X'80' LD R1,GW0	MVC S1(8),S2 XI S1,X'80'

OP1 and OP2 may be indirectly addressed.

Long Float Negation, 1 Operand

1. Format of the macro

(1)	(2)	(1)	(6)
F2	000A	35	OP1

2. Generated code

OP1=REGISTER	OP1=STORAGE
LCDR R1,R1	XI S1,X'80'

OP1 may be indirectly addressed.

Long Float Assignment

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	36	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LDR R1,R2	STD R2,S1
OP2= STOR.	LD R1,S2	MVC S1(8),S2

OP1 and OP2 may be indirectly addressed.

Long Float Exponentiation (Integer)

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)
F2	0016	37	X	TARGET	N

2. Generated code

```
LA      1,X
LA      2,N
LA      3,TARGET
L       15,N'93'
BALR    14,15
```

X and TARGET may be indirectly addressed.

Float Comparison

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	38	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	CDR R1,R2	STD 0,GW0 LD 0,S1 CDR 0,R2* LD 0,GW0
OP2= STOR.	CD R1,S2	STD 0,GW0 LD 0,S1 CD 0,S2 LD 0,GW0

*R2 must not be 0.

OP1 and OP2 may be indirectly addressed.

Long Float General Exponentiation

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)
F2	0016	39	X	TARGET	Y

2. Generated code

```
LA      1,Y
LA      3,TARGET
LA      2,X
L       15,N'97'
BALR    14,15
```

ABS, Long Float

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	3B	OP1	OP2

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LPDR R1,R2	STD R2,S1 NI S1,X'7F'
OP2= STOR.	LD R1,OP2 LPDR R1,R1	MVC S1(8),S2 NI S1,X'7F'

OP1 and OP2 may be indirectly addressed.

Character String Concatenation

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)	(1)	(1)
F2	0018	40	OP1	OP2	OP3	L1	L2

2. Generated code

```
MVC      S1(L1),S2
MVC      S1+L1(L2),S3
```

OP2 and OP3 may be indirectly addressed.

CONVERSION CV5

1. Format of the macro

(1)	(2)	(1)	(1)	(1)	(6)	(6)	(2)
F2	0014	42	I	RN	OP1	OP2	D

2. Generated code

```
                   LA     1,S2
                   LA     2,S1
if I=0:            LA     3,D
if I=1:            LA     3,N'3'+D
                   L      15,N'RN'
                   BALR   14,15
```

OP1 and OP2 may be indirectly addressed.

CONVERSION CV4

1. Format of the macro

(1)	(2)	(1)	(1)	(1)	(6)	(6)	(2)	(2)
F2	0016	41	I	RN	OP1	OP2	D1	D2

2. Generated code

```
                        LA     1,S2
if I = 0 or 1:          LA     2,D2
if I = 2 or 3:          LA     2,N'3'+D2
                        LA     3,S1
if I = 0 or 2:          LA     4,D1
if I = 1 or 3:          LA     4,N'3'+D1
                        L      15,N'RN'
                        BALR   14,15.
```

OP1 and OP2 may be indirectly addressed.

Short Float to Long Float Assignment

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	46	OP1	OP2

2. Generated code

	OP1 = REGISTER	OP1 = STORAGE
OP2= REG.	SDR R1,R1 LER R1,R2	XC S1(8),S1 STE R2,S1
OP2= STOR.	SDR R1,R1 LE R1,S2	XC S1(8),S1 MVC S1(4),S2

OP1 and OP2 may be indirectly addressed.

Decimal Fixed to Binary Integer Conversion CV30

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)
F2	0012	50	OP1	OP2	L	Z

2. Generated code

```
SHIFT   GW0(8),S2(L),Z
CVB     R1,GW0
```

The submacro SHIFT is described after FIXED DECIMAL ADDITION.

OP2 may be indirectly addressed.

Decimal Fixed to Zoned Decimal (T) Conversion, CV31

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)
F2	0012	51	OP1	OP2	L1	L2

2. Generated code

```
UNPK    S1(L1),S2(L2)
```

OP1 and OP2 may be indirectly addressed.

Decimal Fixed to Zoned Decimal Conversion, CV32

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)
F2	0012	52	OP1	OP2	L1	L2

2. Generated code

```
UNPK    S1(L1),S2(L2)
OI      S1+L1-1,X'F0'
```

OP1 and OP2 may be indirectly addressed.

Binary Integer to Binary Float Conversion, CV33

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)
F2	0012	53	OP1	OP2	LABEL

2. Generated code

```
                SR     4,4
                LPR    5,R2
                STM    4,5,GW0
                MVI    GW0,X'4E'
                LTR    R2,R2
                BC     2,LABEL
                MVI    GW0,X'CE'
        LABEL:  LD     R1,GW0
                AD     R1,=(0)
```

Binary Integer to Bit String Conversion, CV34

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)
F2	0011	54	OP1	OP2	Z

2. Generated code

	OP1=REGISTER	OP1=STORAGE
OP2= REG.	LPR R2,R2 LR R1,R2 SLL R1,Z	LPR R2,R2 SLL R2,Z ST R2,S1
OP2= STOR.	L R1,S2 LPR R1,R1 SLL R1,Z	L 5,S2 LPRR R5,R5 SLL 5,Z ST 5,S1

OP1 and OP2 may be indirectly addressed.

Zoned Decimal (T) to Decimal Fixed Conversion, CV35

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)
F2	0012	55	OP1	OP2	L1	L2

2. Generated code

```
PACK    S1(L1),S2(L2)
```

OP1 and OP2 may be indirectly addressed.

Bit String to Binary Integer Conversion, CV36

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)
F2	0011	56	OP1	OP2	L

2. Generated code

```
MVC     GW0(4),S2
L       R1,GW0
SRL     R1,MIN(32-L,1)
```

OP2 may be indirectly addressed.

Binary Integer to Decimal Fixed, CV37

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)
F2	0011	57	OP1	OP2	L

2. Generated code

```
CVD     R2,S1
MVC     S1(L),S1+8-L
```

OP1 may be indirectly addressed.

Character String Comparison

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)	(6)
F2	0012 0018	58	OP1	OP2	L1	L2	OP3

If L1=L2, the length of the macro is 18 bytes instead of 24 bytes.

2. Generated code

```
a.  L1=L2:  CLC    S1(L1),S2

b.  L1<L2:  MVI    S3,X'40'
            MVC    S3+1(L2-1),S3
            MVC    S3(L1),S1
            CLC    S3(L2),S2

c.  L1>L2:  MVI    S3,X'40'
            MVC    S3+1(L1-1),S3
            MVC    S3(L2),S2
            CLC    S1(L1),S3
```

OP1 and OP2 may be indirectly addressed.

Shift Right Arithmetic Single

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	59	OP1	OP2

2. Generated code

OP1=REGISTER	OP1=STORAGE
SRA R1,S2	L 4,S1 SRA 4,S2 ST 4,S1

OP1 may be indirectly addressed.

Shift Left Arithmetic Single

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	5A	OP1	OP2

2. Generated code

OP1 = REGISTER	OP1 = STORAGE
SLA R1,S2	L 4,S1 SLA 4,S2 ST 4,S1

OP1 may be indirectly addressed.

Shift Right Arithmetic Double

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	5B	OP1	OP2

2. Generated Code

OP1=REGISTER	OP1=STORAGE
SRDA R1,S2	LM 4,5,S1 SRDA 4,S2 STM 4,5,S1

OP1 may be indirectly addressed.

Shift Left Arithmetic Double

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	5C	OP1	OP2

2.0 Generated Code

OP1=REGISTER	OP1=STORAGE
SLDA R1,S2	LM 4,5,S1 SLDA 4,S2 STM 4,5,S1

OP1 may be indirectly addressed.

Bit String NOT, 2 Operands

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)	(1)
F2	0013	63	OP1	OP2	L		M

2. Generated Code

```
XC      S1(L),S2
NI      S1+L-1,M
```

OP1 and OP2 may be indirectly addressed.

Bit String Assignment

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)	(1)
F2	0013	65	OP1	OP2	L1	L2	M

2. Generated Code

```
a.  L1>L2: XC    S1(L1),S1
           MVC   S1(L2),S2

b.  L1≤L2: MVC   S1(L1),S2
           NI    S1+L1-1,M
```

OP1 and OP2 may be indirectly addressed.

Bit String NOT, 1 Operand

1. Format of the macro

(1)	(2)	(1)	(6)	(1)	(1)
F2	000D	64	OP1	L	M

2. Generated code

```
XC      S1(L),N'2'
NI      S1+L-1,M
```

OP1 may be indirectly addressed.

Bit String AND

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)
F2	0012	66	OP1	OP2	L1	L2

2. Generated code

```
NC      S1(L2),S2 and if L2<L1
XC      S1+L2(L1-L2),S1+L2
```

OP1 and OP2 may be indirectly addressed.

Bit String OR

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)
F2	0012	67	OP1	OP2		L2

2. Generated code

```
OC      S1(L2),S2
```

OP1 and OP2 may be indirectly addressed.

Bit String Comparison

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)
F2	0012	68	OP1	OP2	L1	L2

2. Generated code

```
a.  If L1=L2:  CLC   S1(L1),S2

b.  If L1<L2:  XC    GW0+80(16),GW0+80
               MVC   GW0+80L1),S1
               CLC   GW0+80L2),S2

c.  If L1>L2:  XC    GW0+80(16),GW0+80
               MVC   GW0+80(L2),S2
               CLC   S1(L1),GW0+80
```

OP1 and OP2 may be indirectly addressed.

Shift Right Logical Single

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	69	OP1	OP2

2. Generated code

OP1=REGISTER	OP1=STORAGE
SRL R1,S2	L 4,S1 SRL 4,S2 ST 4,S1

OP1 may be indirectly addressed.

Shift Left Logical Single

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	6A	OP1	OP2

2. Generated Code

OP1 = REGISTER	OP1 = STORAGE
SLL R1,S2	L 4,S1 SLL 4,S2 ST 4,S1

OP1 may be indirectly addressed.

Shift Right Logical Double

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	6B	OP1	OP2

2. Generated Code

OP1 = REGISTER	OP1 = STORAGE
SRDL R1,S2	LM 4,5,S1 SRDL 4,S2 STM 4,5,S1

OP1 may be indirectly addressed.

Shift Left Logical Double

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	6C	OP1	OP2

2. Generated code

OP1 = REGISTER	OP1 = STORAGE
SLDL R1,S2	LM 4,5,S1 SLDL 4,S2 STM 4,5,S1

OP1 may be indirectly addressed.

Branch on Condition

1. Format of the macro

(1)	(2)	(1)	(1)	(6)
F2	000B	70	C	OP1

2. Generated Code

```
OPT     OP1
BC      C,OP1
```

Return to Label Constant

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)	(1)	(2)
F2	0019	71	OP1	OP2	OP3	E5	REG

2. Generated Code

```
OP3:    DC label  (OP2)
        MVC       S1(8),OP3
        MVC       S1+4(2),72(REG)
        LA        1,S1
        L         15,N'13'
        BR        15
```

Define Label Constant

1. Format of the macro

(1)	(2)	(1)	(2)
F2	0007	72	LABEL

2. Generated Code

```
LABEL:
```

Assign Label Constant

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(6)	(1)	(2)
F2	0019	75	OP1	OP2	OP3	E5	REG

2. Generated Code

```
OP3:    DC label  (OP2)
        MVC       S1(8),OP3
        MVC       S1+4(2),72(REG)
```

Pointer Assignment

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	76	OP1	OP2

2. Generated Code

a. OP2 is a pointer (bit 6 of byte 15 is set on)

```
MVC    S1+1(3),S2+1
```

b. OP2 is not a pointer

```
LA     0,S2
ST     0,GW0+80
MVC    S1+1(3),GW0+81
```

IF

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)
F2	0011	7A	OP1	OP2	L

2. Generated Code

OP2=REGISTER	op2=storage
or R2,R2 OPT OP1 BC 8,OP1	OC S2(L),S2 OPT OP1 BC 8,OP1

OP2 may be indirectly addressed.

Pointer Comparison

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	78	OP1	OP2

2. Generated Code

   a. OP2 is a pointer (bit 6 of byte 15 is set on)
```
      CLC    S1+1(3),S2+1
```

   b. OP2 is not a pointer
```
      LA     0,S2
      ST     0,GW0+80
      CLC    S1+1(3),GW0+81
```

Sum

1. Format of the macro

(1)	(2)	(1)	(6)	(1)	(2)
F2	000E	7B	OP1	N	LABEL

2. Generated Code
```
LA      4,OP1
LA      3,N
LABEL:
```
OP1 may be indirectly addressed.

Return to Label Variable

1. Format of the macro

(1)	(2)	(1)	(1)	(6)
F2	000B	80		OP1

2. Generated Code
```
LA      1,S1
L       15,N'13'
BR      15
```

DO Branch

1. Format of the macro

(1)	(2)	(1)	(6)
F2	000A	81	OP1

2. Generated Code
```
LA      1,S1
L       15,N'13'
BR      15
```

Assign Label Variable

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	85	OP1	OP2

2. Generated Code
```
MVC     S1(8),S2
```
OP1 and OP2 may be indirectly addressed.

Store

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	83	OP1	OP2

2. Generated Code
```
ST      R1,S2
```
OP2 may be indirectly addressed.

Store Short

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	84	OP1	OP2

2. Generated Code
```
STE     R1,S2
```
OP2 may be indirectly addressed.

Character String Assignment

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(1)	(1)
F2	0012	86	OP1	OP2	L1	L2

2. Generated Code

   a. L1 ≤ L2:
```
MVC     S1(L1),S2
```
   b. L1 > L2:
```
MVI     S1,X'40'
MVC     S1+1(L1-1),S1
MVC     S1(L2),S2
```
OP1 and OP2 may be indirectly addressed.

Subscripted Variable

1. Format of the macro

(1)	(2)	(1)	(1)	(1)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)
F2	003C	87		N	OP1	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9

2. Generated Code

```
a.  N=0:  LA    R1,S8
          A     R1,S9

b.  N=1:  L     R1,S2
          MH    R1,S3
          A     R1,S9
          LA    5,S8
          AR    R1,5

c.  N=2:  L     R1,S2
          MH    R1,S3
          A     R1,S4
          MH    R1,S5
          A     R1,S9
          LA    5,S8
          AR    R1,5

d.  N=3:  L     R1,S2
          MH    R1,S3
          A     R1,S4
          MH    R1,S5
          A     R1,S6
          MH    R1,S7
          A     R1,S9
          LA    5,S8
          AR    R1,5
```

All operands may be indirectly addressed.

Substring

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (6)
┌───┬────┬───┬───┬───┐
│F2 │0010│88 │OP1│OP2│
└───┴────┴───┴───┴───┘
```

2. Generated Code

if OP2=STORAGE

```
LA    5,S1
USED  REGISTER(0)
A     5,S2
BCTR  5,0
```

or, if OP2 = REG:

```
LA    5,S1
USED  REGISTER(0)
AR    5,R2
BCTR  5,0
```

OP1 and OP2 may be indirectly addressed.

High

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (1)
┌───┬────┬───┬───┬───┐
│F2 │000B│BA │OP1│ N │
└───┴────┴───┴───┴───┘
```

2. Generated Code

```
a.  N=0:  MVI  OP1,X'FF'

b.  N<0:  MVI  OP1,X'FF'
          MVC  OP1+1(N),OP1
```

OP1 may be indirectly addressed.

Load Address of ON Block

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (1)
┌───┬────┬───┬───┬───┐
│F2 │000B│8B │OP1│ R │
└───┴────┴───┴───┴───┘
```

2. Generated Code

```
LA    R,OP1
AH    R,0(R)
```

OP1 may be indirectly addressed.

Initial

1. Format of the macro

```
 (1)  (2)  (1)  (2)  (2)
┌───┬────┬───┬───┬───┐
│F2 │0008│8C │ A │ B │
└───┴────┴───┴───┴───┘
```

2. Generated Code

```
LA    1,A
L     15,B
BALR  14,15
```

Format

1. Format of the macro

```
 (1)  (2)  (1)  (2)
┌───┬────┬───┬───┐
│F2 │0006│8D │ A │
└───┴────┴───┴───┘
```

2. Generated Code

```
LA    1,A
L     15,N'21'
BALR  14,15
```

Load Transmit

1. Format of the macro

(1)	(2)	(1)	(2)
F2	0006	8E	A

2. Generated Code

```
L       15,A
```

Call

1. Format of the macro

(1)	(2)	(1)	(1)	(6)		(6)
F2	N*6+5	90	N	OP1		OP+1

2. Generated code

```
a.  N≤8:    STM  0,4,GW0+80
            LA   14,S3
            LA   15,S4
            LA   0,S5
            .
            .
            .
            LA   RX,SN+1   *
            STM  14,RX,GW0
            L    15,S1     **
            LR   0,RS      **
            LA   1,GW0
            BALR 14,15

b.  N>8:    STM  0,4,GW0+80
            LA   14,S3
            LA   15,S4
            LA   0,S5
            LA   1,S6
            LA   2,S7
            LA   3,S8
            LA   4,S9
            STM  14,4,GW0
            LA   14,S10
            LA   15,S11
            LA   0,S12
            .
            .
            .
            LA   RX,SN+1   *
            STM  14,4,GW0+28
            LM   0,4,GW0+80
            L    15,S1     **
            LR   0,RS      **
            LA   1,GW0
            BALR 14,15
```

3. Format of the operands

The general format of the parameters is

(1)	(2)	(2)	(1)
E1	NAME	MOD	AT

or

(1)	(2)	(1)	(1)	(1)
E1	NAME	X	Y	AT

if the parameter is an entry name.

* This instruction is deleted if the name is X'0000'. If the parameter is an entry name, the instruction is replaced by:

```
MVC       BLOCK(4),S(I)
LA        RX,BLOCK
ST        RS,BLOCK+4 only if RS = 0
BLOCK:    DS 2F
```

**If the entry name is a parameter, this instruction is replaced by:

```
L         5,S1
LM        15,0,0(5)
```

If RS=0, the instruction LR 0,RS is deleted.
RS depends on X and Y as follows:

	X = 0	X = 1	X = 2
Y = 1	RS = 0	RS = 13	RS = 0
Y = 2	RS = 0	RS = 11	RS = 13
Y = 3	RS = 0	RS = 11	RS = 10

Return

1. Format of the macro

(1)	(2)	(1)	(1)
F2	0005	91	I

2. Generated code

```
a.  I=0:    L     13,4(13)
            LM    14,12,12(13)
            BR    14

b.  I=1:    LR    13,11
            L     13,4(13)
            LM    14,12,12(13)
            BR    14

c.  I=2:    LR    13,10
            L     13,4(13)
            LM    14,12,12(13)
            BR    14

d.  I=3:    same as I=0.
```

Prologue

1. Format of the macro

```
  (1)    (2)     (1)  (6)  (6)  (1)  (1)  (1)
┌───┬────────┬───┬───┬───┬───┬───┬───┐──
│F2 │(N+4)*6│92 │OP1│OP2│LN │BN │ N │
└───┴────────┴───┴───┴───┴───┴───┴───┘──

      (1)  (1)  (1)   (2)   (6)     (6)
───┌───┬───┬───┬─────┬───┬───┬───┐
   │ E │ I │FL │LABEL│P1 │   │PN │
───└───┴───┴───┴─────┴───┴───┴───┘
```

OP1	- Entry name
OP2	- gives the address of the ON block
LN	- Level number
BN	- Block number
N	- Number of parameters
E	- indicates whether main entry or secondary entry
I	- indicates whether internal, external (1) or external (2)
FL	- ON-flags
LABEL	- a label-name used in the prologue.

P1, P2, ... PN - Parameters.

2. Generated code

a. Main ENTRY of MAIN PROCEDURE (E=0, I=1)

```
 BALR     15,0
 BEGIN    (BN,LN,OP1)
 NOPR
 BAL      14,10(15)
 DC       A(STATIC)
 L        12,0(14)
 LR       9,15
 LR       3,1

if ONSYSLOG:
 L        15,N'10'

else :
 L        15,N'11'

 BALR     14,0
 LA       14,14(14)
 BALR     1,15
 DC       X'FL'
 DC       AL3(OP2)
 DC       XL4'LENGTH OF DSA'

if indirect library routines in
compilation
 L        1,N'5'
 L        15,N'16'
 BALR     14,15

if N > 0
 MVC      PN(4),0(3)              *
 MVC      P(N-1) (4),4(3)         *
 MVC      P(1) (4),4*(N-1) (3)    *
 MVC      P1(4),4*(N-1) (3)       *
 MVC      80(4,13),4*N(3)
```

b. Secondary ENTRY of MAIN PROCEDURE (E=1, I=1)

```
 CNOP           2,4
 INDIVISIBLE    CODE (17,10)
 BALR           15,0
 BAL            14,8(15)
 DC             A (STATIC)
 L              12,0(14)

LABEL+1:
 L              9,LABEL
 LR             3,1

if ONSYSLOG:
 L              15,N'10'

else :
 L              15,N'11'

 BALR           14,0
 LA             14,14(14)
 BALR           1,15
 DC             X'FL'
 DC             AL3(OP2)
 DC             XL4'LENGTH OF DSA'

if indirect library routines in
compilation
 L              1,N'5'
 L              15,N'16'
 BALR           14,15

if N > 0:
 MVC          PN(4),0(3)                 *
 MVC          P(N-1) (4),4(3)            *
 MVC          P(I) (4),4*(N-I) (3)       *
 MVC          P1(4),4*(N-1) (3)          *

 MVC          80(4,13),4*N(3)

for STATIC STORAGE:
LABEL
DC  A of segment origin (LABEL+1)
```

c. Main ENTRY of no MAIN PROCEDURE (E=0, I=0 or 2)

```
 BEGIN           (BN,LN,OP1)
 STM            14,12,12(13)
 CNOP           0,4
 INDIVISIBLE    CODE (22,12)
 BAL            14,12 (15)
 DC             A(STATIC)
 L              12,0(14)
 LR             9,15
 LR             3,1
 L              15,N'12'
 BALR           14,0
 LA             14,14(14)
 BALR           1,15
 DC             X'FL'
 DC             AL3 (OP2)
 DC             XL4'LENGTH OF DSA'

if level 1, LN=1:
 L              11,76(13)
```

```
if level 2, LN=2:
 L              10,76(13)
 L              11,76(10)


if external, I=2 and indirect
library routines in compilation
 L              1,N'5'
 L              15,N'16'
 BALR           14,15


if N>0:
 MVC         PN(4),0(3)             *
 MVC         P(N-1)  (4),4  (3)     *
 MVC         P(I) (4),4*(N-I) (3)   *
 MVC         P1(4),4*(N-1) (3)      *


 MVC         80(4,13),4*N(3)
```

d. Secondary ENTRY of no MAIN PROCEDURE
(E=1, I=0, or 2)

```
 STM            14,12,12(13)
 CNOP           0,4
 INDIVISIBLE    CODE (22,12)
 BAL            14,12(15)
 DC             A(STATIC)
 L              12,0  (14)

LABEL+1:
 L              9,LABEL
 LR             3,1
 L              15,N'12'
 BALR           14,0
 LA             14,14(14)
 BALR           1,15
 DC             X'FL'
 DC             AL3(OP2)
 DC             XL4'LENGTH OF DSA'

if level 1, LN=1:
 L              11,76(13)

if level 2, LN=2:
 L              10,76(13)
 L              11,76(10)

if external, I=2 and indirect
library routines in compilation
 L              1,N'5'
 L              15,N'16'
 BALR           14,15

if N > 0:
 MVC         PN(4),0(3)             *
 MVC         P(N-1)(4),4(3)
 MVC         P(I) (4),4*(N-I) (3)   *
 MVC         P1(4),4*(N-1)  (3)     *

 MVC         80(4,13),4*N(3)
```

for STATIC STORAGE:
LABEL DC A of SEGMENT ORIGIN
(LABEL+1)

Store Multiple

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (6)
┌───┬────┬───┬───┬───┐
│F2 │0010│93 │OP1│OP2│
└───┴────┴───┴───┴───┘
```

2. Generated code

```
STM    R1,R1+1,S2
```

OP2 may be indirectly addressed.

Store Long

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (6)
┌───┬────┬───┬───┬───┐
│F2 │0010│94 │OP1│OP2│
└───┴────┴───┴───┴───┘
```

2. Generated code

```
STD    R1,S2
```

OP2 may be indirectly addressed.

Move Address

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (6)
┌───┬────┬───┬───┬───┐
│F2 │0010│95 │OP1│OP2│
└───┴────┴───┴───┴───┘
```

2. Generated code

OP1=REGISTER	OP1=STORAGE
LA R1,S2	LA 5,S2 USED REGISTER(0) ST 5,S1

OP1 and OP2 may be indirectly addressed.

Array Expression Begin

1. Format of the macro

```
 (1)    (2)    (1)  (3)   (3)   (2)  (2)
┌───┬───────┬───┬───┬─────┬───┬───┐
│F2 │14+N*6│98 │ A │LABEL│ N │ M │ ──
└───┴───────┴───┴───┴─────┴───┴───┘

     (6)     (6)
──┌───┬───┬───┐
  │OP1│   │OPN│
──└───┴───┴───┘
```

2. Generated code

```
a.  N ≤ 7: STM  14,4,GW0+80
           LA   14,S1
           LA   15,S2
           LA   0,S3
           .
           .
           .
           LA   RX,SN
           STM  14,RX,A
           LM   14,4,GW0+80
           LA   4,M
    LABEL: ST   4,DSA+84

b.  N > 7: STM  14,4,GW0+80
           LA   14,S1
           LA   15,S2
           LA   0,S3
           LA   1,S4
           LA   2,S5
           LA   3,S6
           LA   4,S7
           STM  14,4,A
           LA   14,S8
           LA   15,S9
           LA   0,S10
           .
           .
           .
           LA   RX,SN
           STM  14,RX,A+28
           LM   14,4,GW0+80
           LA   4,M
    LABEL: ST   4,DSA+84
```

All operands (OP1, OP2, ... OPN) may be indirectly addressed.

Low

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (1)
┌───┬────┬───┬───┬───┐
│F2 │000B│9A │OP1│  N│
└───┴────┴───┴───┴───┘
```

2. Generated code

```
a.  N=0:   MVI  OP1,X'00'

b.  N>0:   MVI  OP1,X'00'
           MVC  OP1+1(N),OP1
```

OP1 may be indirectly addressed.

Load Variable

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (2)  (2)
┌───┬────┬───┬───┬───┬───┐
│F2 │000E│9C │OP1│ A │ B │
└───┴────┴───┴───┴───┴───┘
```

2. Generated code

```
LA     1,S1
ST     1,A+B
```

OP1 may be indirectly addressed.

Set Byte

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (1)
┌───┬────┬───┬───┬───┐
│F2 │000B│9D │OP1│ M │
└───┴────┴───┴───┴───┘
```

2. Generated code

```
MVI    S1,M
```

OP1 may be indirectly addressed.

Return Function Value

1. Format of the macro

```
 (1)  (2)  (1)  (1)  (6)  (1)
┌───┬────┬───┬───┬───┬───┐
│F2 │000C│A1 │ I │OP1│ L │
└───┴────┴───┴───┴───┴───┘
```

2. Generated code

```
a.  I=0:    USED   REGISTER(5)
            L      5,80(13)
            MVC    0(L,5),S1
            USED   REGISTER(0)
            L      13,4(13)
            LM     14,12,12(13)
            BR     14

b.  I=1:    USED   REGISTER(5)
            L      5,80(11)
            MVC    0  (L,5),S1
            USED   REGISTER(0)
            LR     13,11
            L      13,4(13)
            LM     14,12,12(13)
            BR     14

c.  I=2:    USED   REGISTER(0)
            L      5,80(10)
            MVC    0(L,5),S1
            USED   REGISTER(0)
            LR     13,10
            L      13,4(13)
            LM     14,12,12(13)
            BR     14

d.  I=3:    same as I = 0.
```

OP1 may be indirectly addressed.

Overlay

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (1)
┌───┬────┬───┬───┬───┐
│F2 │000B│A2 │OP1│ L │
└───┴────┴───┴───┴───┘
```

2. Generated code

```
a.  L<8:   MVI    GW0,X'40'
           MVC    GW0+1(7),GW0
           MVC    GW0(L),S1
           SR     0,0
           LA     1,GW0
           SVC    4

b.  L≥8:   MVC    GW0(L),S1
           SR     0,0
           LA     1,GW0
           SVC    4
```

OP1 may be indirectly addressed.

Load

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (6)
┌───┬────┬───┬───┬───┐
│F2 │0010│A3 │OP1│OP2│
└───┴────┴───┴───┴───┘
```

2. Generated code

```
L      R1,S2
```

OP2 may be indirectly addressed.

Load Short

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (6)
┌───┬────┬───┬───┬───┐
│F2 │0010│A4 │OP1│OP2│
└───┴────┴───┴───┴───┘
```

2. Generated code

```
LE     R1,S2
```

OP2 may be indirectly addressed.

Move Immediate

```
 (1)  (2)  (1)  (6)  (1)
┌───┬────┬───┬───┬───┐
│F2 │000B│A5 │OP1│M  │
└───┴────┴───┴───┴───┘
```

2. Generated code

```
MVI    S1,M
```

OP1 may be indirectly addressed.

Array Expression End

1. Format of the macro

```
 (1)    (2)    (1)  (3)   (3)   (2)  (2)
┌───┬───────┬───┬───┬─────┬───┬───┬───
│F2 │14+N*2 │A8 │A  │LABEL│N  │M  │
└───┴───────┴───┴───┴─────┴───┴───┴───

    (2)  (2)  (2)  (2)
───┬───┬───┬───┬───┐
   │C1 │C2 │   │CN │
───┴───┴───┴───┴───┘
```

2. Generated code

```
a.  N ≤ 7:  STM    14,4,GW0+80
            LM     14,4,A
            LA     14,C1(14)
            LA     15,C2(15)
            LA      0,C3(0)
            A       0,A+8
            LA      1,C4
            .
            .
            .
            LA     RX,CN
            STM    14,RX,A
            LM     14,4,GW0+80
            OPT    (LABEL)
            BCT    4,LABEL

b.  N > 7:  STM    14,4GW0+80
            LM     14,4,A
            LA     14,C1(14)
            LA     15,C2(15)
            LA      0,C3(0)
            A       0,A+8
            LA      1,C4(1)
            LA      2,C5(2)
            LA      3,C6(3)
            LA      4,C7(4)
            STM    14,4,A
            LM     14,4,A+28
            LA     14,C8(14)
            LA     15,C9(15)
            LA      0,C10(0)
            A       0,A+36
            LA      1,C11(1)
            .
            .
            .
            LA     RX,CN(RX)
            STM    14,RX,A+28
            LM     14,4,GW0+80
            OPT    (LABEL)
            BCT    4,LABEL
```

Repeat

1. Format of the macro

```
 (1)  (2)  (1)  (6)  (6)  (1)  (1)
┌───┬────┬───┬───┬───┬───┬───┐
│F2 │0012│AA │OP1│OP2│L  │I  │
└───┴────┴───┴───┴───┴───┴───┘
```

2. Generated code

```
MVC     S1(L),S2
MVC     S1+L(I*L),S1
```

OP1 and OP2 may be indirectly addressed.

Load DED

1. Format of the macro

(1)	(2)	(1)	(6)
F2	000A	AC	OP1

2. Generated code

```
LA      2,S1
```

OP1 may be indirectly addressed.

Load Scalar

1. Format of the macro

(1)	(2)	(1)	(6)
F2	000A	AD	OP1

2. Generated code

```
LA      1,S1
BALR    14,15
```

OP1 may be indirectly addressed.

Load Array

1. Format of the macro

(1)	(2)	(1)	(6)	(2)	(2)	(2)
F2	0010	AE	OP1	A	B	LABEL

2. Generated code

```
      LA      3,B
      LA      0,A
      LA      1,S1
LABEL BALR    14,15
      AR      1,0
      OPT     (LABEL)
      BCT     3,LABEL
```

OP1 may be indirectly addressed.

Call (3)

1. Format of the macro

(1)	(2)	(1)	(3)	(1)	(1)	(1)
F2	000A	B0	S1	X	Y	AT

2. Generated code

```
L       15,S1
LR      0,RS
BALR    14,15
```

or if AT indicates a parameter

```
L       5,S1
LM      15,0,0(5)
BALR    14,15
```

RS depends on X and Y as follows:

	X=0	X=1	X=2
Y=1	RS=0	RS=13	RS=0
Y=2	RS=0	RS=11	RS=13
Y=3	RS=0	RS=11	RS=10

Set True on Condition

1. Format of the macro

(1)	(2)	(1)	(1)	(6)	(2)
F2	000E	B1	C	OP1	LABEL

2. Generated code

```
MVI     S1,X'80'
OPT      (LABEL)
BC      C,LABEL
MVI     S1,X'00'
LABEL
```

OP1 may be indirectly addressed.

Load Multiple

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	B3	OP1	OP2

2. Generated code

```
LM      R1,R1+1,S2
```

OP2 may be indirectly addressed.

Load Long

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	B4	OP1	OP2

2. Generated code

```
LD     R1,S2
```

OP2 may be indirectly addressed.

Move Character

1. Format of the macro

(1)	(2)	(1)	(6)	(6)
F2	0010	B5	OP1	OP2

2. Generated code

a. OP2 = REGISTER:
```
STC     R2,S1
```

b. OP2 = STORAGE:
```
MVC     S1(1),S2+3
```

OP1 and OP2 may be indirectly addressed.

Or Immediate

1. Format of the macro

(1)	(2)	(1)	(6)	(1)
F2	000B	B6	OP1	M

2. Generated code

```
OI     S1,M
```

OP1 may be indirectly addressed.

Multiply Halfword

1. Format of the macro

(1)	(2)	(1)	(6)	(1)
F2	000B	B8	OP1	R

2. Generated code

```
MH     R,OP1
```

OP1 may be indirectly addressed.

Call Library Routine

1. Format of the macro

(1)	(2)	(1)	(3)
F2	0007	BA	NAME

2. Generated code

```
L       15,NAME
BALR    14,15
```

Loop Begin

1. Format of the macro

(1)	(2)	(1)	(6)	(6)	(2)	(2)
F2	0014	BC	OP1	OP2	N	LABEL

2. Generated code

```
      LA     3,N
      LA     4,S1
LABEL ST     4,S2
```

OP1 and OP2 may be indirectly addressed.

Loop End

1. Format of the macro

(1)	(2)	(1)	(2)	(2)
F2	0008	BD	L	LABEL

2. Generated code

```
LA     4,L(4)
OPT    (LABEL)
BCT    3,LABEL
```

Generally, the generation code from the macros can be made using the same algorithm for different macros. There are only a few macros which require special handling. These special macros are processed in the first generator phase.

Further objectives of the first phase are:

1. To translate the macro key and the macro code to a key and an address in the model instruction dictionary.
2. To build a table giving the passes needed in the second phase.
3. To change the text string, the statement identifiers, the attribute tables for declared variables, and the END OF STATEMENT's are eliminated from the text string.
4. To generate code that is independent of the macros. The first instructions of the text string must be the address constants for storage blocks.

```
L'4'  DC  AL2(0)       =   0
      DC  AL2(4096)    =   4K
      DC  AL2(8192)    =   8K
      DC  AL2(12288)   =  12K
      DC  AL2(16348)   =  16K
      DC  AL2(18396)   =  18K
      DC  AL2(20480)   =  20K
      DC  AL2(22528)   =  20K
      DC  AL2(24576)   =  24K
      DC  AL2(26624)   =  26K
      DC  AL2(28672)   =  28K
      DC  AL2(30720)   =  30K
L'6'  DC  AL2(0)
      DC  AL2(0)
```

DESCRIPTION OF ROUTINES

Note: The routines HUI and HUE are described in phase E60.

After initialization, filling of the buffers, and the generation of the address constants for the storage blocks (0K, 4K, 8K, etc.), the text is scanned for macros. Each macro key detected is translated into an address W and an integer Z. If Z = 0, i.e., if the macro is to be processed in this phase, W gives the address of the routine that determines the code to be generated.

The individual routines are shown in flow charts TD - TT. Table MATAB showing the macro key, the routine name, and the corresponding flow chart, is given in Figure 1. The code generated by each of the routines is described in the section The Macros and the Generated Code.

Macro Key	Routine Name (W=routine address)	Flow Chart
X'10'	decimal ADDITION	TH
X'11'	SUBTRACTION	TH
X'12'	MULTIPLICATION	TH
X'13'	DIVISION	TJ
X'14'	NEGATION (1 oprnd.)	TK
X'15'	ASSIGNMENT	TL
X'16'	NEGATION (2 oprnds.)	TK
X'18'	COMPARISON	TH
X'50'	conversion CV30	TM
X'56'	conversion CV38	TN
X'90'	CALL	TR
X'92'	PROLOGUE	TD
X'98'	ARray EXpression HEader	TP
X'A8'	ARray EXpression ENd	TQ
X'AA'	REPEAT	TO
X'B0'	CALL(3)	TR

Figure 1. Macro Keys and Corresponding Processing Routines (MATAB)

SHIFT -- TG

This is a subroutine used by the decimal arithmetic macros. For details refer to the section The Macros and the Generated Code.

LAREG -- TS

This is a submacro used by the routines AREXHE and CALL. It causes the generation of a sequence of "LA REG,OP" instructions.

HEINS

This routine is used for inserting operands in a fixed set of code and moving this code onto the text output medium. The code to be generated is also given in the flow charts.

HMOCO

This routine is used to move a given set of code onto the text output medium.

PHASE PL/IE60/61 (CODE GENERATION II) -- UA, UB

Most of the macros are processed in the second generator phase. Therefore, the phase may make multiple passes over the text string and include multiple changes of the model-instruction dictionary. In the first generator phase, the needed passes of the second generator phase are determined and noted in a table. Only the phase overlays given by this table are used in the second generator phase.

Phase E60 processes only those macros that allow a single algorithm for handling indirect addressing, code generation, and inserting of the operands.

The needed code is determined by the work done on the macro-definition header. If a set of code is determined, a branch is made from the macro-definition header to a model-instruction set. The model-instruction set gives the information for inserting the operand and for the treatment of indirect addressing.

The algorithm for the second generator phase may be separated into several parts:

1. Determination of the phase overlays.0 The phase overlays (phase E61) contain parts of the model-instruction dictionary. These phase overlays may not all be necessary. In the first generator phase the needed parts of the model-instruction dictionary are noted in a table. The table is used for the determination of the phase overlays.

2. Scan of text string. The number of different elements is reduced in the first generator phase. In the second generator phase, the text string may consist of:

   a. Macros

   b. Code

   c. Variable tables

   d. Constant tables

DESCRIPTION OF ROUTINES

Symbols Used in Flow Charts

X	: input pointer
Y	: output pointer
S	: buffer index
B(S)	: start address of buffer(S)
L	: length of text element
RC	: condition code set by the machine
R	: register used for indirect addressing

HDETER -- UC, UD

After a macro is detected, it must be determined whether the macro can be generated in the current pass or not. If the macro can be generated, the code required for the macro is determined by the work done on the macro-definition header. The macro-definition header consists of instructions, (described in the section The Macro Definition Header) giving information for selecting model-instruction sets. The code generated for one macro may consist of more than one model-instruction set.

INSERT -- UE, UF

A model-instruction set determined by the macro definition header contains no complete code. Some operands of the instructions must be taken over from the operand list of the macro. The information for this process is given in the model-instruction set, too. After inserting the operands, the generation is continued in the macro-definition header.

HOPE -- UG

Secondary entry point: HOPEI

This routine is used for moving the operands depending on the three parameters P, M, and LM. For details on these parameters refer to the section Model Instruction Sets. The routine HINAD is called for operands that refer to variables or constants.

HINDAD -- UH

A requirement for the use of indirect addressing is indicated in byte 6 of the corresponding operand. The significant bits of this byte are:

Bit 1:	1 = CONTROLLED
Bit 3:	1 = EXTERNAL 0 = INTERNAL
Bit 5:	1 = PARAMETER (formal)

If an operand OPX is controlled by B (OPX CONTROLLED (B)), OPX is replaced by B. The corresponding byte (6) is also changed; however, bit 1 is not changed.

Code Used with Indirect Addressing

Suppose that in the code there appears an instruction A WR,OPX where OPX requires indirect addressing. The code which replaces the above instruction is shown in Figure 1.

OPX=	CODE
PARAMETER	L REG, OPX A WR, O (REG)
EXTERNAL	L REG, AOP A WR, O (REG) . . . . AOP DC A (OPX)
CONTROLLED (B)	L REG, B A WR, O (REG)
CONTROLLED (B) and B = PARAMETER	L REG, OPX L REG, O (REG) A WR, O (REG)
CONTROLLED (B) and B = EXTERNAL	L REG, AOP L REG, O (REG) A WR, O (REG) . . . . AOP DC A (B-1)
Note: The following cases are invalid: OPX = PARAMETER and EXTERNAL = EXTERNAL and PARAMETER = PARAMETER and CONTROLLED = CONTROLLED (B) where B itself is CONTROLLED	

Figure 1. Code Used with Indirect Addressing

HUE -- UJ, UK

The routine has four entry points:

HUES	: used for skipping in the input text.
HUEI	: used for moving from input buffers to any location except output buffer.
HUEO	: used for moving from any location except the input buffers into the putput buffer.
HUEIO	:used for moving from the input buffers into the output buffer.

The following parameters are used:

X	=FROM address
Y	=TO address
L	=length of text to be moved or skipped
B(S) or BS	= index of buffer being used
BUFL	= length of buffers

HMOVE -- UL

This routine is called by HUE for moving L bytes of information from address X to address Y. L may be greater than 256 bytes.

MASURO

At this point, control is transferred to the address contained in the address constant. For details, refer to General Description of the Generator Phases, under Operations, macro subroutine.

The routines branched to handle all cases that occur rarely and are not handled by operations and instructions contained in the model instruction set.

LOAD

This is a supervisor macro, which is used in this phase.

PL/IF25 (SORTING CONSTANTS AND VARIABLES) -- W9

This phase performs the functions discussed in the subsequent paragraphs. For further details refer to Description of Routines below.

Text Scan. The text-input string is scanned for constants and generated variables. When constants or generated variables are found, they are written onto a work file as the prestatement table PRETAB. Assembler instructions are written onto the text-output file TXTOUT.

PRETAB Scan. PRETAB is scanned for generated variables. These variables are converted to entries for DSTAB. A DSTAB entry contains the following information:

Byte (s)	Contents
0-1	Internal name of the variable.
2-3	Length of the variable during object time.
4	Block and level number
5	Attributes as follows:

Bit	Indication
0	0 = AUTOMATIC
	1 = STATIC
1-3	000 = Scalar variable
	001 = STRUCTURE
	010 = ARRAY
	100 = Pointer
	111 = Parameter
4-7	Left-hang bits if STRUCTURE

DSTAB entries are written on TXTOUT as the output buffer is filled.

Any constants in PRETAB are written on the text input file (TXTIN) following the assembler instructions.

SYMTAB Scan. (For generation of the symbol table, refer to phase B20.) SYMTAB is scanned for variables for which STATIC or AUTOMATIC storage is to be allocated. These variables are converted to DSTAB entries as described above and written on TXTOUT.

DSTAB entries for the abovementioned variables are also written, as table DSTAB, on the work file.

Sorting Constants. The constants written on TXTIN during the PRETAB scan are sorted in the following order:

1. Constants or address constants that are not optimizable.
2. Optimizable address constants.
3. Optimizable 8-byte constants.
4. Optimizable 4-byte constants.
5. Optimizable 2-byte constants.
6. Other optimizable constants.

The sorting of constants is accomplished by 6 scans through the unsorted constants on TXTIN. The first scan is for constants and address constants which cannot be optimized. Any such constants are written on the work file.

The second scan is for optimizable address constants, and these constants are written on the work file. This procedure is continued for the remaining types of constants as shown above. At the end of the sixth scan, all constants are written on the work file in the abovementioned order.

Phase Input and Output

The input for this phase is a text string containing Assembler code, constants, and generated variables.

Assembler Instructions. The format of these instructions is as follows:

Key X'F6'	Length of following instructions + 3	One or more Assembler instructions

Generated Variables. These entries have a fixed length of eight bytes and appear in the text string in a format as follows:

Key X'F0'	Length of following entries + 3	One or more 8-byte entries

Each entry contains the following information:

Byte(s)	Contents	
0-1	Internal representation of the name.	
2	Bit	Indication
	0-3	Reserved.
	4-7	Internal length of the variable.
3	0	0 = AUTOMATIC 1 = STATIC
	1	not used
	2	1 = POINTER
	3-5	not used
	6	1 = DELETE
	7	1 = CONSTANT 0 = VARIABLE
4	0-4	not used
	5	1 = LABEL
	6-7	not used
5	0-2	not used
	3	1 = string data
	4	1 = BIT string 0 = CHARACTER string
	5	1 = FIXED 0 = FLOAT
	6	1 = BINARY 0 = DECIMAL
	7	not used
6	If character-string type data: length of string.	
7	0-1	block level
	2-7	block number

Constants. Constants that are to be allocated storage in STATIC storage during object time appear in the following format:

Key X'F3'	Length of following constants + 3	One or more constants

Each constant has the following format:

Byte(s)	Contents	
0-1	Internal name of the constant	
2	Attributes as follows:	
	Bit	Indication
	0-1	00 = X-type DC 01 = A-type DC 10 = V-type DC 11 = AL3-type DC
	2-3	00 = Optimizable 01 = To be deleted 10 = Not optimizable, but containing block is optimizable 11 = Not to be deleted.
	4	1 = V-type constants of a length of three bytes
	5	1 = word boundary if not optimizable.
	6-7	00 = not used 01 = A-type DC for label assignment 10 = A-type DC for an entry point 11 = double-word boundary if not optimizable
3-4	Length of constant (if an address constant, the length must be four bytes).	
5-n	Internal representation of constant or, if an address constant, name and modifier.	

Symbol Table SYMTAB. The entries of this table have a fixed length of 20 bytes each. For further information about table SYMTAB see the description of the generation of the table in the preceding phases.

Output of the Phase:

1. A text string on TXTIN containing Assembler code only.

2. Table DSTAB on a work file. This table contains generated and declared variables, for which STATIC or AUTOMATIC storage is to be allocated.

3. Constant table CONTAB on the work file. This table contains the constants in sorted order.

DESCRIPTION OF ROUTINES

Symbols used in flow charts:

INPT	: input pointer for text
INPO	: input pointer for text
BUFFL	: buffer length
BULE	: buffer length
EOP	: end of program
OPT1	: output pointer for text
OPT2	: output pointer for text
REC	: number of records required from text input
RECLE	: record length
COMPL	: length of available input area
RECN	: record counter for DSTAB
LENGTH	: length of variable
DSTA	: buffer for DSTAB entry

Text Scan -- XA through XF

The text string to be scanned contains

1. Assembler code (key = X'F6'),

2. generated variables (key = X'F0'),

3. constants (key = X'F3'), and

4. end-of-program key X'FF'.

Each key, except the end-of-program key, is followed by 2 bytes containing the skippable length. When the text scan is completed, a text string containing Assembler code, excluding keys and skippable length indications, on TXTOUT and the table PRETAB on the work file are provided for further processing.

The text string is consecutively read into input buffers BUFF1 and BUFF2 and processed as follows:

1. Assembler instructions (key X'F6' and no skippable length) are moved into OBUF1 and written on TXTOUT.

2. A constant or a generated variable causes the program to branch to the subroutine CONVAR. This routine moves the string of constants (or generated variables) into OBUF2 and writes them on the work file as table PRETAB.

3. When the end-of-program key is encountered, the records in OBUF1 and OBUF2 are written out and control is passed to the routines that cause the table PRETAB to be scanned.

<u>INITIAL1 -- XB</u>

This subroutine sets the input and output pointers (INPT, OPT1, and OPT2 to 0 and calculates the buffer addresses.

<u>TXTIN -- XC</u>

This subroutine reads text from TXTIN into the input buffers as determined by parameter REC.

<u>WAIT1 -- XC</u>

This subroutine tests whether or not the next text record is ready for processing. If not, the subroutine loops until the I/O instruction is completely executed.

<u>ASSCODE -- XD</u>

This subroutine causes Assembler code, excluding the key X'F6' and the skippable-length value to be moved into OBUF1.

If OBUF1 are full, its contents is written on TXTOUT, output pointer OPT1 is reset to zero, and Assembler code is moved into the next buffer. If OBUF1 is not full, the routine only increases OPT1 by the length of the Assembler code moved into the buffer.

<u>OUTASS -- XD</u>

This subroutine writes the contents of OBUF1 on TXTOUT. OBUF1 contains Assembler code. OPT1 is reset to 0.

<u>CONVAR -- XE</u>

Entry point: PRET.
This subroutine controls the writing of constants and of generated variables on a work file. If the length of the string to be written exceeds the length of the input buffer, the string is written out in sections.

<u>PRET -- XF</u>

This subroutine causes the constants or generated variables to be moved from the input buffers into OBUF2. If the length of the constants (or variable) string to be moved exceeds the length of OBUF2, the string must be written out in sections.

Constants or variables are moved into OBUF2 until this output buffer is full. The output buffer is then written onto the work file.

<u>PUTPRE -- XF</u>

This subroutine writes the contents of OBUF2 on the work file and resets OPT2 to 0.

<u>SKIPIN -- XG</u>

This subroutine increases INPT by the skippable length. If necessary, read-in of the next record into input buffer 2 is initiated.

<u>PRETAB Scan -- XH through XO</u>

The data contained in PRETAB (on a work file) is scanned. This data consists of constants and generated variables. The format of the PRETAB entries is the same as described for constants and general variables under <u>Phase Input and Output</u>).

The End-of-PRETAB key is X'FF'.

When the PRETAB scan is completed, one 6-byte DSTAB entry is provided for each entry in PRETAB. In addition, all constants are written (unsorted) on TXTIN following the program string.

The data contained in PRETAB is consecutively read from the work file into the input buffers. Each generated variable is converted to a 6-byte DSTAB entry by the subroutine GENVAR and written on TXTOUT. Constants are written on TXTIN. When the End-of-PRETAB key is encountered, control is transferred to the routines that scan SYMTAB.

<u>INITIAL2 -- XI</u>

This subroutine calculates the addresses for the input and output buffers. The input and output pointers are set to 0 and the parameters for the input and output routines are set to their initial values.

GETPRE -- XJ

This subroutine reads PRETAB from the work file into the input buffers as determined by parameter REC.

WAIT2 -- XJ

This subroutine controls the processing of input buffer BUFF1 and loops until the I/O operation to fill buffer BUFF2 is completed.

GENVAR -- XK

This subroutine converts the entry for the generated variable in PRETAB to a 6-byte DSTAB entry.

RTESTIN -- XL

This subroutine determines if the next PRETAB record is to be read or if a waiting loop is to be entered to wait for the completion of the preceding I/O instruction.

OPT -- XM

This subroutine causes the DSTAB entry for the variable to be moved into the output buffer area and OPT1 to be increased by 6. If OPT1 is greater than the buffer length, subroutine DSPUT is called.

DSPUT -- XM

This subroutine causes the contents of OBUF1 to be written on TXTOUT. The contents of OBUF2 are moved to OBUF1. OPT1 is updated.

CONSTA -- XN

This subroutine controls the writing of the constants on TXTIN.

OUT -- XN

The subroutine causes the constants to be moved from the input area to the output buffer area. Additional control functions are required for strings greater than the buffer area.

PUTCO -- XO

This subroutine causes the contents of the output buffer which contains constants to be written on TXTIN.

SKIPRE -- XO

Input pointer INPT is increased by buffer length and set to the next key in PRETAB.

PRETEND -- XO

The end key X'FFF' of the constant table is moved into the output buffer and the last record of constants is written on TXTIN.

SYMTAB Scan -- XP through XY

Symbol table SYMTAB is consecutively read into input buffers BUFF1 and BUFF2 and processed as follows:

1. Variables, for which no storage will be allocated in STATIC or AUTOMATIC storage, are skipped.

2. Variables for which storage must be allocated cause the following information to be moved into the output buffer:

   a. Internal name of the variable. This name is contained in bytes 2 and 3 of the SYMTAB entry.
   b. Length of the variable. In case of a character string, the length is contained in byte 8. Otherwise in byte 4 (bits 4 through 7). For structures, the length is contained in bytes 12 and 13. For arrays, the length of the array element must be multiplied by the number of array elements.
   c. Block and level number.
   d. Special attributes required during storage allocation.

3. If an end-of-block key is encountered, a new record must be read because the next SYMTAB entry is in the new record.

4. When the end key of SYMTAB is encountered, the TXTOUT tape is rewound and the 6-byte entries are read into BUFF1 and written on the work file as table DSTAB.

INITIAL3 -- XQ

This subroutine causes the parameters required to perform the SYMTAB scan to be set to their initial values.

GETSYM -- XQ

This subroutine causes SYMTAB to be read into the input buffers as determined by parameter REC.

INCR -- XQ

Input pointer INPT is increased by the length of the SYMTAB entry. If the value of INPT exceeds the buffer length, the contents of BUFF2 are moved to BUFF1 and the next SYMTAB record is read.

ALLOC -- XR

The subroutine tests the SYMTAB entry. If it is a variable for which storage must be allocated in STATIC or AUTOMATIC storage, the subroutine ALLVAR is called. Otherwise, the entry is skipped.

ALLVAR -- XS

This subroutine generates a DSTAB entry for the variables in SYMTAB that require storage to be allocated. For the information contained in a DSTAB entry, refer to the introductory paragraphs of this section.

DSEND -- XT

This subroutine causes (1) the end-of-DSTAB key (X'FFFF') to be moved into the output buffer and (2) the last DSTAB record to be written on TXTOUT.

PUTNV -- XT

This subroutine reads DSTAB from TXTOUT into the input buffer and causes the entries to be written out on the work file.

Sorting Constants -- XU through XY

All constants that are included in the text string from TXTIN are sorted in the order described in the introductory paragraphs of this section. For the format of the contents and the type of information they contain, refer to Phase Input and Output. The overall functions for the sorting of constants are as follows:

1. The TXTIN tape is rewound and the table of constants is scanned for constants that cannot be optimized and for address constants. The constants are read consecutively from TXTIN into BUFF1 and BUFF2; the attribute byte is tested and, if the constant cannot be optimized, moved into the output buffer (see subroutine OUTPUT) from which blocks of sorted constants are written on the work file. If the attribute byte indicates that the constant can be optimized, control is transferred to the subroutine SKIP in order to skip the constant by increasing the value of INPT accordingly.

2. When the end-of-constant key is encountered, the TXTIN tape is rewound and the table of constants is scanned for optimizable address constants.

3. This procedure of rewinding the TXTIN tape and scanning the table of constants for a specific type of constants is repeated until all constants have been written on the work file in the desired order.

Flow charts XU and XV show the main functions to be performed to properly sort the constants.

INITIAL4 -- XW

This subroutine causes (1) the buffer addresses to be calculated and (2) the parameters required to write out the sorted constants on the work file to be set to their initial values.

GETIN -- XW

This subroutine causes (1) the TXTIN tape to be rewound to the beginning of the first constant by means of a POINT macro and (2) the subroutine GECON to be called.

GECON -- XW

The input buffers are filled as determined by the value of REC.

OUTPUT -- XX

This subroutine tests the constant to determine if the DELETE bit is set. If this bit is set, the constant is skipped. If the delete bit is not set, the subroutine causes the constant to be moved into the output buffer and OPT1 to be increased by the length of the constant. If the value of OPT1 is greater than the buffer length, the contents of the output buffer are written out on the work file.

COUT -- XX

This subroutine writes the contents of the output buffers on the work output buffers to be written on the work file.

SKIP -- XY

INPT is increased by the length of the constant. If its value is greater than the buffer length, the next record is read from TXTIN.

CONEND -- XY

The end key for the constant table CONTAB is moved into the output buffer and the records in the output buffer that have not yet been written out are written on the work file.

During phase PL/I F25, the constants were sorted into the table CONTAB in this order:

1. Constants and address constants that are not optimizable.
2. Optimizable address constants.
3. Optimizable 8-byte constants.
4. Optimizable 4-byte constants.
5. Optimizable 2-byte constants.
6. Other optimizable constants.

Constants that are not optimizable are allocated storage in the order as they are written in the table.

Constants are compared for identity. If two constants are equal, storage is allocated only to one constant and an equate entry is made for the other. An equate entry has following format:

Byte (s)	Contents
0-1	Name of equated constant.
2-3	Name of based constant (storage allocated).
4-5	Modifier of based constant.

The modifier indicates an offset as shown by the example below:

```
Constant (A) :      DC C'ABCDEFGH'
Constant (B) :      DC C'EFGH'
```

Constant B is contained in constant A. In this case, constant B is flagged by the DELETE bit and an equate entry B EQU A+4 is generated.

The constants are read from the work file into the table space. If the constants do not fit into the table space, the constants are optimized only within the limits of this area.

The equate entries are moved into the output buffers and are written on TXTOUT. After all constants contained in the table area have been tested for identity, these constants are written on TXTIN following the program string. The next part of CONTAB is read into the table space and processed as described above. This procedure is repeated until the end of CONTAB is reached.

All equate entries that have been generated are read from TXTOUT into the input buffers and written as table CONEQU on the work file.

Those constants which were written out on TXTIN are (1) read into the input buffers, (2) moved into the output buffers if they do not have a DELETE bit, and (3) written on the work file as new constant table CONTAB.

Phase Input and Output

The input for this phase are the constants contained in the table CONTAB on the work file. These constants are sorted in the order as described in phase F25.

The output consists of the following:

1. Equate table CONEQU on the work file. This table contains 6-byte entries as follows:

Byte (s)	Contents
0-1	Name of equated constant.
2-3	Name of based constant (for which storage is allocated).
4-5	Modifier of based constant.

2. Constant table CONTAB on the work file containing only those constants for which storage must be allocated. The format of the constants in this new CONTAB is the same as for the constants in the input table CONSTA.

STORAGE AREAS

The subsequent paragraphs describe the storage areas used during this phase. Figure 1 shows the layout of these areas. The following symbols are used:

TS	Begin-address of table-space area
TS1	TS + one buffer-length
BUFF1	Begin-address of buffer area
TS8	TS + 12 buffer-lengths
TSE	TS + 13 buffer-lengths (end of table-space area
OBUF1	Begin-address of first of two output buffers (same as TSE)

The constant table CONTAB is read from the work file into the table space TS up to TSE.

During initialization, INPT1 is set to point to the first constant, i.e., to TS.

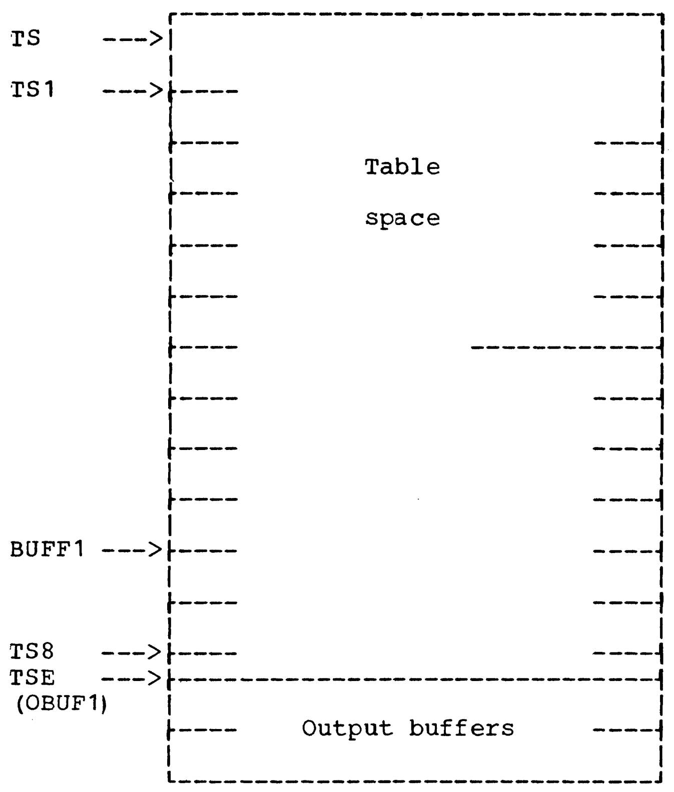


Figure 1. Layout of Storage Areas

The constants in the table-space area are then compared with each other. If identical constants are found, an Equate entry is generated and written on TXTOUT. If all constants in the table-space area are optimized, these constants are written on TXTIN following the end of the program string. The constants between TS8 and TSE are moved to TS, INPT1 is updated and, if not all of the constants have yet been processed, the table-space area is filled with the following part of CONTAB beginning at TS. This procedure is repeated until all constants have been optimized.

DESCRIPTION OF ROUTINES

Symbols used in flow charts:

LCON :	Length of table CONTAB
INPT :	input pointer
INPO :	input pointer
REPO1:	Record counter
REPO2:	Record counter
OPT :	Output pointer
TS0 :	Pointer in table space
TS1 :	Pointer in table space

INITIAL -- YB

This subroutine calculates the addresses of the input and output buffers and sets the pointers to their initial values.

FILL -- YB

This subroutine reads from the work file into the table space. This causes eight or nine buffers to be filled with constants from CONTAB.

SAVE -- YC

INPT2 is set to INPT1 + 5 + the length of the constant; thus, INPT2 points to the address of the next constant. This address is stored.

TEST -- YC

This subroutine determines the type of the constant INPT1 points to and transfers control to the appropriate subroutine.

NOTOPT -- YD

This subroutine scans the table space for optimizable blocks of constants. If two blocks of constants compare equal, an equate entry is generated.

ADCON -- YE

This subroutine scans the table space for identical address constants. Two address constants are equal if (1) they have identical types and (2) they are identical in bytes 5 through 9. In this case, the equate information is moved into the output buffer and written on TXTOUT when the output buffer is full.

OPTIM8 -- YF

This subroutine compares the base constant with all constants that have a length of eight bytes or less. The base constant is pointed to by INPT1. Constants of a length of two, four or eight bytes are compared only for identical boundary alignment. The compare operation is terminated when the end of CONTAB or the end of the table space is reached.

OPTIM4 -- YG

Constants of a length of less than five bytes are compared with the base constant. The base constant, which is pointed to by INPT2, has a length of four bytes. Constants of a length of two bytes are compared only for identical boundary alignment. Also, the constants are compared only within the limits of the table space. If an identical constant is found, control is transferred to the subroutine EQU.

MOD -- YG

This subroutine updates the input pointer INPTO and the modifier MODIF.

OPTIM2 -- YH

When this subroutine is entered, INPT1 points to a 2-byte base constant. INPT2 points to the constants the base constant is compared with.

If a subsequent constant has a length of two bytes or only one, this constant is compared with the base constant. If the two constants compare equal, a DELETE bit is set and an equate entry is moved into the output buffer.

The compare operation is terminated when the end of the table space or the end of CONTAB is reached.

OPTIM -- YI

Constants of a length other than eight, four, or two bytes are compared with each other.

Two constants are considered to be equal if they are identical in length and if their internal representations are the same. In this case, an equate entry is moved into the output buffer and control is transferred to the subroutine EQU.

The compare operation is terminated when the end of CONTAB or the end of the table space is reached.

EQU -- YJ

The subroutine causes the equate information, i.e., the name of the equated constant, the name of the base constant, and the modifier, to be moved into the output buffers. The output pointer OPT is increased by 6 and, if the value of OPT exceeds the buffer length, control is transferred to the subroutine.

EQUOUT -- YJ

The contents of OBUF1 are written on TXTOUT. OBUF2 is moved to OBUF1 and OPT is updated.

CONOUT -- YK

This subroutine causes the constants contained in the table space to be written on TXTIN following the end of the program string.

EQUSR -- YK

The TXTOUT tape is rewound and the equate entries are read from TXTOUT into the input buffer and then written on the work file as table CONEQU.

CONSCR -- YL

This subroutine causes the TXTIN tape to be positioned to the beginning of constants following the program string. The constants are read into the input buffers and tested for DELETE bits. Constants without a DELETE bit are moved to the output buffers and written on the work file. Constants with a DELETE bit are skipped.

SKIP -- YL

The input pointer INPT is increased by the length of the constant. If the value of INPT exceeds the buffer length, BUFF2 is moved to BUFF1. The next block of records is read from TXTIN and INPT is updated.

PUTOUT -- YM

The subroutine causes the output pointer OPT to be increased by the length of the constant. If the value of OPT exceeds the buffer length, the contents of OBUF1 are written on the work file. The contents of OBUF2 are moved to OBUF1 and OPT is updated.

PUT -- YM

This subroutine causes the records contained in OBUF2 and not yet written out to be written on the work file.

GETCON -- YN

This subroutine causes constants from CONTAB to be read into the table space.

PHASE PL/IF75 (STORAGE ALLOCATION) -- Y0

Storage to be allocated may be STATIC or AUTOMATIC. This section describes how STATIC or AUTOMATIC storage is arranged and how these types of storage are allocated.

Static Storage

At object time, STATIC data is arranged as shown in Figure 1.

Not Optimizable Constants
Optimizable Constants
Simple Variables
Arrays
Structures

Figure 1. Arrangement of STATIC Storage

Automatic Storage

Each invocation of a procedure or BEGIN-END block at object time requires a DSA (Dynamic Storage Area). The DSA consists of a block of storage aligned at a double-word boundary. The size of DSA is determined by the size of its fields. Figure 2 shows the arrangement of a DSA.

Fixed Area
Parameters
Variables and Work Storage
Arrays
Structures

Figure 2. Arrangement of a DSA

The fixed area contains the length of DSA and a register save area. The length of the fixed area is always 88 bytes.

Storage Allocation

Storage is said to be allocated for a variable or a constant when a certain region of storage is assigned to the variable (or constant).

Storage Allocation for Constants from CONTAB

The constant table CONTAB (for generation see phase F35) is consecutively read into the input buffers. The location counter for STATIC storage (LCST) is set to 0 before the allocation of storage is started for the first string of data from CONTAB.

For each constant, an entry for the offset table OFFTAB1 is generated and moved into the ouptut buffer. The format of an OFFTAB1 entry is as follows:

Byte(s)	Contents
0-1	Internal name of constant.
2-3	Offset (= location counter LCST).
4	Attribute byte (for constants, this byte contains only the STATIC bit, i.e., bit 2 = 1).

Before the value of the location counter is moved into the output buffer, this value is set to a specific boundary, if required. Following the move operation, the location counter is increased by the length of constant and the OFFTAB1 entry for the next constant is generated. This procedure is repeated until one OFFTAB1 entry has been generated for all constants in CONTAB. The generated OFFTAB1 entries are written on TXTOUT as the output buffer is filled.

Storage Allocation for Character-String Constants in CARTAB

These constants, which are contained in the work file, are chained together and handled as one constant. The name of this constant is a reserved name and has the internal representation 3. The length of CARTAB is contained in the interphase communication region.

To allocate storage for these constants, an OFFTAB1 entry is moved into the output buffer and the location counter is increased by the length contained in the communication region.

Storage Allocation for STATIC and AUTOMATIC Variables

The table DSTAB contains entries for the STATIC and AUTOMATIC variables. These entries are not sorted. In order to allocate the variables in the order as described above, several passes through DSTAB are required. In the first pass, storage is allocated for parameters; passes 2 through 5 allocate storage for 8-byte, 4-byte, 2-byte, and other scalar variables. Pass 6 allocates storage for arrays and pass 7 for structures.

The number of procedures or BEGIN-END blocks in the compilation was stored in the interphase communication region during phase A50.

At the outset of storage allocation, location counters are provided for each block. These location counters (LC1 through LCn) are initially set to 88, which is the length of the fixed area in a DSA. During the first pass, DSTAB entries are consecutively read into the input buffers. For each parameter encountered, an OFFTAB1 entry is generated as follows:

Byte (s)	Contents
0-1	Internal name of parameter
2-3	Offset (value of location counter for the block)
4	Attributes:

Bit	Indication
0-1	Level number
2	1 = STATIC, 0 = AUTOMATIC
3-7	Set to 0.

The information contained in bytes 0, 1, and 5 is taken from the DSTAB entry. In addition, the DSTAB entry contains the block number used to find the proper location counter.

After the OFFTAB1 entry has been moved into the output buffer, the value of the location counter is increased by the length of the parameter and the next variable is tested for being a parameter. This procedure is continued until the end of DSTAB is reached. The generated OFFTAB1 entries are written on TXTOUT as the output buffer is filled.

During the second pass, the DSTAB is scanned for 8-byte variables. Each 8-byte variable encountered is tested to determine whether it is STATIC or AUTOMATIC. If it is STATIC, the value of the location counter LCST for STATIC storage is moved into the OFFTAB1 entry; otherwise, the value of the location counter for the associated block is moved into the OFFTAB1 entry.

Before the value of the location counter is moved into the output buffer, this value is set to the next double-word boundary, if necessary.

The format of an OFFTAB1 entry for an 8-byte variable is the same as for an OFFTAB1 entry for a parameter.

After the OFFTAB1 entry has been moved into the output buffer, the value of the location counter is increased by the length of the variable and the next DSTAB entry is tested.

When the end of DSTAB is reached, the next pass is started to allocate storage for the 4-byte variables.

The remaining passes handle storage allocation in the same way as the second pass. Each time, all DSTAB entries are tested for the desired type of variable, and the OFFTAB1 entries for variables, arrays, or structures are moved into the output buffer and written on TXTOUT.

Phase Input and Output

Input:

1. Constants
   The constant table CONTAB, which was written on the work file during the preceding phase. CONTAB contains all declared and generated constants, except user-defined character-string constants. CONTAB is used to allocate storage for the constants. The format of CONTAB is described in phase F25.

2. Character String
   The user-defined character-string constants were gathered in one string and written on the work file as table CARTAB during a phase A45. For storage allocation, only the length of the character string is required. The length is stored in the interphase communication region.

3. Variables
   During phase F25, the table DSTAB was built up. This table contains one entry for each variable for which storage is to be allocated in STATIC or AUTOMATIC storage. For a description of this table see the section describing phase F25.

Output:

1. Offset table OFFTAB1 on TXTOUT. The format of OFFTAB1 entries has already been described.

2. Block table BLTAB on the work file. This table contains the lengths of DSA's aligned at double-word boundary.

DESCRIPTION OF ROUTINES

Flow chart ZA shows the main functions for the allocation of storage for constants; flow chart ZH for the allocation of storage for variables.

Symbols used in flow charts:

INPT	input pointer
OPT	output pointer
BUFFL	buffer length
LCST	location counter for static storage
LOCO	location counter for static storage
REC	number of work file records requested

Storage Allocation for Constants -- ZA-ZG

CONTAB is read consecutively into input buffers BUFF1 and BUFF2. At the beginning, input pointer INPT points to the first constant. The location counter LCST for STATIC storage is set to 0.

For each constant, an OFFTAB1 entry is generated and moved into the ouput buffer. The value of the location counter LCST, the name of the constant, and the STATIC bit are moved into the output buffer. The value of the location counter may have to be set to the required boundary before the move operation is executed. The value of the location counter is increased by the length of the constant and the next constant is processed. The OFFTAB1 entries are written out on TXTOUT. When the end of CONTAB is reached, an OFFTAB1 entry is produced for CARTAB. The length of CARTAB is contained in the communication region.

INITIAL1 -- ZB

The addresses of the input and output buffers are calculated and the pointers are set to 0. Bit 2 in the TABTAB entry for CONTAB is set to 0 to start reading the CONTAB entries.

CONTAB -- ZB

This subroutine causes one or two records of CONTAB to be read from the work file into the input buffer(s).

OFFTAB1 -- ZC

This subroutine causes the value of the location counter, the internal name of the constant, and the STATIC bit to be moved into the output buffer. Control is then transferred to the subroutine OFFOUT to write the OFFTAB1 entries on TXTOUT. The value of the location counter is increased by the length of the constant before control is returned.

ALIGN -- ZC

Before the value of the location counter is moved into the output buffer, the constant is tested to determine if it requires a boundary. This subroutine tests the constant for the required alignment.

LOCAL -- ZD

The subroutine causes the value of the location counter LOCO to be set to the boundary specified by AL8, AL4, or AL2.

LINCR -- ZD

The subroutine LINCR determines if the constant is an address constant of a length of three bytes. If it is, the length specified in bytes 3 and 4 of the constant is changed to 3 because the location counter is to be increased by 3.

OFFOUT -- ZE

OPT is increased by 5. If its value exceeds the buffer length, the contents of OBUF1 are written on TXTOUT. The contents of OBUF2 are moved to OBUF1 and OPT is updated.

SKIPC -- ZF

The input pointer INPT is increased by the length of the constant. If the value of INPT exceeds the buffer length, the contents of the input buffer BUF2 is moved to BUF1 and the next CONTAB record is read into BUFF2. INPT is updated.

CARTAB -- ZG

This subroutine generates an OFFTAB1 entry for the character string table CARTAB. The value of the location counter is increased by the length of CARTAB (obtained from the interphase communication region).

RERRTEST -- ZG

The location counter is tested and, if its value is greater than or equal to 64K-1, an error bit is set in the compiler communication region in the field IJKMWC and the diagnostic phase G31 is called.

Storage Allocation for Variables -- ZH-ZL

Seven passes of scanning the DSTAB entries are required to properly sort the variables and to allocate storage to them.

When the desired type of variable is encountered, control is transferred to OFFSET. That subroutine causes an appropriate OFFTAB1 entry to be generated and written on TXTOUT.

INITIAL -- ZI

A 2-byte location counter for each procedure or BEGIN-END block is reserved in the table space. These location counters are set to their initial values, i.e., the length of the fixed area in the DSA (19 words).

PICK -- ZI

This subroutine initiates the reading of the data table DSTAB from the work file. The input pointer INPT is set to 0.

DSTAB -- ZI

The subroutine causes (1) the data table DSTAB to be read into the input buffers and (2) INPT to be updated.

OFFSET -- ZJ

This subroutine causes the offset value, the internal name, and the attributes to be moved a into the output buffer. In addition, this subroutine uses various other subroutines to complete the generation of and the writing out of the OFFTAB1 entry.

GETLOC -- ZJ

For an AUTOMATIC variable, the appropriate location-counter value is taken from the table space and moved to LOCO. For a STATIC variable, the value in location counter LCST is moved to LOCO.

PUTLOC -- ZK

For an AUTOMATIC variable, the value in LOCO is returned to the table space BLTAB of the block concerned. For a STATIC variable, LOCO is returned to LCST.

LEFTH -- ZK

The appropriate location counter value is moved to LOCO using the subroutine GETLOC. The value in LOCO is then adjusted as determined by the lefthang of structure.

SKIPV -- ZK

This subroutine skips to the next DSTAB entry by increasing INPT. If the value in INPT becomes greater than the buffer length, the subroutine DSTAB is called to read the next record.

LAST -- ZL

The end-of-OFFTAB1 key (X'FFFF') is moved into the output buffer, and offset table entries not yet written out are written on TXTOUT.

BLTAB -- ZL

When this subroutine is entered, the location counters in TS contain the lengths of the generated DSA's. These lengths are adjusted to double-word boundaries and written on the work file as table BLOCK1. The length of STATIC storage (in LCST) is stored in the compiler communication region.

PHASE PL/IF90 (BUILDING OF OFFSET TABLE) -- AA

This phase builds up the final offset table OFFTAB which contains all offsets of data in the following form:

bytes	0 - 1	offset
byte	2	attributes

The entries are sorted in ascending order of their interal names.

Phase Input and Output

The input used by this phase is

symbol table SYMTAB on SYS001
equate table CONEQU on SYS001
offset table OFFTAB1 (built up and described in phase F75) on TXTOUT

Output is the final offset table OFFTAB on SYS001 or in storage. It contains all offsets of data in ascending order of their internal names.

Switches

Switches are located in byte WSWITCH:

```
bits 0 - 5 not used
bit  6 = 0 switch CON off: SYMTAB must be
                           retrieved
       = 1 switch CON on : CONEQU must be
                           retrieved
bit  7 = 0 switch TXT off: EQUTAB is on
                           SYSS001
       = 1 switch TXT on : EQUTAB is on
                           TXTIN.
```

I/O Concept

Two adjacent buffers, referred to as buffers A and B, are used as I/O buffers. The beginning of a table is read into both buffers. The individual entries of the table are processed sequentially from left to right (beginning in buffer A). The buffer pointer R3 is increased each time by the entry length until it points to an entry in buffer B. If necessary, buffer A is moved into the output buffer in order to put it onto TXTIN, TXTOUT or SYS001. Buffer B is moved into buffer A and the buffer pointer is reset to the next entry to be processed. The next record will be read into buffer B in overlapped mode.

Communication with Other Phases

The address of the table space is decreased in this phase by the length of TABTAB. The number N2 of records of OFFTAB that can be stored in the area between the new beginning of the table space and the end of the second I/O buffer is stored in IJKMIP+2. If OFFTAB does not exceed N2 records, it remains in storage at the end of this phase and the beginning of OFFTAB is identical to the new beginning of the table space.

FUNCTIONAL DESCRIPTION

This phase performs three main functions: gathering of equates, sorting of offsets and calculating the offsets of equates, and inserting equate offsets into the offset table.

1. Gathering of Equates

Equate table EQUTAB1 is built up. Each entry contains the following information:

bytes 0-1 :	Internal name of defined variable or equated constant
bytes 2-3 :	internal name of based variable or based constant
bytes 4-5 :	modifier (0 if defined variable).

Variables with the attribute DEFINED are handled as equates. SYMTAB is scanned and, if a defined variable is found, an entry in EQUTAB1 is made.

Equate table CONEQU contains all equated constants in the following form:

bytes 0-1 :	Internal name of equated constant.
bytes 2-3 :	internal name of the based constant.
bytes 4-5 :	modifier.

All entries of CONEQU are appended to EQUTAB1. If the source text contains no defined variables, gathering of equates is skipped and CONEQU is used as EQUTAB1.

2. Sorting of Offsets and Calculating the Offsets of Equates

If the offset table entries are sorted in ascending order of their internal names, no internal name is required within the entry since an entry in the final offset table OFFTAB can be found by using the internal name as relative address in the offset table.

Offset table OFFTAB1 contains all entries in unsorted order. An entry consists of:

bytes 0-1	internal name
bytes 2-3	offset
byte 4	attributes

The format of the entries in the final offset tabel OFFTAB is as follows:

bytes 0-1	offset
byte 2	attributes

The area in which OFFTAB is built up is called work area. The number of OFFTAB entries that fit into the work area is called M. MIN is equal to the smallest internal name of an entry that fits into the work area; MAX is equal to the internal name greater than the greatest internal name of an entry that fits into the work area.

Sorting of offsets starts with MIN=0 and MAX=M. OFFTAB1 is read successively into the input buffers. The offsets and attributes of all entries of OFFTAB1 whose internal names are greater than or equal to MIN and less than MAX are stored in the work area in ascending order of their internal names. After scanning of OFFTAB1, EQUTAB1 is read successively into the input buffers. Entries of EQUTAB1 with internal names of based data greater than or equal to MIN and less than MAX are processed, e.g., the internal names of based data of these entries are replaced by their offsets retrieved from the work area. The modifier is added to each retrieved offset. The first byte of the modifier is replaced by the attributes of the based data. If EQUTAB1 is read from TXTIN, the processed EQUTAB1 is written onto SYS001 and vice versa.

The offset table built up in the work area is named OFFTAB2. The entries and length of OFFTAB2 are the same as those of OFFTAB. The two tables differ in that OFFTAB2 contains gaps to be replaced by the entries of equated data in OFFTAB. If OFFTAB2 is completely stored in the work area, the offsets are stored and the offsets of equates are calculated in one pass. Otherwise, the part of OFFTAB2 that is in the work area is written on TXTOUT. MIN and MAX are increased by M and sorting of offsets and calculating the offsets of equates is continued until OFFTAB2 is completely on TXTOUT.

The processed equate table is named EQUTAB. The EQUTAB entries have the following format:

bytes 0-1	: internal name of equated data
bytes 2-3	: offset of equated data
byte 4	: attributes of equated data (equal to attributes of based data)
byte 5	: not used.

3. Inserting Equate Offsets into the Offset Table

If all entries of OFFTAB2 are stored in the work area, EQUTAB is successively read into the input buffers. The offsets and attributes of the equated data are inserted into the work area according to their internal names.

If OFFTAB is greater than the work area, inserting of equate offsets starts with MIN=0 and MAX=M. The records of OFFTAB2 that contain entries of internal names greater than or equal to MIN and less than MAX are read into the work area. EQUTAB is successively read into the input buffers. The offsets and attributes of equated data whose internal names are greater than or equal to MIN and less than MAX are inserted into the work area according to their internal names. If all entries of EQUTAB whose internal names are not less than MIN or greater than MAX have been inserted, the part of OFFTAB that is in the work area is written onto SYS001. MIN and MAX are increased by M and the insertion of equate offsets is continued until OFFTAB is completely on SYS001.

DESCRIPTION OF ROUTINES

Initialization and Gathering of Equates -- AB, AC

Buffer pointer R3 is set to the beginning of buffer A and table space pointer R8 is set to the beginning of the table space.

If the source program contains defined variables, SYMTAB is scanned. If a defined variable is found, an entry in EQUTAB1 is made. SYMTAB is scanned in buffers A and B. EQUTAB1 is built up in the table space and written onto TXTIN.

If scanning of SYMTAB is finished, switch CON is set. CONEQU is read into buffers A and B, and all entries of it are added to EQUTAB1. If EQUTAB1 has been completed and written onto TXTIN, switch TXT is set to indicate the presence of EQUTAB1 on TXTIN.

Construction of Work Area -- AD

The space consisting of parts of the phase (beginning with WBEG2), the table space, and the first three buffers is used as work area to build up the offset tables.

If the length of the work area divided by 3 is not less than the number of offsets, the entire offset table can be placed in the work area. Otherwise, the offset tables must be built up in several passes. Between these passes, the parts of an offset table that have been processed are written onto TXTOUT or SYS001 in the length of multiples of the buffer length. To avoid gaps in the offset tables, the length of the work area actually used must be three times the buffer length times floor of length of the work area divided by three times the buffer length.

The number of entries of the offset table that can be placed in the work area is called M.

Sorting of Offsets (TXTOUT) -- AE

OFFTAB1 is read from TXTOUT into the input buffers. The offset and attributes of each OFFTAB1 entry whose internal names are less than M are stored in the work area according to the internal name. The table built up in the work area is called OFFTAB2. If there are entries of OFFTAB1 with internal names equal to or greater than M, OFFTAB1 is written onto SYS001.

Calculating Offsets of Equates -- AF

EQUTAB1 is on TXTIN if switch TXT is on; otherwise, it is on SYS001. EQUTAB1 is read into the input buffers. If the entry of the based data of an EQUTAB1 entry is in the work area, the based data is replaced by its offset. Its modifier is added to the offset and the first byte of the modifier is replaced by the attributes of the based data.

After processing, EQUTAB1 is written onto SYS001 if switch TXT is on; otherwise, it is written onto TXTIN. Switch TXT is altered.

Sorting of Offsets (SYS001) -- AG

The part of OFFTAB2 which is built up in the work area is written onto TXTOUT if OFFTAB2 is not completely contained in the work area. If OFFTAB2 is completely on TXTOUT and EQUTAB is on SYS001, EQUTAB is written onto TXTIN.

If the construction of OFFTAB2 is not finished, MIN and MAX are increased by M and OFFTAB1 is read from SYS001 into the input buffers. The offset and attributes of each OFFTAB1 entry whose internal name is greater than or equal to MIN and less than MAX are stored in the work area according to its internal name to continue the construction of OFFTAB2.

Inserting of Offsets of Equates -- AH

EQUTAB, which contains the internal name, offset, and attributes of all equated data, is read into the input buffers. If OFFTAB2 is completely in the work area, the offset and attributes of each entry of EQUTAB are stored in the work area according to its internal name.

If OFFTAB2 is not completely in the work area, the insertion of offsets of equated data is started with MIN=0 and MAX=M. The records of OFFTAB2 which contain the offsets of internal names greater than or equal to MIN and less than MAX are read into the work area. The offset and attributes of each EQUTAB entry whose internal name is greater than or equal to MIN and less than MAX are stored in the work area according to its internal name. If EQUTAB has been completely scanned, the records of OFFTAB that are in the work area are written onto SYS001. If the construction of OFFTAB is not finished, MAX and MIN are increased by M and the insertion of offsets of equates is continued by scanning EQUTAB again.

End-of-Buffer Routine -- AI

Entry point: WSR21

Input parameters:
R3 points to the element to be scanned in the input buffers.
R6 contains the address of buffer A.
R7 contains the address of buffer B.

If pointer R3 points to an element in buffer B, buffer B is moved into buffer A, buffer pointer R3 is reset, and the routine is left through exit B. Otherwise, the routine is left through exit A.

Move Buffer B to Buffer A -- AI

Entry point: WSR21A

The function is the same as described for the routine WSR21, but without testing whether pointer R3 points to an element in buffer B.

Put End of Table on SYS001 -- AJ

Entry point: WSR51

Input parameters:
R3 points to the beginning of the end key of a table in buffer A.
R7 contains the add ress of buffer B.

It is tested whether the entire end key of a table is in buffer A. If not, buffer B is written onto SYS001 and the routine is left through exit B. Otherwise, the routine is left through exit A.

Work up an Entry of OFFTAB1 -- AK

Entry point: WSR11

Input parameter:
R3 points to the entry of OFFTAB1 being scanned.

If the internal name of the entry pointed to by R3 is not less than MIN and less than MAX, the offset and the attributes of this entry are stored in the work area according to its internal name.

Output parameter:
R3 points to the second byte of the OFFTAB1 entry following the scanned entry.

Work up an Entry of EQUTAB -- AK

Entry point: WSR61

Input parameters:
R2 contains the internal name of the EQUTAB entry being scanned.
R3 points to the EQUTAB entry scanned.

If the internal name in R2 is not less than MIN and less than MAX, the offset and the attributes of the entry pointed to by R3 are stored in the work area according to its internal name.

Output parameter:
R3 points to the EQUTAB entry following the scanned entry.

Work up an Entry of EQUTAB1 -- AK

Entry point: WSR31

Input parameter:
R3 points to the EQUTAB1 entry scanned.

If the internal name of the based data of the entry pointed to by R3 is not less than MIN and less than MAX, the offset of the equated data is calculated by the offset of the based data and the modifier. The based data is replaced by the offset of the equated data, and the first byte of the modifier is replaced by the attributes of the based data.

Output parameter:
R3 points to the EQUTAB1 entry following the scanned entry.

Compare Internal Name with MIN and MAX -- AK

Entry point: WSRE1

Input parameters:
R2 : internal name;
R4 : address of the table space;
R9 : MIN;
R10 : MAX.

If the internal name is less than MIN or not less than MAX, the routine is left through exit A. Otherwise, the allocation of the entry in the work area is calculated by the internal name and the routine is left through exit B.

Output parameter:
R2 contains the address of the internal name in the work area, if the routine is left through exit B.

Calculate end of Filled Work Area -- AL

Entry point: WSRB1

Input parameters:
R4 : address of the work area;
R9 : MIN;
R11 : K = maximum number of OFFTAB2 or OFFTAB entries.

The address of the end of the filled work area is calculated.

Output parameter:
R3 contains the address of the end of the filled work area.

Put Work Area on TXTOUT -- AL

Entry point: WSRC1

Input parameters:
R3 contains the address of the end of the filled work area.
R4 contains the address of the work area.

The contents of the filled work area are written onto TXTOUT.

Put Work Area on SYS001 -- AL

Entry point: WSRD1

Input parameters:
R3 contains the address of the end of the filled work area.
R4 contains the address of the work area.

The contents of the filled work area are written onto SYS001.

Set Pointer and Clear Work Area -- AL

Entry point: WSR41

Input parameters:
R4 contains the address of the work area.
R6 contains the address of buffer A.
R8 contains M.

Buffer pointer R3 is set to the beginning of buffer A, and the work area is filled with zeros.

Output parameter:
R3 points to the beginning of buffer A.

Work up an Entry of SYMTAB -- AJ

Entry point: WSR22

Input parameters:
R3 points to the SYMTAB entry being scanned.
R8 points to the location in the table space where the next entry of EQUTAB1 is stored.

If the SYMTAB entry contains a defined variable, the internal names of the defined and the based variable are moved into the entry in the table space pointed to by R8.

Output parameters:
R3 points to the entry of SYMTAB that follows the one being checked.
R8 points to the next EQUTAB1 entry.

Write Record of Table Space on TXTIN -- AJ

Entry point: WSR12

Input parameters:
R4 contains the address of the beginning of the table space.
R8 points to the location in the table space where the next EQUTAB1 entry is stored.
R11 contains the address of the end of an EQUTAB1 record in the table space.

If R8 points to an address not less than that of the end of an EQUTAB1 record, a record of EQUTAB1 is written onto TXTIN and the remaining bytes of EQUTAB1 are moved to the beginning of the table space. Table space pointer R8 is reset.

Fill Begin of Table Space with Zeros -- AJ

Entry point: WSR02

Input parameter:
R8 points to the location of the table space where the next entry of EQUTAB1 is stored.

The area of the table space used to build up EQUTAB1 is filled with zeros starting at the location pointed to by R8.

These phases can be divided into two groups. The first group comprises the phases F95 - G15; the second group comprises the phases G20 - G55. Phase G17 is organized differently and not described here.

PHASES F95 - G15

These phases prepare the program text for final output, i.e., all code and all information required for the TXT and RLD cards has been prepared upon completion of these phases.

Phase F95 generates the code for offsets greater than 4K using the offset table OFFTAB, which contains the offsets of the variables and constants of static and automatic storage. As far as possible, the offsets are inserted into the text string, even if no code generation is required.

Phase G00 optimizes the "maximum" code produced by phase F95. The offsets of the labels are inserted into the label table LABTAB, and all program blocks are divided into segments of 12K length. For all branches within the same segment, the preceding pseudo Assembler instruction ADD (generated in previous phases) is deleted. LABTAB is updated accordingly.

Phase G01 inserts the label offsets into OFFTAB so that the missing offsets may be retrieved from only one table. It also lists OFFTAB if the SYM option was specified in the OPTION job control statement.

In the first part of phase G15, the text string is scanned and the remaining offsets are inserted. In the second part, the format of the constants in static storage is changed from that of the constant table CONTAB to that of the text string, and the offsets for the address constants are calculated. If the source program contains no file declarations, phase G15 transfers control to phase G20 instead of to phase G17.

Input/Output Handling. The handling of input and output is the same for all phases (except phase G17). Storage is divided into four parts:

1. Compiler interface

2. Program space (4K)

3. Table space (TS). This space is used to read OFFTAB from SYS001 and to build up LABTAB.

4. Buffer area. It is used for I/O of the text string.

Actually, there are five buffers in the buffer area. However, only three are used for text I/O so that the first two buffers may be considered as belonging to the table space. The last three buffers are used as follows:

buffer 1:	output buffer (OBUF)
buffer 2:	input buffer 1 (IBUF1)
buffer 3:	input buffer 2 (IBUF2)

The start addresses of these buffers are B0, B1, and B2, respectively. Pointer POU is used in OBUF; pointer POI is used in IBUF1 and IBUF2. The buffer length is referred to as BUFL.

ISU, MOO, MODIF (F95, G00, G15). The text is read into the input buffers and scanned using pointer POI. When POI becomes greater than B2, the record is moved from IBUF2 to IBUF1, POI is adjusted, and a new text record is read into IBU1. This action is performed by the routine ISU.

PHASES G20 - G55

The text of the length L (given in register 1) is moved from the address pointed by POI to the address pointed to by POU by means of the routine MOO. This routine also performs the output if OBUF is full and adjusts all pointers. Reading and writing of the text records is performed by the external routines IJKGI and IKJPO. The external move routine IJKMN is used for move operations. MODIF is used to evaluate the correct modifier.

Table Handling. Each table to be used in several phases has an 8-byte entry in the master table TABTAB. Each TABTAB entry has the following format:

bytes 0-1	flag bytes. The first three bits indicate the following: bit 0 on = table on SYS001 bit 1 on = table in storage bit 2 on = transfer to or from SYS001 has been started.
bytes 2-3	position on SYS001 (key)
bytes 4-5	number of records
bytes 6-7	record length

If OFFTAB, which is built in phase F90, is small enough, it remains in the table space. Otherwise, it is written onto SYS001 (TABTAB entry ZTAB11). Phase F95 checks whether or not OFFTAB is in the table space. If it is not, the first part is read into the table space by means of the external routine ZTIN. The text is scanned and the available offsets are inserted. Then the next part of OFFTAB is read. This process continues until all parts of OFFTAB have been in storage.

If OFFTAB was not on SYS001, it is written onto it by the external routine ZTOUT in order to free the table space for the construction of LABTAB in phase G00.

If LABTAB becomes greater than the table space, it is intermediately written onto ten onto it by the external routine ZTOUT in order to free the table space for the construction of LABTAB in phase G00.

If LABTAB becomes greater than the table space, it is intermediately written onto ten onto it by the external routine ZTOUT in order to free the table space for the construction of LABTAB in phase G00.

If LABTAB becomes greater than the table space, it is intermediately written onto ten onto it by the external routine ZTOUT in order to free the table space for the construction of LABTAB in phase G00.

If LABTAB becomes greater than the table space, it is intermediately written onto ten onto it by the external routine ZTOUT in order to free the table space for the construction of LABTAB in phase G00.

If LABTAB becomes greater than the table space, it is intermediately written onto ten onto it by the external routine ZTOUT in order to free the table space for the construction of LABTAB in phase G00.

If LABTAB becomes greater than the table space, it is intermediately written onto ten onto it by the external routine ZTOUT in order to free the table space for the construction of LABTAB in phase G00.

If LABTAB becomes greater than the table space, it is intermediately written onto SYS001 (TABTAB entry ZTAB19). It is read again into storage for updating. The final LABTAB is written onto SYS001 (TABTAB entry ZTAB20) for use by phase G25.

In phase G01, the label offsets are inserted from LABTAB into OFFTAB. For this purpose, OFFTAB is read into the table space and LABTAB is read (record by record) into the buffer area. The updated OFFTAB remains in the table space unless it becomes too large. In this case, it is written onto SYS001 (TABTAB entry ZTAB07).

In phase G15, OFFTAB is read into the table space in order to insert the missing offsets into the instructions. The text is scanned only once if OFFTAB fits into the table space; otherwise, the text is scanned as many times as parts of OFFTAB have to be read. After this phase, OFFTAB is no longer used.

The constant table CONTAB is also processed in phase G15. CONTAB (TABTAB entry ZTAB08) is read into the buffers record by record. After this phase, CONTAB is no longer used.

As initialization for the following phases, the external name table EXTAB (TABTAB entry ZTAB04) is read from SYS001 at the end of phase G15 and written onto TXTIN, which becomes TXTOUT in the following phases.

PHASES G20 - G55

Phase G20 arranges the different cards for the files to prepare one file module and writes the cards onto SYS001.

Phase G25 generates the ESD cards for the object-program module and writes them onto SYS001.

Phase G30 generates the TXT and RLD cards for the object-program module as well as the END card. The cards are written onto SYS001 (TABTAB entry ZTAB16).

Phase G31 is called if the compilation must be terminated before phase G55 is called. It lists the flagged errors, closes all files, and terminates the compilation by the EOJ macro instruction.

Phase G40 lists the object code.

Phase G55 writes all cards (for both the first and the second module) onto IJSYSLN (in blocks of 322 bytes) if the LINK option is on. If the DECK option is on, it punches the object deck. If the SYM option is on, it lists the external symbol table and the block table.

This phase generates code for operands that have an offset greater than or equal to 4K or 12K, respectively. The code is generated by means of the four general registers 5 - 8 and some constants contained in static storage. Registers 7 and 8 are loaded with 4K and 8K, respectively. Registers 5 and 6 are used for indirect addressing. The constants in static storage are 0K, 12K, 16K, 20K, 24K, and 28K. They are stored in this sequence under the internal name N'0004' (one half-word each).

1. Offset less than 12K

    a. No additional code is required if the offset is less than 4K.

        Note: No code generation is required for RS instructions, since these instructions are never used with offsets greater than or equal to 4K.

    b. For instructions with offsets between 4K and 12K, an additional register is used for addressing. In RX instructions, the index register has not been used so that registers 7 (containing 4K) and 8 (containing 8K) can be used as index registers for these instructions. For instance, in the instruction

        ```
        L  R,NAME1
        ```

        NAME1 is an address with the offset 5000. This instruction is modified to

        ```
        L  R,904(7,9)
        ```

        where 9 is the base register.

        In all other instructions, the base register is increased by 4K or 8K. For example, in the instruction

        ```
        MVC NAME2(4),NAME3
        ```

        NAME2 is an address with the offset 6000 and NAME3 has the offset 10000. The instruction is modified to

        ```
        (ST  6,SAVE)
        LA   6,0(7,9)
        LA   5,0(8,9)
        MVC  1904(4,6),1806(5)
        (L   6,SAVE)
        ```

        The instructions in brackets are required if, for example, register 6 is not free.

2. Offset greater than 12K-1 and less than 32K. In this case, one of the constants 12K, 16K, 20K, 24K, or 28K is loaded into a register as follows:

    ```
    LH 6,=A(16K).
    ```

    This register can then be used as an index register in RX instructions so that no additional code generation is required. In all other cases, an additional ADD instruction for the base register must be generated, and register 6 or 5 is inserted as base register. For instance,

    ```
    AR      6,9
    STM     R,S,50,(6)
    ```

3. Offset greater than 32K-1. One of the constants is loaded into register 5 or 6 and multiplied by 2 by adding the register to itself, e.g.,

    ```
    LH  6,=A(20K)   (20K)
    AR  6,6         (40K)
    ```

    For 36K, 44K, 52K, and 60K, additional 4K must be added, e.g.,

    ```
    AR  6,7         (44K)
    ```

    The following instructions are the same as described under item 2.

Phase Input and Output

The input is read from the text input file and consists of machine instructions, pseudo Assembler instructions, and end key. The format of these instructions is described in the section Instruction Formats. The input is read into the two input buffers B1 and B2. The input pointer is POI.

The offset table OFFTAB is read into the table space. If OFFTAB is small enough, phase F90 has left it in the enlarged table space of the length M*buffer length. If the size of OFFTAB exceeds the table space, OFFTAB is read from SYS001. The table space was enlarged in phase F90 by 180 bytes in low core and two I/O buffers in high core. Thus, M is given by the length of the table space divided by the buffer length. M is stored at IJKMIP+2. The start address of the table space (stored at IJKMTS) was reduced by 180 bytes.

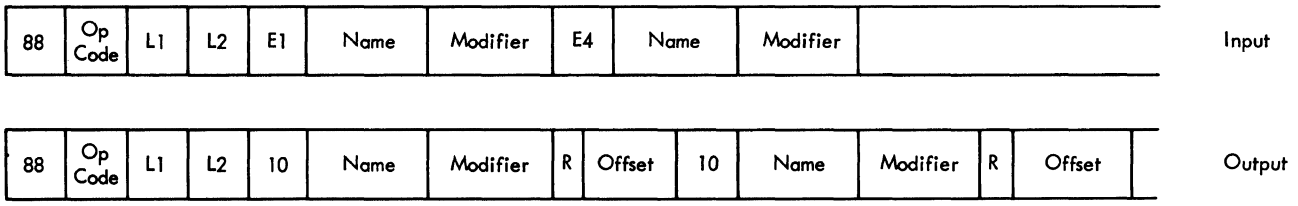


Figure 1. Input and Output of Phase F95

All instructions whose offset has already been determined are modified by inserting a half-word after the modifier. The inserted half-word contains the base register in the first four bits and the offset in the remaining bits. The key is changed to 10.

<u>Note</u>: The pseudo Assembler instruction

DC AL3

is not modified.

The correlation between the input and output of this phase is shown in Figure 1.

NAME and MODIFIER are not deleted because they are still used in phase G40. The output is written on the text output file using the output buffer (OBUF) B0 with the pointer POU. At the end of this phase, OFFTAB is written on SYS001, if it is not already there.

<u>Instruction Formats</u>

The format of the individual machine instructions is shown in Figure 2. The first byte contains the key 88. The second byte contains the operation code. R1, R2, and R3 are registers. L1 and L2 are lengths. I is an immediate constant.

The format of the Assembler pseudo instructions is shown in Figure 3. The first byte contains the key 80.

The end key (01) is used to determine the end of the text.

The keys used in the operands have the following meaning:

E1	Declared variable
E4	Generated variable
E9	Constant
E5	Register
00	Absolute address
11	Label in DC A
10	New key (worked up)
18	New key (worked-up E5)

The first three keys denote entries in OFFTAB, from where the offset is retrieved. The key E5 indicates that the operand consists of a register with a displacement given by the modifier. The key 00 denotes an absolute address. The key 11 denotes a label in a DC A or DC AL3 instruction. The keys 10 and 18 are written if the offset has already been retrieved and inserted into a half-word following the element.

FUNCTIONAL DESCRIPTION

Phase F95 does the following:

1. To retrieve the offset from OFFTAB, the first part of OFFTAB is read into the table space if it is not already there. The offsets of all text operands with entries in this part of OFFTAB are determined. Then, the next part of OFFTAB is read into the table space, and the text is scanned again from the beginning. This process is repeated until all parts of OFFTAB have been in storage.

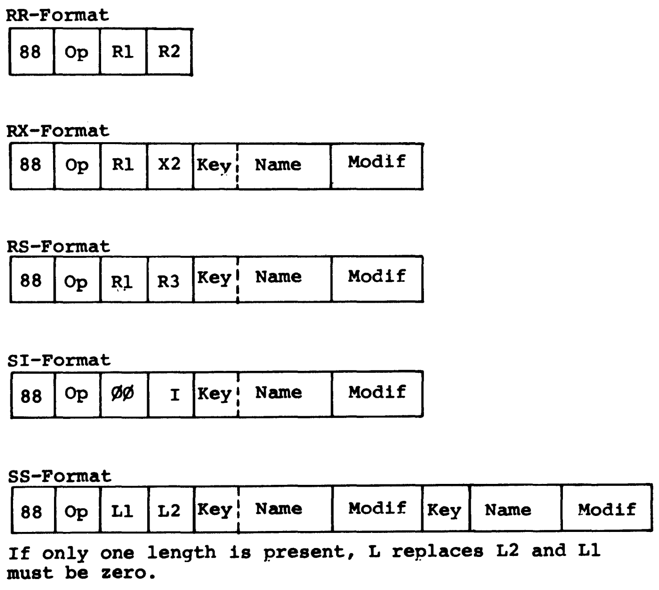
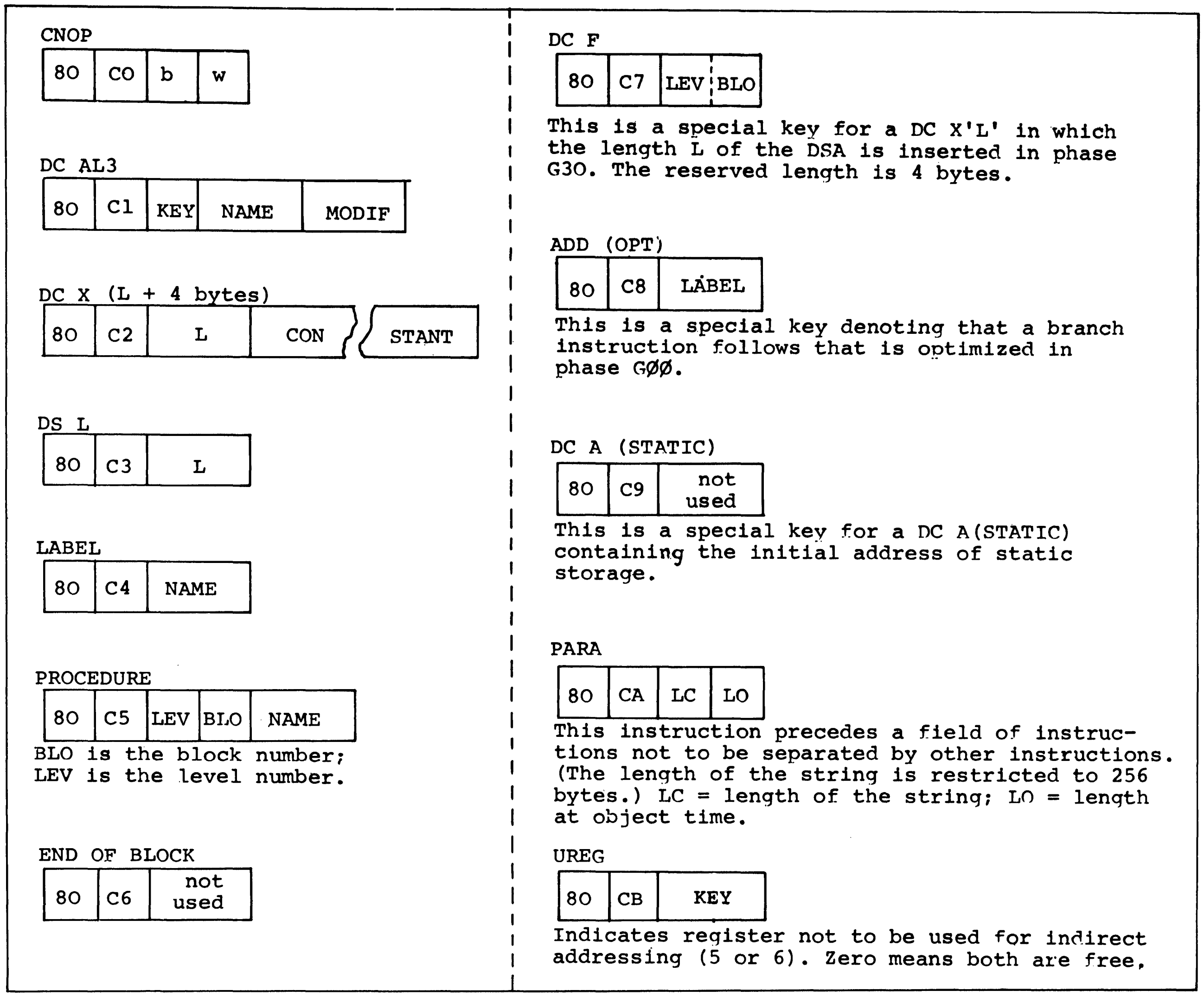


Figure 2. Machine Instruction Formats



BLO is the block number; LEV is the level number.

This is a special key for a DC X'L' in which the length L of the DSA is inserted in phase G30. The reserved length is 4 bytes.

This is a special key denoting that a branch instruction follows that is optimized in phase GØØ.

This is a special key for a DC A(STATIC) containing the initial address of static storage.

This instruction precedes a field of instructions not to be separated by other instructions. (The length of the string is restricted to 256 bytes.) LC = length of the string; LO = length at object time.

Indicates register not to be used for indirect addressing (5 or 6). Zero means both are free.

Figure 3. Assembler Pseudo Instruction Formats

2. To evaluate the base register, it is determined whether the variable is in static or in automatic storage. General register 12 is used for static storage. One of the general registers 13, 11, or 10 is used for automatic storage. Which of these three registers is used depends on the block in which the variable appears and in which it is called (see BAS -- AS).

3. The length of static and automatic storage and of the text string is restricted to 64K.

To get no displacement greater than or equal to 4K, storage is divided into 4K-blocks. Each block is pointed to by the corresponding offset. When code is to be generated, the 4K-block is determined and the address of the block is loaded into a register which is then used as a base or index register.

4. The code to be generated depends (1) on the type of instruction and (2) on the 4K-block pointed to by the offset of the operand. Examples of code generation for blocks with an offset greater than 4K are shown in Figure 4.

Block pointed to by offset	RX instruction	SS or SI instructions
4K 8K	Reg. 7 as index Reg. 8 as index	LA 5,Ø(7,BASE) LA 5,Ø(8,BASE)
12K	LH 5,A(12K) Reg. 5 as index	LH 5,A(12K) AR 5,BASE Reg. 5 as base reg.
16K	LH 5,A(16K) Reg. 5 as index	LH 5,A(16K) AR 5,BASE Reg. 5 as base reg.
2ØK	LH 5,A(2ØK) Reg. 5 as index	LH 5,A(2ØK) AR 5,BASE Reg. 5 as base reg.
24K	LH 5,A(24K) Reg. 5 as index	LH 5,A(24K) AR 5,BASE Reg. 5 as base reg.
28K	LH 5,A(28K) Reg. 5 as index	LH 5,A(28K) AR 5,BASE Reg. 5 as base reg.
32K	LH 5,A(16K) AR 5,5 Reg. 5 as index	LH 5,A(16K) AR 5,5 AR 5,BASE Reg. 5 as base reg.
36K	LH 5,A(16K) AR 5,5 AR 5,7 Reg. 5 as index	LH 5,A(16K) AR 5,5 AR 5,7 AR 5,BASE Reg. 5 as base reg.
4ØK	LH 5,A(2ØK) AR 5,5 Reg. 5 as index	LH 5,A(2ØK) AR 5,5 AR 5,BASE Reg. 5 as base reg.

5 = one of the registers 5 or 6.
A(...) = address of the corresponding constant.
BASE = base register.

Figure 4. Generation of Code for Blocks with Offset greater than 4K

Logical Flow

If the first part of OFFTAB is not yet in the table space, it is read from SYS001. The I/O buffers are filled with text. The text is scanned for entries in this part of OFFTAB, and the corresponding offset and the attribute byte are moved into a special stack. The base register and the 4K-block are determined and the corresponding code is generated. The key and the instruction followed by a half-word containing the base register and the offset (modulo 4K) are then put into the output text string.

When the end key is found, the next part of OFFTAB is read into the table space and the text is scanned again from the beginning. This process is repeated until all parts off OFFTAB have been in storage.

Note: The modifier becomes negative if it is greater than X'FFF8'.

DESCRIPTION OF ROUTINES

Note: The routines ISU, MOO, and MODIF are described in the section General Description of Phases F95 - G55.

Symbols used in flow charts:

BN	Number of OFFTAB records
TS	Table space
M	Number of buffers in TS
B0, B1, B2	Initial I/O buffer addresses
POU	Output pointer
POI	Input pointer
STA	Stack
STAP	Pointer in STA
STAR	Pointer in STA
HW	Half-word
SSS	Switch for SS instructions
BOTH	Switch indicating that R6 has been saved by a store instruction
OFEN	Stack for offset
LEVS	Stack for level
RY	Register containing object-time base register
REWO	1-byte stack indicating the free register
R1, R2, etc.	General registers
OBUF	Output buffer
IBUF	Input buffer

WUP -- AP, AQ

This routine scans the text for end-of-statement, machine instructions, and Assembler instructions.

If an end-of-statement of the format

```
 (1)     (3)             (2)
┌────┬─────────────┬────────────────┐
│ EA │ not used    │Statement No.   │
└────┴─────────────┴────────────────┘
```

is detected, it is transformed into a pseudo Assembler instruction of the format

```
 (1)  (1)    (2)        (2)
┌────┬────┬───────┬────────────────┐
│ 80 │ C5 │ FF FF │ Statement No.  │
└────┴────┴───────┴────────────────┘
```

This format is the same as for the PROCEDURE statement. The only difference is that it contains X'FFFF' in bytes 3 and 4.

Assembler and machine instructions are differentiated by their first byte, which is 80 for Assembler instructions and 88 for machine instructions. The individual types of Assembler and machine instructions are then determined by means of the second byte (the code byte).

The Assembler instructions are processed by individual routines branched to via the branch table shown in Figure 5. The routines are described later.

Key	Branch to	Handles
C0	B CNOP	CNOP
C1	B DCAL3	DCAL3
C2	B DCX	DC X
C3	B DSL	DS
C4	B LABEL	Label
C5	B PROCE	Begin of block
C6	B ENDBL	End of block
C7	B DCF	DC X with length of DSA
C8	B ADD	Optimizable branch
C9	B DCSTA	DC A (static)
CA	B PARA	Connected field
CB	B UREG	Free register (6 or 5)

Figure 5. Format of the Branch Table

For machine instructions, the instruction format (RR, RX, RS, SI, and SS) is checked. The routine then scans the text for operands with one of the keys E1, E9, E4, E5, and 00. The scanned element is moved into a special 18-byte stack. STAR always points to the first free byte in STA. Pointer STAP is used to indicate the position of the keys.

If one of the keys E1, E9, or E4 is found, the routine OGE is called to retrieve the offset from OFFTAB, add the modifier, determine the base register and the 4K-block and generate code, if necessary, insert the base register and the offset into the half-word following the modifier as shown in Figure 1.

If one of the keys E5 or 00 is found, only a part of OGE (OGE1) is used to generate code if the modifier is greater than or equal to 4K, and to insert offset and register (for key E5 only) into the half-word following the the modifier half-word in STA.

OGE --AR

OGE determines whether or not the name at STAP+5 is an entry in the present part of OFFTAB. If it is an entry, OGE moves offset and attribute byte to OFEN+2, determines the base register, changes the key, and moves base register plus offset into the corresponding column of STA. If additional code is required, it is generated by means of the routine KBT.

BAS -- AS

This routine determines the base register by means of the variable level (in the attribute byte at OFEN+4) and the current block level (in LEVS).

The attribute byte has the following format:

bits 0-1: level
bit 2 1 = automatic
bits 3-7: off

For static variables, register 12 is used as base register.

For automatic variables, the block containing the variable is determined. Figure 6 shows the register used for the current block level and the level of the variable. The base register is returned in RY.

	CURRENT BLOCK LEVEL		
VARIABLE LEVEL	1	2	3
1	13	11	11
2	ERROR	13	10
3	ERROR	ERROR	13

Figure 6. Base Registers Used for Block and Variable Level

MOK -- AT

MOK inserts the base register (contained in RY) into the leftmost 4 bits of the half-word at STAR and the rightmost 12 bits of the offset into the other 12 bits.

KBT -- AT, AU, AV, AW

KBT determines the 4K-block and distinguishes the offsets as follows:

1. offset smaller than 12K
2. offset greater than or equal to 12K and smaller than 32K
3. offset greater than or equal to 32K

In each of these cases it determines whether or not the instruction is an RX instruction.

It generates code by means of 3 masks (see MIO -- AY) corresponding to the 4K-block and the type of instruction and returns the number of the base register in RY. The number of the index register, if any, is inserted into the instruction at STA+3.

FRR -- AX

This routine determines which of the two registers 5 or 6 is free and returns the number of the free register in R2. If both registers are free, 6 is returned. If no register is free, R2 is set to zero. Whether or not a register is free is indicated by the byte REWO (see Figure 7).

If a free register is used for the first operand of an SS instruction (SWITCH=1), bits 4 (for R5) and 5(for R6) are set to prevent the use of these registers for the second operand. Therefore, these bits are always tested when a free register is found. If the corresponding bit is on, the other register is used and its contents are saved, if required.

Bit	Meaning
0-3	Not used
4	If on, register 5 is used for first operand of an SS instruction
5	If on, register 6 is used for first operand of an SS instruction
6	If on, register 5 is free
7	If on, register 6 is free

Figure 7. Available Registers as Indicated by Byte REWO

MIO -- AY

MIO puts a mask, the initial address of which has been inserted into R0, into OBUF. It identifies the mask and thus its length. The hexadecimal formats of the masks are as follows:

LH mask (9 bytes):
88 48 00 00 E1 00 04 00 00

AR mask (4 bytes):
88 1A 00 00

LA mask (9 bytes):
88 41 00 00 E5 00 00 00 00

L mask (9 bytes):
88 58 06 00 E1 00 06 00 00

ST mask (9 bytes):
88 50 06 00 E1 00 06 00 00

CNOP -- AZ

This routine is identical to MULTI.

DC AL3 -- AZ

The element (length 7 bytes) is moved into OBUF.

DCX -- AZ

The length of the element is determined by adding 4 to the length half-word. The element is then put into the output stream.

DSL, LABEL, DCF, DCSTA, ENDBL

All these routines are identical to MULTI.

MULTI -- AZ

This routine moves the element (length 4 bytes) into OBUF.

PROCE -- BA

This routine determines whether there is a statement number or not. If there is a statement number, the element is put out. Otherwise, this routine stores the level from POI+2 at LEVS and calls MOO to move the element into OBUF.

ADD -- BB

The pseudo assembler instruction and the following branch are put into OBUF.

PARA -- BA

The following elements of length LC (restricted to 256) are regarded as one unit and put into OBUF together with the preceding assembler pseudo instruction PARA. No additional ADD, CNOP, or LABEL may occur in the string.

UREG -- BB

The free register (5 or 6) is flagged in a special flag byte REWO. If bit 7 of REWO is on, register 6 is available. If bit 6 is on, register 5 is available. The register that may not be used is specified in the second half-word of the instruction. If it contains 5, this means that register 5 is not free. If it contains zero, both registers are free. The element is then put into OBUF.

This phase performs the following functions:

1. It constructs a label table LABTAB which contains the internal names of the labels, the location counter values relative to the beginning of each program block, and the number of the program block;

2. It constructs a program block table PBT which contains the names and addresses of the program blocks;

3. It optimizes the code for branches to label constants inside the same 12K segment and updates LABTAB and PBT. The code generated for branches that cannot be optimized is shown in the section Generated Code and Optimization.

Block Structure

The source program has a special block structure; blocks may be nested into one another up to a level of 3. This block structure is assorted in a previous phase, so that there is no longer any nesting.

Each program block may have a maximum length of 32K bytes. Because most of the branch instructions branch to labels inside of the same program block, the label handling is optimized within a program block. All branches to labels outside of the block are not optimized.

If any program block is larger than 12K bytes, it has to be divided into segments of 12K. At the end of each (full) 12K segment, an instruction

```
   BALR 9,0
```

is generated so that register 9 is always loaded with the initial address of the current 12K segment. For the first 12K segment of each program block, register 9 is loaded by the prologue.

A branch within a 12K segment may require a displacement greater than or equal to 4K. Therefore, the general registers 7 (loaded with 4K) and 8 (loaded with 8K) are used as index registers. Thus, any branch inside of a 12K segment is possible without using a displacement greater than or equal to 4K.

Generated Code and Optimization

In phase E50, a 4-byte pseudo Assembler instruction of the format

80	C8	LABEL

is generated in front of each branch to a label constant inside the same program block so that the code can be optimized as follows:

1. The branch instruction and the label appear in the same 12K segment of the same program block:
   The pseudo Assembler instruction is deleted and the branch may be modified by inserting register 7 or 8 as index register (if the displacement is greater than 4K-1 or 8K-1, respectively).

2. The branch instruction and the label appear in different 12K segments of the same program block:
   The code to be generated depends on the type of branch instruction.

   a. Absolute branch

   ```
   AH 9,=X'3000' (or = X'6000')
   BC F,LABEL
   ```

   b. Conditional branch

   ```
   BALR  14,0
   AH    9,=X'3000'
   SPM   14,0
   BC    8,LABEL
   SH    9,=X'3000'
   ```

   c. BCT and BAL

   ```
   AH    9,=X'3000'
   BAL   LINK,LABEL
   SH    9,=X'3000'
   ```

   If the label has an offset smaller than that of the branch instruction, the AH will be replaced by an SH and the SH by an AH. The constants 12K and 24K are half-words contained in static storage.

Construction of LABTAB and PBT

LABTAB and PBT are constructed in the routine LATA. The text is scanned for the pseudo Assembler instructions PROCEDURE, END OF BLOCK, and LABEL.

1. When PROCEDURE is found, the number of the program block is stored in a stack, the name and the location counter LOC1, which counts the offset from the beginning of the program, are inserted in PBT at a location given by the block number (PBN). The second location counter LOCO, which counts the offset from the beginning of the program block, is set to zero to start a new count. The entry name is inserted in LABTAB with the location 0.

   The format of PBT and LABTAB entries is as follows:

   PBT:
   Name (2 bytes)
   Offset (LOC1) (2 bytes)

   LABTAB: Name (2 bytes)
   Offset (LOCO) (2 bytes)
   PBN (1 byte)

2. When END OF BLOCK is found, it is checked whether LOCO is greater than 32K and whether LOC1 is greater than 64K. An error is indicated if either one of these conditions is detected. The part of LABTAB that pertains to this block is written on SYS001.

3. When LABEL is found, the name, LOCO, and the block number PBN are inserted at the appropriate location in LABTAB. While PBT remains in storage for phase G15, LABTAB is written on SYS001 for the phases G01 and G25.

Optimization of Text Code

The text is scanned for the 4-byte special pseudo Assembler instruction ADD, which indicates an optimizable branch. This instruction has the format.

```
┌────┬────┬───────┐
│ 80 │ C8 │ LABEL │
└────┴────┴───────┘
```

The present part of LABTAB is scanned for the label and, if the label is found, the label offset is compared with the current location counter LOCO if the label and the ADD instruction are located in the same program block. If label and instruction appear in the same 12K segment, the pseudo Assembler instruction is deleted and the following instruction is modified. The label offsets of this program block in LABTAB and all offsets of following blocks in PBT are updated.

If label and instruction do not appear in the same 12K segment, code is generated as described in the section Generated Code and Optimization.

At the end of a 12K segment, an instruction

```
BALR 9,0
```

is generated to load register 9 with the initial address of the following segment. This instruction must be exactly at the end of the 12K segment. This is achieved by inserting one or more instructions of the type

```
BCR 0,0
```

if necessary.

Critical Cases and Boundary Problems

There are some critical cases at the end of each 12K segment. Because the LABTAB was constructed when maximum code was present or simulated by the pseudo Assembler instruction, some labels may get from one 12K segment into the other during this phase due to the deletion of code.

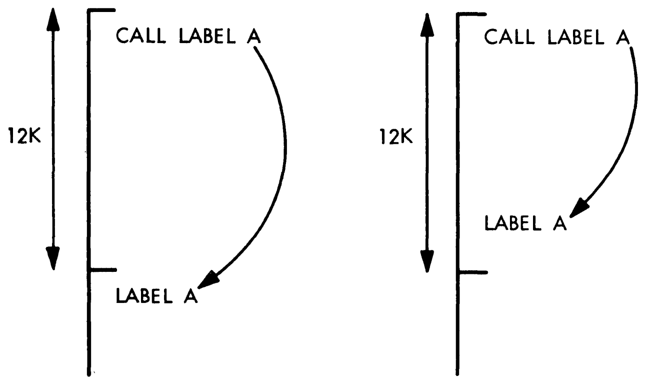


Figure 1. Calling of Label A at Different Moments of Compile Time

Figure 1 shows two cases of calling LABEL A at different moments of compile time. While in case 1 instruction and label are situated in different segments, they are in the same segment in case 2, although instructions 1 and 2 are situated in the same segment. These cases are handled differently as follows:

Case 1:

The ADD instruction is not deleted because it is not known whether or not LABEL A will go into the same segment. In phase G15, where it can be determined whether label and instruction are in the same segment, the generated AH instruction is modified to a NOPR instruction by inserting the address of a zero half-word into the AH instruction, so that zero is added to register 9.

Case 2:

The pseudo Assembler instruction is deleted and no code is generated. Another critical case occurs, for instance, if there is only one label to be called in a 12K segment, and this is situated just before the last instruction of the segment. Because a

    BALR 9,0

instruction has to be generated at the end of each 12K segment, it may happen that the label is moved into the next segment so that the optimization performed before becomes wrong. To exclude this case, A will be regarded as being situated in the next block, so that no optimization is performed.

Due to the limited table space, only a part of LABTAB may fit into storage so that the rest remains on SYS001. Because each LABTAB entry has 5 bytes, the maximum number of present labels is restricted to FLOOR [M*BUFFERLENGTH:5] =NPL where M is the number of buffers in the table space. For a 16K system with a buffer length of 256 bytes, the number of labels in the table space is NPL = FLOOR (9x256:5) =460.

If NPL is smaller than the number of labels of one 12K segment, some labels, although belonging to the same segment, may not be present in the table space and, therefore, be regarded as belonging to another segment. The corresponding branches are not optimized. Optimization is stopped when coming to a branch with current offset greater than the offset of the last present label.

Phase Input and Output

Phase input is the output text string of phase F95. The I/O buffers B1 and B2 are used for accommodating the text input (the pointer is POI). Input is controlled by the subroutine ISU.

LABTAB is intermediately written on SYS001 if it becomes larger than the table space. It is put out in records of buffer length (ZTAB19) and read again into the table space for updating. At the end of each block, the corresponding updated part of LABTAB is written on SYS001 (ZTAB20). Thus, LABTAB is completely on SYS001 at the end of the phase. PBT is left in storage. It is placed at the end of the program space by an ORG instruction.

Buffer B0 and the pointer POU are used for output. The output consists of the optimized text string which contains only one new element:

(1)	(1)	(1)	(1)	(1)	(1)	(2)	(2)
80	C0	OP code	X	00	E1	NAME	0000

This element represents the optimized branch instruction. The special pseudo Assembler instruction ADD is changed in this phase to

(1)	(1)	(1)	(1)	(1)	(1)	(2)	(2)
80	C8	4A or 4B	9	00	E1	NAME	0

The CNOP instruction

80	C0	B	W

is replaced by a corresponding number of

    BCR 0,0

instructions of the format

88	07	00	00

so that the key C0 is available for the other instructions, and the instructions

80	CA	LC	LO

and

80	CB	KEY

are deleted.

DESCRIPTION OF ROUTINES

Note: The routines ISU, MOO, and MODIF are described in the section General Description of Phases F95 - G55.

Symbols used in flow charts:

B0, B1, B2	- Initial addresses of the I/O buffers
BUFL	- Buffer length
OBUF	- Output buffer (for text), same as B0
IBUF	- Input buffer (for text)
POU	- Pointer for OBUF
POI	- Pointer for input buffers
LAPO	- Pointer for LABTAB in the I/O buffers
TS	- Initial address of table space

TSE	- End of table space (points to first byte after table space)
LSP	- Pointer for LABTAB in table space
NLB	- Number of LABTAB buffers
LAPO	- Pointer in table space to insert labels into LABTAB
BA	- BUFL of B0
LASTAD	- Address of last updated LABTAB element + 5
BTS	- TS + BUFL
LOCO	- Location counter from beginning of a program block
LOC1	- Location counter from beginning of program
UPCO	- Update counter
UP	- Update counter for present LABTAB part
PBTCO	- Pointer for PBT and other tables
BRAN1	- Branch table
N	- Number of current 12K segment
SSS	- Switch for SS format
SRS	- Switch for labels
CCB	- Switch for conditional branches
NLBSTA	- Stack for NLB
NO	- Number of program blocks
R0, R1, R2, RX, RY	- Working registers
PBT	- Program block table
OFFTAB	- Offset table
LABTAB	- Label table
M	- Number of buffers in TS

LATA -- BI

LATA constructs the first version of LABTAB. It scans the text string for the pseudo Assembler instructions LABEL, PROCEDURE, and END of block. Two location counters are used. LOCO counts from the beginning of each program block; LOC1 counts from the beginning of the program.

1. When LABEL is found, LOCO is stored in LABTAB together with the label name and the number of the current program block.

2. When PROCEDURE is found, the number of the program block is stored in a stack. The name and LOC1 are inserted into PBT at a location given by the block number (PBN). LOCO is set to zero to start a new count. The entry name is inserted into LABTAB with the location 0. The format of the entries in PBT is as follows:

   ```
   bytes 1 - 2 :  name
   bytes 3 - 4 :  offset (LOC1)
   ```

   The format of the entries in LABTAB is as follows:

   ```
   bytes 1 - 2 :  name
   bytes 3 - 4 :  offset (LOCO)
   byte    5   :  program block number
   ```

3. When END OF BLOCK is found, an error is indicated if LOC1 is greater than 64K or if LOCO is greater than 32K.

LABE -- BJ

If the pseudo Assembler instruction LABEL is found, LOCO is stored in LABTAB together with the internal name of the label and the current program block number contained in the corresponding stack. LABTAB is written on SYS001 if an overflow occurs. The element LABEL is put into OBUF.

UDA -- BK

UDA updates those offsets of the present labels in LABTAB of the current program block that are greater than the value of the current location counter LOCO with the value given in UP. The same updating is performed for the offsets in PBT that are greater than the value of LOC1.

MOC -- BL

If the half-word pointed to by LSP contains the end key (X'FFFF'), LABTAB is written on SYS001. It is tested whether the byte pointed to by LSP + 4 (block number) is equal to PBN. If it is not, the routine returns. If the label location is greater than LOCO, the routine returns. If it is not, LSP is increased by 5. If LSP becomes greater than TS + BUFL, the first buffer in the table space is written on SYS001. LSP is decreased by BUFL and a new LABTAB record, if any, is read in and updated.

SCAL -- BM

If switch SRS is off, SCAL scans the present part of LABTAB for the label at POI + 2. If SRS is on, the label is regarded as not belonging to the same 12K segment. Thus, optimization is suppressed for branches of that part of a 12K segment the labels of which are not present in the table space.

Before scanning LABTAB, the current program block number PBN and LOCO are compared with the number of the last present LABTAB entry. If the PBNs are identical and LOCO is greaer than the label offset, SRS is set to one. SRS is reset if the end of the program block or the 12K segment is found.

If the label is in the present part and belongs to the same program block and the same 12K segment, SCAL returns RX $\neq$ 0; otherwise, it returns RX = 0.

MADM -- BN

This routine moves the ADD mask to OBUF by means of the routine MOC.

SBO -- BN

This routine is used for the handling at the 12K segment boundaries. It generates

```
   BCR 0,0
```

instructions until LOCO has the value N*12K-2. The instruction

```
   BALR 9,0
```

is generated to load register 9 with the initial address of following segment.

The location counters and the update counter are adjusted. The present labels are updated (by calling UDA), and new labels are read into the table space (by calling MOC).

Branch Table BRAN1 and Routines

To determine the various Assembler instructions, the code byte is used as offset in a branch table BRAN1 which contains branches to corresponding subroutines. For the format of the branch table refer to phase F95.

The individual routines branched to via the branch table perform the following functions:

CNOP (Test N) -- BO

N = 0 : Word boundary: Add 2 to location counters. Double-word boundary: Add 6 to location counters. Skip the element.

N ≠ 0 : Evaluate the increment of the location counters. Generate a corresponding number of

instructions. Update the present label offsets of LABTAB, change UPCO and update PBT.

DCAL3 -- BO

The location counters are increased by 3 and the element is moved into OBUF.

DCX -- BO

The second half-word (L) of the element is added to the location counters and the element is put into OBUF.

DSL -- BP

The second half-word (L) of the element is added to the location counters and the element is moved into OBUF.

LABEL (Test N) -- BP

N = 0 : Branch to LABE.

N ≠ 0 : Move the element into OBUF.

PROCE -- BQ

The routine checks whether there is a statement number. If there is not, it stores the program block number in stack PBN and sets LOCO to zero. It tests whether N = 0.

N = 0 : Insert the name of the block and LOC1 into PBT and the name with LOCO into LABTAB.

N ≠ 0 : Move the element into OBUF.

If it is a statement number, the element is moved into OBUF.

ENDBL -- BQ

An error message is produced by calling phase G31 if either LOCO ≥ 32K or LOC1 ≥ 64K. Call MOC to write the labels of the current block on SYS001 and read the next part of LABTAB into the table space, if possible. Move the element into OBUF.

DCF -- BQ

The location counters are increased by 4 and the element is moved into OBUF.

ADD -- BR

N = 0 : Determine the type of the following branch instruction and increase the location counters accordingly. Skip the element and the following branch instruction.

N ≠ 0 : Call SCAL to scan LABTAB for the label (second half-word). Determine whether the label belongs to the same 12K segment as the instruction.

1. If not the same 12K segment:

   Insert name and instruction key into the ADD mask of following format:

(1)	(1)	(1)	(1)	(1)	(1)	(2)	(2)
80	C8	4A 4B	9	0	E3	NAME	0

a. An absolute branch follows: The ADD mask followed by the branch instruction is put into OBUF.

b. A BAL or BCT instruction follows: The ADD mask is put into OBUF followed by the branch instruction and an SH instruction of corresponding format.

c. A conditional branch follows: The following sequence of instructions is generated:

```
BALR  14,0
AH    9,....
SPM   14
BC    ...
SH    9,....
```

The location counters are increased accordingly. If the generated instructions should not fit in the current 12K segment, corresponding NOPR instructions are inserted and the instructions are put into the next segment.

2. Same 12K segment:

Delete the pseudo Assembler instruction by adjusting the pointer POI and change the following branch instruction to the format

(1)	(1)	(1)	(1)	(1)	(1)	(2)	(2)
80	C0	Op. code	X	0	E1	NAME	0

Put this element into OBUF and increase the location counters by 4. Update the label table LABTAB and PBT.

<u>DCSTA -- BP</u>

The location counters are increased by 4 and the element is moved into OBUF.

<u>PARA -- BT</u>

N = 0 : Increase the location counters by the length LO (contained in the byte at POI + 3). Skip the element of length 4 and the following string of length LC.

N ≠ 0 : Delete the pseudo Assembler instruction (POI + 4). Test whether LOCO + LO is greater than N*12K - 2. If the value is not greater, increase LOCO and LOC1 by LO. Put the string at POI (length LC) into OBUF. If the value is greater, evaluate

N*12K - 2 - LOCO = ABS.

Generate ABS: 2 instructions of the type

BCR 0,0

Increase the location counters by ABS + 2 and N by 1. Then generate an instruction

BALR 9,0;

update present labels and branch to the action listed under item 1.

<u>UREG -- BT</u>

This instruction is deleted during the construction of LABTAB by increasing the pointer POI by 4.

The offset table OFFTAB is listed if the SYM option is active. This phase inserts the label offsets and block numbers from the label table LABTAB into the offset table OFFTAB. The offset table is put into the table space in subsequent parts. The label table is read into the I/O buffers record by record.

Each LABTAB record is scanned. For each entry, the offset half-word and the following program block number are inserted into the present part of OFFTAB, if possible. The scan of LABTAB is repeated until all parts of OFFTAB have been in the table space.

Phase Input and Output

The input consists of the two tables OFFTAB and LABTAB retrieved from SYS001. OFFTAB is read into the table space in parts of M records. The value of M is stored at IJKMIP+2.

LABTAB is read (record by record) into the I/O buffers B0 and B1 with the pointer LAPO. Each record is processed separately. If the end of LABTAB is found, the next part of OFFTAB is written into the table space and reading of LABTAB starts again with the first record.

The output (used by phase G15) consists of the changed offset table which is written onto SYS001 under ZTAB07.

OFLIS -- CC

This routine lists the offset table OFFTAB in the following format (in the example, the start address of the static storage is assumed to be 4000):

INTERNAL NAME	OFFSET	TYPE	MODULE OFFSET
0101	0024	STATIC	004024
0102	0038	AUTOMATIC	

Variables, constants, or labels that do not have an offset in static or automatic storage are not listed. For entry names, the offset of the generated address constants in static storage are written.

RAN -- CD

RAN inserts one line of the offset table into the print buffer. For the format of the line see the section OFLIS -- CC. Then the line is printed by the routine ZPRNT.

TRANS -- CD

This routine translates hexadecimal values into EBCDIC. The bytes (the number of which is given in R2) at 0(R1) are translated and moved into the print buffer at the location given in R0.

WOLA -- CB

WOLA inserts the offsets and block numbers of the present part of LABTAB into the present part of OFFTAB.

It takes the first half-word of a LABTAB entry, multiplies it by 3, and takes the value as offset in the offset table. It then inserts offset and block number at the position indicated by the offset. The pointer LAPO is increased by 5 for adjustment to the next entry. The routine returns when it finds the end key or reaches the end of the first buffer.

PHASE PL/IG15 (FINAL OFFSET PREPARATION) -- CH

This phase inserts the offsets from OFFTAB together with the corresponding base Registers into the instructions. A half-word following the instruction is reserved for this purpose as described in phase F95. For address constants, three bytes are reserved for the offset.

The phase consists of two main parts:

1. Processing of the text string and
2. Processing of the constants contained in static storage.

Text String

The text output of phase G00 is the input of this phase. The text is scanned for RX, RS, SI, SS, and DC AL3 instructions. The format of the operands of these instructions is as follows:

byte 1	key
bytes 2-3	name
bytes 4-5	modifier

The key may have the following values:

X'E1'	declared variable
X'E9'	constant
X'E4'	generated variable
X'E5'	register
X'00'	absolute address
X'11'	DC AL3 (label)
X'10'	already processed
X'20'	DC VL3

a. The first three keys mark the following field as entry in OFFTAB. In these cases, the offset is taken from OFFTAB. The modifier is added, the corresponding base register evaluated, and both are inserted into half-word following the operand.
The key is changed to X'10' to mark that the element has already been processed.
Address constants require special treatment because the offset may become greater than 64K. Therefore, the offset is inserted into the following 3 bytes.

b. The key X'E5' denotes that the operand (the half-word name) is a register. The register is taken as base register and the modifier as offset. Both are inserted into the following half-word as described under item a. The key is changed to X'10'.

c. The key X'00' denotes an absolute address, i.e., the name field is free, no base register has to be taken, and the modifier has to be inserted as offset.

Though most of the instructions were processed in phase F95, some are left because the offsets for the labels could not be inserted; some strings, the elements of which should be connected, have not yet been processed; and the address constants could not be processed because some of them might contain labels.

d. The key X'10' is used to mark operands already processed, i.e., that the half-word with offset and base register has already been inserted.

Besides those normal cases, two special instructions have to be regarded. Those are the special pseudo assembler instructions with the code byte X'C0' or X'C8', respectively. (See phase G00). Special base and index registers have to be inserted besides the offset in these cases.

Constants

The constants are on SYS001 in the table CONTAB. They have the following format:

bytes 1 - 2	internal name
byte 3	attribute
bytes 4 - 5	length
bytes 6 - n	constant

The attribute byte has the following format:

```
bits 0-1 : 00 = constant of type DC X
           01 = constant of type DC A
           10 = constant of type DC V
           11 = constant of type DC AL3
bit   2  :  0 = not optimizable
            1 = optimizable

bit   3  :      not used
bit   4  :  1 = DCVL3 if bits 0-1 are on.
bit   5  :  1 = word boundary
bits 6-7 : 01 = DC X for label assignment
                DC A for segment origin
           10 = DC A of an entry name
           11 = double-word boundary
```

The constant itself is given in the internal representation or, if it is an address constant, as name plus modifier (4 bytes).

	Byte (s)									
Instruction	1	2	3	4	5	6	7	8	9	10
DC X	80	C2	length		constant					
DC A	80	CA	10	name		modifier		offset		
DC V	80	CB	10	name		modifier		offset		
DC AL3	80	C1	10	name		modifier		offset		
DC VL3	80	C1	13	name		modifier		offset		

Figure 1. Output Format of Translated Constants

These constants are translated into the same format as those in the text string, and the offset for address constants is inserted. For DC A, the code byte X'CA' and for DC V the code byte X'CB' is used. The constants are put onto the text medium behind the normal text; the end key is put at the end of the constants so that there is no difference between the two parts in following phases. At the begin of the constants, a label FFFF is generated denoting the begin of static storage.

The output is shown in Figure 1.

Two keys are used for DC AL3 because two types of instructions may occur: 1) the normal DC AL3 with the key X'10' and 2) a DC VL3 (flagged by bit 4 of the attribute byte) with the key X'30'.

The names in front of the constants, used to address them, are inserted into a pseudo assembler instruction LABEL.

Special constants:

a. For GOTOs from one program block to a label in another one, the following DC instructions are used:

```
L'aaaa' DC A(BLOCK + 12K segment)
        DC X'OFFSET'
```

The DC A constant contains the address of the program block and a multiple of 12K as modifier representing the initial offset of the 12K segment corresponding to the label.

The 4-byte DC X constant is the label offset in the block reduced by the offset of the corresponding 12K segment.

The output formats of these constants are shown in Figure 2.

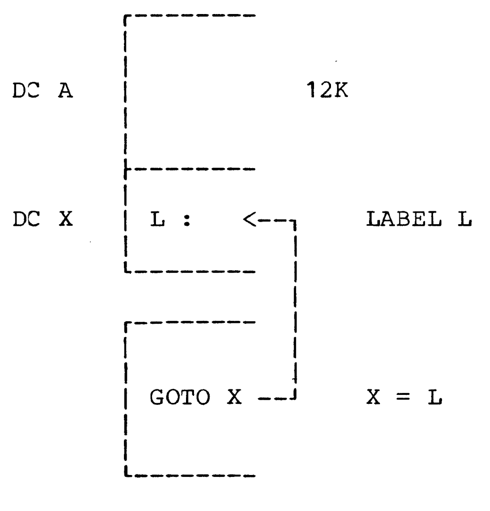
80	C4 label aaaa

80	CA	10	BLOCK NAME	N*12K	OFFSET

N = 0 or 1 or 2

80	C2	4	CONSTANT

Figure 2. Output Formats of DC A and DC X Constants Generated for GOTOs.



b. To correctly load register 9 (in the prologue) when calling an entry name, a constant

'Label DC A (BLOCK + 12K segment)'

is used. Its output is the same as that of the previous instruction.

General Flow

The offset table is read into the table space and the text is scanned and processed. If the End key is found, the constant table CONTAB is read into the buffers. The format is changed, the offset inserted, and the constants are written behind the text string.

If the offset table is greater than the table space, the text followed by the constants is scanned again. This process is repeated until all parts of OFFTAB have been in the table space. At the end of the phase it is tested whether a file table exists. If there is a file table, phase G17 is called. Otherwise, phase G17 is skipped and phase G20 is called.

Phase Input and Output

The input consists of the text string as generated in phase G00 and of the following tables:

OFFTAB	Offset table to be found on SYS001 under ZTAB07 or, if small enough, in the table space.
CONTAB	Constant table to be found on SYS001 under ZTAB08. It is read into the I/O buffers.
CARTAB	Character-string table to be found on SYS001 under ZTAB00. It is read in and put behind the text string.
EXTTAB	External name table to be found on SYS001 under ZTAB04. It is written onto the free text medium at the end of the phase.
PBT	Program block table in the program space (generated by phase G00).

The text is read into the I/O buffers B1 and B2 with the pointer POI and put out by means of the output buffer B0 with the pointer POU.

Symbols used in flow charts

B0, B1, B2	- I/O buffer begin addresses
IBUF	- Input buffers
OBUF	- Output buffers
POI	- Input pointer
POU	- Output pointer
BRAN2	- Branch table
LOCO	- Location counter
DCXMA	- Address of the DC X mask
OFEN	- Storage area for offset
HW	- Half-word
BN	- Number of OFFTAB records left
R0, R1, R2, RY	- Working registers
SSS	- Switch for SS instructions
SW	- Switch for the constants
REN	- Register containing the name
LAMA	- Address of LABEL mask
CN	- Number of constant records
PSC	- Pointer for constants
AMA	- Address of DC A mask
DSLMA	- Address of DS mask
SWO	- Switch for DC SO
M	- Number of buffers in the table space

Functional Description

This phase has the following functions:

1. To scan the text for instructions not yet processed.

2. To generate the half-word containing base register and offset after the operand.

3. To change the format of the constants in static storage.

4. To insert the offsets for address constants into the succeeding 3 bytes.

Since OFFTAB may become greater than the table space, only a part of OFFTAB might be present in storage. Therefore, the text string and the constants may have to be scanned several times, once for each part of OFFTAB.

Machine Instructions

The instruction format is determined by testing bits 0 and 1 of the operation-code byte.

RR Format

The element is moved into OBUF and LOCO is increased by 2.

RX, RS, or SI Format

LOCO is increased by 4. The key at POI+4 is tested. If the key is E1, E9 or E4, the offset is taken from OFFTAB together with the attribute byte. The modifier is added to the rightmost 12 bits of the offset, and the base register is evaluated. The element is put into output after the key has been changed. The half-word containing base register and offset is put behind the element.

If the key is E5, the offset half-word is constructed from register and modifier. If the key is 00, the modifier is used as offset half-word. If the key is 10, the element is moved into the output.

SS Format

Both operands of the instruction are handled as described for the RX, RS, and SI instructions. The element is put into the output in two parts. LOCO is increased by 6.

Assembler Instructions

The individual assembler instructions are determined by the second byte which contains the instruction key. The keys are:

C0	- optimized BRANCH
C1	- DC AL3
C2	- DC X
C3	- DS L
C4	- LABEL
C5	- PROCEDURE
C6	- END OF BLOCK
C7	- DC X'LENGTH OF DSA'
C8	- special ADD
C9	- DC A(STATIC)
CA*	- DC A
CB*	- DC V
CC*	- DC LASS
CD*	- DC SO

* used for static constants

The key byte is put into a general register and X'C0' is subtracted. The resulting number is used as an offset in the branch table (BRAN2) that contains the corresponding branches to the processing subroutines.

The branch table BRAN2 has the format shown below.

```
BRAN2  B  OBRA    0
       B  DCAL3   1
       B  DCX     2
       B  DSL     3
       B  LABEL   4
       B  PROCE   5
       B  ENDBL   6
       B  DCF     7
       B  ADD     8
       B  DCSTA   9
       B  DCA     A
       B  DCV     B
       B  DCLASS  C
       B  DCSO    D
```

The names contained in the table are the names of the subroutines. The displacement within the table is found by multiplying the number in the right-hand column by four.

OBRA -- CL

The instruction (branch) has the following format:

Byte(s)	Contents
1- 2	X'80C0'
3	operation code
4	X
5- 6	X'00E1'
7- 8	name
9-10	zero

LOCO is increased by 4. The name field is tested to determine whether the corresponding offset is contained in the present part of OFFTAB. If it is not contained, the element is put into OBUF unchanged. If it is contained, the first byte (80) is deleted. The second byte (C0) is replaced by X'88'. The offset is taken from OFFTAB.

```
0  ≤ offset module 12K < 4K byte 4=X'00'
4K ≤ offset module 12K < 8K byte  4=X'07'
8K ≤ offset module 12K < 12K byte 4=X'08'
```

The key E1 is changed to 10 and the element is put into OBUF. The offset modulo 4K and the base register 9 are inserted into the following half-word.

The new format is:

Byte(s)	Contents
1	X'88'
2	operation code
3	X
4	0, 7, or 8
5	X'10'
6- 7	name
8- 9	zero
1-11	left-hand four bits: X'9' remainder: offset

DC AL3 -- CM

The instruction has the following format:

Byte(s)	Contents
1- 2	X'80C1'
3	key
4- 5	name
6- 7	modifier
8-10	used for offset

The location counter is increased by 3. The key is tested for 10 and 11. If it is 10, the element has already been processed and is put into OBUF unchanged. In this case, the succeeding 3 bytes contain the offset. If it is 11, the key denotes that the operand is a label. The offset is taken from OFFTAB together with the program block number. The offset is increased by the sum of modifier and initial block-address offset to be taken from PBT. This sum is inserted into the succeeding three bytes. The element is then put into OBUF after the key has been changed.

If the key is neither 10 nor 11, the operands are constants in static storage. The offset is taken from OFFTAB (if present). The sum of offset, modifier, and length of program is inserted into the succeeding 3 bytes after the element with the changed key has been moved into OBUF.

DCX -- CM

Since the constant may become greater than 256 bytes (for STATIC), a special treatment for I/O is necessary. Thus, the element is not changed but only put into the output (LOCO + length of constant). It is tested whether the element is fully contained in the input buffers B1, B2. If it is, MOO is called to move the element into OBUF. If it is not, MOO is called for the present part. The input pointer and the element length are updated and new input is read into B1 and B2. This loop is repeated until the entire element has been brought from input to output.

DS L -- CN LOCO is increased by the length L (HW at POI+2). The element is put into OBUF unchanged (call MULTI).

LABEL -- CN (Same as MULTI) -- CN. The element is put into OBUF unchanged.

PROCE -- CN LOCO is set to zero to start a new count. The element is then put into OBUF (call MULTI).

ENDBL -- CN. (Same as MULTI). The element is put into OBUF unchanged.

DC F -- CN The element is put into OBUF unchanged and LOCO is increased by 4.

MULTI -- CN. The element of length 4 is moved from POI to POU by calling MOO.

ADD -- CN The element has the following format:

Byte(s)	Contents
1- 2	X'80C8'
3	X'5A' or X'5B'
4-10	X'0900E300040000'

LOCO is increased by 4. The name field is tested to determine whether the corresponding offset is contained in the present part of OFFTAB.

If it is not contained, the element is moved into OBUF unchanged. If it is contained, the first byte (80) is deleted. The second byte (C8) is set to X'88'. The key is changed to X'10'. The offset is taken from OFFTAB. Then it is tested whether or not LABEL and instruction are contained in the same 12K segment. The modifier is replaced by 0 if it is the same 12K segment, by 12K if it is a neighbouring segment, and 24K in all other cases. Base register 12 together with the offset of the constant which is stored in a special stack during the initialization are inserted into the succeeding half-word.

The new format is:

Byte(s)	Contents
1	X'88'
2	X'5A' or X'58'
3- 7	X'0900100004'
8- 9	modifier
10-11	left-hand four bits: X'C' remainder: offset

88	5A or 58	09	00	10	0004	MODIFIER	C OFFSET

The internal name 0004 is reserved for the special constants 0K, 12K, 16K, 20K, 24K and 28K. These are stored as half-words at the beginning of static storage. Their offset is contained in OFFTAB. During the initialization of the phase, the offset of these constants has to be taken and stored into a stack so that it is accessible during the entire phase.

Constant	Name	Modifier
0K	0004	0000
12K	0004	0002
16K	0004	0004
20K	0004	0006
24K	0004	0008
28K	0004	000A

DC STA -- CO. The length of the program is retrieved from the interphase communication region. It is inserted to the last three bytes of the DC A mask. The format of the DC A mask is as follows:

Byte(s)	Contents
1- 3	X'80CA10'
4- 7	zero
8-10	length of program

The name field is set to zero to sign the special DCA. This mask is moved into OBUF, and LOCO is increased by 4.

DC A and DC V -- CP These routines are used for the constants of static storage after they have been brought to a new format.

Byte(s)	Contents
1	X'80'
2	X'CA' or X'CB
3	key
4-5	name
6-7	modifier
8-9	offset

The key is tested for 10 and 11. The DC V routine is contained in DC A.

The key 10 indicates that the element has already been processed. The three succeeding bytes contain the offset. The element is put into OBUF. The key 11 denotes that the internal name represents a label. In this case, the offset together with the program block number, if present, are taken from OFFTAB. The offset is increased by the sum of modifier and initial block-address offset to be taken from PBT. This sum is inserted into the succeeding three bytes and the element put into OBUF after the key has been changed.

If the key is neither 10 nor 11, the offset is taken from OFFTAB, if present. The sum of offset, modifier, and length of program is inserted into the succeeding three bytes. The element is put into OBUF after the key has been changed.

DC LASS -- CQ The instruction has the following format:

80	CC	E1	Int. NAME OF LABEL	0	

The name of the label is tested to determine whether the corresponding offset is contained in the present part of OFFTAB.

If it is not contained, the element is put into OBUF unchanged. If it is contained, the second byte is changed to CA and the third to 10. The offset together with the block number is taken from OFFTAB. The block number is inserted into the name field and the 12K segment is determined. Corresponding to this, 0, 12K, or 24K is inserted as modifier. The modifier is then added to the offset of the program block to be taken from PBT. The sum is inserted into the following three bytes.

80	CA	10	NUMBER OF BLOCK	0, 12K or 24K	OFFSET OF CORRES.12K segm.

Following this, the instruction DC X (containing the offset of the label inside the 12K segment) is generated. Its format is as follows:

80	C2	LENGTH=4	OFFSET

The length 4 is inserted into the DC X mask. The label offset is reduced by the modifier of the preceding DC A and stored into the following 4 bytes. The element is put into OBUF.

DC SO -- CQ. This routine consists of the first part of DC STA. At the beginning, switch SWO is set to one so that the routine is terminated before generating the second instruction.

End Key

When the End key is found for the first time, a routine will be called to transform the static storage constants into the same format as the constants of the text string. The routine inserts the offset, if possible, and moves it into OBUF. After this a switch will be set.

At the beginning of the static storage, a label 'FFFF' is generated while the end key is put at the end of the output.

In all cases, the contents of the last output buffer are written onto text medium. If all parts of OFFTAB have not yet been in storage, a new part of OFFTAB is read into the table space. The work files are switched and rewound, and the phase is called again until all offsets have been inserted.

The character string is retrieved from SYS001 and put behind the text. TXTIN is rewound and EXTAB is written on it. It is tested whether a file table exists. If it does, phase G17 is called. Otherwise, that phase is skipped and G20 is called.

EXTAB -- CR

This routine reads EXTAB record by record into the I/O buffers B0 and B1 in overlapped mode and puts it onto TEXTOUT in the same way.

OGA -- CR

OGA determines whether the corresponding offset is contained in the present part of OFFTAB. If not, it moves the rest of the element (5 bytes) into OBUF and returns. Otherwise, the offset is taken from OFFTAB together with the attribute byte and stored in OFEN+2. The base register is determined by the routine BAS; the rest of the element (5 bytes) is followed by a half-word containing the base register in the first 4 bits and the offset mod 4K in the following 12 bits is moved into OBUF.

GEO -- CS

GEO moves the OFFTAB entry, if present, to OFEN+2 (R1 = 0), or it returns (R1 = 1) after having moved the element into OBUF.

MOK -- CS

Moves the half-word at OFEN+2 into OBUF.

ROFF -- CT

ROFF determines the final offset by adding the modifier (at POI + R1) to the offset, taking the sum modulo 4K, storing the result in OFEN+2 and ORing base register RY on the initial 4 bits of OFEN+2.

BAS -- CU

This routine determines the base register by means of the variable level contained in the attribute byte at OFEN+4 and the current block level contained in LEVS. The format of the attribute byte is as follows:

```
bits 0 - 1 : level
bit    2   : 1 = static, 0 = automatic
bits 3 - 7 : zeros
```

If bit 2 is on (static), register 12 is used as base register. If it is off (automatic), it has to be determined in which block the variable is to be found. Figure 3 shows all possible cases and the base register used. The base register is returned in RY.

		Current Block Level		
		1	2	3
Level of the Variable	1	13	11	11
	2	ERROR	13	10
	3	ERROR	ERROR	13

Figure 3. Basic Register Used for Automatic Variables

TRAN -- CV, CW, DA DB

TRAN transforms the constants of static storage given in a table on SYS001 into the instruction formats used for the text. The constant table CONTAB is taken from SYS001 and put into the input buffers B1 and B2. The format of the constant in the table is:

DC X

Byte(s)	Contents
1-2	name
3	attributes
4-5	length
6-8	constant

DC A, DC AL3, DC V, DC VL3

1-2	name
3	attributes
4-5	4
6-7	internal name
8-9	modifier

All elements with a 'delete' bit are deleted. The name field is translated into the pseudo assembler instruction LABEL. The DC X instructions are transformed into the following format:

Byte(s)	Contents
1-2	X'80C2'
3-4	length L
5-7	constant

This is done by inserting the length into the DC X mask:

Byte(s)	Contents
1-2	X'80C2'
3-4	length L

and moving the mask followed by the constant itself into OBUF. In the case of an optimizable DC X, LOCO is set to boundary depending on the length of the constant:

Length	Boundary
2	half-word
4	word
8	double-word

This is done by generating a corresponding DSL instruction by means of the DSL mask:

Byte(s)	Contents
1-2	X'80C3'
3-4	length L

There are special constants for label assignments which are transformed to the format:

Byte(s)	Contents
1-2	X'80CC'
3	key
4-5	label
6	zero

The routine DC LASS is called to insert the offsets, to generate the following DC X and to put the elements into OBUF. These special constants originally have the following format in CONTAB:

1. DC A

Byte(s)	Contents
1-2	name of DC A
3	attributes bits 0-1 = 00: DC X bit 2 = 0 : not optimizable bits 3-5 always zero bits 6-7 = 01: label assignment
4-5	length = 4
6-7	label
8-9	zero

2. DC X

Byte (s)	Contents
1-2	zero
3	attributes

bits 0-1 = 00: DC X
bits 2-7 always zero

The DC AL3 and the DC VL3 instructions are transformed by means of the DC A mask:

Byte (s)	Contents
1	X'80'
2	X'C1', X'CA', X'CB', or X'CC'
3	key
4-5	internal name
6-7	modifier
8-10	place for offset

The key C1 is inserted into byte 1, the internal name and the modifier into the corresponding two half-words. The key at byte 2 is set to 11 for entry names, to 20 for DC VL3, and to E1 for all other items. The routine DC AL3 is called to insert the offset, if possible, and to put the element into OBUF.

The same mask is used for DC A and DC V instructions, CB is inserted into byte 1 for DC V and CA or CC for DC A, after LOCO has been brought to word-boundary.

In the case of a DC V, the element is put into OBUF. The modifier is taken as offset and put into the three bytes following the element. The key at byte 2 is set to 10.

For DC A instructions the following three cases may occur: The operand is an entry name: The key (byte 2) is set to 11, and the routine DC A is called to put the element followed by the offset (if present) which is formed by the sum (modifier + offset from OFFTAB + length of program block) into OBUF.

Segment origin:

The format of this instruction is:

Byte (s)	Contents
1-2	name of DC A
3	attributes
4-5	length = 4
1-7	entry
8-9	zero

The format of the attribute byte is as follows:

bits 0-1 : 01 = DC A
bit 2 : 0 = optimizable
bits 3-5 : zero
bits 6-7 : 01 = segment origin

It is transformed to:

Byte (s)	Contents
1-2	X'80CD'
3	key
4-5	label
6-7	zero

and the routine DC SO is called to insert the offsets and to put the element into OBUF.

In all other cases, the key (byte 2) is set to E1 and the routine DC A is called to put the element followed by the offset which is formed by the sum (modifier + offset from OFFTAB + length of program) into OBUF.

BOU -- DC

BOU sets LOCO to the boundary given in R2.

R2 = 2 half-word boundary
R2 = 4 word boundary
R2 = 8 double-word boundary

All other values of R2 cause no boundary alignment. LOCO is set to boundary by generating an instruction DSL with the corresponding length L.

MOK1 -- DD

This routine moves a constant from PSC into OBUF even if it is not fully contained in the I/O buffers. It moves the present part, reads new input into the input buffers, and moves the next part. This procedure is repeated until the constant is fully moved into OBUF.

ISCR -- DE

ISCR supervises the input buffers B1 and B2. It compares the pointer PSC with B2. If B2 is greater than PSC, the routine returns. If it is less or equal, the input is moved from B2 to B1 (with the length L = BUFL) and new input is read into B2. PSC is adjusted by subtracting BUFL, and the routine returns.

BRG -- DF

BRG is called by the routines OBRA and ADD. It prepares the elements generated in phase G00 for the branch instruction. It takes the offset from OFFTAB and inserts some keys into the instruction.

ISU, MOO, MODIF

These routines are described in the section General Description of Phases F95 - G55)

PHASES PL/IG17, B, D, E, R, S (FILE GENERATION) -- DJ

These phases generate the tables required for each file: the DTF appendages, the DTF tables, and the buffers.

These tables are generated in the form of assembler output cards (ESD, TXT, and RLD cards). For each file, a special control section is produced. The name of the control section is the file name. External references are the module name in each table and, depending on special file parameters, certain library routine names (e.g. the name of the end-of-file processing routine). The first part of the text is the appendage (from START to the label TABAD in the tables shown in Appendix D). For the format see description of the library. The following text applies to the normal logical IOCS DTF table, which is followed by the buffer areas, if required.

The above control sections are generated as follows: First, the addresses of the work spaces (buffers) to be used by the phases are established and the first entries of the file table and of the external name table (on the text output medium) are read. Reading and writing is performed in non-overlapped mode, using the external routines IJKARO and IJKAPI and the standard routine ZTIN. The phase itself provides a partial overlapping of these functions. A record count is maintained in register RD for the file table entries, and the processing is terminated when this table is exhausted. Then the file entries are scanned. The file table was constructed and written on SYS001 by phase B25. The external name table was constructed and written on SYS001 by phase B97. The latter table is written on TXTOUT by phase G15.

All file entries, which are not to be processed by the present phase are bypassed. In the first of the six phases, a bit is set in the communication region for each bypassed file entry. This bit later on initiates calling of the appropriate phase to process the entry. If a file entry to be processed by one of the phases of group G17 is encountered in the corresponding phase, a mask is selected according to certain file parameters, and the output pointer (register RE) is set to the address of this mask. Then the external name table is scanned for the external name of the file. The external name table is scanned record by record. Each record contains several entries. A new record is fetched whenever the previous record has been completely scanned.

If no matching file name is found, the file entry is associated to a file parameter and consequently ignored. Otherwise, the matching external name is inserted into the selected mask. The remaining part of the mask is set up according to the file table attributes. The EDIT routine is called to edit the modified mask into the text input medium, immediately following the program text. The next phase is fetched according to the setting of the communication bits. If the next phase is G20, an END card image is additionally produced and edited.

The EDIT routine writes one output record (card image: 80 bytes), starting with the byte selected by the output pointer. Then the output pointer is consequently incremented by 80.

Phase Input and Output

This phase uses the file table (see phase B25) and the external name table (see phase B97) as input. The external name table was written onto the text output medium by the preceding phase. The text input medium was not rewound so that the output cards can be written onto the medium, immediately following the text.

The output file tables are based on preassembled masks in the form of assembler output cards. All files are divided into several groups, each group is characterized by special file parameters. For each group, one mask is generated. The remaining file parameters, which may change and which are not required for the selection of the group, affect the mask internally. The file groups and the modification of masks are discussed later.

General Rules

The following rules apply to all files except those explicitly mentioned:

1. The file name is inserted as SD name in the first ESD card.
2. The CSECT length in the SD entry is incremented for all files except the unbuffered files.
   The increment value is:
   - one blocksize for files with BUFFERS(1);
   - two blocksizes for files with BUFFERS(2) and STREAM, and
   - two blocksizes + MOD (blocksize,8) for files with BUFFERS(2) and RECORD.

PL/IG17 (CARD, PRINT, UNBUFFERED FILES)

This phase processes all card, print, and unbuffered files.

CARD FILES

Four different masks are used for card files:

FILECDI :		Input card files
FILECD01:	1442	Output card files
FILECD02:	2520	Output card files
FILECD03:	2540	Output card files

The following rules apply to all card files:

Location	Action	
001E	AL1(0)	if SYSIPT, SYSPCH
	AL1(1)	if SYS000 to SYS245
001F	AL1(1)	if SYSIPT
	AL1(2)	if SYSPCH
	AL1(0) to AL1(245)	if SYS000 to SYS245
0028	7th character	Z if BUFFERS(1)
		I if BUFFERS(2)
	8th character	0 if 2540,INPUT
		1 if 1442
		2 if 2520
		3 if 2501
		4 if 2540,OUTPUT

Rules for FILECDI:

Location	Action	
002D	AL1(0)	if 2501
	AL1(1)	if 2540
	AL1(2)	if 2520
	AL1(32)	if 1442
	Add 4 to all values if BUFFERS(2)	
0030	The address is incremented	
	by blocksize	if BUFFERS(2) STREAM, and
	by blocksize + MOD (blocksize,8)	if BUFFERS(2), RECORD.
0038	4th parameter:	insert blocksize
0040	Insert LA 2,0(14)	if BUFFERS(2)

Rules for FILECD01, 2, 3:

Location	Action	
002D	AL1(16)	if BUFFERS(1)
	AL1(20)	if BUFFERS(2)
002E	AL1(65)	if 2520, 2540
002F	AL1(193)	if 1442
003A	Insert LA 2,0(14)	if BUFFERS(2)
003E	X'00'	if 2540
	X'01'	if 2520
	X'02'	if 1442
0040	1st parameter:	see location 002E
	2nd parameter:	the address is increased by blocksize
		if BUFFERS(2) STREAM, and
	by blocksize + MOD (blocksize,8)	if BUFFERS(2), RECORD
	4th parameter:	insert blocksize

PRINT FILES

Two different masks are used for print files:

FILEPRR : print files with record
FILEPRP : print files with print

The following rules apply to print files:

Location		Action	
FILEPRR	FILEPRP		
001E	0026	AL1(0)	if SYSLST
		AL1(1)	if SYS000 to SYS245
001F	0027	AL1(3)	if SYSLST
		AL1(0) to AL1(245)	if SYS000 to SYS245
0028	0030	7th character: Z	if BUFFERS(1)
		I	if BUFFERS(2)
002D	-	AL(16)	if BUFFERS(1)
		AL1(20)	if BUFFERS(2)
	0035	AL1(48)	if BUFFERS(1)
		AL1(52)	if BUFFERS(2)
003A	0042	NOP	if BUFFERS(1)
		LA 2,0(14)	if BUFFERS(2)
0040	-	2nd parameter: the address is increased by blocksize + MOD (blocksize,8)	if BUFFERS(2)
		4th parameter:	insert blocksize - 1
-	0048	2nd parameter: the address is increased by blocksize	if BUFFERS(2)
		4th parameter:	insert blocksize - 1

UNBUFFERED TAPE FILES

One mask is used for unbuffered tape files: FILETAUN. The following rules apply to this file group:

Location	Action	
0016	AL1(1)	if SYS000 to SYS245
	AL1(0)	if SYSIPT, SYSPCH, SYSLST
0017	AL1(1)	if SYSIPT
	AL1(2)	if SYSPCH
	AL1(3)	if SYSLST
	AL1(0) to AL1(245)	if SYS000 to SYS245
0025	Increase value by 16	if BACKWARDS, and by 128 if LEAVE
0028	Insert blocksize	
002A	Insert blocksize	
002C	X'02'	if without BACKWARDS
	X'0C'	if BACKWARDS
0038	X'00000000'	if without BACKWARDS
	X'00400000'	if BACKWARDS

UNBUFFERED DISK FILES

Two different masks are used for unbuffered disk files:

FILEDIUN: no UPDATE
FILEDIUU: with UPDATE.

The following rules apply to unbuffered disk files:

Location		Action
FILEDIUN	FILEDIUU	
0026	002E	Insert filename
0038	0040	Insert blocksize
004E	0056	Insert blocksize
0050	0058	Increment value by 64 if VERIFY
0054	005C	Increment value by 16 if VERIFY

PL/IG17B (BUFFERED TAPE FILES)

This phase processes all tape files except the unbuffered ones. Eight different masks are used for buffered tape files:

FILETAFI: Input tape files with fixed records
FILETAFO: Output tape files with fixed records
FILETASP: Tape files with PRINT option
FILETAFB: Tape files with BACKWARDS option
FILETAVI: Input tape files with variable records
FILETAVO: Output tape files with variable records
FILETAUI: Input tape files with undefined records
FILETAUO: Output tape files with undefined records

The following rules apply to all buffered tape files:

Location		Action	
FILETASP	All others		
0026	001E	AL1(0)	if SYSIPT, SYSPCH, SYSLST
		AL1(1)	if SYS000 to SYS245
0027	001F	AL1(1)	if SYSIPT
		AL1(2)	if SYSPCH
		AL1(3)	if SYSLST
		AL1(0) to AL1(245)	if SYS000 to SYS245
0034	002C	X'11'	if NOLABEL
		X'12'	if OUTPUT without NOLABEL
		X'13'	if INPUT, BACKWARDS without NOLABEL.
		X'14'	if INPUT without BACKWARDS and NOLABEL
0035	002D	Increment by 64	if blocksize ≠ record size or if VARIABLE
		by 32	if BUFFERS (2)
		by 4	if BACKWARDS
0036	002E	Insert file name	
0040	0038	Increment value	
		by 128	if NOLABEL not specified,
		by 16	if LEAVE,
		by 8	if BACKWARDS specified.

Rules for FILETAFI, FILETAFO, FILETASP, FILETAFB:

Location		Action	
FILETASP	All others		
0054	004C	Insert NOP if neither blocksize unequal to record size nor BUFFERS(2) specified.	
0058	0050	2nd parameter:	
		add (blocksize -1)	if BACKWARDS
		4th parameter:	
		insert blocksize	
0060	0058	Add to address blocksize	if BUFFERS(2) STREAM,
		blocksize + MOD (blocksize, 8)	if BUFFERS(2) RECORD,
		add extra (blocksize -1)	if BACKWARDS.
0064	005C	Add (blocksize - record size)	if BACKWARDS,
		blocksize	if OUTPUT, BUFFERS(2), STREAM
		blocksize + MOD (blocksize, 8)	if OUTPUT, BUFFERS(2) RECORD
0068	0060	Insert (-recordsize)	if BACKWARDS,
		recordsize in all other cases.	
006C	0064	Add (blocksize - recordsize)	if BACKWARDS,
		blocksize	if OUTPUT, BUFFERS(1),
		2 blocksizes	if OUTPUT, BUFFERS(2), STREAM
		2 blocksizes+MOD(blocksize,8)	if OUTPUT, BUFFERS(2), RECORD
0070	0068	Insert blocksize	
0072	006A	Insert (blocksize + 1)	if BACKWARDS,
		(blocksize -1) in all other cases	
0074	006C	Insert (recordsize -1).	

Rules for FILETAVI and FILETAVO:

Location	Action	
0050	4th parameter: insert blocksize	
0058	Add blocksize + MOD (blocksize, 8)	if BUFFERS(2)
005C	Insert blocksize	
0060	Add blocksize + MOD (blocksize,8)	if BUFFERS(2)
006C	Add blocksize + MOD (blocksize,8)	if BUFFERS(2)
0070	Insert blocksize -4	if OUTPUT,
	blocksize	if INPUT
0076	Insert (blocksize -1)	

Rules for FILETAUI and FILETAUO:

Location	Action	
004C	Insert NOP	if neither BACKWARDS nor BUFFERS(2)
0050	1st parameter:	
	X'01'	if OUTPUT
	X'02'	if INPUT without BACKWARDS
	X'0C'	if INPUT, BACKWARDS
	2nd parameter:	
	add (blocksize -1)	if BACKWARDS
	4th parameter: insert blocksize	
0058	Add blocksize + MOD (blocksize, 8)	if BUFFERS(2)
	add extra (blocksize -1)	if BACKWARDS
005C	Add blocksize + MOD (blocksize, 8)	if BUFFERS(2)
0060	Insert BCTR 14,0	if BACKWARDS
	NOPR	
0064	Insert blocksize	
0066	Insert (blocksize -1)	

PL/IG17D, E (BUFFERED CONSECUTIVE DISK FILES

These two phases process all consecutive disk files except the unbuffered ones. Ten different masks are used for buffered, consecutive disk files:

FIDIINFI:	Input disk file, fixed records
FIDIINVA:	Input disk file, variable records
FIDIINUN:	Input disk file, undefined records
FIDIOUFI:	Output disk file, fixed records
FIDIOUPR:	Disk file with PRINT option
FIDIOUVA:	Output disk file, variable records
FIDIOUUN:	Output disk file undefined records
FIDIUPFI:	Update disk file, fixed records
FIDIUPVA:	Update disk file, variable records
FIDIUPUN:	Update disk file, undefined records

Exception to General File Rules:

The CSECT length is additionally increased by 8 bytes for all disk files with OUTPUT, BUFFERS(1), and by 16 bytes for all disk files with OUTPUT, BUFFERS(2).

Rules for all Buffered, Consecutive Disk Files:

Location		Action
FIDIOUPR	All others	
0035	002D	Increment value by 64 if blocksize unequal to recordsize
0036	002E	Insert file name
004C	0044	If INPUT/UPDATE, BUFFERS(2), STREAM, add blocksize to address. If INPUT/UPDATE, BUFFERS(2), RECORD, add blocksize + MOD (blocksize, 8) to address.
0062	005A	If OUTPUT, insert blocksize
0068	0060	If FIXED, insert records per track (RT: see below). Otherwise insert X'FF'.
006A	0062	Insert (blocksize -1)
0074	006C	If BUFFERS(1) and blocksize is equal to recordsize, insert NOP.
0080	0078	If OUTPUT, add (blocksize +8) to address. Otherwise, add blocksize to address.

0084	007C	Increment value
		by 4 if VARIABLE, INPUT/UPDATE;
		by 8 if FIXED, INPUT/UPDATE, BUFFERS(2);
		by 16 if VERIFY, OUTPUT/UPDATE

00A0	0098	2nd parameter: Add (blocksize +8) to address if OUTPUT, BUFFERS(2), STREAM. Add (blocksize +8) + MOD (blocksize, 8) to address if OUTPUT, BUFFERS(2), RECORD. 3rd parameter: Insert 64 if OUTPUT, VERIFY 4th parameter: Insert blocksize if INPUT/UPDATE. Insert (blocksize +8) if OUTPUT.

Rules for Evaluation of RT (Records per Track):

Length of normal record:
LNR = [blocksize multiplied by 537/512 + 61]

Normal records per track:
NRT = [3625/LNR]

Number of records per track for INPUT/UPDATE:
RTI = NRT or
RTI = NRT + 1 if 3625 - NRT multiplied by LNR ≥ blocksize

Number of records per track for OUTPUT: RTO = RTI - 1

PL/IG17R, S (REGIONAL DISK FILES)

These two phases process all regional disk files. Ten different masks are used for regional disk files:

FIDIINR1:	Input disk files, regional 1
FIDIONR1:	Output disk files, regional 1, no verify
FIDIOVR1:	Output disk files, regional 1, with verify
FIDIUNRI:	Update disk files, regional 1, no verify
FIDIUVRI:	Update disk files, regional 1, with verify
FIDIINR3:	Input disk files, regional 3
FIDIONR3:	Output disk files, regional 3, no verify
FIDIOVR3:	Output disk files, regional 3, with verify
FIDIUNR3:	Update disk files, regional 3, no verify
FIDIUVR3:	Update disk files, regional 3, with verify

Exception to General File Rules

For FIDIINR3, the CSECT length is increased by blocksize + key length.For FIDIONR3, FIDIOVR3, FIDIUNR3, and FIDIUVR3, the CSECT length is increased by blocksize + key length + 8.

Rules for all Regional Files:

Location	Action
0010	If REGIONAL 3, OUTPUT/UPDATE, add (keylength + 8) to address.
0018	If INPUT, REGIONAL 3, add blocksize to address.
001C	Insert RT if REGIONAL 1
0024	Insert keylength
005E	Insert filename
0094	Insert blocksize
00B0	2nd parameter: If INPUT, REGIONAL3 add blocksize to address. 4th parameter: insert keylength if INPUT/UPDATE, REGIONAL3.
00B6	If OUTPUT, REGIONAL 3, insert keylength.
00B8	2nd parameter: add (keylength + 8) to address if OUTPUT/UPDATE, REGIONAL 3 4th parameter: insert blocksize.
00D0	If OUTPUT/UPDATE, REGIONAL 3, insert blocksize + keylength. Otherwise insert blocksize.
00F0	4th parameter: If OUTPUT/UPDATE, REGIONAL3, insert blocksize + keylength + 8.

This phase moves the block table from SYS001 into the table space. The label table is written from SYS001 onto TXTIN. If no files are declared, the end of the phase is reached.

If files do exist, ESD, TXT, and RLD cards for each user- defined file are processed.0 These cards are stored on TXTIN in the preceding sequence. Following the cards for all files, only one END card is generated. The output of G17 is changed in the following manner:

First, all ESD cards are sorted out and written onto SYS001. Then all TXT cards are written behind the ESD cards, and finally all RLD cards followed by the END card are written onto SYS001. The ESID numbers of the cards are changed to provide only one module for the files.

Phase Input and Output

Input:

1.	TXTIN :	The text (i.e., the changed source program) is located on TXTIN. Information about the existing files is written following the text. This information consists of ESD, TXT, and RLD cards and one END card as described in phase G17. TXTIN is positioned at the end of the text if no files exist and at the end of the files if files do exist.
2.	TXTOUT :	The external name table is located on TXTOUT, which is positioned at its beginning.
3.	SYS001 :	The block table and the label table are located on SYS001.

Output:

1.	TXTIN :	At the end of this phase, the unchanged text and the unchanged files followed by the label table are contained on TXTIN, which is positioned at the beginning of the label table.
2.	TXTOUT :	The external name table is contained on TXTOUT, which is positioned at its beginning.
3.	SYS001 :	The work file SYS001 contains the updated files.

The block table is contained in the first 128 bytes of the table space.

Interphase

1. The NOTE information about the beginning of the files written on TXTIN is contained in ZTAB19.

2. The NOTE information about the end of the external name table written on TXTOUT is contained in ZTAB18.

3. The NOTE information about the end of the DTF table writen on SYS001 is contained in KSAVE8.

4. If the third bit in the first byte of ZTAB03 is 1, files do exist.

5. At the end of this phase, the number of the cards which are still in the output buffer and have not been written on SYS001 is stored in the first byte behind the output buffer.

Main Routine -- DO and DR

This routine moves the block table into the table space and writes the label table onto TXTIN. SYS001 is reset to the end of the DTF table and, if no files exist, the end of the phase is called; otherwise, the files are read in.

The ESD cards are changed and written onto SYS001, the TXT cards that have been processed are written onto TXTOUT, and the RLD cards are stored in the table space. If an overflow occurs in the table space, the RLD cards are written onto TXTIN. If the END card is detected, the TXT cards are moved from TXTOUT onto SYS001 and the RLD cards are written onto SYS001. Finally, the END card is written on SYS001. Each card written on SYS001 is given a consecutive card number in colunms 77-80 and the identification FILE in columns 73-76. At the end of this phase, TXTIN is positioned at the beginning of the label table, and TXTOUT is repositioned at its beginning.

The parameters for ZTIN are loaded and the block table record indicated by ZTAB13 is read into the table space. Control is transferred to the routine LABTAB, which moves the label table onto TXTIN. The tables contained on SYS001 are saved or no longer used; therefore, the work file is positioned at the end of the DTF table, and the other tables may be overwritten. If files exist, routine INI2 is called. TXTOUT is set to the first free record following the external-name table, and TXTIN is set at the beginning of the files.

Buffer handling is as shown in Figure 1.

```
    TS           TSB                                   B0
    ↑            ↑                                      ↑
    |            |                                      |
    +------------+------------------------------   -----+
    | block table | table space used for RLD cards      |
    +------------+------------------------------   -----+
    |            |                                      |
    IJKMTS       IJKMTS+128
```

Input Buffers:

```
B0          B0+80          B1            B1+80  B3   BA
|           |              |             |      |    |
+-----------+--------------+-------------+------+----+
|           | input files  | output TXT  |free  |free|
+-----------+--------------+-------------+------+----+

IJKMBS+2 * IJKMBL                        IJKMBS+ 3+IJKMBL
```

Output Buffer:

```
BA                                            BB or B5

|                                                    |
+------------+-------------+------------+------------+
|            |             |            |            |
+------------+-------------+------------+------------+
IJKMBS+3 * IJKMBL+8                            BA+320
                                   320 BYTES = 4 CARDS
```

Figure 1. Buffer Handling

2. Processing of the files.

   The first card of the file is read into the input buffer at B0+80, and the program enters a loop as follows:

   LOOP: The card in B0+80 is moved into buffer B0 and the next record is read into B0+80. The card type in B0 is checked:

   a. An ESD card is detected: Subroutine ESD changes the ESID numbers in the ESD card and subroutine MOOK moves the card into the output buffer BA.
   b. A TXT card is detected: The ESID number of the last ESD card that contains an SD entry is contained in SDNO. It is put into the TXT card. The card is moved into output-buffer B1 and written onto TXTOUT.
   c. An RLD card is detected: Subroutine RLD changes the ESID numbers in the RLD cards, and subroutine MOR moves the card into the table space or onto TXTIN.
   d. If an END card is detected, the end of the loop is reached.

   Subroutine TXTSYS moves the TXT cards behind the ESD cards; subroutine RLDSYS moves the RLD cards behind the TXT cards. The number of those cards that are still in output buffer BA (contained in POUS) is stored in the first byte behind B5. TXTIN is set to the beginning of the label table and, after repositioning TXTOUT to its beginning, phase G25 is called.

INI1 -- DT

This routine initializes some pointers. These pointers are used mainly to move the label table and the block table. All pointers, counters, and values used for handling files are initialized in INI2. The transfer bits of the block table (ZTAB13) and the label table (ZTAB20) are set to 0. The initial buffer address (IJKMBS) is increased by 2 * IJKMBL and stored in B0. B0 is increased by IJKMBL and stored in B1.

INI2 -- DU

This routine is called if files exist. It evaluates the buffer start addresses and initializes some counters and pointers used to process the files.

The address of the output buffer for the TXT cards is stored in B1 (B1 = B0+160).

The address of the output buffer for the sorted cards is stored in BA (BA = B0+IJKMBL+8), and the end of this buffer is stored in BB (BB = BA+320).

The TABTAB entry of the output (ZTAB16) is updated. The buffer length is changed to 320; the transfer bit is set to 0. SDNO, which contains the ESID number of the last ESD card with an SD entry, is set to 0, and ESID, which contains the current ESID number, is set to 1. The output pointer CSP is set to BA, TSB is evaluated, and SPEI is set equal to TSB.

LABTAB -- DV

This routine moves the label table from SYS001 onto TXTIN.

The beginning of the label table on TXTIN is noted by the NOTE macro, and the number of label table records is moved from ZTAB20 into register R3.

If R3 is 0, the label table does not exist, and the routine returns to the main routine. Otherwise, the first label table record is read into buffer B0.

Buffers:

```
┌────────┬──────────────────────────────────┐
│        │                                  │
└────────┴──────────────────────────────────┘
↑        ↑                                  ↑
B0       B1 = B0+IJKMBL             B1+IJKMBL
```

Both buffers have the buffer length and both are used alternately as input and output buffers. Next, the program enters a loop as follows:

First, R3 is decreased by 1 and a test is performed to determine if it is equal to 0. If it is 0, the end of the loop is reached; otherwise, the next record is read into B1 or B0 and the preceding record is written onto TXTIN from B0 or B1, respectively.

The program waits for the end of the last reading. Then the last record is written onto TXTIN, the end of the label table is noted, and the routine returns control back to the main routine to the point where it was called.

ESD -- DW

This routine inserts the current ESID number contained in ESID into the ESD cards. SDNO is updated and ESID is increased by the number of entries in the ESD card.

This routine is called if an ESD card is identified. If its first entry is an SD entry, SDNO, (the ESID number of the last ESD card containing an SD entry) is set to the current value of ESID. ESID is inserted into the ESD card and increased by the number of bytes contained in the card divided by the length of one ESD entry. Then the routine returns to the main routine.

MOOK -- DX

This routine moves the cards that have been processed into the output buffer BA after inserting the successive card number. If the buffer is filled, a 320-byte record is written on SYS001.

This routine is called if a card that has been processed has to be moved from the place pointed to by R1 into the output buffer BA. The pointer CSP points to the next available byte in the buffer. NOS contains the current card number in binary representation. It is increased by 1, and, after it was converted to decimal, it is moved into the last four columns of the card. The identification ;FILE; is put into columns 73-76. Then the card is moved into the output buffer, CSP is increased by 80, and POUS, the counter of the cards in BA, is increased by 1.

If CSP does not point to the end of the output area, the end of the routine is reached; otherwise, the four cards contained in the buffer are written on SYS001. CSP is reset to BA, POUS is cleared, and the program waits for the end of the write operation before the end of the routine is reached.

RLD -- DY

This routine updates the relocation and position headers of the RLD cards.

This routine is called when an RLD card is found in buffer B0.

First, the ESID numbers of the first entry are updated. The first ESID number, the relocation header, is increased by SDNO-1 and the second, the position header, is set to SDNO. The next entries are handled in the following manner:

If the flag byte is on, the four-byte entry is skipped. If the flagbyte is off, the entry is handled like the first entry. If all entries are processed, the routine returns to the main routine.

MOR -- DZ and EA

This routine moves the RLD cards out of B0 into the table space. If the area reserved for this purpose is filled, the cards are written in records of 1920 bytes onto TXTIN following the label table.

SPEI (the current pointer for the area TSB reserved for the RLD cards) is increased by 80 and then compared with B0. If SPEI+80 is not greater than B0, the RLD card is moved into the area, SPEI is increased by 80, and the end of this routine is reached.

If SPEI+80 is greater than B0, the current position of TXTIN is noted and the file is set to the end of the label table or to the end of the last RLD card written on TXTIN. Next, the contents of the table space are divided into 1920-byte records which are written onto TXTIN. Left over cards are moved to the beginning of the area TSB and the RLD card in B0 is moved behind them. The end of the last record written on TXTIN is noted and the file is reset to the current position in the file module. ZEHLS, which counts the number of records put onto TXTIN, is updated and SPEI is set to the location of the next available byte in the table space. Next a branch is made to the beginning of this routine and the next RLD card is processed.

TXTSYS -- EB

This routine moves the TXT cards written on TXTOUT onto SYS001 behind the ESD cards.

This routine is called after the END card is found in the buffer B0.

The TXTOUT work file is repositioned to the beginning of the TXT cards. They are read into buffer B1, and moved by subroutine MOOK into output buffer BA. If all cards are read in and processed (ZAHLS = 0), the routine returns to the main routine.

RLDSYS -- EC and ED

This routine moves the RLD cards from the table space or TXTIN onto SYS001 behind the TXT cards.

This routine is called by the main routine. Two cases have to be distinguished.

1. If SWI = 0, (i.e., no RLD cards have been written on TXTIN) the RLD cards contained in the table space are put onto SYS001 by subroutine MOOK. When all cards have been processed, the program branches back to the main routine.

2. In the second case, RLD cards have been written on TXTIN by subroutine MOR. The file is positioned to the end of the last record. The RLD cards still contained in the table space are moved onto TXTIN (buffer length = 1920) and the number of written records is updated. Then the file is repositioned to the beginning of the RLD cards and the cards are moved from TXTIN onto SYS001, controlled by the counter of the RLD records and the counter of the cards contained in the last record. After having moved all records, the routine returns to the main routine.

The function of this phase is to generate ESD (External Symbol Dictionary) cards The format of ESD cards is shown in Figure 1. ESD cards are generated in the following cases:

1. Program:
   One ESD-SD card for the program control section including the STATIC storage.

2. External references:
   One ESD-ER card for each of the following:

   a. Library routine used during object time
   b. External procedure referenced by this compilation
   c. File name

3. Entries:
   One ESD-LD card for each entry point of the external procedure.

4. External variables:
   One ESD-SD for each external variable.

Column	Contents
1	Multiple punch
2- 4	ESD
11-12	Number of bytes of information contained in this card
15-16	ESID number of the first SD or ER on this card
17-72	Variable information: 8 positions - name 1 position - Type code to indicate SD,LD or ER 3 positions - assembled origin 1 position + Blank 3 positions - Length, if SD type. If an LD type, this field contains the external symbol identification.
73-76	First four characters of the program name or zeros
77-80	Sequential card number

Figure 1. Format of the ESD Card

In the first part of this phase (general flow charts EH-EI) the external name table EXTAB is scanned and the following ESD cards are produced:

1. On finding an entry name with the block and level number 0, the ESD-SD card for the program is produced.

2. If the entry name has the block number 1 and the level number 0, this is a secondary entry point of the external procedure and an ESD-LD card is produced.

3. On finding an entry name with a level and block number different from 0,0 and 0,1 an ESD-ER card must be produced.

4. On finding a file name, an ESD-ER card is produced.

5. Entries for built-in functions are skipped.

6. Other entries are considered as external variables and ESD-SD cards are produced.

If the external variable is a structure with a lefthang different from 0, an ESD-SD card with a generated name and an ESD-LD card with the name of the structure is produced.

If lefthang is 0, only an ESD-SD card is generated.

In the second part of the phase (general flow chart ET) the ESD-ER cards for the library routines are produced. The library bit string in the communication area indicates which routines are needed during object time.

Appendix XYZ contains a listing of the library routines and their corresponding internal names in decimal and hexadecimal representation.

<u>Phase Input:</u>

1. Label table LABTAB on the TXTIN work file. The format of the LABTAB entries is as follows:

Bytes 0-1	:	Internal name of the label or entry name
Bytes 2-3	:	Offset from program begin
Byte 4	:	Block number

2. External name table EXTAB on the TXTOUT work file.

3. Block table BLTAB in the table space at IJKTS. Length of BLTAB: 128 bytes.

4. Library bit string in the table space at IJKTS+128. Length of library bit string: 32 bytes.

5. Length of the control section PROGRAM+ STATIC STORAGE in the first 4 bytes of the TABTAB entry ZTAB05.

6. Remaining ESD, TXT, and RLD cards and the END card which are in a 320-byte buffer at BUFF3 of the buffer area.

7. Information bytes needed by the phase, at BUFF3+320.

Phase Output:

1. ESD table on SYS001 (TABTAB entry = ZTAB16). If the last I/O buffer is not full, the remaining ESD cards are stored at BUFF3. The number of remaining ESD cards is stored at BUFF3+320.

2. ESID table containing a maximum of 255 entries. The ESID entries have the following format:

   Bytes 0-1 : Internal name
   Byte 2 : ESID number

   The start address of the ESID table is stored at BUFF3+324

3. Block table containing the lengths of DSA's (Dynamic Storage Areas). This table, located in the first 128 bytes of table space, is not changed by this phase. The address of the block table is stored in BUFF3+328.

4. If the number of ESD cards is greater than 255, bit 4 in IJKMWC is set to 1 and phase G31 is called.

DESCRIPTION OF ROUTINES

Symbols used in flow charts:

ABLT :	Address of block table
ALIB :	Address of library bit string
AESID:	Address of ESID table
EXBU1:	Input buffer for EXTAB
EXBU2:	Input buffer for EXTAB
LABU1:	Input buffer for LABTAB
LABU2:	Input buffer for LABTAB
ESBU :	ESD output buffer
REC1 :	Number of input records required
CACN :	Number of cards in ESD buffer
BCDP :	BCD pointer
SVAR :	ESD variable field

INIT1 -- EJ

The initialization routine INIT1 determines the addresses of the buffers and tables used during the phase. The contents of the table space (starting address TS) and the buffer area (starting address BS) are shown in Figure 2.

Location	Contents	Length
TS+0	Block table BLTAB	128
TS+128	Library bit string LIBTAB	32
TS+160	ESID-table ESIDTAB	max. 765
TS+925	Input buffers for EXTAB	2 buffer lengths
BS+1BUFFL	Input buffers for LABTAB	2 buffer lengths
BS+3BUFFL	ESD output buffer	320
BS+3BUFFL +320	Interphase communication	20

Note:
TS = Table space starting address
BS = Buffer area starting address
BUFFL = Buffer length

Figure 2. Contents of the Table Space and the Buffer Area

GETEX -- EK

This subroutine reads one or two records of the external name table EXTAB from the TXTOUT medium according to the specification in REC1. If only one record is required, buffer EXBU2 is moved to EXBU1 and EXBU2 is filled up.

GETLA -- EK

This subroutine reads 2 records of the label table LABTAB from the TXTIN work file and waits for I/O termination.

MESID -- EL

This subroutine moves the internal representation of the external name and the ESID number to the ESID table area. The output pointer OPT2 is increased by the length of the ESID entry.

ESFIN -- EM

This subroutine controls the variable field in the ESD card, i.e., columns 17-60. If the ESD card is full or if the last ESD entry was made, the routine ESMO is called in order to move the ESD card into the output buffer ESBU.

SDPRO -- EL

This subroutine moves the following information into the ESD entry SVAR:

1. Program name (8 bytes) as given in the external name table

2. Origin = 000

3. Type of ESD entry = X'00'

4. Length of the control section including program and STATIC storage. The length is contained in the TABTAB entry ZTAB02.

Finally, the origin is increased by the length of the section and aligned on double-word boundary. The ESID numer is increased by 1. The length of the variable field in the ESD card is increased by 16 and routine ESFIN is called in order to move the entry into the output buffer.

ESMO -- EN

The ESD card produced at ESENT is moved into the output buffer ESBU. If the output buffer is full, it is written on a work file. Otherwise, the output pointer OPT1 is increased by 80 and the routine returns to the main program.

LDXEC -- EO

This subroutine stores the LD information for the secondary entry points of the external procedure in the variable field of the ESD card:

1. External name (8 bytes)

2. Type code to indicate LD = X'01'

3. Assembled origin (3 bytes) which is taken from the label table LABTAB by means of subroutine LABS.

4. ESD number which is always X'000001'

LABS -- EO

The input parameter for this routine is the name of the secondary entry point of the external procedure in internal representation. This subroutine scans through the label table LABTAB, and, on finding the corresponding name of the secondary entry point, the offset is fetched and stored in the ESD card.

ERCAL -- EP

This subroutine moves the following information into the variable field of the ESD card:

1. External name

2. Type code to indicate ER

3. Assembled origin (must always be X'000000').

An entry in the ESID table is made and the ESID number is increased by 1.

SDLDS -- EQ

If the external structure has a lefthang of 0, only an ESD-SD card is produced in subroutine SDPRO. Otherwise, an ESD-SD card is produced for which an external name must be generated, and an ESD-LD card is made for the structure name.

PAD -- ER

This subroutine generates a name for the ESD-SD card of an external structure. The user-defined name of the external structure must not be longer than 6 characters. If it is exactly 6 characters, the leftmost 2 bytes of the 8-byte name in the ESD card are padded with X'5B5B'. If the user-defined name is shorther than 6 bytes, it is moved right-aligned into the ESD card, and the leftmost bytes are padded with X'5B'.

SKIP -- ES

This subroutine increases the input pointer INPT1 by the length of the EXTAB entry. To ensure that the second input buffer EXBU2 is read, the routine waits for completion of input from the TXTOUT work file to determine if the input pointer is greater than EXBU2-30. If the input pointer is greater than EXBU2, buffer 2 is moved into buffer 1 and the next record is read from buffer 2.

LIBER -- ET

In the second part of phase G25, the ESD-ER cards are produced for the library routines needed during object time. The library bit string located at IJKMLB in the communication region indicates the library routines for which ESD cards must be produced. The BCD names of the library routines have a length of 6 or 7 bytes. The first 3 characters are always IJK. The variable characters are listed in the library table and are inserted in the ESD card.

ERLI -- EU

This subroutine moves the following information into the variable field of the ESD card:

1. BCD name of the library routine.

2. Type code to indicate ER

3. Assembled origin must always be X'0000'.

An entry in the ESID table is made, and the ESID number is increased by 1.

PHASE PL/IG30 (GENERATION OF TXT AND RLD CARDS) -- FH

This phase generates the TXT and RLD cards for the program module.

TXT Cards

TXT cards (see Figure 1) contain machine instructions and constants.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2- 4	TXT. Identifies the type of load card.
5	Blank.
6- 8	24-bit start address in storage where the information from the card is to be loaded (in extended card code).
9-10	Blank.
11-12	Number of bytes of text to be loaded from the card (in extended card code).
13-14	Blank.
15-16	External Symbol Identification (ESID.) number, in extended card code, assigned to the program segment in which the text occurs. (Here 01)
17-72	A maximum of 56 bytes of instructions and/or constants assembled in extended card code.
73-80	Not used by the loader; used for identification: the first 4 characters of the external procedure name followed by a sequence number.

Figure 1. Format of the TXT Card

RLD Cards

RLD cards are generated for all address constants. They contain the location of the constant in reference to the corresponding TXT card, the ESID numbers for the reference (relocation header), and the position (position header). Figure 2 shows the format of the RLD card.

The length of the information to be put in the RLD card for each constant is 8 bytes or 4 bytes depending on whether the relocation and position headers change or not. If they do not change, the headers must not be repeated. This is flagged by the continuation flag bit. The information for a maximum number of 13 address constants may be contained in one card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2- 4	RLD. Identifies the type of load card.
5-10	Blank.
11-12	Number, in extended card code, of bytes of information in the variable field (card columns 17-72) of this card. The range is from 8 to a maximum of 56.
13-16	Blank.
17-72	Variable field (in extended card code). Consists of the following subfields: 1. <u>Relocation Header</u>. (Two bytes) An ESID with a value of from 01 through 256. Whether or not the value is 01 or from 02 through 256 depends on whether the symbol it points to is internal or external to the particular program segment. 2. <u>Position Header</u>. (Two bytes) The ESID assigned to this program segment. 3. <u>Flag Byte</u> (bits 0 through 3 are not used). This byte contains three items: a. <u>Size</u>. (Bits 4 and 5) Two bits which indicate the length (in bytes) of the adjusted address cell (AA cell) a. 00 - one-byte cell b. 01 - two-byte cell c. 10 - three-byte cell d. 11 - four-byte cell b. <u>Complement Flag</u>. (Bit 6) When this bit is 1, it means that the value (or address) of the symbol is to be subtracted from the contents of the AA cell. When this bit is 0, the value of the symbol is to be added to the contents of the AA cell.

Figure 2. Format of the RLD Card (Part 1 of 2)

Column	Contents
	c. Continuation Flag. (Bit 7) When this bit is 1, it means that this is one of a series of addresses to be adjusted. When this bit is 0, this is the only AA cell to be adjusted or the last in a series using the same relocation and position headers.
	4. Address. The three-byte address of the location of the AA cell. The flag byte and address may be repeated for AA cells as long as the continuation flag bit is on in the current four-byte entry.
73-80	Not used by the loader; used for identification: the first 4 characters of the external procedure name followed by a sequence number.

Figure 2. Format of the RLD Card (Part 2 of 2)

END Card

The last card to be generated is the END card. Figure 3 shows the format of the Load End card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2- 4	END. Identifies the type of load card.
5	Blank.
6-8	Address (may be blank), in extended card code, of the point in the program segment to which control may be transferred at the end of the loading process.
9-14	Blank.
15-16	External Symbol Identification (ESID). (May be blank.)
17-72	Blank.
73-80	Used for identification as in the ESD, TXT, and RLD cards.

Figure 3. Format of the Load End Card

If there is a main procedure (flagged in the communication area), the address columns will be set to 0 and the ESID number is set to 01. In all other cases the address columns and columns for the ESID number contain blanks.

The cards are constructed in an 80-byte mask and written on SYS001 in physical records of 320 bytes each. The cards for the files and the ESD cards of the program module are already present on SYS001. All cards are used as input for the final output phase (G55).

There are two modules, one for files and one for the program and the external variables. A module may consist of several control sections.

```
              ┌──
              │
              │  ESD - cards
              │  TXT - cards
              │  RLD - cards
              │
MODULE1       │                ( FILES )
              │
              │
              │
              │  END - card
              │
              └──

              ┌──
              │
              │  ESD - cards
              │  TXT - cards   ( PROGRAM )
              │  RLD - cards
              │
MODULE2       │
              │
              │
              │
              │  END - card
              │
              └──
```

General Flow of the Phase -- FH

The program string is scanned to determine whether an element is a machine or an assembler instruction.

For machine instructions the format is determined. The instruction is moved (according to the format) into the current TXT-card mask which is moved into the output buffer if it is full.

For assembler instructions the kind of instruction is determined first. The corresponding information is inserted in the current TXT card. For address constants, the current location counter (LOC1) is stored in a save buffer together with the corresponding ESID numbers.

When all TXT cards have been generated, i.e., when the END-key is found, the RLD cards are generated from the records in the save buffer.

All cards are written on SYS001 in 320-byte records.

Phase Input/Output

The input consists of the program string and three tables: the ESD table, the character-string and the BLOCK table. The program string is located on TXTIN, followed by the character-string and is the output of phase G15 where it has been described. The program string is read sequentially using buffers B1 and B2 as input buffers with the pointer POI.

The ESID and BLOCK tables are inserted in the table space by the phases G25 and G20, respectively. Their addresses are located in the second and third word of the master table TABTAB. The ESID table is restricted to 255 3-byte entries with the following format:

```
         2        1
      ┌──────────────┐         ┌───────┐
ESID  │INTERNAL  ESID│  BLOCK  │LENGTH │
      │ NAME      NO │         │OF DSA │
      ├──────────────┤         ├───────┤
      │   .       .  │         │   .   │
      │   .       .  │         │   .   │
      │   .       .  │         │   .   │
      │              │         │       │
```

The BLOCK table is restricted to a maximum of 63 2-byte entries each containing the length of a DSA.

If the save buffer B0 is full, it is written on TXTOUT intermediately. The information is read in again for generation of the RLD cards using the input buffers B1 and B2.

The output is written on SYS001 just behind the ESD cards constructed in phase G25. The output consists of TXT cards, RLD cards, and the END card.

For output, two 320-byte buffers are used with the pointer CPO. Which of the two buffers is used is determined by register ABU and switch AE. One buffer is filled while the other is written on SYS001. The first buffer is located at B3+8, and the second one is at the end of the phase.

If phase G25 does not fill the last 320-byte record, this record is left in the I/O buffer and is filled with TXT cards in phase G30.

Detailed Description

The scan of the text string is handled as in previous phases. It must be determined whether an element is a machine instruction or an assembler instruction.

Machine Instructions

The format is determined first. Then the information contained in the instructions is inserted (according to the format) in the TXT-card mask which is moved into the output buffer if it is full.

1. RR instructions

   a.

```
┌────┬────┬────┬────┐
│    │OP. │    │    │
│ 88 │C.  │ 0X │ 0Y │
└────┴────┴────┴────┘
```

   Format of the instruction. (4 bytes)

   b.

```
┌────┬──┬──┐
│OP. │  │  │
│C.  │X │Y │
└────┴──┴──┘
```

   Information inserted in the TXT card (2 bytes)

2. RX and RS instructions

   a.

```
┌────┬────┬────┬────┬────┬─────────┐
│    │Op. │    │    │    │         │
│88  │C.  │ 0X │ 0Y │ 10 │  NAME   │
└────┴────┴────┴────┴────┴─────────┘
┌─────────┬──┬──────┐
│MODIFIER │B │DISPL.│   (11 bytes)
└─────────┴──┴──────┘
```

   b.

```
┌────┬──┬──┬──┬──────┐
│Op. │  │  │  │      │
│C.  │X │Y │B │DISPL.│   (4 bytes)
└────┴──┴──┴──┴──────┘
```

3. SI instructions

   a.

```
┌────┬────┬────┬────┬────┬────┐
│    │Op. │    │    │    │    │
│ 88 │C.  │ 00 │  I │KEY │NAME│
└────┴────┴────┴────┴────┴────┘
┌────────┬──┬──────┐
│MODIFIER│B │DISPL.│   (11 bytes)
└────────┴──┴──────┘
```

   b.

```
┌────┬──┬──┬──────┐
│Op. │  │  │      │
│C.  │ I│ B│DISP. │   (4 bytes)
└────┴──┴──┴──────┘
```

4. SS instructions

a.

```
┌─────┬─────┬─────┬─────┬────┬────────────┐
│88   │Op.  │0L1  │ 0L2 │KEY │   NAME     │
│     │C.   │     │     │    │            │
└─────┴─────┴─────┴─────┴────┴────────────┘
┌──────────┬──┬───────┬────┬────────────┐
│MODIFIER  │B │DISPL. │KEY │   NAME     │
└──────────┴──┴───────┴────┴────────────┘
┌──────────┬──┬───────┐
│MODIFIER  │B │DISPL. │   (18 bytes)
└──────────┴──┴───────┘
```

b.

```
   (1)   (1)     (2)      (2)
┌─────┬──┬──┬─┬──────┬─┬──────┐
│Op.C.│L1│L2│B│DISPL.│B│DISPL.│ (6 BYTES)
└─────┴──┴──┴─┴──────┴─┴──────┘
```

For all instructions the location counter is adjusted in the same way.

Abbreviations:

X, Y	are registers,
B	is the base register,
Op.C.	is the operation code,
I	is the immediate constant,
DISPL	is the displacement
L1,L2	are lengths

Assembler Instructions

These instructions must be distinguished in the code byte (second byte).

Code byte	Instruction
C1	DC AL3
C2	DC X
C3	DS L
C4	LABEL
C5	PROCEDURE
C6	END OF BLOCK
C7	DC X'LENGTH OF DSA'
CA	DC A
CB	DC V

The code byte is put into a general register and X'C0' is subtracted. The result is used as offset in the branch table BRAN4 with the corresponding branches to subroutines as entries.

```
BRAN4  ----->  ┌──────────────┐
               │ B   ERR      │  0
               │ B   DC AL3   │  1
               │ B   DC X     │  2
               │ B   DS L     │  3
               │ B   LABEL    │  4
               │ B   PROCE    │  5
               │ B   ENDBL    │  6
               │ B   DC F     │  7
               │ B   ERR      │  8
               │ B   ERR      │  9
               │ B   DC A     │  A
               │ B   DC V     │  B
               └──────────────┘
```

Note: ERR is an error routine for elements which must not occur.

1. DC AL3   (DC VL3)

The location counter is stored in the save buffer together with the corresponding ESID number. The ESID number is 01 for the key 10 (for DC AL3), otherwise it has to be taken from the ESID table (for DC VL3).

The leftmost bit of the entry is set to 1 to flag the length of the address (3 bytes).

```
Save buffer:   ┌──────────────────┐
               │ CURRENT     ESID │
               │ LOCATION     NO  │
               │ COUNTER          │
               ├──────────────────┤
               │    .         .   │
               │    .         .   │
               │    .         .   │
               │                  │
```

The location counter is increased by 3. The instruction has the format:

a.

```
 (1)   (1)   (1)      (2)
┌────┬─────┬─────┬──────────┐
│80  │C1   │KEY  │  NAME    │
└────┴─────┴─────┴──────────┘
          (2)          (3)
┌───────────────┬──────────┐
│   MODIFIER    │ OFFSET   │  (10 bytes)
└───────────────┴──────────┘
```

The following information is put into the TXT-card mask:

b.

```
┌──────────────┐
│    OFFSET    │           (3 bytes)
└──────────────┘
```

2. DC X

   The location counter is increased by the length of the constant found in the second halfword and the constant itself is moved into the TXT-card mask.

3. DS L

   The location counter is increased by the length given in the second halfword, and, if the length is greater than 8, a new TXT card is started, otherwise, zeros are inserted.

4. LABEL

   This element is skipped by increasing the pointer POI by 4.

5. PROCE

   This element is not used and, therefore, it is skipped by adding 6 to POI.

6. ENDBL

   Adding 4 to POI causes this element to be skipped, too. (Same routine as LABEL)

7. DC F

   This instruction has the format

80	C7	LEV	BLO

   The block number BLO is used to find the corresponding entry in the BLOCK table, where the length of the DSA is found in a halfword. This halfword is expanded to a fullword by inserting leading zeros and the fullword is stored in the TXT card mask.

   The location counter is increased by 4.

8. DC A

   The location counter is stored in the save buffer together with the ESID number 01. The location counter is increased by 4. The offset, found in the last three bytes of the instruction, is expanded to a fullword and stored in the TXT-card mask.

9. DC V

   The location counter is stored in the save buffer together with the corresponding ESID number, which is taken from the ESID table.

   The location counter is increased by 4. The offset, found in the last three bytes of the instruction, is expanded to a fullword and stored in the TXT-card mask.

END Key

The END key is directly followed by the character string, the length of which is found in the communication area. The character string is inserted in the current TXT-card mask and the mask is written on SYS001.

The save buffer, which might be written on TXTOUT intermediately, is read into the I/O buffers, and for each entry in the save buffer an entry in the current RLD mask is inserted.

When the records of the save buffer have been processed, this phase ends by generating the END card.

Abbreviations

B0, B1, B2, B3	- I/O buffers
POI	- Input pointer
POU	- Output pointer for save buffer
BRAN4	- Branch Table
LOC1	- Location Counter
BUFL	- Buffer length
RLDCOU	- Counter for the RLD mask
TXTCOU	- Counter for the TXT mask
CPO	- Pointer for the 320-byte buffers
AE	- Switch for the 320-byte buffers
AB1	- Initial address of first 320-byte buffer
AB2	- Initial address of second 320-byte buffer
TXT	- Address of TXT-card mask
L	- Length

INRE -- FL

This routine inserts the register in the rightmost 4 bits of the byte POI+2 into the leftmost four bits of POI+2, and the register in the rightmost 4 bits of the byte at POI+3 into the rightmost 4 bits of POI+2.

MEX -- FL

This routine moves a string of length L (given in R1) from POI+1 into the TXT-card mask at the point denoted by the pointer TXTCOU and supervises the TXT-card mask, moves it into output if it is full and adjusts the starting address and TXTCOU.

MOSC -- FM

This routine moves a record of length 80 bytes from the mask initial address TXT to the current output buffer. It adusts the pointer CPO and writes the output buffer, if full, on SYS001.

Before moving the card into output, it inserts the TXTCOU or RLDCOU in bytes 11 and 12 of the mask giving the number of information bytes in the card.

The output buffers have a length of 320 bytes.

MOKK -- FM

This routine moves the four bytes, consisting of the location counter and ESID number, into OBUF (B0) by means of routine MOO. The OBUF is supervised and the output pointer is adjusted.

ELO -- FN

This routine determines the information to be inserted into the RLD mask. This will be done by means of an 8-byte mask with the following format:

-2- RELOCATION HEADER	-2- POSITION HEADER	-1- FLAG BYTE	-3- ADDRESS

1. The relocation header is taken from the save-buffer where it is the fourth byte of each entry (ESID number).

2. The position header is always X'01', because all constants arise in the program control section.

3. The flag byte has to be constructed in the following format:

```
bits 0-3 : zeros
     4,5 : 10 - For DC AL3 and DC VL3
                constants.
           11 - For all other
                constants.
       6 : zero
       7 :  0 - If another header or
                no more instructions
                follow in current card.
            1 - For the same headers.
```

4. The address of the DC instruction relative to the beginning of the program control section.

ELO inserts this information in the mask and sets a switch (SW8) to 1 if the headers change and to 0 if they do not.

MAX -- FO

This routine inserts the 8-byte mask. It determines whether there is room enough to insert the information of length 4 (SW8 = 0) or 3 (SW8 = 1) into the current RLD mask. If there is room enough, the mask is inserted and counter RLDCOU is increased; if not, the RLDCOU is inserted into bytes 11-12 of the mask (number of information bytes in the card) and the mask is written on SYS001. Then the 8-byte mask is inserted into the card in each case, and the RLDCOU is updated.

If SW8 = 0, the last bit of the previous flag byte is set to 1 (means continuation).

RLDCA -- FP

This routine changes the TXT-card mask to an RLD-card mask which is used to generate the RLD cards. It reads the save records one by one into the I/O buffer B0 and uses the pointer POI. For each entry it determines the bytes to be inserted by routine ELO, inserts them into the card (by MAX) and increases POI. This is done until the END key X'FF' is found. Then the last RLD card is written out by MOSC.

INLE -- FQ

This routine inserts the length-1 in the byte at POI+3 into one byte together with the length-1 at POI+2 if there is any.

ESI -- FQ

This routine fetches the corresponding ESID number from the ESID table by scanning the table for the name in the current DCV.

PHASE PL/IG31 (FINAL DIAGNOSTIC) -- GA

This phase lists the errors that occur after phase E25. It terminates the compilation if one or more of these errors occur or if neither LINK, SYM, nor DECK was specified in the OPTION job control statement. Which error has occurred is indicated by the bits of byte IJKMWC as follows:

Bit	Message*
0	5G01I
1	5G02I
2	5G03I
3	5G04I
4	5G05I
5	5G06I
6	5G07I

*For the actual diagnostic message refer to the DOS/TOS PL/I Programmer's Guide.

If no error is detected, all files are closed and the compilation is terminated. If an error is detected, the corresponding message is written on SYSLST, and the message

5E01I JOBSTEP PL/I TERMINATED, LINK OPTION RESET is written on SYSLOG.

The linkage bits in the communication area are set off, the files are closed, and the compilation is terminated.

Note: For the tape version, SYSRES is rewound at the beginning of the phase.

COPR -- GB

The current line is written on SYSLST. If SYSLOG is not the same device as SYSLST, the message

5E01 JOBSTEP PL/I TERMINATED, LINK OPTION RESET

is written on SYSLOG.

This phase lists the object code produced by the compilation and the constants of the static storage. The listing is produced if the LISTX option is specified; otherwise, the next phase (G55) is called. Figure 1 shows the format of the object code listing.

Print Position	Subheader	Contents
2	LOC.	6 hexadecimal digits representing the current value of the location counter.
10	OBJECT CODE	The instruction printed in hexadecimal digits containing operation code, registers or lengths, and displacements.
31	LABEL	The internal name is listed in the format L'aaaa', where 'a' is a hexadecimal digit of the internal name.
40	OP.	Mnemonic operation code using the Assembler mnemonics.
46	OPERANDS	Up to 35 print positions containing the operands of the instruction.
90		Statement number NN
Note: Registers, lengths, displacements names, and labels are listed in hexadecimal notation. The statement number is listed in decimal notation. It indicates that the code generated from the preceding statement number from the beginning of the program) to this one belongs to the statement marked by the statement number. This number may appear more than once.		

Figure 1. Format of the Object Code Listing

There are two formats for the operands:

1. If the internal name replaces a declared name or a library routine, it is listed in the format N'aaaa'+mmmm, where aaaa is the internal name and mmmm is the modifier. A zero modifier or heading zeros in it are left out.

2. In all other cases, the format of the operands is X'ddd'(B), where ddd is the hexadecimal displacement and B is the base register.

In addition, there are some pseudo Assembler instructions with the following formats which differ from Assembler language:

1. At the beginning of a block, 'BEGIN OF BLOCK NN' is written, where NN is the block number.

2. At the end of a block, 'END OF BLOCK' is written.

3. For the length of a DSA, the comment 'LENGTH OF DSA OF BLOCK NN' is written.

4. At the beginning of the static storage,

'L'FFFF' STATIC STORAGE'

is listed starting in column 31 and preceded by a blank line.

5. The listing is terminated with 'END' in columns 40-42.

Figure 2 shows examples for each type of instruction format.

```
|000000                       L'0102'  BEGIN OF BLOCK 01
|000000  05F0                          BALR  F,0
|000002  45E0 F00A                     BAL   E,X'00A'(F)
|000008  00000040                      DC    A(N'FFFF')
|00000C  905A D078                     STM   5,A,N'0103'+8
|000010  05E0                          BALR  E,0
|000012  4A60 C002                     AH    6,X'002'(C)
|000016  040E                          SPM   E
|000018  0A02                          SVC   02
|                                      STATEMENT NUMBER 7
|00001A  9216 D080            L'0107'  MVI   N'0106'+4,X'16'
|00001E  05E0                          BALR  E,0
|000020  41E1 E00E                     LA    E,X'00E'(1,E)
|000024  03                            DC    X'03'
|000025  000118                        DC    AL3(N'0105')
|000028  00000C80                      DC    LENGTH OF DSA OF BLOCK 01
|00002C  D255 E007 D112                MVC   X'007'(56,E),N'0110'+36
|000032  F842 D104 D080                ZAP   N'0111'-7(5),N'0110'(3)
|000038  94EE 6022            L'0112'  NI    X'022'(6),X'EE'
|00003C                                END OF BLOCK
|00003C                                DS    CL0004
|
|                             L'FFFF'  STATIC STORAGE
|000040  0000300040005000     L'0004'  DC    X'0000300040005000'
|        60007000                      DC    X'60007000'
|00004C  00                   L'0119'  DC    X'00'
|00004D  000000                        DC    VL3(N'0100')
|000050  00000000             L'0019'  DC    V(N'0019')
|000054  00000038             L'0112'  DC    A(N'0112')
|000058  000060               L'0120'  DC    AL3(N'0119')
|00005B                       L'0003'  DC    C'/$$$ IJXG40'
|                                      END
|-----------------------------------------------------------------
|Note: The preceding listing is not a logical program
```

Figure 2. Example of All Instruction Formats

In some cases, address constants occur containing the same internal name as their own label, e.g.

L'0101' DC A(N'0101')

This is because the internal label name occurs twice, once in the program part and once in static storage. In the program string, the constant is addressed via this name. In the static storage, the program string label is addressed. Only one entry in the offset table is used for both. This is because the offset of the constant is inserted into the offset table by phase F75 and inserted into the text string by phase F95. In phases G00 and G01, the offset in OFFTAB is replaced by the label offset; in phase G15, this offset is inserted into the instructions.

Phase Input and Output

The input is taken from the TEXT work file and consists of the output of phase G15. The input is read into buffers B1 and B2 with pointer POI.

Another input is the BLOCKTABLE, which is read into B0 during phase G30.

The output is written onto IJSYSLS via two buffers of length 121 bytes each reserved in the I/O buffer B4. The buffers are addressed via register 11.

General Flow of the Phase -- GF

The input is scanned. Each instruction is identified and the corresponding print line inserted into one of the two print buffers. The information is written on IJSYSLS by means of routine ZPRNT. When all instructions have been listed, i.e., when the end key is found, the phase is terminated by printing 'END' and calling phase G55.

The program string is read and scanned sequentially. The following types of instructions are differentiated:

1. Machine instructions (key is X'88')

2. Assembler instructions (key is X'80')

3. End key (X'01')

Machine Instructions. If a machine instruction is detected, the format is determined. This is done by means of the code byte which contains a special hexadecimal key for each instruction. For each format there are translate tables for the corresponding mnemonics. The code byte is translated with a translate vector containing the offsets in the corresponding table of mnemonics. The mnemonics are moved from the mnemonic tables into the corresponding entry of the print buffer. The mnemonic tables are shown in Figures 4-7. A summary of these tables is shown in Figure 3.

Instruction	Tables of Mnemonics	Translate Vectors
RR	RR1	RR2
RX	RX1	RX2
RS	SI1	SI2
SI	SI1	SI2
SS	SS1	SS2 (for X'D.')
		SS3 (for X'F.')

Figure 3. Summary of Mnemonics Tables and Translate Vectors

RR1	0 SPM	20 LPDR	RR2	X'00' 00	X'08' 14	X'17' 28
	• BALR	• LNDR		00 01	09 15	18 29
	• BCTR	• LCDR		00 02	0A 16	19 2A
	• SCR	• LDR		00 03	0B 17	1A 2B
	• SVC	• CDR		26 04	0C 18	1B 2C
	5 LPR	25 ADR		01 05	0D 19	1C 2D
	• LNR	• SDR		02 06	0E 1A	00 2E
	• LCR	• MDR		03 07	0F 1B	00 2F
	• NR	• DDR		00 08	10 1C	1D 30
	• CLR	• LPER		00 09	11 1D	1E 31
	10 OR	30 LNER		04 0A	12 1E	29 32
	• XR	• LCER		00 0B	13 1F	1F 33
	• LR	• LER		00 0C	14 20	00 34
	• CR	• CER		00 0D	15 21	00 35
	• AR	• AER		00 0E	28 22	00 36
	15 SR	35 SER		00 0F	16 23	00 37
	• MR	• MER		05 10	00 24	20 38
	• DR	• DER		06 11	00 25	21 39
	• ALR	• SPM		27 12	00 26	22 3A
	• SLR	• LTR		07 13	00 27	23 3B
		40 LTDR				24 3C
		• LTER				25 3D

Figure 4. RR Tables

RX1	0 0000	25 M	RX2	X'01' 40	X'16' 59	X'00' 72
	1 STH	• D		02 41	17 5A	00 73
	2 LA	• AL		03 42	18 5B	00 74
	3 STC	• SL		04 43	19 5C	00 75
	4 IC	• STD		05 44	1A 5D	00 76
	5 EX	30 LD		06 45	1B 5E	27 77
	6 BAL	• CD		07 46	1C 5F	28 78
	7 BCT	• AD		08 47	1E 60	29 79
	8 BC	• SD		09 48	00 61	2A 7A
	9 LH	• MD		0A 49	00 62	2B 7B
	10 CH	35 DD		0B 4A	00 63	2C 7C
	• AH	• AW		0C 4B	00 64	
	• SH	• SW		0D 4C	00 65	
	• MH	• STE		00 4D	00 66	
	• CVD	• LE		0E 4E	00 67	
	15 CVB	40 CE		0F 4F	1F 68	
	• ST	• AE		10 50	20 69	
	• M	• SE		00 51	21 6A	
	• CL	• NE		00 52	22 6B	
	• O	• DE		00 53	23 6C	
	20 X	45		11 54	24 6D	
	• L			12 55	25 6E	
	• C			13 56	26 6F	
	• A			14 57	00 70	
	• S			15 58	00 71	

Figure 5. RX Tables

SI1		SI2	
0	0000	X'00'	80
•	SRL	00	81
•	SLL	00	82
•	SRA	00	83
•	SLA	00	84
5	SRDL	00	85
•	SLDL	00	86
•	SRDA	00	87
•	SLDA	01	88
•	STM	02	89
10	TM	03	8A
•	MVI	04	8B
•	NI	05	8C
•	CLI	06	8D
•	OI	07	8E
15	XI	08	8F
•	LM	09	90
		0A	91
		0B	92
		00	93
		0C	94
		0D	95
		0E	96
		0F	97
		10	98

Figure 6. RS and SI Tables

SS1		SS2		SS3	
0	0000	X'00'	D0	X'00'	F0
1	MVN	01	D1	08	F1
2	MVC	02	D2	09	F2
3	MVZ	03	D3	0A	F3
4	NC	04	D4	00	F4
5	CLC	05	D5	00	F5
•	OC	06	D6	00	F6
•	XC	07	D7	00	F7
•	MVO			0B	F8
•	PACK			0C	F9
10	UNPK			0D	FA
•	ZAP			0E	FB
•	CP			0F	FC
•	AP			10	FD
•	SP				
15	MP				
•	DP				

Figure 7. SS Tables

Besides the mnemonics, the location counter, the current label, and the operands are moved into the print buffer. Before it is inserted, all the information is translated into EBCDIC by routine TRANS.

Assembler Instructions

These instructions must be distinguished in the code byte (second byte). The instruction codes follow.

Contents of Code byte	Instruction
C1	DC AL3 (DC VL3)
C2	DC X
C3	DS L
C4	LABEL
C5	PROCEDURE OR STATEMENT NUMBER
C6	END OF BLOCK
C7	DC X'LENGTH OF DSA'
CA	DC A
CB	DC V

The code byte is put into a general register and X'C0' is subtracted. The result is used as offset in the branch table BRAN3 with corresponding branches to subroutines as entries. The format of BRAN3 follows:

BRAN3	
B ERR	0
B DC AL3	1
B DC X	2
B DS L	3
B LABEL	4
B PROCE	5
B ENDBL	6
B DC F	7
B ERR	8
B ERR	9
B DC A	A
B DC A	B

Note: ERR is an error routine for elements which must not occur.

1. DC AL3, DC VL3, DC A, and DC V

The information contained in the instruction is inserted into the print buffer. The print line has the following format:

LOC.	OBJECT CODE	LABEL	OP.	OPERANDS
000088	000742	L'0137'	DC	AL3(N'0140')
00008B	000000	L'0138'	DC	VL3(N'0138')
000090	00000806	L'0139'	DC	A(N'0142')
000094	000000	L'0037'	DC	V(N'0037')

LOC is increased by 3 or 4 corresponding to the type of instruction. POI is increased by 10.

2. DC X

The information is inserted into the print buffer as follows:

Starting Print Position	Contents
2	Location counter
10	Constant, but not more than 8 bytes of it
31	Label
40	DC
46	X'CONSTANT' (not more than 8 bytes)

If the constant has more than 8 bytes, a new line is printed for each 8 bytes, but without updating the location counter each time. The location counter is increased by the entire length of the constant.

3. DS L

The print line has the following format:

	Column
LOCATION	2
LABEL	31
CODE DS	40
OPERAND CL followed by the length	46

LOC is increased by length L, which is found in the second half-word. POI is increased by 4.

4. LABEL

The label is moved into stack LAFI, and POI is increased by 4. When the label X'FFFF' (denoting the beginning of the static storage) is found, the line

' L'FFFF' STATIC STORAGE

is printed.

5. PROCE

Test whether statement number or not. If not, the message 'BEGIN OF BLOCK NN', the label, and the location counter are inserted into the print buffer. POI is increased by 6.

If there is a statement number, it is checked if the number is zero. If it is not, it is listed in the form

STATEMENT NUMBER NNNN

starting at print position 90. NNNN is the statement number translated into a decimal value. Leading zeros are left out. If an immediately following sta tement number has a lower value, it is skipped. POI is increased by 4.

6. ENDBL

The expression 'END OF BLOCK' and the location counter are inserted into the print buffer. POI is increased by 4.

7. DC F

The length of the corresponding DSA is taken from the block table expanded to a full-word and inserted at R11+10. The message

' DC LENGTH OF DSA OF BLOCK NN '

(where NN is the block number) is listed. The location counter and, in some cases, a label are also inserted. The location counter and POI are increased by 4.

END key

When the end key is found, the word END is put into the print buffer. ZPRNT is called to print the line. It determines whether the DECK and LINK bits are off. If both are off, the compilation is terminated by calling phase G31, which prints the message

5G02I SUCCESSFUL COMPILATION

closes all files, and calls EOJ. Otherwise, the phase is terminated by calling phase G55.

LOCAT -- GL

The routine translates location counter LOC into EBCDIC and inserts it into the corresponding entries of the print buffer.

LAB -- GL

This routine first translates the label found in the label field (LAFI) into EBCDIC. The label, preceded by L' and followed by a quote is moved into the print buffer. The label field is then set to zero.

TRANS -- GM

This routine translates hexadecimal values into EBCDIC. The bytes (the number of which is given in R2) at 0(R1) are translated and moved into the print buffer at the location given in R0.

CHAMO -- GM

This routine complements the half-word at POI+<R1>. The half-word is moved into a work area to ensure boundary alignment, then loaded into register R2, and restored at POI+<R1>.

TRA -- GN

This routine translates a string from POI+4 of length L (given in R3) into EBCDIC and inserts it into the print buffer at location R11+47. A quote is inserted at the end of the translated string.

NAME -- GN

This routine inserts the name, in the form N'aaaa', into the print buffer at the location given in RN. It determines whether or not the modifier is 0. If it is not 0, it inserts the modifier after the name. The modifier has no leading zeros and is preceded by a + or - sign.

TRANSL -- GO

This routine translates the length for SS instructions with only one length. It takes the length from POI+3 and inserts it unpacked into the stack ARBS. Length - 1 is inserted into the print buffer at R11+11.

TRANLS -- GO

This routine translates the lengths for SS instructions with two lengths. The procedure is the same as described in TRANSL.

TRANSR -- GO

This routine translates the register contents into EBCDIC and inserts them into the print buffer.

CHAR -- GP

This routine lists the character string with the label L'0003' followed by

DC C'aaa •••'.

No more than 32 characters are put into one line. The location counter is updated and printed for each line.

This phase provides the final output for the Linkage Editor, writes the object cards on IJSYSPH and/or writes the external symbol table on IJSYSLS if LINK,DECK, and/or SYM are flagged in the job control switches of the communication area within the Supervisor Nucleus.

The address of the communication area is inserted into register 1 via the macro instruction COMRG. The job control switches are located in bytes 56-59 of the communication area.

Job Control Bytes:

Byte: 56	Job control byte
Byte: 57	Linkage control byte
Byte: 58	Language processor control byte
Byte: 59	Job duration indicators

NOTE:

a) bit 0 of byte 57 denotes LINK if on,
b) bit 0 of byte 58 denotes DECK if on,
c) bit 3 of byte 58 denotes SYM if on.

(If all these bits are off, the compilation is terminated previous to this phase.)

The input of this phase is made up of the cards generated in previous phases and described there. The cards have been written onto SYS001 in physical records of 320 bytes and logical records of 80 bytes.

The first 16 bytes of each card (logical record) are fixed while the following 56 bytes contain information corresponding to the type of card. The final 8 bytes (73 - 80) are used for identification, containing the first 4 letters of the name of the external procedure and a current number.

Output for Linkage Editor

The output for the Linkage Editor is written onto IJSYSLN in physical records of 322 bytes without overlap. The first two bytes of each physical record contain the following information:

Byte 1: number of logical records in the physical one
Byte 2: length of the logical records.

Because four logical records of 80 bytes (in card-format) are inserted into one physical record, the bytes get the following values:

Byte 1: 4
Byte 2: 80

The cards are taken from SYS001 in records of 320 bytes and inserted into a mask of 322 bytes. From there they are put onto IJSYSLIN by means of routine ZLEDI which uses one 322-byte output record in the I/O-buffer region.

Card Output

The cards are punched as they are written onto SYS001. This is done by the routine ZPCH.

The cards are moved into one of the two output buffers (length 80 bytes) in the I/O-buffer area. The initial address of the output buffer is found in register 10. The use of two output buffers makes overlapping possible. Then routine ZPCH is called to move the card onto IJSYSPH.

Listing of the External Symbol Table

The input is scanned for ESD cards. These are printed in the following format:

SYMBOL	TYPE	ESID	ADDR	LENGTH	ESID
*	SD	*	*	*	
*	ER	*			
*	LD		*		*
Print 8 Pos.	Print 2 Pos.	Print 2 Pos.	Print 6 Pos.	Print 4 Pos.	Print 2 Pos.

* means 'used in this case'.

The printing is done by inserting the information in one of two 121 byte buffers at B5 which are used to process overlapped. Then routine ZPRINT is called to print the line.

Listing of the Block Table

At the end of the phase, the block table is listed. The format is of the block table is shown below:

Block	Length of DSA	Blocktable
01	01C7	
02	0060	
03	0070	
04	0058	

It contains, besides the block number (printed in hexadecimal notation), the length of the corresponding DSA (also in hexadecimal notation).

When all cards have been punched, printed, or written onto IJSYSLN, the compilation is terminated.

For this the comment

5W01I SUCCESSFUL COMPILATION

is put on IJSYSLS: all files are closed and the LINKEDIT bits (bits 0 and 1 of byte 57) are set.

If the compilation is in error, byte IJKMWC+1 is set to X'FF' and the message

5W02I COMPILATION IN ERROR

is listed.

Then the macro END OF JOB (EOJ) is used to terminate the compilation.

Detailed Description

In this phase the following three options are tested:

a. Output on IJSYSLN (LINK-bit is on)
b. Output on IJSYSPH (DECK-bit is on)
c. Output on IJSYSLS (SYM-bit is on)

The resulting 7 cases

1. a, b, and c
2. a and b
3. a and c
4. b and c
5. a
6. b
7. c

are processed in four special routines:

PLI	for cases 1 and 2
LINK	for cases 3 and 5
PCH	for cases 4 and 6
PRT	for case 7

Input and Output

The input consists of the cards written onto SYS001 in records of 320 bytes (one record containing 4 cards).

These records are read into the second input buffer EBU, moved into the first input buffer ABU, and processed there while another record is read into EBU.

Register CPO is used as an input pointer.

The output is written onto IJSYSLN, IJSYSPH and/or IJSYSLS depending on the options specified. While for IJSYSPH and IJSYSLS two output buffers are reserved (addressed via register 10 or 11, respectively), the output on IJSYSLN uses only one buffer of 322 bytes.

The buffers for IJSYSLS are located in the last I/O buffer B4 and have a length of 121 bytes. The buffers for IJSYSPH are located in the external I/O buffers B5 and B6 and have a length of 80 bytes each.

The listing of the block table uses the table at IJKMBS, which contains the lengths of the DSAs.

In order to test the option bits, the address of the corresponding communication area is put into register 1 by means of the macro COMRG. The first bit of byte 57 is the LINK bit, the first bit of byte 58 is the DECK bit, and the fourth bit of byte 58 is the SYM bit.

When all input has been processed, the block table is listed and this phase terminates the compilation by printing a final comment, inserting bits into the communication area, and closing all files. The final instruction is the macro EOJ.

GET -- HC

This routine processes overlapped by moving the contents of the second input buffer EBU into the first input buffer ABU. It tests to determine whether there are any more records on SYS001. If there are no more records, switch NC1 is set to 1. Otherwise, the next record is read into EBU, and the number of records NC is updated.

PLI -- HC

This routine puts all input records from ABU onto IJSYSLN, punches all cards, and prints the ESD table if the SYM switch is on.

The output on IJSYSLN is processed by routine LINKS, the punching by PUNCH, and the listing by PRINT.

LNK -- HD

LNK writes all input records from ABU onto IJSYSLN and prints the ESD table if the SYM switch is on. It uses routines LINKS, PRINT, and GET.

PCH -- HE

This routine punches all cards by calling routine PUNCH and prints the ESD cards by means of routine PRINT. It uses routine GET for input.

PRT -- HF

This routine prints the external symbol table by means of routine ZPRNT, which uses two print buffers of length 121, the current address of which is to be found in register 11.

The information is taken from the cards found in the buffer ABU and inserted into the corresponding slots of the print buffer.

PRINT -- HG

This routine prints an ESD card in the way described in PRT. It scans the card and prints one line for each ESD entry in the card.

PUNCH -- HH

This routine is used to punch a card located at CPO. This is done by moving the card into one of the punch buffers (the address of which is found in register 10) and calling routine ZPCH.

LINKS -- HH

This routine moves one record of 322 bytes from ABO into the link buffer and writes it onto IJSYSLN by calling routine ZLEDI.

TRA -- HH

This routine translates hexadecimal values (length given in R2) into EBCDIC code and inserts the translated string into the print buffer at the location given in R0. The location of the hexadecimal value is given in R1.

PRIBLO -- HI

This routine prints the rightmost byte in register RY (block number) and the halfword at CPO (length of the current block DSA).

BLOCKT -- HI

This routine prints the block table by means of routine PRIBLO. It inserts the subheader and uses CPO as a pointer in the block table and RY as a counter for the block numbers. The greatest block number is found in IJKMBC.

APPENDIX A. SYNTAX NOTATION OF PL/I INPUT STREAM

The metalanguage used in this section must not be considered to be of universal significance. It is a combination of the IBM syntax notation, the Backus/Naur form and a few extensions. The following rules apply:

- <a series of lower case letters> is the common form of metalinguistic variables.
- ::= means "is defined as". Each metalinguistic definition contains one such symbol.
- { } denotes grouping.
- | indicates that a choice is to be made.
- [ ] denotes options. Anything enclosed in square brackets may appear once or not at all.
- ℰ 3 "Combinatorial" brackets indicate that the options enclosed by them may appear in any order, however not more than once.
- min 1
  If used in connection with { }, the enclosed syntactical unit must appear at least once.
  If used in connection with combinatorial brackets, at least one of the enclosed options must appear.
- max m
  If used in connection with { }, the enclosed syntactical unit must not appear more than m times.
  If used in connection with combinatorial brackets, a maximum of m of the enclosed options may appear.
- min 1 max m
  Both limitations apply (see preceding text).
- <A SERIES OF CAPITAL LETTERS> indicates the internal 3-byte representation of the corresponding keyword.
- hex , where and are hexadecimal digits, indicates the literal occurence of the hexadecimal digits.
- All other symbols maintain their original meaning.

Examples:
(Note that most of these examples are no valid definitions of PL/I statements.)

<digit> ::= 0|1|2|3|4|5|6|7|8|9

Meaning: a digit is defined as the literal occurence of a 0,1,2 ... etc.

<function reference> ::= <ident> [(expression list>)]

Meaning: a function reference is defined as an identifier optionally followed by an expression list enclosed in parentheses.

<replication factor> ::= (min 1 max 3 {<digit>})

Meaning: a replication factor is defined as a series of 1 to 3 digits enclosed in parentheses.

<open item> ::=
<FILE>(<ident>) ℰ [<INPUT>] [<BACKWARDS>] [<page>] 3

Meaning: an open item is defined as the internal representation of the keyword FILE, followed by a left parenthesis, followed by an identifier, followed by a right parenthesis, followed by a list of three options which may appear in any order (each option not more than once).

<A> ::= <links> hex 0003

<arithmetic constant> ::= <decimal fixed constant> | <binary fixed constant> | <decimal float constant> | <binary float constant>

<assign> ::= <left> hex 0E <right>

<assignment statement> ::= <assign> <name> = <expr>

<B> ::= <links> hex 0001

<BACKWARDS> ::= <links> hex 0164

<BEGIN> ::= <left> hex 06 <right>

<begin statement> ::= <BEGIN>

<binary constant> ::= <binary integer> [.[<binary integer>]] |. <binary integer>

<binary digit> ::= 0|1

<binary fixed constant> ::= <binary integer> <B>

<binary float constant> ::= <binary constant> <exponent> <B>

<binary integer> ::= min 1 {<binary digit>}

<bit string constant> ::= [<replic>] '<binary integer>'<B>

<blend> ::= <left> hex 07 <right>

<block> ::= <data character>

<BY> ::= <links> hex 0010

<CALL> ::= <left> hex 09 <right>

<call statement> ::= <CALL> <ident> [(<expression list>)]

<character string constant> ::= <character string constant key> <data character> <character string constant key> <data character> <data character>

<character string constant key> ::= hex E3

<CLOSE> ::= <left> hex 30 <right>

<close list> ::= <file option> | <close list>, <file option>

<close statement> ::= <CLOSE> <close list>

<COLUMN> ::= <links> hex 012F

<condition> ::= <FIXEDOVERFLOW> | <UNDERFLOW> | <SIZE> | <ERROR> | <ZERODIVIDE> | <ENDFILE> <enclosed ident> | <TRANSMIT> <enclosed ident> | <KEY> <enclosed ident> | <RECORD> <enclosed ident> | <ENDPAGE> <enclosed ident> | <OVERFLOW> | <CONVERSION>

<constant> ::= <sterling constant> | <bit string constant> | <character string constant> | <binary fixed constant> | <decimal fixed constant> | <binary float constant> | <decimal float constant>

<CONVERSION> ::= <links> hex 016A

<decimal fixed constant> ::= <integer> [. [<integer>]] | . <integer>

<decimal float constant> ::= <decimal fixed constant> <exponent>

<DECLARE> ::= <left> hex 40 <right>

<declare statement> ::= <DECLARE> min 0 {<data character>}

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<DISPLAY> ::= <left> hex 32 <right>

<display statement> ::= <DISPLAY> <single> [<REPLY> <enclosed name>]

<DO> ::= <left> hex 12 <right>

<do statement> ::= <DO> [<while clause> | <q-name> = <specification list>]

<E> ::= <links> hex 0002

<EDIT> ::= <links> hex 0055

<ELSE> ::= <left> hex 11 <right>

<else statement> ::= <ELSE>

<enclosed ident> ::= (<ident>)

<enclosed name> ::= (<name>)

<end of block statement> ::= <blend>

<end of group statement> ::= <grend>

<end of statement> ::= <eos> <no error> <level> <block> <statement number> | <eos>
        <error> <level> <block> <statement number> <error tail>

<ENDFILE> ::= <links> hex 0147

<ENDPAGE> ::= <links> hex 014B

<ENTRY> ::= <left> hex 0B <right>

<entry statement> ::= <ENTRY> [(<identifier list>)]

<eos> ::= hex EA

<error> ::= hex 80 | hex 40

<fixedoverflow> ::= <links> hex 0177 hex 40

<ERROR> ::= links hex 0107

<error key> ::= hex EB

<error number> ::= <data character>

<error tail> ::= <error key> <error number> | <error tail> <error key> <error number>

<exponent> ::= <E> [+ | -] <integer>

<expr> ::= <constant> | <name> | <function reference> | <single> | <prefix operator>
        <expr> | <expr> <infix operator> <expr>

<expression list> ::= <expr> | <expression list>,<expr>

```
<F> ::= <links> hex 0004

<FILE> ::= <links> hex 0047

<file option> ::= <FILE> <enclosed ident>

<fixedoverflow> ::= <links> hex 0177

<FORMAT> ::= <left> hex 35 <right>

<format element> ::= [<integer>] <format item> | <integer> <format list>

<format element list> ::= <format element> | <format element list>, <format element>

<format item> ::= <F> (<integer> [,<integer> [,[+ | -] <integer>]]) | <E> (<integer>,
         <integer> [,<integer>]) | <B> [(<integer>)] | <A> [(<integer>)] | <X>
         (<integer>) | <SKIP> [(<integer>)] | <LINE> (<integer>) | <COLUMN> (<integer>) |
         <PAGE> | <R> (<q-name>)

<format list> ::= (<format element list>)

<format statement> ::= <FORMAT> <format list>

<FROM> ::= <links> hex 005A

<function reference> ::= <ident> [(<expression list>)]


<GET> ::= <left> hex 33 <right>

<get statement> ::= <GET> [<file option> | <STRING> <enclosed name>] <input data
         specification>

<GO> ::= <links> hex 000E

<goto> ::= <left> hex 0A <right>

<GOTO> ::= <links> hex 004D

<goto statement> ::= <goto> <name>

<grend> ::= <left> hex 13 <right>


<heading> ::= min 0 {<ident>:}


<ident> ::= <ident key> <data character> <data character>

<ident key> ::= hex E1

<identifier list> ::= <ident> | <identifier list>, <ident>

<IF> ::= <left> hex 10 <right>

<if statement> ::= <IF> <expr>

<infix operator> ::= + | - | * | / | ** | = | < | > | ¬= | <= | <= | || | | | ¬> | ¬<

<INPUT> ::= <links> hex 0183

<input data element> ::= <name> | <input repetitive specification>

<input data element list> ::= <input data element> | <input data element list>, <input
         data element>
```

<input data list> ::= (<input data element list>)

<input data specification> ::= <input list specification> | <input edit specification>

<input edit list> ::= <input data list> <format list> | <input edit list> <input data list> <format list>

<input edit specification> ::= <EDIT> <input edit list>

<input list specification> ::= <LIST> <input data list>

<input repetitive specification> ::= (<input data element list> <do> <q-name> = <specification list>)

<integer> ::= <u>min</u> 1 {<digit>}

<INTO> ::= <links> <u>hex</u> 00D9

<iteration> ::= <expr> [<TO> <expr> [<BY> <expr>] | <BY> <expr> [<TO> <expr>]] [<while clause>]

<KEY> ::= <links> <u>hex</u> 0097

<KEYFROM> ::= <links> <u>hex</u> 01C9

<KEYTO> ::= <links> <u>hex</u> 00FD

<L> ::= <links> <u>hex</u> 0088

<left> ::= <u>hex</u> E000

<level> ::= <data character>

<LINE> ::= <links> <u>hex</u> 00D8

<links> ::= <u>hex</u> E1

<LIST> ::= <links> <u>hex</u> 00D6

<LOCATE> ::= <left> <u>hex</u> 38 <right>

<locate statement> ::= <LOCATE> <ident> <file option> {<SET> <enclosed name> [<KEYFROM> <single>] | [<KEYFROM> <single>] <SET> <enclosed name>}

<name> ::= <q-name> [(<subscript list>)]

<NEWPAGE> ::= <links> <u>hex</u> 01C8

<no error> ::= <u>hex</u> 00

<nop> ::= <left> <u>hex</u> 0D <right>
<null statement> ::= <nop>

<ON> ::= <left> <u>hex</u> 22 <right>

<on statement> ::= <ON> <condition> [<SYSTEM> | <GO> <TO> <ident> | <GOTO> <ident>

<OPEN> ::= <left> <u>hex</u> 31 <right>

<open item> ::= <file option> [<PAGESIZE> <single> | <OUTPUT> | <INPUT>]

```
<open list> ::= <open item> | <open list>, <open item>

<open statement> ::= <OPEN> <open list>

<OUTPUT> ::= <links> hex 011C

<output data element> ::= <expr> | <output repetitive specification>

<output data element list> ::= <output data element> | <output data element list>,
         <output data element>

<output data list> ::= (<output data element list>)

<output data specification> ::= <output list specification> | <output edit specification>

<output edit list> ::= <output data list> <format list> | <output edit list> <output data
         list> <format list>

<output edit specification> ::= <EDIT> <output edit list>

<output list specification> ::= <LIST> <output data list>

<output repetitive specification> ::= (<output data element list> <do> <q-name> =
         <specification list>

<OVERFLOW> ::= <links> hex 0152


<PAGE> ::= <links> hex 0057

<PAGESIZE> ::= <links> hex 0159

<pence integer> ::= 10 | 11 | [0] <digit>

<pence part> ::= <pence integer> [. [<integer>]]

<prefix operator> ::= + | - | ¬

<PROCEDURE> ::= <left> hex 05 <right>

<procedure statement> ::= <PROCEDURE> [(<identifier list>)]

<PUT> ::= <left> hex 34 <right>

<put statement> ::= <PUT> [<file option >] {{{ε[<page>] [<line> <SINGLE>]3| <skip>
         [<SINGLE>]} <OUTPUT DATA SPECIFICATION>} | {{MIN 1ε[<page> [<line> <SINGLE>]3|
         <skip> [<SINGLE>]} [<OUTPUT DATA SPECIFICATION>]}}| <put> <string> <ENCLOSED
         NAME> <OUTPUT DATA SPECIFICATION>


<q-name> ::= <ident> | <q-name> . <ident>


<R> ::= <links> hex 000A

<READ> ::= <left> hex 36 <right>

<read statement> ::= <READ> <file option> {{<SET> <enclosed name> | <INTO> <enclosed
         ident>} [<KEYTO> <enclosed name> | <KEY> <single>] | [<KEYTO> <enclosed name> |
         <KEY> <single>] {<SET> <enclosed name> | <INTO> <enclosed ident>}}

<RECORD> ::= <links> hex 0114

<replic> ::= (min 1 max 3 {<digit>})
```

<REPLY> ::= <links> hex 0105

<RETURN> ::= <left> hex 0C <right>

<return statement> ::= <RETURN> [<single>]

<REVERT> ::= <left> hex 21 <right>

<revert statement> ::= <REVERT> <condition>

<REWRITE> ::= <left> hex 39 <right>

<rewrite statement> ::= <REWRITE> <file option>{[<KEY> <single>] [<FROM> <enclosed ident>]}

<right> ::= <data character> <data character> <data character>

<SET> ::= <links> hex 0018

<shilling part> ::= [0 | 1] <digit>

<SIGNAL> ::= <left> hex 20 <right>

<signal statement> ::= <SIGNAL> <condition>

<single> ::= (<expr>)

<SIZE> ::= <links> hex 0052

<SKIP> ::= <links> hex 005D

<specification list> ::= <iteration> | <specification list>, <iteration>

<statement> ::= <heading> <statement body> <end of statement>

<statement body> ::= <null statement> | <assignment statement> | <call statement> | <display statement> | <goto statement> | <return statement> | <signal statement> | <revert statement> | <stop statement> | <do statement> | <if statement> | <open statement> | <close statement> | <read statement> | <write statement> | <format statement> | <else statement> | <begin statement> | <end of block statement> | <end of group statement> | <procedure statement> | <entry statement> | <on statement> | <declare statement> | <get statement> | <put statement> | <locate statement> | <rewrite statement>

<statement number > ::= <data character> <data character>

<sterling constant> ::= <integer> . <shilling part> . <pence part>

<STOP> ::= <left> hex 23 <right>

<stop statement> ::= <STOP>

<STRING> ::= <links> hex 0113

<subscript> ::= <q-name> | <arithmetic constant> | {+ | -} <subscript> | <subscript> {*|+|-} <subscript>

<subscript list> ::= <subscript> | <subscript list>, <subscript>

<SYSTEM> ::= <links> hex 011F

<TO> ::= <links> hex 000F

<TRANSMIT> ::= <links> hex 0158

<UNDERFLOW> ::= <links> hex 0162

<WHILE> ::= <links> hex 007F

<while clause> ::= <WHILE> <single>

<WRITE> ::= <left> hex 37 <right>

<write statement> ::= <WRITE> <file option> {<FROM> <enclosed ident> [<KEYFROM> <single>]
          | [<KEYFROM> <single>] <FROM> <enclosed ident>}

<X> ::= <links> hex0006

<ZERODIVIDE> ::= <links> hex 0169

APPENDIX B. SYNTAX NOTATION OF PL/I OUTPUT STREAM

<a> ::= <links> hex 0003

<and> ::= hex E205EC

<arithmetic constant> ::= <decimal fixed constant> | <binary fixed constant> | <decimal float constant> | <binary float constant>

<assign> ::= <left> hex 0E <right>

<assignment statement> ::= <assign> <name> <ist> <expr>

<b> ::= <links> hex 0001

<backwards> ::= <links> hex 0164

<BEGIN> ::= <left> hex 06 <right>

<begin statement> ::= <BEGIN>

<binary fixed constant> ::= <binary fixed constant key> <length> min 2 {<data character>}

<binary fixed constant key> ::= hex F9

<binary float constant> ::= <binary float constant key> <length> min 4 {<data character>}

<binary float constant key> ::= hex FA

<bit string constant> ::= <bit string constant key> <length> min 4 {<data character>}

<bit string constant key> ::= hex FB

<blend> ::= <left> hex 08 <right>

<branch> ::= <links> hex 0355

<by> ::= <links> hex 0010

<CALL> ::= <left> hex 09 <right>

<call statement> ::= <CALL> <ident> [<lnb> <expression list> <rnb>]

<cat> ::= hex E203EA

<character string constant> ::= <character string constant key> <data character> <data character> <character string constant key> <data character> <data character>

<character string constant key> ::= hex E3

<CLOSE> ::= <left> hex 30 <right>

<close list> ::= <file option> | <close list> <comma> <file option>

<close statement> ::= <CLOSE> <close list>

<col> ::= <links> hex 012F

<comma> ::= hex E200E8

<condition> ::= <fixedoverflow> | <underflow> | <size> | <fieldoverflow> | <endfile> <enclosed ident> | <transmit> <enclosed ident> | <key> <enclosed ident> | <record> <enclosed ident> | <endpage> <enclosed ident> | <overflow> | <error> | <zerodivide> | <conversion>

<constant> ::= <sterling constant> | <bit string constant> | < character string constant> | <binary fixed constant> | <decimal fixed constant> | <binary float constant> | <decimal float constant>

<conversion> ::= <links> hex 016A

<DECLARE> ::= <left> hex 40 <right>

<declare statement> ::= <DECLARE> min 0 {<data character>}

<decimal fixed constant> ::= <decimal fixed constant key> <length> min 1 {<data character>}

<decimal fixed constant key> ::= hex F7

<decimal float constant> ::= <decimal float constant key> <length> min 3 {<data character>}

<decimal float constant key> ::= hex F8

<DISPLAY> ::= <left> hex 32 <right>

<display statement> ::= <DISPLAY> <single> [<reply> <enclosed name>]

<DO> ::= <left> hex 12 <right>

<do> ::= <links> hex 0054

<do statement> ::= <DO> [<while clause> | <q-name> <ist> <specification list>]

<e> ::= <links> hex 0002

<edit> ::= <links> hex 0055

<ELSE> ::= <left> hex 11 <right>

<else statement> ::= <ELSE>

<enclosed ident> ::= <llb> <ident> <rlb>

<enclosed name> ::= <llb> <name> <rlb>

<endfile> ::= <links> hex 0147

<end of block statement> ::= <blend>

<end of group statement> ::= <grend>

<end of procedure statement> ::= <prend>

<end of statement> ::= <eos> <no error> <level> <block> <statement number> | <eos> <error> <level> <block> <statement number> <error tail>

<endpage> ::= <links> hex 014B

<ENTRY> ::= <left> hex 0B <right>

<entry statement> ::= <ENTRY> [<lnb> <identifier list> <rnb>]

```
<eos> ::= hex EA

<eq> ::= hex E207ED

<ERROR> ::= <links> hex 0107

<error> ::= hex 80 | hex 40

<error key> ::= hex EB

<error number> ::= <data character>

<error tail> ::= <error key> <error number> | <error tail> <error key> <error number>

<expr> ::= <constant> | <name> | <function reference> | <single> <prefix operator> <expr>
         | <expr> <infix operator> <expr>

<expression list> ::= <expr> | <expression list> <comma> <expr>


<f> ::= <links> hex 0004

<file> ::= <links> hex 0047

<file option> ::= <file> <enclosed ident>

<fixedoverflow> ::= <links> hex 0177

<FORMAT> ::= <left> hex 35 <right>

<format element> ::= [<format integer>] <format item> | <format integer> <format list>

<format element list> ::= <format element> | <format element list> <comma> <format
          element>

<format integer> ::= <format integer key> <length> min 1 {<data character>}

<format integer key> ::= hex FE

<format item> ::= <f> <llb> <format integer> [<comma> <format integer> [<comma> [<uplus>
          | <umin>] <format integer>]] <rlb> | <e> <llb> <format integer> <comma> <format
          integer> [<comma> <format integer>] <rlb> | <b> [<llb> <format integer> <rlb>] |
          <a> [<llb> <format integer> <rlb>] | <x> <llb> <format integer> <rlb> | <skip>
          [<llb> <format integer> <rlb>] | <line> <llb> <format integer> <rlb> | <col>
          llb> <integer> <rlb> | <page> | <r> <llb> <q-name> <rlb>

<format list> ::= <llb> <format element list> <rlb>

<format statement> ::= <FORMAT> <format list>

<from> ::= <links> hex 005A

<function reference> ::= <ident> [<lnb> <expression list> <rnb>]


<gauche> ::= hex E2 <data character>

<ge> ::= hex E207F2

<GET> ::= <left> hex 33 <right>

<get statement> ::= <GET> [<file option> | <string> <enclosed name>] <input data
          specification>

<goto> ::= <left> hex 0A <right>
```

<goto statement> ::= <goto> <name>

<grent> ::= <left> hex 13 <right>

<gt> ::= hex E207F0

<heading> ::= min 0 {<ident>:}

<ident> ::= <ident key> <data character> <data character>

<identifier list> ::= <ident> | <identifier list> <comma> <ident>

<ident key> ::= hex E1

<IF> ::= <left> hex 10 <right>

<if statement> ::= <IF> <expr>

<infix operator> ::= <plus> | <minus> | <slash> | <pot> | <star> | <eg> | <lt> | <gt> | <ne> | <le> | <ge> | <and> | <or> | <cat>

<input> ::= <links> hex 0183

<input data element> ::= <name> | <input repetitive specification>

<input data element list> ::= <input data element> | <input data element list> <comma> <input data element>

<input data list> ::= <llb> <input data element list> <rlb>

<input data specification> ::= <input list specification> | <input edit specification>

<input edit list> ::= <input data list> <format list> | <input edit list> <input data list> <format list>

<input edit specification> ::= <edit> <input edit list>

<input list specification> ::= <list> <input data list>

<input repetitive specification> ::= <llb> <input data element list> <do> <q-name> <ist> <specification list> <rlb>

<into> ::= <links> hex 00D9

<ist> ::= <gauche> hex FB

<iteration> ::= <expr> [<to> <expr> [<by> <expr>] | <by> <expr> [<to> <expr>]] [<while clause>]

<key> ::= <links> hex 0097

<keyfrom> ::= <links> hex 01C9

<keyto> ::= <links> hex 00FD

<le> ::= hex E207F1

<left> ::= hex E0 <data character>

<length> ::= <data character> <data character>

<line> ::= <links> hex 00D8

<links> ::= hex EF

<list> ::= <links> hex 00D6

<llb> ::= <gauche> hex FD

<lnb> ::= hex E200E6

<LOCATE> ::= <left> hex 38 <right>

<locate statement> ::= <LOCATE> <ident> <file option> {<set> <enclosed name> [<keyfrom> <single>] | [<keyfrom> <single>] <set> <enclosed name>}

<lt> ::= hex E207EF

<minus> ::= hex E208F4

<name> ::= <q-name> [<lnb> <subscript list> <rnb>]

<ne> ::= hex E207EE

<newpage> ::= <links> hex 01C8

<no error> ::= hex 00

<nop> ::= <left> hex 0D <right>

<not> ::= hex E20AF9

<null> ::= <links> hex 0356

<null statement> ::= <nop>

<ON> ::= <left> hex 22 <right>

<on statement> ::= <ON> <condition> {<system> | <branch> <ident> | <null>}

<OPEN> ::= <left> hex 31 <right>

<open item> ::= <file option> [<pagesize> <single> | <output> | <input>]

<open list> ::= <open item> | <open list> <comma> <open item>

<open statement> ::= <OPEN> <open list>

<or> ::= hex E204EB

<output> ::= <links> hex 011C

<output data element> ::= <expr> | <output repetitive specification>

<output data element list> ::= <output data element> | <output data element list> <comma> <output data element>

<output data list> ::= <llb> <output data element list> <rlb>

<output data specification> ::= <output list specification> | <output edit specification>

<output edit list> ::= <output data list> <format list> | <output edit list> <output data list> <format list>

<output edit specification> ::= <edit> <output edit list>

<output list specification> ::= <list> <output data list>

<output repetitive specification> ::= <llb> <output data element list> <do> <q-name> <ist> <specification list> <rlb>

<overflow> ::= <links> hex 0152

<page> ::= <links> hex 0057

<pagesize> ::= <links> hex 0159

<period> ::= <links> hex 0360

<plus> ::= hex E208F3

<pot> ::= hex E20AFA

<prefix operator> ::= <uplus> | <umin> | <not>

<prend> ::= <left> hex 07 <right>

<PROCEDURE> ::= <left> hex 05 <right>

<procedure statement> ::= <PROCEDURE> [<lnb> <identifier list> <rnb>]

<PUT> ::= <left> hex 34 <right>

<put statement> ::= <PUT> [<file option>] {{{ε[<page>] [<line> <single>]3| <skip> [<single>]} <output data specification>}|{{min 1ε[<page>] [<line> <single>]3| <skip> [<single>]} [<output data specification>]}}| <PUT> <string> <enclosed name> <output data specification>

<q-name> ::= <ident> | <q-name> hex 2B <ident>

<r> ::= <links> hex 000A

<READ> ::= <left> hex 36 <right>

<read statement> ::= <READ> <file option> {{<set> <enclosed name> | <into> <enclosed ident>} [<keyto> <enclosed name> | <key> <single>] | [<key> <single> | <keyto> <enclosed name>] {<set> <enclosed name> | <into> <enclosed ident>}}

<record> ::= <links> hex 0114

<rlb> ::= <gauche> hex FE

<rnb> ::= hex E200E7

<reply> ::= <links> hex 0105

<RETURN> ::= <left> hex 0C <right>

<return statement> ::= <RETURN> [<single>]

<REVERT> ::= <left> hex 21 <right>

<revert statement> ::= <REVERT> <condition>

<REWRITE> ::= <left> hex 39 <right>

<rewrite statement> ::= <REWRITE> <file option> ℰ[<key> <single>] [<from> <enclosed ident>]3

<right> ::= <data character> <data character> <data character>

<rnb> ::= hex E200E7

<set> ::= <links> hex 0018

<SIGNAL> ::= <left> hex 20 <right>

<signal statement> ::= <SIGNAL> <condition>

<single> ::= <lnb> <expr> <rnb>

<size> ::= <links> hex 0052

<skip> ::= <links> hex 005D

<slash> ::= hex E209F6

<specification list> ::= <iteration> | <specification list> <specom> <iteration>

<specom> ::= <gauche> hex FC

<star> ::= hex E209F5

<statement> ::= <heading> <statement body> <end of statement>

<statement body> ::= <null statement> | <assignment statement> | <call statement> | <display statement> | <goto statement> | <return statement> | <signal statement> | <revert statement> | <stop statement> | <do statement> | <if statement> | <open statement> | <close statement> | <read statement> | <write statement> | <get statement> | <put statement> | <locate statement> | <rewrite statement> | <format statement> | <else statement> | <begin statement> | <end of block statement> | <end of procedure statement> | <end of group statement> | <procedure statement> | <entry statement> | <on statement> | <declare statement>

<statement number> ::= <data character> <data character>

<sterling constant> ::= <sterling constant key> <length> min 6 {<data character>}

<sterling constant key> ::= hex FC

<STOP> ::= <left> hex 23 <right>

<stop statement> ::= <STOP>

<string > ::= <links> hex 0113

<subscript> ::= <q-name> | <arithmetic constant> | {<uplus> | <umin>} <subscript> | <subscript> {<plus> | <minus> | <star>} <subscript>

<subscript list> ::= <subscript> | <subscript list> <comma> <subscript>

<system> ::= <links> hex 011F

<to> ::= <links> hex 000F

<transmit> ::= <links> hex 0158

<umin> ::= hex E20AF8

```
<underflow> ::= <links> hex 0162

<uplus> ::= hex E20AF7


<while> ::= <links> hex 007F

<while clause> ::= [<while> <single>]

<WRITE> ::= <left> hex 37 <right>

<write statement> ::= :WRITE> <file option> {<from> <enclosed ident> [<keyfrom> <single>]
         | [<keyfrom> <single>] <from> <enclosed ident>}


<x> ::= <links> hex 0006


<zerodivide> ::= <links> hex 0169
```

APPENDIX C. LIBRARY ROUTINES

INT. NAME		MOD. NAME Pos. 4-7	DESCRIPTION
HEX.	DEC.		
10	16	SZCA	MAIN
11	17	SZCM	MAIN
12	18	SZCN	PROLOGUE
13	19	SZCP	GO TO
14	20	SZCS	SIGNAL
15	21	SZCT	STOP
16	22	SZLM	ENTRY MOVE ROUTINE
17	23	ZWSA	LIBRARY WORK SPACE
18	24	TOPM	OPEN
19	25	TCLM	CLOSE
1A	26	TPSM	PAGESIZE
1B	27		
1C	28		STREAM-DIRECTED LIST INPUT
1D	29		INITIAL STRING LIST INPUT
1E	30		STRING TRANSMITTED LIST INPUT
1F	31		
20	32	TFDM	STREAM DIRECTOR
21	33	TFMM	FORMAT DECODER
22	34	TGDS	STRING DIRECTOR
23	35	TSTM	GET/PUT FILE INITIAL
24	36	TGDI	GET STRING INITIAL
25	37	TGDO	PUT STRING INITIAL
26	38		
27	39	TXRM	EXTENT, FB TO PDI
28	40	VBCM	
29	41	VTCM	FLOAT TO PDI
2A	42	VPCM	FIXED DECIMAL TO PDI
2B	43	VFCM	NUM. FIELD FLOAT TO PDI
2C	44	VECM	E, F TO PDI
2D	45	VGIM	CHAR. STRING TO BIT STRING
2E	46	VIGM	BIT STRING TO CHAR. STRING
2F	47	TSTR	X, PAGE, SKIP
30	48	TLCM	LINE, COLUMN
31	49	VCBM	PDI TO BIN. FIXED
32	50	VCTM	PDI TO FLOAT
33	51	VCPM	PDI TO FIXED DECIMAL
34	52	VCFM	PDI TO NUM. FIELD FLOAT
35	53	VCEM	PDI TO E, F
36	54	SYSI	SYSIN
37	55	SYSA	SYSPRT
38	56	TCBM	CONSECUTIVE BUFFERED TRANSMITTER
39	57	TCUM	CONSECUTIVE UNBUFFERED TRANSMITTER
3A	58	TRGM	REGIONAL TRANSMITTER
3B	59	TDPD	DISPLAY
3C	60	TDPR	DISPLAY : REPLY
3D	61	TXCF	EOF
3E	62	TXCW	WR LENGTH
3F	63	TXCR	ERROPT
40	64	VTBM	BIN. FLOAT TO BIN. FIXED
41	65	VBTM	BIN. FIXED TO BIN. FLOAT
42	66	VIIM	BIT STRING TO BIT STRING
43	67	VRPM	STERLING TO DECIMAL FIXED
44	68	VNPM	FIXED NUM. FIELD TO DEC. FIXED
45	69	FPNM	DEC. FIXED TO FIXED NUM. FIELD
46	70	VPRM	DEC. FIXED TO STERLING NUM. FIELD
47	71		
48	72		
49	73		
4A	74		
4B	75		
4C	76		
4D	77		
4E	78		
4F	79		
50	80	STMM	TIME
51	81	SDTM	DATE
52	82		
53	83		
54	84	QQSM	SQUARE ROOT (SHORT)
55	85	QQLM	SOUARE ROOT (LONG)
56	86	QASM	EXPONENTIATION (SHORT)
57	87	QALM	EXPONENTIATION (LONG)
58	88	QLSA	LOG (SHORT)
59	89	QLLA	LOG (LONG)
5A	90	QLSC	LOG2 (SHORT)
5B	91	QLLC	LOG2 (LONG)
5C	92	QLSB	LOG10 (SHORT)
5D	93	QLLB	LOG10 (LONG)
5E	94	QSSD	SINE (SHORT)
5F	95	QSLD	SINE (LONG)
60	96	QSSC	SINE-DEGREE (SHORT)
61	97	QSLC	SINE-DEGREE (LONG)
62	98	QSSB	COSINE (SHORT)
63	99	QSLB	COSINE (LONG)
64	100	QSSA	COSINE-DEGREE (SHORT)
65	101	QSLA	COSINE-DEGREE (LONG)
66	102	QTSB	TAN (SHORT)
67	103	QTLB	TAN (LONG)
68	104	QTSA	TAN-DEGREE (SHORT)
69	105	QTLA	TAN-DEGREE (LONG)
6A	106	QCSA	SINH (SHORT)
6B	107	QCLA	SINH (LONG)
6C	108	QCSB	COSH (SHORT)
6D	109	QCLB	COSH (LONG)
6E	110	QDSA	TANH (SHORT)
6F	111	QDLA	TANH (LONG)
70	112	QBSA	ATANH (SHORT)
71	113	QBLA	ATANH (LONG)
72	114	QRSB	ERF (SHORT)
73	115	QRLB	ERF (LONG)
74	116	QRSA	ERFC (SHORT)
75	117	QRLA	ERFC (LONG)
76	118	QNSD	ATAN (SHORT)

Note: The underlined module names are primary entry points.

INT. NAME HEX.	INT. NAME DEC.	MOD. NAME Pos. 4-7	DESCRIPTION
77	119	QNLD	ATAN (LONG)
78	120	QNSC	ATAN-DEGREE (SHORT)
79	121	QNLC	ATAN-DEGREE (LONG)
7A	122	QNSB	ATAN- (X,Y) SHORT)
7B	123	QNLB	ATAN- (X,Y) (LONG)
7C	124	ANSA	ATAN-DEGREE (X,Y) (SHORT)
7D	125	QNLA	ATAN-DEGREE (X,Y) (LONG)
7E	126	RBKB	REPEAT BIT
7F	127	RBKA	BIT CONCATENATION
80	128	RBIM	INDEX BIT
81	129	RGIM	INDEX CHARACTER
82	130		
83	131		
84	132	RBBM	BOOL
85	133	RGKM	REPEAT CHARACTER
86	134	RMSX	MAX (FLOAT SHORT)
87	135	RMLX	MAX (FLOAT LONG)
88	136	RMBX	MAX (FIXED BINARY)
89	137	RMPX	MAX (FIXED DECIMAL)
8A	138	RMSN	MIN (FLOAT SHORT)
8B	139	RMLN	MIN (FLOAT LONG)
8C	140	RMBN	MIN (FIXED BINARY)
8D	141	RMPN	MIN (FIXED DECIMAL)
8E	142		SUBSTR BIT (RIGHT)
8F	143		SUBSTR CHAR (RIGHT)
90	144		SUBSTR BIT (LEFT)
91	145		SUBSTR CHAR (LEFT)
92	146	RESM	EXP (FLOAT SHORT + INTEGER)
93	147	RELM	EXP (FLOAT LONG + INTEGER)
94	148	REPM	EXP (DECIMAL + INTEGER)
95	149	REBM	EXP (BIN. FIXED) INTEGER
96	150	RXSA	EXP (GENERAL SHORT)
97	151	RXLM	EXP (GENERAL LONG)
98	152		
99	153		
9A	154		
9B	155		
9C	156		
9D	157		
9E	158		
9F	159		
A0	160		ABS
A1	161		SIGN
A2	162		FLOOR
A3	163		CEIL
A4	164		UNSPEC
A5	165		
A6	166		BIT
A7	167		CHAR
A8	168		
A9	169		BINARY
AA	170		DECIMAL
AB	171		FLOAT
AC	172		FIXED
AD	173		PRECISION
AE	174		
AF	175		ADD
B0	176		MULTIPLY
B1	177		DIVIDE
B2	178		HIGH
B3	179		LOW
B4	180		SUM
B5	181		PROD
B6	182		ALL
B7	183		ANY
B8	184		ADDRESS
B9	185		STRING
BA	186		
BB	187		
BC	188		
BD	189		
BE	190		
BF	191		
C0	192		
C1	193		
C2	194	SDMP	DYNDUMP
C3	195		
C4	196		
C5	197		
C6	198		
C7	199		
C8	200	RTSM	FLOOR (FLOAT SHORT)
C9	201	RTLM	FLOOR (FLOAT LONG)
CA	202	RTBM	FLOOR (BIN. FIXED)
CB	203	RTPM	FLOOR (DEC. FIXED)
CC	204	RVSM	CEIL (FLOAT SHORT)
CD	205	RVLM	CEIL (FLOAT LONG)
CE	206	RVBM	CEIL (BIN. FIXED)
CF	207	RVPM	ceil (DEC. FIXED)
D0	208	RSSM	MOD (FLOAT SHORT)
D1	209	RSLM	MOD (FLOAT LONG)
D2	210	RSBM	MOD (BIN.FIXED)
D3	211	RSPM	MOD (DEC. FIXED)
D4	212	RUSM	ROUND (FLOAT SHORT)
D5	213	RULM	ROUND (FLOAT LONG)
D6	214	RUBM	ROUND (BIN. FIXED)
D7	215	RUPM	ROUND (DEC. FIXED)
D8	216	RWSM	TRUNC (FLOAT SHORT)
D9	217	RWLM	TRUNC (FLOAT LONG)
DA	218	RWBM	TRUNC (BIN. FIXED)
DB	219	RWPM	TRUNC (DEC. FIXED)
DC	220		ABS (FLOAT SHORT)
DD	221		ABS (FLOAT LONG)
DE	222		ABS (BIN. FIXED)
DF	223		ABS (DEC. FIXED)

Note: The underlined module names are primary entry points.

APPENDIX D. DTF TABLES

```
000000   FILECDI    START   0                             CARD INPUT
000000              DC      X'C2'                         OPEN MASK
000001              DC      AL3(TABAD)                    TABLE ADDRESS
000004              DC      X'01'                         FLAG BYTE ONE
000005              DC      AL3(0)                        CHAIN ADDRESS
000008              DC      X'0000'                       FLAG BYTE TWO, COMM BYTE
00000A              DC      H'0'                          RECORD LENGTH
00000C              DC      A(0)
000010              DC      A(IOA1)                       BUFFER ADDRESS
000014              DC      H'0'                          REMAINING DATA
000016              DC      H'0'                          DATA LENGTH
000018              DC      0D'0'
000018   TABAD      DC      X'000080000000'               RES. COUNT. COM., STATUS BTS
00001E              DC      AL1(0)                        LOGICAL UNIT CLASS
00001F              DC      AL1(0)                        LOGICAL UNIT
000020              DC      A(CCWAD)                      CCW-ADDRESS
000024              DC      4X'00'                        CCB-ST BYTE,CSW CCW ADDR.
000028              DC      V(IJCFZIZ0)                   ADDR OF LOGIC MODULE
00002C              DC      X'02'                         DTF TYPE  (READER)
00002D              DC      AL1(1)                        SWITCHES
00002E              DC      AL1(2)                        NORMAL COMM. CODE
00002F              DC      AL1(2)                        CNTROL COMM. CODE
000030              DC      A(IOA1)                       ADDR OF IOAREA1
000034              DC      V(IJKTXCF)                    EOF ADDRESS
000038   CCWAD      CCW     2,IOA1,X'20',80
000040              NOP     0
000044              NOP     0
000048              DC      X'0000'
00004A
000050   IOA1       DC      0D'0'                         IOAREA1
                    END
```

```
000000   FILECD01   START   0                             CARD OUTPUT, DEVICE 1442
000000              DC      X'A2'                         OPEN MASK
000001              DC      AL3(TABAD)                    TABLE ADDRESS
000004              DC      X'01'                         FLAG BYTE ONE
000005              DC      AL3(0)                        CHAIN ADDRESS
000008              DC      X'0000'                       FLAG BYTE TWO, COMM BYTE
00000A              DC      H'0'                          RECORD LENGTH
00000C              DC      A(0)
000010              DC      A(IOA1)                       BUFFER ADDRESS
000014              DC      H'0'                          REMAINING DATA
000016              DC      H'0'                          DATA LENGTH
000018              DC      0D'0'
000018   TABAD      DC      X'000080000000'               RES. COUNT,COM. BYTE,STATUS BTS
00001E              DC      AL1(0)                        LOGICAL UNIT CLASS
00001F              DC      AL1(0)                        LOGICAL UNIT
000020              DC      A(CCWAD)                      CCW ADDRESS
000024              DC      4X'00'                        CCB-ST BYTE,CSW CCW ADDR.
000028              DC      V(IJCFZOI0)                   ADDR OF LOGIC MODULE
00002C              DC      X'04'                         DTF TYPE = PUNCH
00002D              DC      AL1(16)                       SWITCHES
00002E              DC      AL1(65)                       NORMAL COMM. CODE
00002F              DC      AL1(65)                       CONTROL COMM. CODE
000030              DC      A(IOA1)                       ADDR. OF DATA IN IOAREA1
000034              DC      CL4' '                        BUCKET
000038              NOPR    0
00003A              NOP     0
00003E              DC      X'00'                         SWITCH 2
00003F              DC      C' '                          BLANK FOR EJECT LAST PRG. CARD
000040   CCWAD      CCW     65,IOA1,X'20',80
000048   IOA1       DC      0D'0'                         IOAREA1
                    END
```

```
000000   FILECD02  START   0                               CARD OUTPUT, 2520B1
000000             DC      X'A2'                           OPEN MASK
000001             DC      AL3(TABAD)                      TABLE ADDRESS
000004             DC      X'01'                           FLAG BYTE ONE
000005             DC      AL3(0)                          CHAIN ADDRESS
000008             DC      X'0000'                         FLAG BYTE TWO, COMM BYTE
00000A             DC      H'0'                            RECORD LENGTH
00000C             DC      A(0)
000010             DC      A(IOA1)                         BUFFER ADDRESS
000014             DC      H'0'                            REMAINING DATA
000016             DC      H'0'                            DATA LENGTH
000018             DC      0D'0'
000018   TABAD     DC      X'000080000000'                 RES. COUNT,COM. BYTES,STATUS BTS
00001E             DC      AL1(0)                          LOGICAL UNIT CLASS
00001F             DC      AL1(0)                          LOGICAL UNIT
000020             DC      A(CCWAD)                        CCW2 ADDRESS
000024             DC      4X'00'                          CCB-ST BYTE,CCW ADDR.
000028             DC      V(IJCFZOZ2)                     ADDR. OF LOGIC MOD
00002C             DC      X'04'                           DTF TYPE (PUNCH)
00002D             DC      AL1(16)                         SWITCHES
00002E             DC      AL1(65)                         NORMAL COMM. CODE
00002F             DC      AL1(65)                         CONTROL COMM. CODE
000030             DC      A(IOA1)                         ADDR. OF DATA IN IOAREA1
000034             DC      CL4' '                          BUCKET
000038             NOPR    0
00003A             NOP     0
00003E             DC      X'01'                           SWITCH 2
00003F             DC      C' '                            BLANK FOR EJECT LAST PRG. CARD
000040             CCW     65,IOA1,X'20',80
000048   CCWAD     CCW     1,*-9,X'20',1                   FOR EJECT LAST PROG. CARD
000050   IOA1      DC      0D'0'                           IOAREA1
                   END


000000   FILECD03  START   0                               CARD OUTPUT,DEVICE 2540
000000             DC      X'A2'                           OPEN MASK
000001             DC      AL3(TABAD)                      TABLE ADDRESS
000004             DC      X'01'                           FLAG BYTE ONE
000005             DC      AL3(0)                          CHAIN ADDRESS
000008             DC      X'0000'                         FLAG BYTE TWO, COMM BYTE
00000A             DC      H'0'                            RECORD LENGTH
00000C             DC      A(0)
000010             DC      A(IOA1)                         BUFFER ADDRESS
000014             DC      H'0'                            REMAINING DATA
000016             DC      H'0'                            DATA LENGTH
000018             DC      0D'0'
000018   TABAD     DC      X'000084000400'                 RES. COUNT,COM. BYTE,STATUS BTS
00001E             DC      AL1(0)                          LOGICAL UNIT CLASS
00001F             DC      AL1(0)                          LOGICAL UNIT
000020             DC      A(CCWAD)                        CCW ADDRESS
000024             DC      4X'00'                          CCB-ST BYTE,CSW CCW ADDR.
000028             DC      V(IJCFZOI4)                     ADDR OF LOGIC MODULE
00002C             DC      X'04'                           DTF TYPE = PUNCH
00002D             DC      AL1(16)                         SWITCHES
00002E             DC      AL1(65)                         NORMAL COMM. CODE
00002F             DC      AL1(65)                         CONTROL COMM. CODE
000030             DC      A(IOA1)                         ADDR. OF DATA IN IOAREA1
000034             DC      CL4' '                          BUCKET
000038             NOPR    0
00003A             NOP     0
00003E             DC      X'00'                           SWITCH 2
00003F             DC      C' '                            BLANK FOR EJECT LAST PRG. CARD
000040   CCWAD     CCW     65,IOA1,X'20',80
000048             CCW     1,*+8,X'20',80                  FOR PUNCH ERROR RETRY
000050             DC      CL80' '                         AREA FOR SAVE CARD IMAGE
0000A0   IOA1      DC      0D'0'                           IOAREA1
                   END
```

```
000000   FILEPRR   START   0                      PRINTER, RECORD ORIENTED
000000             DC      X'A2'                  OPEN MASK
000001             DC      AL3(TABAD)             TABLE ADDRESS
000004             DC      X'09'                  FLAG BYTE ONE
000005             DC      AL3(0)                 CHAIN ADDRESS
000008             DC      X'0000'                FLAG BYTE TWO, COMM.BYTE
00000A             DC      H'00'                  RECORD LENGTH
00000C             DC      A(0)
000010             DC      A(IOA1)                BUFFER ADDRESS
000014             DC      H'0'                   REMAINING DATA
000016             DC      H'0'                   DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000084000400'        RES. CNT, COM. BYTES, STATUS BTS
00001E             DC      AL1(0)                 LOGICAL UNIT CLASS
00001F             DC      AL1(0)                 LOGICAL UNIT
000020             DC      A(CCWAD)               CCW ADDR.
000024             DC      4X'00'                 CCB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJDFZPZZ)            ADDRESS OF LOGIC MODULE
00002C             DC      X'08'                  DTF TYPE (PRINTER)
00002D             DC      AL1(16)                SWITCHES
00002E             DC      X'09'                  NORMAL COMM. CODE
00002F             DC      X'09'                  CONTROL COMM. CODE
000030             DC      A(IOA1)                ADDRESS OF DATA IN IOAREA1
000034             DC      4X'00'                 BUCKET
000038             NOPR    0
00003A             NOP     0
00003E             DC      2X'00'                 NOT USED
000040   CCWAD     CCW     9,IOA1,X'20',120
000048   IOA1      DC      0D'0'
                   END
```

```
000000   FILEPRP   START   0                      PRINTER WITH PRINT OPTION
000000             DC      X'A2'                  OPEN MASK
000001             DC      AL3(TABAD)             TABLE ADDRESS
000004             DC      X'0D'                  FLAG BYTE ONE
000005             DC      AL3(0)                 CHAIN ADDRESS
000008             DC      X'0000'                FLAG BYTE TWO, COMM.BYTE
00000A             DC      H'0'                   RECORD LENGTH
00000C             DC      A(0)
000010             DC      A(IOA1-1)              BUFFER ADDRESS
000014             DC      H'0'                   REMAINING DATA
000016             DC      H'0'                   DATA LENGTH
000018             DC      H'0'                   PAGE SIZE
00001A             DC      H'0'                   CURRENT LINE
00001C
000020   TABAD     DC      0D'0'
000020             DC      X'000080000000'        RES. COUNT, COM. BYTES, STATUS BTS
000026             DC      AL1(0)                 LOGICAL UNIT CLASS
000027             DC      AL1(0)                 LOGICAL UNIT
000028             DC      A(CCWAD)               CCW ADDR.
00002C             DC      4X'00'                 CCB-ST BYTE,CSW CCW ADDRESS
000030             DC      V(IJDFAZZZ)            ADDRESS OF LOGIC MODULE
000034             DC      X'08'                  DTF TYPE (PRINTER)
000035             DC      AL1(48)                SWITCHES
000036             DC      X'09'                  NORMAL COMM. CODE
000037             DC      X'09'                  CONTROL COMM. CODE
000038             DC      A(IOA1)                ADDRESS OF DATA IN IOAREA1
00003C             DC      4X'00'                 BUCKET
000040             NOPR    0
000042             NOP     0
000046             DC      2X'00'                 NOT USED
000048   CCWAD     CCW     9,IOA1,X'20',120
000050             DC      X'00'                  CONTROL CHARACTER FIELD
000051   IOA1      DC      0C'0'
                   END
```

```
000000   FILETAUN  START    0                   TAPE UNBUFFERED FILE
000000             DC       X'83'               OPEN MASK
000001             DC       AL3(TABAD)          TABLE ADDRESS
000004             DC       X'01'               FLAG BYTE ONE
000005             DC       AL3(0)              CHAIN ADDRESS
000008             DC       X'4600'             FLAG BYTE TWO, COMM. BYTE
00000A             DC       H'0'                RECORD LENGTH
00000C             DC       X'40000000'
000010   TABAD     DC       0D'0'
000010             DC       X'000082000000'     CCB
000016             DC       AL1(0)              LOGICAL UNIT CLASS
000017             DC       AL1(0)              LOGICAL UNIT
000018             DC       AL4(CCWAD)          CCW ADDRESS
00001C             DC       4X'00'              CCB-ST BYTE,CSW CCW ADDRESS
000020             DC       V(IJFWEZZZ)         ADDR OF LOGICAL MODULE
000024             DC       X'10'               DTF TYPE
000025             DC       AL1(32)             LOGICAL INDICATORS
000026             DC       X'0000'
000028             DC       H'0'                RECORD LENGTH
00002A             DC       H'0'                BLOCKSIZE
00002C             DC       X'02'               READ OP CODE
00002D             DC       AL3(IJKTXCF)        END OF FILE ADDRESS
000030   CCWAD     CCW      X'02',*,X'20',1     CHANNEL PROGRAM
000038             DC       F'0'                BLOCK COUNT
00003C             DC       AL1(128)            READ ERROR OPTION INDIC.
00003D             DC       AL3(IJKTXCR)        READ ERROR ROUTINE
                   EXTRN    IJKTXCF,IJKTXCR
                   END
```

```
000000   FILEDIUN  START   0                        DISK UNBUFFERED FILE, NO UPDATE
000000             DC      X'83'                    OPEN MASK
000001             DC      AL3(TABAD)               TABLE ADDRESS
000004             DC      X'01'                    FLAG BYTE ONE
000005             DC      AL3(0)                   CHAIN ADDRESS
000008             DC      X'4600'                  FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                     RECORD LENGTH
00000C             DC      A(0)
000010             DC      0D'0'
000010   TABAD     DC      X'000082000000'          CCB
000016             DC      X'FFFF'                  CCB-LOGICAL UNIT
000018             DC      A(CCWAD)                 CCB-CCW ADDRESS
00001C             DC      4X'00'                   CCB-ST BYTE,CSW CCW ADDRESS
000020             DC      V(IJGWEZZZ)              LOGIC MODULE ADDRESS
000024             DC      X'20'                    DTF TYPE
000025             DC      AL1(32)                  OPEN/CLOSE INDICATORS
000026             DC      CL8'FILEDIUN'            FILENAME
00002E             DC      H'3625'                  TRACK CAPACITY COUNTER
000030             DC      7X'00'
000037             DC      X'09'                    UPPER HEAD LIMIT
000038             DC      H'0'                     RECORD LENGTH
00003A             DC      14X'00'
000048             DC      X'0000FF00'              SEARCH ADDRESS-CCHH
00004C             DC      2X'00'
00004E             DC      H'0'                     MAXIMUM RECORD LENGTH
000050             DC      AL1(32)                  VERIFY CHAIN BIT
000051             DC      AL3(IJKTXCF)             EOF ADDRESS
000054             DC      AL1(128)                 LOGICAL INDICATORS
000055             DC      AL3(IJKTXCR)             USER'S ERROR ROUTINE
000058   CCWAD     CCW     7,*-18,64,6              SEEK
000060             CCW     X'31',*-24,64,5          SEARCH ID EQUAL
000068             CCW     8,*-8,0,0                TIC
000070             CCW     3,*,32,1                 WRITE CKD OR READ DATA
000078             CCW     5,*,32,1                 WRITE DATA/READ COUNT
000080             CCW     X'31',*-56,64,5          SEARCH ID EQUAL
000088             CCW     8,*-8,0,0                TIC
000090             CCW     X'1E',*+16,48,8          VERIFY
000098             CCW     X'12',*+8,0,8            READ COUNT
0000A0             DC      X'0000000001000000'      COUNT AREA
                   EXTRN   IJKTXCF,IJKTXCR
                   END
```

```
000000   FILEDIUU  START   0                        DISK UNBUFFERED FILE, WITH UPDATE
000000             DC      X'9B'                    OPEN MASK
000001             DC      AL3(TABAD)               TABLE ADDRESS
000004             DC      X'01'                    FLAG BYTE ONE
000005             DC      AL3(0)                   CHAIN ADDRESS
000008             DC      X'4600'                  FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                     RECORD LENGTH
00000C             DC      A(0)
000010             DC      A(0)
000014
000018             DC      0D'0'
000018   TABAD     DC      X'000082000000'          CCB
00001E             DC      X'FFFF'                  CCB-LOGICAL UNIT
000020             DC      A(CCWAD)                 CCB-CCW ADDRESS
000024             DC      4X'00'                   CCB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJGWEZZZ)              LOGIC MODULE ADDRESS
00002C             DC      X'20'                    DTF TYPE
00002D             DC      AL1(32)                  OPEN/CLOSE INDICATORS
00002E             DC      CL8'FILEDIUU'            FILENAME
000036             DC      H'3625'                  TRACK CAPACITY COUNTER
000038             DC      7X'00'
00003F             DC      X'09'                    UPPER HEAD LIMIT
000040             DC      H'0'                     RECORD LENGTH
000042             DC      14X'00'
000050             DC      X'0000FF00'              SEARCH ADDRESS-CCHH
000054             DC      2X'00'
000056             DC      H'0'                     MAXIMUM RECORD LENGTH
000058             DC      AL1(32)                  VERIFY CHAIN BIT
000059             DC      AL3(IJKTXCF)             EOF ADDRESS
00005C             DC      AL1(128)                 LOGICAL INDICATORS
00005D             DC      AL3(IJKTXCR)             USER'S ERROR ROUTINE
000060   CCWAD     CCW     7,*-18,64,6              SEEK
000068             CCW     X'31',*-24,64,5          SEARCH ID EQUAL
000070             CCW     8,*-8,0,0                TIC
000078             CCW     3,*,32,1                 WRITE CKD OR READ DATA
000080             CCW     5,*,32,1                 WRITE DATA/READ COUNT
000088             CCW     X'31',*-56,64,5          SEARCH ID EQUAL
000090             CCW     8,*-8,0,0                TIC
000098             CCW     X'1E',*+16,48,8          VERIFY
0000A0             CCW     X'12',*+8,0,8            READ COUNT
0000A8             DC      X'0000000001000000'      COUNT AREA
                   EXTRN   IJKTXCF,IJKTXCR
                   END
```

```
000000   FILETAFI  START   0                          TAPE FILE FIXED INPUT
000000             DC      X'C2'                      OPEN MASK
000001             DC      AL3(TABAD)                 TABLE ADDRESS
000004             DC      X'01'                      FLAG BYTE ONE
000005             DC      AL3(0)                     CHAIN ADDRESS
000008             DC      X'0000'                    FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                       RECORD LENGTH
00000C             DC      X'40000000'
000010             DC      A(IOA1)                    BUFFER ADDRESS
000014             DC      H'0'                       REMAINING DATA
000016             DC      H'0'                       DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000082000000'            CCB
00001E             DC      AL1(0)                     LOGICAL UNIT CLASS
00001F             DC      AL1(0)                     LOGICAL UNIT
000020             DC      AL4(CCWAD)                 CCW ADDRESS
000024             DC      4X'00'                     CCB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJFFZZZZ)                ADDRESS OF LOGICAL MODULE
00002C             DC      X'11'                      DTF TYPE
00002D             DC      AL1(8)                     LOGICAL IOCS SWITCHES
00002E             DC      CL8'FILETAFI'              FILE NAME
000036             DC      X'0200'
000038             DC      AL1(0)                     SWITCH ONE FOR OPEN AND CLOSE
000039             DC      AL3(0)
00003C             DC      AL1(32)                    SWITCH TWO FOR OPEN AND CLOSE
00003D             DC      AL3(IJKTXCF)               EOF-ADDRESS
000040             DC      F'0'                       BLOCKCOUNT
000044             BXH     11,12,24(15)               DEBLOCKING FORWARD
000048             LA      14,1(14)                   INCREASE BLOCKCOUNT BY ONE
00004C             L       2,IJF2-TABAD(1)            LOAD USER REGISTER
000050   CCWAD     CCW     X'02',IOA1,X'00',0
000058             DC      A(IOA1)                    ADDRESS OF IOAREA
00005C   IJF2      DC      F'0'                       DEBLOCKER 1
000060             DC      F'0'                       DEBLOCKER 2
000064             DC      F'0'                       DEBLOCKER 3
000068             DC      Y(0)                       BLOCKSIZE
00006A             DC      Y(0)                       BLOCKSIZE-1
00006C             DC      Y(0)                       RECSIZE-1
00006E             DC      H'0'                       NOT USED
000070             DC      A(IJKTXCW)                 WLR-ADDRESS
000074             DC      A(IJKTXCR)                 ERROR EXIT
000078   IOA1      DC      0D'0'
                   EXTRN   IJKTXCF,IJKTXCW,IJKTXCR
                   END
```

```
000000  FILETAFO  START   0                      TAPE FILE FIXED OUTPUT
000000            DC      X'A2'                  OPEN MASK
000001            DC      AL3(TABAD)             TABLE ADDRESS
000004            DC      X'01'                  FLAG BYTE ONE
000005            DC      AL3(0)                 CHAIN ADDRESS
000008            DC      X'0000'                FLAG BYTE TWO, COMM. BYTE
00000A            DC      H'0'                   RECORD LENGTH
00000C            DC      X'40000000'
000010            DC      A(IOA1)                BUFFER ADDRESS
000014            DC      H'0'                   REMAINING DATA
000016            DC      H'0'                   DATA LENGTH
000018  TABAD     DC      0D'0'
000018            DC      X'000080000000'        CCB
00001E            DC      AL1(0)                 LOGICAL UNIT CLASS
00001F            DC      AL1(0)                 LOGICAL UNIT
000020            DC      AL4(CCWAD)             CCW ADDRESS
000024            DC      4X'00'                 CCB-ST BYTE,CSW CCW ADDRESS
000028            DC      V(IJFFZZZZ)            ADDRESS OF LOGICAL MODULE
00002C            DC      X'11'                  DTF TYPE
00002D            DC      AL1(0)                 LOGICAL IOCS SWITCHES
00002E            DC      CL8'FILETAFO'          FILE NAME
000036            DC      X'0100'
000038            DC      AL1(0)                 SWITCH ONE FOR OPEN AND CLOSE
000039            DC      AL3(0)
00003C            DC      AL1(0)                 SWITCH TWO FOR OPEN AND CLOSE
00003D            DC      AL3(*)                 EOF-ADDRESS
000040            DC      F'0'                   BLOCKCOUNT
000044            BXH     11,12,24(15)           DEBLOCKING FORWARD
000048            LA      14,1(14)               INCREASE BLOCKCOUNT BY ONE
00004C            L       2,IJF2-TABAD(1)        LOAD USER REGISTER
000050  CCWAD     CCW     X'01',IOA1,X'00',0
000058            DC      A(IOA1)
00005C  IJF2      DC      A(IOA1)                DEBLOCKER 1
000060            DC      F'0'                   DEBLOCKER 2
000064            DC      A(IOA1-1)              DEBLOCKER 3
000068            DC      Y(0)                   BLOCKSIZE
00006A            DC      Y(0)                   BLOCKSIZE-1
00006C            DC      Y(0)                   RECSIZE-1
00006E
000070  IOA1      DC      0D'0'
                  END
```

```
000000   FILETASP  START   0                         TAPE STREAM FILE WITH PRINT OPTION
000000             DC      X'A2'                     OPEN MASK
000001             DC      AL3(TABAD)                TABLE ADDRESS
000004             DC      X'0D'                     FLAG BYTE ONE
000005             DC      AL3(0)                    CHAIN ADDRESS
000008             DC      X'C000'                   FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                      RECORD LENGTH
00000C             DC      X'40000000'
000010             DC      A(IOA1)                   BUFFER ADDRESS
000014             DC      H'0'                      REMAINING DATA
000016             DC      H'0'                      DATA LENGTH
000018             DC      H'0'                      PAGE SIZE
00001A             DC      H'0'                      CURRENT LINE
00001C
000020   TABAD     DC      0D'0'
000020             DC      X'000080000000'           CCB
000026             DC      AL1(0)                    LOGICAL UNIT CLASS
000027             DC      AL1(0)                    LOGICAL UNIT
000028             DC      AL4(CCWAD)                CCW ADDRESS
00002C             DC      4X'00'                    CCB-ST BYTE,CSW CCW ADDRESS
000030             DC      V(IJFFZZZZ)               ADDRESS OF LOGICAL MODULE
000034             DC      X'11'                     DTF TYPE
000035             DC      AL1(0)                    LOGICAL IOCS SWITCHES
000036             DC      CL8'FILETASP'             FILE NAME
00003E             DC      X'0100'
000040             DC      AL1(0)                    SWITCH ONE FOR OPEN AND CLOSE
000041             DC      AL3(0)
000044             DC      AL1(0)                    SWITCH TWO FOR OPEN AND CLOSE
000045             DC      AL3(*)                    EOF-ADDRESS
000048             DC      F'0'                      BLOCKCOUNT
00004C             BXH     11,12,24(15)              DEBLOCKING FORWARD
000050             LA      14,1(14)                  INCREASE BLOCKCOUNT BY ONE
000054             L       2,IJF2-TABAD(1)           LOAD USER REGISTER
000058   CCWAD     CCW     X'01',IOA1,X'00',0
000060             DC      A(IOA1)
000064   IJF2      DC      A(IOA1)                   DEBLOCKER 1
000068             DC      F'0'                      DEBLOCKER 2
00006C             DC      A(IOA1-1)                 DEBLOCKER 3
000070             DC      Y(0)                      BLOCKSIZE
000072             DC      Y(0)                      BLOCKSIZE-1
000074             DC      Y(0)                      RECSIZE-1
000076   IOA1      DC      OH'0'
                   END
```

```
000000  FILETAFB  START   0                        TAPE FILE BACK FIXED RECORDS
000000            DC      X'C2'                    OPEN MASK
000001            DC      AL3(TABAD)               TABLE ADDRESS
000004            DC      X'03'                    FLAG BYTE ONE
000005            DC      AL3(0)                   CHAIN ADDRESS
000008            DC      X'4500'                  FLAG BYTE TWO, COMM. BYTE
00000A            DC      H'0'                     RECORD LENGTH
00000C            DC      X'40000000'
000010            DC      A(IOA1)                  BUFFER ADDRESS
000014            DC      H'0'                     REMAINING DATA
000016            DC      H'0'                     DATA LENGTH
000018  TABAD     DC      0D'0'
000018            DC      X'000082000000'          CCB
00001E            DC      AL1(0)                   LOGICAL UNIT CLASS
00001F            DC      AL1(0)                   LOGICAL UNIT
000020            DC      A(CCWAD)                 CCW ADDRESS
000024            DC      4X'00'                   CCB-ST. BYTE, CSW CCW ADDRESS
000028            DC      V(IJFFBZZZ)              ADDRESS OF LOGICAL MODULE
00002C            DC      X'11'                    DTF TYPE
00002D            DC      AL1(8)                   LOGICAL IOCS SWITCHES
00002E            DC      CL8'FILETAFB'            FILE NAME
000036            DC      X'0C00'
000038            DC      AL1(0)                   SWITCH ONE FOR OPEN AND CLOSE
000039            DC      AL3(0)
00003C            DC      AL1(32)                  SWITCH TWO FOR OPEN AND CLOSE
00003D            DC      AL3(IJKTXCF)             EOF-ADDRESS
000040            DC      F'0'                     BLOCKCOUNT
000044            BXLE    11,12,24(15)             DEBLOCKING BACKWARD
000048            BCTR    14,0                     DECREASE BLOCKCOUNT BY ONE
00004A            NOPR    0
00004C            L       2,IJF2-TABAD(1)          LOAD USER REGISTER
000050  CCWAD     CCW     X'0C',IOA1,X'00',0
000058            DC      A(IOA1)                  ADDRESS OF IOAREA
00005C  IJF2      DC      A(IOA1)                  DEBLOCKER 1
000060            DC      F'0'                               2
000064            DC      A(IOA1)                            3
000068            DC      Y(0)                     BLOCKSIZE
00006A            DC      Y(0)                     BLOCKSIZE+1
00006C            DC      Y(0)                     RECORDSIZE-1
00006E            DC      H'0'
000070            DC      V(IJKTXCW)               WLR-ADDRESS
000074            DC      V(IJKTXCR)               ERROR EXIT
000078  IOA1      DC      0D'0'
                  EXTRN   IJKTXCF
                  END
```

```
000000  FILETAVI  START   0                        TAPE INPUT FILE VARIABLE RECORDS
000000            DC      X'C2'                    OPEN MASK
000001            DC      AL3(TABAD)               TABLE ADDRESS
000004            DC      X'11'                    FLAG BYTE ONE
000005            DC      AL3(0)                   CHAIN ADDRESS
000008            DC      X'4500'                  FLAG BYTE TWO, COMM. BYTE
00000A            DC      H'0'                     RECORD LENGTH
00000C            DC      X'40000000'
000010            DC      A(IOA1)                  BUFFER ADDRESS
000014            DC      H'0'                     REMAINING DATA
000016            DC      H'0'                     DATA LENGTH
000018  TABAD     DC      0D'0'
000018            DC      X'000082000000'          CCB
00001E            DC      AL1(0)                   LOGICAL UNIT CLASS
00001F            DC      AL1(0)                   LOGICAL UNIT
000020            DC      AL4(CCWAD)               CCW ADDRESS
000024            DC      4X'00'                   CCB-ST BYTE,CSW CCW ADDRESS
000028            DC      V(IJFVZZZZ)              ADDRESS OF LOGICAL MODULE
00002C            DC      X'11'                    DTF TYPE
00002D            DC      AL1(72)                  LOGICAL IOCS SWITCHES
00002E            DC      CL8'FILETAVI'            FILE NAME
000036            DC      X'0200'                  INPUT
000038            DC      AL1(0)                   SWITCH ONE FOR OPEN AND CLOSE
000039            DC      AL3(0)
00003C            DC      AL1(32)                  SWITCH TWO FOR OPEN AND CLOSE
00003D            DC      AL3(IJKTXCF)             EOF-ADDRESS
000040            DC      F'0'                     BLOCKCOUNT
000044            NOP     0(0)
000048            LA      14,1(14)                 INCREASE BLOCKCOUNT BY ONE
00004C            L       2,IJF4-TABAD(1)          LOAD USER IOREG
000050  CCWAD     CCW     X'02',IOA1,X'00',0
000058            DC      A(IOA1)                  ADDRESS OF IOAREA
00005C  IJF3      DC      F'0'
000060  IJF2      DC      A(IOA1)                  DEBLOCKER 3,
000064            DC      2F'0'                              4,5,
00006C  IJF4      DC      A(IOA1+4)                          1,
000070            DC      F'0'                               6
000074            DC      Y(0)                     BLOCKSIZE
000076            DC      Y(0)                     BLOCKSIZE-1
000078            DC      Y(0)
00007A            DC      H'0'
00007C            DC      V(IJKTXCW)               WLR-ADDRESS
000080            DC      V(IJKTXCR)               ERROR EXIT
000084
000088  IOA1      DC      0D'0'
                  EXTRN   IJKTXCF
                  END
```

```
000000  FILETAVO  START   0                      TAPE OUTPUT FILE VARIABLE RECORDS
000000            DC      X'A2'                  OPEN MASK
000001            DC      AL3(TABAD)             TABLE ADDRESS
000004            DC      X'11'                  FLAG BYTE ONE
000005            DC      AL3(0)                 CHAIN ADDRESS
000008            DC      X'4500'                FLAG BYTE TWO, COMM. BYTE
00000A            DC      H'0'                   RECORD LENGTH
00000C            DC      X'40000000'
000010            DC      A(IOA1)                BUFFER ADDRESS
000014            DC      H'0'                   REMAINING DATA
000016            DC      H'0'                   DATA LENGTH
000018  TABAD     DC      0D'0'
000018            DC      X'000080000000'        CCB
00001E            DC      AL1(0)                 LOGICAL UNIT CLASS
00001F            DC      AL1(0)                 LOGICAL UNIT
000020            DC      A(CCWAD)               CCW ADDRESS
000024            DC      X'0000'                CCB-ST BYTE, CSW CCW ADDRESS
000026
000028            DC      V(IJFVZZZZ)            ADDRESS OF LOGICAL MODULE
00002C            DC      X'11'                  DTF TYPE
00002D            DC      AL1(64)                LOGICAL IOCS SWITCHES
00002E            DC      CL8'FILETAVO'          FILE NAME
000036            DC      X'0100'                OUTPUT
000038            DC      AL1(0)                 SWITCH ONE FOR OPEN AND CLOSE
000039            DC      AL3(0)                 USER LABEL ROUTINE
00003C            DC      AL1(0)                 SWITCH TWO FOR OPEN AND CLOSE
00003D            DC      AL3(*)                 EOF-ADDRESS
000040            DC      F'0'                   BLOCKCOUNT
000044            L       3,IJF3-TABAD(1)
000048            LA      14,1(14)               INCREASE BLOCKCOUNT BY ONE
00004C            L       2,IJF4-TABAD(1)
000050  CCWAD     CCW     X'01',IOA1,X'00',0
000058            DC      A(IOA1)                ADDRESS OF IOAREA
00005C  IJF3      DC      F'0'
000060  IJF2      DC      A(IOA1)                DEBLOCKER 3,
000064            DC      2F'0'                            4,5,
00006C  IJF4      DC      A(IOA1+4)                        1,
000070            DC      F'0'                             6
000074            DC      Y(0)                   BLOCKSIZE-4
000076            DC      Y(0)                   BLOCKSIZE-1
000078            DC      Y(0)
00007A
000080  IOA1      DC      0D'0'
                  END
```

```
000000   FILETAUI   START    0                          TAPE INPUT FILE UNDEFINED RECORDS
000000              DC       X'C2'                      OPEN MASK
000001              DC       AL3(TABAD)                 TABLE ADDRESS
000004              DC       X'31'                      FLAG BYTE ONE
000005              DC       AL3(0)                     CHAIN ADDRESS
000008              DC       X'4500'                    FLAG BYTE TWO, COMM. BYTE
00000A              DC       H'0'                       RECORD LENGTH
00000C              DC       X'40000000'
000010              DC       A(IOA1)                    BUFFER ADDRESS
000014              DC       H'0'                       REMAINING DATA
000016              DC       H'0'                       DATA LENGTH
000018   TABAD      DC       0D'0'
000018              DC       X'000082000000'            CCB
00001E              DC       AL1(0)                     LOGICAL UNIT CLASS
00001F              DC       AL1(0)                     LOGICAL UNIT
000020              DC       AL4(CCWAD)                 CCW ADDRESS
000024              DC       4X'00'                     CCB-ST. BYTE, CSW CCW ADDRESS
000028              DC       V(IJFUZZZZ)                ADDRESS OF LOGICAL UNIT
00002C              DC       X'11'                      DTF TYPE
00002D              DC       AL1(8)                     LOGICAL IOCS SWITCHES
00002E              DC       CL8'FILETAUI'              FILE NAME
000036              DC       X'0200'
000038              DC       AL1(0)                     SWITCH ONE FOR OPEN ADN CLOSE
000039              dc       AL3(0)
00003C              DC       AL1(32)                    SWITCH TWO FOR OPEN AND CLOSE
00003D              DC       AL3(IJKTXCF)               EOF-ADDRESS
000040              DC       F'0'                       BLOCKCOUNT
000044   IJF4       DC       F'0'                       DEBLOCKER 1
000048              L        4,IJF4-TABAD(1)            GIVE USER RECSIZE
00004C              L        2,IJF2-TABAD(1)            LOAD USER IOREG
000050   CCWAD      CCW      X'00',IOA1,X'00',0
000058              DC       A(IOA1)
00005C   IJF2       DC       A(IOA1)                    DEBLOCKER 2
000060              LA       14,1(14)                   CHANGE BLOCKCOUNT
000064              DC       Y(0)                       BLOCKSIZE
000066              DC       Y(0)                       BLOCKSIZE-1
000068              NOPR     0
00006A              DC       H'0'
00006C              DC       V(IJKTXCR)                 ERROR EXIT
000070   IOA1       DC       0D'0'
                    EXTRN    IJKTXCR,IJKTXCF
                    END
```

```
000000   FILETAUO  START   0                              TAPE OUTPUT FILE UNDEFINED RECORDS
000000             DC      X'A2'                          OPEN MASK
000001             DC      AL3(TABAD)                     TABLE ADDRESS
000004             DC      X'31'                          FLAG BYTE ONE
000005             DC      AL3(0)                         CHAIN ADDRESS
000008             DC      X'4500'                        FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                           RECORD LENGTH
00000C             DC      X'40000000'
000010             DC      A(IOA1)                        BUFFER ADDRESS
000014             DC      H'0'                           REMAINING DATA
000016             DC      H'0'                           DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000080000000'                CCB
00001E             DC      AL1(0)                         LOGICAL UNIT CLASS
00001F             DC      AL1(0)                         LOGICAL UNIT
000020             DC      AL4(CCWAD)                     CCW ADDRESS
000024             DC      4X'00'                         CCB-ST. BYTE, CSW CCW ADDRESS
000028             DC      V(IJFUZZZZ)                    ADDRESS OF LOGICAL MODULE
00002C             DC      X'11'                          DTF TYPE
00002D             DC      AL1(0)                         LOGICAL IOCS SWITCHES
00002E             DC      CL8'FILETAUO'                  FILE NAME
000036             DC      X'0100'                        OUTPUT
000038             DC      AL1(0)                         SWITCH ONE FOR OPEN AND CLOSE
000039             DC      AL3(0)
00003C             DC      AL1(0)                         SWITCH TWO FOR OPEN AND CLOSE
00003D             DC      AL3(0)                         EOF-ADDRESS
000040             DC      F'0'                           BLOCKCOUNT
000044   IJF4      DC      F'0'                           DEBLOCKER 1
000048             NOP     0(0)
00004C             L       2,IJF2-TABAD(1)                LOAD USER IOREG
000050   CCWAD     CCW     X'01',IOA1,X'00',0
000058             DC      A(IOA1)
00005C   IJF2      DC      A(IOA1)                        DEBLOCKER 2
000060             LA      14,1(14)                       CHANGE BLOCKCOUNT
000064             DC      Y(0)                           BLOCKSIZE
000066             DC      Y(0)                           BLOCKSIZE-1
000068             LR      12,4                           PICK UP RECSIZE
00006A
000070   IOA1      DC      0D'0'
                   END
```

```
000000   FIDIINFI  START   0                        DISK FILE
000000             DC      X'C2'                    OPEN MASK
000001             DC      AL3(TABAD)               TABLE ADDRESS
000004             DC      X'01'                    FLAG BYTE ONE
000005             DC      AL3(0)                   CHAIN ADDRESS
000008             DC      X'0000'                  FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                     RECORD LENGTH
00000C             DC      X'80000000'
000010             DC      A(IOA1)                  BUFFER ADDRESS
000014             DC      H'0'                     REMAINING DATA
000016             DC      H'0'                     DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000082000000'          CCB
00001E             DC      X'FFFF'                  CCB-LOGICAL UNIT
000020             DC      A(CCWAD)                 CCB-CCW ADDRESS
000024             DC      4X'00'                   XXB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJGFIEZZ)              LOGIC MODULE
00002C             DC      X'20'                    DTF TYPE
00002D             DC      AL1(2)                   OPEN/CLOSE INDICATOR
00002E             DC      CL8'FIDIINFI'            FILENAME
000036             DC      8X'00'
00003E             DC      X'08'                    OPEN COMMUNICATIONS BYTE
00003F             DC      2X'00'
000041             DC      AL3(*)                   USER'S LABEL ADDRESS
000044             DC      A(IOA1)                  ADDRESS OF IOAREA
000048             DC      X'80000000'              CCHH ADDR OF USER LABEL TRACK
00004C             DC      6X'00'
000052   FILENS    DC      2X'00'                   SEEK ADDRESS-BB
000054             DC      X'0000FF00'              SEARCH ADDRESS-CCHH
000058             DC      X'00'                    RECORD NUMBER
000059             DC      AL3(IJKTXCF)             EOF ADDRESS
00005C             DC      4X'00'                   CCHH CONTROL FIELD
000060             DC      AL1(0)                   CONTROL FIELD
000061             DC      X'00'                    SWITCHES
000062             DC      H'0'                     SIZE OF BLOCK-1
000064             DC      5X'00'                   CCHHR BUCKET
000069             DC      AL3(IJKTXCW)             WLERR ADDRESS
00006C             L       2,88(1)                  LOAD USER'S IOREG
000070             DC      A(IOA1)                  DEBLOCKER-INITIAL POINTER
000074             DC      F'0'                     DEBLOCKER-RECORD SIZE
000078             DC      A(IOA1-1)                DEBLOCKER LIMIT
00007C             DC      AL1(128)                 LOGICAL INDICATORS
00007D             DC      AL3(IJKTXCR)             USER'S ERROR ROUTINE
000080   CCWAD     CCW     7,*-46,64,6              SEEK
000088             CCW     X'31',*-52,64,5          SEARCH ID EQUAL
000090             CCW     8,*-8,0,0                TIC
000098             CCW     6,IOA1,0,0               READ DATA
0000A0   IOA1      DC      0D'0'
                   EXTRN   IJKTXCF,IJKTXCW,IJKTXCR
                   END
```

```
000000   FIDIINVA  START   0                      DISK FILE
000000             DC      X'C2'                  OPEN MASK
000001             DC      AL3(TABAD)             TABLE ADDRESS
000004             DC      X'11'                  FLAG BYTE ONE
000005             DC      AL3(0)                 CHAIN ADDRESS
000008             DC      X'0000'                FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                   RECORD LENGTH
00000C             DC      X'80000000'
000010             DC      A(IOA1)                BUFFER ADDRESS
000014             DC      H'0'                   REMAINING DATA
000016             DC      H'0'                   DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000082000000'        CCB
00001E             DC      X'FFFF'                CCB-LOGICAL UNIT
000020             DC      A(CCWAD)               CCB-CCW ADDRESS
000024             DC      4X'00'                 XXB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJGVIEZZ)            LOGIC MODULE
00002C             DC      X'20'                  DTF TYPE
00002D             DC      AL1(66)                OPEN/CLOSE INDICATOR
00002E             DC      CL8'FIDIINVA'          FILENAME
000036             DC      8X'00'
00003E             DC      X'08'                  OPEN COMMUNICATIONS BYTE
00003F             DC      2X'00'
000041             DC      AL3(*)                 USER'S LABEL ADDRESS
000044             DC      A(IOA1)                ADDRESS OF IOAREA
000048             DC      X'80000000'            CCHH ADDR OF USER LABEL TRACK
00004C             DC      6X'00'
000052   FILENS    DC      2X'00'                 SEEK ADDRESS-BB
000054             DC      X'0000FF00'            SEARCH ADDRESS-CCHH
000058             DC      X'00'                  RECORD NUMBER
000059             DC      AL3(IJKTXCF)           EOF ADDRESS
00005C             DC      4X'00'                 CCHH CONTROL FIELD
000060             DC      X'FF'                  CONTROL FIELD
000061             DC      X'00'                  SWITCHES
000062             DC      H'0'                   SIZE OF BLOCK-1
000064             DC      5X'00'                 CCHHR BUCKET
000069             DC      AL3(IJKTXCW)           WLERR ADDRESS
00006C             L       2,88(1)                LOAD USER'S IOREG
000070             DC      A(IOA1+4)              DEBLOCKER-INITIAL POINTER
000074             DC      F'0'                   DEBLOCKER-RECORD SIZE
000078             DC      A(IOA1-1)              DEBLOCKER LIMIT
00007C             DC      AL1(128)               LOGICAL INDICATORS
00007D             DC      AL3(IJKTXCR)           USER'S ERROR ROUTINE
000080   CCWAD     CCW     7,*-46,64,6            SEEK
000088             CCW     X'31',*-52,64,5        SEARCH ID EQUAL
000090             CCW     8,*-8,0,0              TIC
000098             CCW     6,IOA1,64,0            READ DATA
0000A0             CCW     X'92',*+8,0,8          READ COUNT
0000A8             DC      2F'0'                  COUNT AREA
0000B0   IOA1      DC      0D'0'
                   EXTRN   IJKTXCF,IJKTXCW,IJKTXCR
                   END
```

```
000000   FIDIINUN  START   0                  DISK FILE
000000             dc      X'C2'              OPEN MASK
000001             DC      AL3(TABAD)         TABLE ADDRESS
000004             DC      X'31'              FLAG BYTE ONE
000005             DC      AL3(0)             CHAIN ADDRESS
000008             DC      X'0000'            FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'               RECORD LENGTH
00000C             DC      X'80000000'
000010             DC      A(IOA1)            BUFFER ADDRESS
000014             DC      H'0'               REMAINING DATA
000016             DC      H'0'               DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000082000000'    CCB
00001E             DC      X'FFFF'            CCB-LOGICAL UNIT
000020             DC      A(CCWAD)           CCB-CCW ADDRESS
000024             DC      4X'00'             XXB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJGUIEZZ)        LOGIC MODULE
00002C             DC      X'20'              DTF TYPE
00002D             DC      AL1(2)             OPEN/CLOSE INDICATOR
00002E             DC      CL8'FIDIINUN'      FILENAME
000036             DC      8X'00'
00003E             DC      X'08'              OPEN COMMUNICATIONS BYTE
00003F             DC      2X'00'
000041             DC      AL3(*)             USER'S LABEL ADDRESS
000044             DC      A(IOA1)            ADDRESS OF IOAREA
000048             DC      X'80000000'        CCHH ADDR OF USER LABEL TRACK
00004C             DC      6X'00'
000052   FILENS    DC      2X'00'             SEEK ADDRESS-BB
000054             DC      X'0000FF00'        SEARCH ADDRESS-CCHH
000058             DC      X'00'              RECORD NUMBER
000059             DC      AL3(IJKTXCF)       EOF ADDRESS
00005C             DC      4X'00'             CCHH CONTROL FIELD
000060             DC      X'FF'              CONTROL FIELD
000061             DC      X'00'              SWITCHES
000052             DC      H'0'               SIZE OF BLOCK-1
000064             DC      5X'00'             CCHHR BUCKET
000069             DC      3X'00'
00006C             L       2,88(1)            LOAD USER'S IOREG
000070             DC      A(IOA1)            DEBLOCKER-INITIAL POINTER
000074             L       4,96(1)
000078             DC      A(IOA1-1)          DEBLOCKER LIMIT
00007C             DC      AL1(128)           LOGICAL INDICATORS
00007D             DC      AL3(IJKTXCR)       USER'S ERROR ROUTINE
000080   CCWAD     CCW     7,*-46,64,6        SEEK
000088             CCW     X'31',*-52,64,5    SEARCH ID EQUAL
000090             CCW     8,*-8,0,0          TIC
000098             CCW     6,IOA1,96,0        READ DATA
0000A0             CCW     X'92',*+8,0,8      READ COUNT
0000A8             DC      2F'0'              COUNT AREA
0000B0   IOA1      DC      0D'0'
                   EXTRN   IJKTXCF
                   EXTRN   IJKTXCR
                   END
```

```
000000  FIDIOUFI  START   0                          DISK FILE
000000            DC      X'A2'                      OPEN MASK
000001            DC      AL3(TABAD)                 TABLE ADDRESS
000004            DC      X'01'                      FLAG BYTE ONE
000005            DC      AL3(0)                     CHAIN ADDRESS
000008            DC      X'0000'                    FLAG BYTE TWO, COMM. BYTE
00000A            DC      H'0'                       RECORD LENGTH
00000C            DC      X'80000000'
000010            DC      A(IOA1+8)                  BUFFER ADDRESS
000014            DC      H'0'                       REMAINING DATA
000016            DC      H'0'                       DATA LENGTH
000018  TABAD     DC      0D'0'
000018            DC      X'000082000000'            CCB
00001E            DC      X'FFFF'                    CCB-LOGICAL UNIT
000020            DC      A(CCWAD)                   CCB-CCW ADDRESS
000024            DC      4X'00'                     XXB-ST BYTE,CSW CCW ADDRESS
000028            DC      V(IJGFOEZZ)                LOGIC MODULE
00002C            DC      X'20'                      DTF TYPE
00002D            DC      AL1(0)                     OPEN/CLOSE INDICATOR
00002E            DC      CL8'FIDIOUFI'              FILENAME
000036            DC      8X'00'
00003E            DC      X'08'                      OPEN COMMUNICATIONS BYTE
00003F            DC      2X'00'
000041            DC      AL3(*)                     USER'S LABEL ADDRESS
000044            DC      A(IOA1)                    ADDRESS OF IOAREA
000048            DC      X'80000000'                CCHH ADDR OF USER LABEL TRACK
00004C            DC      6X'00'
000052  FILENS    DC      2X'00'                     SEEK ADDRESS-BB
000054            DC      X'0000FF00'                SEARCH ADDRESS-CCHH
000058            DC      X'00'                      RECORD NUMBER
000059            DC      X'00'                      KEY LENGTH
00005A            DC      H'0'                       DATA LENGTH
00005C            DC      4X'00'                     CCHH CONTROL FIELD
000060            DC      AL1(0)                     CONTROL FIELD
000061            DC      X'00'                      SWITCHES
000062            DC      H'0'                       SIZE OF BLOCK-1
000064            DC      5X'00'                     CCHHR BUCKET
000069            DC      X'00'
00006A            DC      H'3625'                    TRACK CAPACITY CONSTANT
00006C            L       2,88(1)                    LOAD USER'S IOREG
000070            DC      A(IOA1+8)                  DEBLOCKER-INITIAL POINTER
000074            DC      F'0'                       DEBLOCKER-RECORD SIZE
000078            DC      A(IOA1-1)                  DBLOCKER LIMIT
00007C            DC      AL1(128)                   LOGICAL INDICATORS
00007D            DC      AL3(IJKTXCR)               USER'S ERROR ROUTINE
000080  CCWAD     CCW     7,*-46,64,6                SEEK
000088            CCW     X'31',*-52,64,5            SEARCH ID EQUAL
000090            CCW     8,*-8,0,0                  TIC
000098            CCW     X'1D',IOA1,0,0             WRITE COUNT, KEY AND DATA
0000A0            CCW     X'31',FILENS+2,64,5        SEARCH ID EQUAL
0000A8            CCW     8,*-8,0,0                  TIC
000080            CCW     30,*,48,1
0000B8  IOA1      DC      0D'0'
                  EXTRN   IJKTXCR
                  EXTRN   IJKTXCF
                  END
```

```
000000  FIDIOUPR  START   0                        DISK FILE
000000            DC      X'A0'                    OPEN MASK
000001            DC      AL3(TABAD)               TABLE ADDRESS
000004            DC      X'0D'                    FLAG BYTE ONE
000005            DC      AL3(0)                   CHAIN ADDRESS
000008            DC      X'0000'                  FLAG BYTE TWO, COMM. BYTE
00000A            DC      H'0'                     RECORD LENGTH
00000C            DC      X'80000000'
000010            DC      A(IOA1+8)                BUFFER ADDRESS
000014            DC      H'0'                     REMAINING DATA
000016            DC      H'0'                     DATA LENGTH
000018            DC      H'0'                     PAGE SIZE
00001A            DC      H'0'                     CURRENT LINE
00001C
000020  TABAD     DC      0D'0'
000020            DC      X'000082000000'          CCB
000026            DC      X'FFFF'                  CCB-LOGICAL UNIT
000028            DC      A(CCWAD)                 CCB-CCW ADDRESS
00002C            DC      4X'00'                   XXB-ST BYTE,CSW CCW ADDRESS
000030            DC      V(IJGFOEZZ)              LOGIC MODULE
000034            DC      X'20'                    DTF TYPE
000035            DC      AL1(0)                   OPEN/CLOSE INDICATOR
000036            DC      CL8'FIDIOUPR'            FILENAME
00003E            DC      8X'00'
000046            DC      X'08'                    OPEN COMMUNICATIONS BYTE
000047            DC      2X'00'
000049            DC      AL3(*)                   USER'S LABEL ADDRESS
00004C            DC      A(IOA1)                  ADDRESS OF IOAREA
000050            DC      X'80000000'              CCHH ADDR OF USER LABEL TRACK
000054            DC      6X'00'
00005A  FILENS    DC      2X'00'                   SEEK ADDRESS-BB
00005C            DC      X'0000FF00'              SEARCH ADDRESS-CCHH
000060            DC      X'00'                    RECORD NUMBER
000061            DC      X'00'                    KEY LENGTH
000062            DC      H'0'                     DATA LENGTH
000064            DC      4X'00'                   CCHH CONTROL FIELD
000068            DC      AL1(0)                   CONTROL FIELD
000069            DC      X'00'                    SWITCHES
00006A            DC      H'0'                     SIZE OF BLOCK-1
00006C            DC      5X'00'                   CCHHR BUCKET
000071            DC      X'00'
000072            DC      H'3625'                  TRACK CAPACITY CONSTANT
000074            L       2,88(1)                  LOAD USER'S IOREG
000078            DC      A(IOA1+8)                DEBLOCKER-INITIAL POINTER
00007C            DC      F'0'                     DEBLOCKER-RECORD SIZE
000080            DC      A(IOA1-1)                DEBLOCKER LIMIT
000084            DC      AL1(128)                 LOGICAL INDICATORS
000085            DC      AL3(IJKTXCR)             USER'S ERROR ROUTINE
000088  CCWAD     CCW     7,*-46,64,6              SEEK
000090            CCW     X'31',*-52,64,5          SEARCH ID EQUAL
000098            CCW     8,*-8,0,0                TIC
0000A0            CCW     X'1D',IOA1,0,0           WRITE COUNT, KEY AND DATA
0000A8            CCW     X'31',FILENS+2,64,5      SEARCH ID EQUAL
0000B0            CCW     8,*-8,0,0                TIC
0000B8            CCW     30,*,48,1
0000C0  IOA1      DC      0D'0'
                  EXTRN   IJKTXCR
                  EXTRN   IJKTXCF
                  END
```

```
000000   FIDIOUVA  START   0                         DISK FILE
000000             DC      X'A2'                     OPEN MASK
000001             DC      AL3(TABAD)                TABLE ADDRESS
000004             DC      X'11'                     FLAG BYTE ONE
000005             DC      AL3(0)                    CHAIN ADDRESS
000008             DC      X'0000'                   FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                      RECORD LENGTH
00000C             DC      X'80000000'
000010             DC      A(IOA1+8)                 BUFFER ADDRESS
000014             DC      H'0'                      REMAINING DATA
000016             DC      H'0'                      DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000082000000'           CCB
00001E             DC      X'FFFF'                   CCB-LOGICAL UNIT
000020             DC      A(CCWAD)                  CCB-CCW ADDRESS
000024             DC      4X'00'                    XXB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJGVOEZZ)               LOGIC MODULE
00002C             DC      X'20'                     DTF TYPE
00002D             DC      AL1(64)                   OPEN/CLOSE INDICATOR
00002E             DC      CL8'FIDIOUVA'             FILENAME
000036             DC      8X'00'
00003E             DC      X'08'                     OPEN COMMUNICATIONS BYTE
00003F             DC      2X'00'
000041             DC      AL3(*)                    USER'S LABEL ADDRESS
000044             DC      A(IOA1)                   ADDRESS OF IOAREA
000048             DC      X'80000000'               CCHH ADDR OF USER LABEL TRACK
00004C             DC      6X'00'
000052   FILENS    DC      2X'00'                    SEEK ADDRESS-BB
000054             DC      X'0000FF00'               SEARCH ADDRESS-CCHH
000058             DC      X'00'                     RECORD NUMBER
000059             DC      X'00'                     KEY LENGTH
00005A             DC      H'0'                      DATA LENGTH
00005C             DC      4X'00'                    CCHH CONTROL FIELD
000060             DC      X'FF'                     CONTROL FIELD
000061             DC      X'00'                     SWITCHES
000062             DC      H'0'                      SIZE OF BLOCK-1
000064             DC      5X'00'                    CCHHR BUCKET
000069             DC      X'00'
00006A             DC      H'3625'                   TRACK CAPACITY CONSTANT
00006C             L       2,88(1)                   LOAD USER'S IOREG
000070             DC      A(IOA1+12)                DEBLOCKER-INITIAL POINTER
000074             DC      F'0'                      DEBLOCKER-RECORD SIZE
000078             DC      A(IOA1-1)                 DEBLOCKER LIMIT
00007C             DC      AL1(128)                  LOGICAL INDICATORS
00007D             DC      AL3(IJKTXCR)              USER'S ERROR ROUTINE
000080   CCWAD     CCW     7,*-46,64,6               SEEK
000088             CCW     X'31',*-52,64,5           SEARCH ID EQUAL
000090             CCW     8,*-8,0,0                 TIC
000098             CCW     X'1D',IOA1,0,0            WRITE COUNT, KEY AND DATA
0000A0             CCW     X'31',FILENS+2,64,5       SEARCH ID EQUAL
0000A8             CCW     8,*-8,0,0                 TIC
0000B0             CCW     30,*,48,1
0000B8             DC      F'0'                      SPACE REMAINING IN OUTPUT AREA
0000BC             DC      H'3625'                   TRACK CAPACITY BUCKET
0000BE             L       3,160(1)                  LOAD USER'S VARBLD REGISTER
0000C2
0000C8   IOA1      DC      0D'0'
                   EXTRN   IJKTXCR
                   EXTRN   IJKTXCF
                   END
```

```
000000   FIDIOUUN  START   0                      DISK FILE
000000             DC      X'A2'                  OPEN MASK
000001             DC      AL3(TABAD)             TABLE ADDRESS
000004             DC      X'31'                  FLAG BYTE ONE
000005             DC      AL3(0)                 CHAIN ADDRESS
000008             DC      X'0000'                FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                   RECORD LENGTH
00000C             DC      X'80000000'
000010             DC      A(IOA1+8)              BUFFER ADDRESS
000014             DC      H'0'                   REMAINING DATA
000016             DC      H'0'                   DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000082000000'        CCB
00001E             DC      X'FFFF'                CCB-LOGICAL UNIT
000020             DC      A(CCWAD)               CCB-CCW ADDRESS
000024             DC      4X'00'                 XXB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJGUOEZZ)            LOGIC MODULE
00002C             DC      X'20'                  DTF TYPE
00002D             DC      AL1(0)                 OPEN/CLOSE INDICATOR
00002E             DC      CL8'FIDIOUUN'          FILENAME
000036             DC      8X'00'
00003E             DC      X'08'                  OPEN COMMUNICATIONS BYTE
00003F             DC      2X'00'
000041             DC      AL3(*)                 USER'S LABEL ADDRESS
000044             DC      A(IOA1)                ADDRESS OF IOAREA
000048             DC      X'80000000'            CCHH ADDR OF USER LABEL TRACK
00004C             DC      6X'00'
000052   FILENS    DC      2X'00'                 SEEK ADDRESS-BB
000054             DC      X'0000FF00'            SEARCH ADDRESS-CCHH
000058             DC      X'00'                  RECORD NUMBER
000059             DC      X'00'                  KEY LENGTH
00005A             DC      H'0'                   DATA LENGTH
00005C             DC      4X'00'                 CCHH CONTROL FIELD
000060             DC      X'FF'                  CONTROL FIELD
000061             DC      X'00'                  SWITCHES
000062             DC      H'0'                   SIZE OF BLOCK-1
000064             DC      5X'00'                 CCHHR BUCKET
000069             DC      X'00'
00006A             DC      H'3625'                TRACK CAPACITY CONSTANT
00006C             L       2,88(1)                LOAD USER'S IOREG
000070             DC      A(IOA1+8)              DEBLOCKER-INITIAL POINTER
000074             STH     4,66(1)
000078             DC      A(IOA1-1)              DEBLOCKER LIMIT
00007C             DC      AL1(128)               LOGICAL INDICATORS
00007D             DC      AL3(IJKTXCR)           USER'S ERROR ROUTINE
000080   CCWAD     CCW     7,*-46,64,6            SEEK
000088             CCW     X'31',*-52,64,5        SEARCH ID EQUAL
000090             CCW     8,*-8,0,0              TIC
000098             CCW     X'1D',IOA1,0,0         WRITE COUNT KEY AND DATA
0000A0             CCW     X'31',FILENS+2,64,5    SEARCH ID EQUAL
0000A8             CCW     8,*-8,0,0              TIC
0000B0             CCW     30,*,48,1
0000B8             DC      H'3625'                TRACK CAPACITY BUCKET
0000BA
0000C0   IOA1      DC      0D'0'
                   EXTRN   IJKTXCR
                   EXTRN   IJKTXCF
                   END
```

```
000000  FIDIUPFI  START  0                        DISK FILE
000000            DC     X'9A'                    OPEN MASK
000001            DC     AL3(TABAD)               TABLE ADDRESS
000004            DC     X'01'                    FLAG BYTE ONE
000005            DC     AL3(0)                   CHAIN ADDRESS
000008            DC     X'0000'                  FLAG BYTE TWO, COMM. BYTE
00000A            DC     H'0'                     RECORD LENGTH
00000C            DC     X'80000000'
000010            DC     A(IOA1)                  BUFFER ADDRESS
000014            DC     H'0'                     REMAINING DATA
000016            DC     H'0'                     DATA LENGTH
000018  TABAD     DC     0D'0'
000018            DC     X'000082000000'          CCB
00001E            DC     X'FFFF'                  CCB-LOGICAL UNIT
000020            DC     A(CCWAD)                 CCB-CCW ADDRESS
000024            DC     4X'00'                   XXB-ST BYTE,CSW CCW ADDRESS
000028            DC     V(IJKFUEZZ)              LOGIC MODULE
00002C            DC     X'20'                    DTF TYPE
00002D            DC     AL1(2)                   OPEN/CLOSE INDICATOR
00002E            DC     CL8'FIDIUPFI'            FILENAME
000036            DC     8X'00'
00003E            DC     X'08'                    OPEN COMMUNICATIONS BYTE
00003F            DC     2X'00'
000041            DC     AL3(*)                   USER'S LABEL ADDRESS
000044            DC     A(IOA1)                  ADDRESS OF IOAREA
000048            DC     X'80000000'              CCHH ADDR OF USER LABEL TRACK
00004C            DC     6X'00'
000052  FILENS    DC     2X'00'                   SEEK ADDRESS-BB
000054            DC     X'0000FF00'              SEARCH ADDRESS-CCHH
000058            DC     X'00'                    RECORD NUMBER
000059            DC     AL3(IJKTXCF)             EOF ADDRESS
00005C            DC     4X'00'                   CCHH CONTROL FIELD
000060            DC     AL1(0)                   CONTROL FIELD
000061            DC     X'00'                    SWITCHES
000062            DC     H'0'                     SIZE OF BLOCK-1
000064            DC     5X'00'                   CCHHR BUCKET
000069            DC     AL3(IJKTXCW)             WLERR ADDRESS
00006C            L      2,88(1)                  LOAD USER'S IOREG
000070            DC     A(IOA1)                  DEBLOCKER-INITIAL POINTER
000074            DC     F'0'                     DEBLOCKER-RECORD SIZE
000078            DC     A(IOA1-1)                DEBLOCKER LIMIT
00007C            DC     AL1(128)                 LOGICAL INDICATORS
00007D            DC     AL3(IJKTXCR)             USER'S ERROR ROUTINE
000080  CCWAD     CCW    7,*-46,64,6              SEEK
000088            CCW    X'31',*-52,64,5          SEARCH ID EQUAL
000090            CCW    8,*-8,0,0                TIC
000098            CCW    6,IOA1,0,0               READ DATA
0000A0            CCW    X'31',FILENS+2,64,5      SEARCH ID EQUAL
0000A8            CCW    8,*-8,0,0                TIC
0000B0            CCW    6,*,48,1                 VERIFY
0000B8  IOA1      DC     0D'0'
                  EXTRN  IJKTXCF,IJKTXCW,IJKTXCR
                  END
```

```
000000   FIDIUPVA  START   0                          DISK FILE
000000             DC      X'9A'                      OPEN MASK
000001             DC      AL3(TABAD)                 TABLE ADDRESS
000004             DC      X'11'                      FLAG BYTE ONE
000005             DC      AL3(0)                     CHAIN ADDRESS
000008             DC      X'0000'                    FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                       RECORD LENGTH
00000C             DC      X'80000000'
000010             DC      A(IOA1)                    BUFFER ADDRESS
000014             DC      H'0'                       REMAINING DATA
000016             DC      H'0'                       DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000082000000'            CCB
00001E             DC      X'FFFF'                    CCB-LOGICAL UNIT
000020             DC      A(CCWAD)                   CCB-CCW ADDRESS
000024             DC      4X'00'                     XXB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJGVUEZZ)                LOGIC MODULE
00002C             DC      X'20'                      DTF TYPE
00002D             DC      AL1(66)                    OPEN/CLOSE INDICATOR
00002E             DC      CL8'FIDIUPVA'              FILENAME
000036             DC      8X'00'
00003E             DC      X'08'                      OPEN COMMUNICATIONS BYTE
00003F             DC      2X'00'
000041             DC      AL3(*)                     USER'S LABEL ADDRESS
000044             DC      A(IOA1)                    ADDRESS OF IOAREA
000048             DC      X'80000000'                CCHH ADDR OF USER LABEL TRACK
00004C             DC      6X'00'
000052   FILENS    DC      2X'00'                     SEEK ADDRESS-BB
000054             DC      X'0000FF00'                SEARCH ADDRESS-CCHH
000058             DC      X'00'                      RECORD NUMBER
000059             DC      AL3(IJKTXCF)               EOF ADDRESS
00005C             DC      4X'00'                     CCHH CONTROL FIELD
000060             DC      X'FF'                      CONTROL FIELD
000061             DC      X'00'                      SWITCHES
000062             DC      H'0'                       SIZE OF BLOCK-1
000064             DC      5X'00'                     CCHHR BUCKET
000069             DC      AL3(IJKTXCW)               WLERR ADDRESS
00006C             L       2,88(1)                    LOAD USER'S IOREG
000070             DC      A(IOA1+4)                  DEBLOCKER-INITIAL POINTER
000074             DC      F'0'                       DEBLOCKER-RECORD SIZE
000078             DC      A(IOA1-1)                  DEBLOCKER LIMIT
00007C             DC      AL1(128)                   LOGICAL INDICATORS
00007D             DC      AL3(IJKTXCR)               USER'S ERROR ROUTINE
000080   CCWAD     CCW     7,*-46,64,6                SEEK
000088             CCW     X'31',*-52,64,5            SEARCH ID EQUAL
000090             CCW     8,*-8,0,0                  TIC
000098             CCW     6,IOA1,64,0                READ DATA
0000A0             CCW     X'92',*+32,0,8             READ COUNT
0000A8             CCW     X'31',FILENS+2,64,5        SEARCH ID EQUAL
0000B0             CCW     8,*-8,0,0                  TIC
0000B8             CCW     6,*,48,146                 VERIFY
0000C0             DC      2F'0'                      COUNT AREA
0000C8             DC      2F'0'                      COUNT SAVE AREA
0000D0             DC      2F'0'                      COUNT SAVE AREA FOR 2I/O
                   EXTRN   IJKTXCF,IJKTXCW,IJKTXCR
0000D8   IOA1      DC      0D'0'
                   END
```

```
000000   FIDIUPUN  START   0                          DISK FILE
000000             DC      X'9A'                      OPEN MASK
000001             DC      AL3(TABAD)                 TABLE ADDRESS
000004             DC      X'31'                      FLAG BYTE ONE
000005             DC      AL3(0)                     CHAIN ADDRESS
000008             DC      X'0000'                    FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                       RECORD LENGTH
00000C             DC      X'80000000'
000010             DC      A(IOA1)                    BUFFER ADDRESS
000014             DC      H'0'                       REMAINIG DATA
000016             DC      H'0'                       DATA LENGTH
000018   TABAD     DC      0D'0'
000018             DC      X'000082000000'            CCB
00001E             DC      X'FFFF'                    CCB-LOGICAL UNIT
000020             DC      A(CCWAD)                   CCB-CCW ADDRESS
000024             DC      4X'00'                     XXB-ST BYTE,CSW CCW ADDRESS
000028             DC      V(IJGUUEZZ)                LOGIC MODULE
00002C             DC      X'20'                      DTF TYPE
00002D             DC      AL1(2)                     OPEN/CLOSE INDICATOR
00002E             DC      CL8'FIDIUPUN'              FILENAME
000036             DC      8X'00'
00003E             DC      X'08'                      OPEN COMMUNICATIONS BYTE
00003F             DC      2X'00'
000041             DC      AL3(*)                     USER'S LABEL ADDRESS
000044             DC      A(IOA1)                    ADDRESS OF IOAREA
000048             DC      X'80000000'                CCHH ADDR OF USER LABEL TRACK
00004C             DC      6X'00'
000052   FILENS    DC      2X'00'                     SEEK ADDRESS-BB
000054             DC      X'0000FF00'                SEARCH ADDRESS-CCHH
000058             DC      X'00'                      RECORD NUMBER
000059             DC      AL3(IJKTXCF)               EOF ADDRESS
00005C             DC      4X'00'                     CCHH CONTROL FIELD
000060             DC      X'FF'                      CONTROL FIELD
000061             DC      X'00'                      SWITCHES
000062             DC      H'0'                       SIZE OF BLOCK-1
000064             DC      5X'00'                     CCHHR BUCKET
000069             DC      3X'00'
00006C             L       2,88(1)                    LOAD USER'S IOREG
000070             DC      A(IOA1)                    DEBLOCKER-INITIAL POINTER
000074             L       4,96(1)
000078             DC      A(IOA1-1)                  DEBLOCKER LIMIT
00007C             DC      AL1(128)                   LOGICAL INDICATORS
00007D             DC      AL3(IJKTXCR)               USER'S ERROR ROUTINE
000080   CCWAD     CCW     7,*-46,64,6                SEEK
000088             CCW     X'31',*-52,64,5            SEARCH ID EQUAL
000090             CCW     8,*-8,0,0                  TIC
000098             CCW     6,IOA1,96,0                READ DATA
0000A0             CCW     X'92',*+32,0,8             READ COUNT
0000A8             CCW     X'31',FILENS+2,64,5        SEARCH ID EQUAL
0000B0             CCW     8,*-8,0,0                  TIC
0000B8             CCW     6,*,48,146                 VERIFY
000000             DC      2F'0'                      COUNT AREA
0000C8             DC      2F'0'                      COUNT SAVE AREA
0000D0             DC      2F'0'                      COUNT SAVE AREA FOR 2I/O
                   EXTRN   IJKTXCF
                   EXTRN   IJKTXCR
0000D8   IOA1      DC      0D'0'
                   END
```

```
000000  FIDIINR1  START  0
000000            DC     X'C5'                      OPEN MASK
000001            DC     AL3(TABAD)                 TABLE ADDRESS
000004            DC     X'01'                      FLAG BYTE ONE
000005            DC     AL3(0)                     CHAIN ADDRESS
000008            DC     X'2800'                    FLAG BYTE TWO, COMM. BYTE
00000A            DC     H'0'                       RECORD LENGTH
00000C            DC     X'80'                      DEVICE CODE
00000D            DC     AL3(0)
000010            DC     A(IOA1)                    BUFFER ADDRESS
000014            DC     X'00'                      REGIONAL TYPE
000015            DC     AL3(IJKTXRP)               ADDRESSING ROUTINE
000018            DC     A(0)
00001C            DC     A(0)
000020            DC     X'0000'                    LOGICAL UNIT
000022  ERRBYTE   DC     X'0000'                    ERROR BYTE
000024            DC     H'0'                       KEYLENGTH
000026            DC     X'00'
000027  SEEKADR   DC     15X'00'
000036            DC     H'10'
000038            DC     6X'0'
00003E            DC     H'10'
000040            DC     6X'0'
000046            DC     H'10'
000048  TABAD     DC     0D'0'
000048            DC     H'0'                       FIRST CCB BYTES
00004A            DC     X'88'
00004B            DC     5X'0'
000050            DC     AL4(CCAD)                  CC ADDR IN CCB
000054            DC     F'0'
000058            DC     V(IJIFZZZZ)                FILE TYPE
00005C            DC     X'22'
00005D            DC     B'00000000'
00005E            DC     CL8'FIDIINR1'
000066            DC     X'0104'
000068            DC     F'0'                       LABEL ROUTINE ADDRESS
00006C            DC     V(IJKTXRM)                 EXTENT ROUTINE ADDRESS
000070            DC     X'0'
000071            DC     AL3(ERRBYTE)
000074            DC     H'0'                       TEST SWITCH
000076            DC     Y(CCWAD-TABAD-32)          POINTER
000078            DC     H'0'                       IJICB2
00007A            DC     X'88'
00007B            DC     5X'0'
000080            DC     AL4(FILENZ)
000084            DC     4X'0'
000088  FILENZ    CCW    X'07',SEEKADR+1,X'00',6
000090            XI     36(2),C'0'
000094            DC     H'0'                       MAXIMUM DATA LENGTH
000096            DC     YL1(FILEN0-TABAD-1)        PTR TO READ ID STRING
000097            DC     YL1(FILEN1-TABAD-1)        READ KEY
000098            DC     YL1(FILEN2-TABAD-1)        WRITE ID
000099            DC     YL1(FILEN3-TABAD-1)        WRITE KEY
00009A            DC     YL1(FILEN4-TABAD-1)        RZERO
00009B            DC     YL1(FILEN5-TABAD-1)        AFTER
00009C            DC     H'00'
00009E            DC     H'61'                      RIC CONSTANT
0000A0            DC     D'0'
0000A8  FILENC    CCW    X'31',SEEKADR+3,X'40',5
0000B0            DC     1F'0'
0000B4            DC     H'0'
0000B6            DC     H'0'
0000B8            CCW    X'06',IOA1,X'40',0
0000C0            DC     1D'0'
0000C8            CCW    X'39',SEEKADR+3,X'40',4
0000D0            CCW    X'0E',IOA1,X'40',0
0000D8  FILEN0    EQU    *
0000D8            DC     X'871814'
```

```
000049   FILEN1     EQU      TABAD+1
000049   FILEN2     EQU      TABAD+1
000049   FILEN3     EQU      TABAD+1
000049   FILEN4     EQU      TABAD+1
000049   FILEN5     EQU      TABAD+1
0000DB
0000E0   CCWAD      CCW      X'07',SEEKADR+1,X'40',6
0000E8              DC       7D'0'
000120   IOA1       DC       0D'0'
                    EXTRN    IJKTXRP
                    END
```

```
000000   FIDIONR1  START   0
000000             DC      X'A5'                       OPEN MASK
000001             DC      AL3(TABAD)                  TABLE ADDRESS
000004             DC      X'01'                       FLAG BYTE ONE
000005             DC      AL3(0)                      CHAIN ADDRESS
000008             DC      X'2800'                     FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                        RECORD LENGTH
00000C             DC      X'80'                       DEVICE CODE
00000D             DC      AL3(0)
000010             DC      A(IOA1)                     BUFFER ADDRESS
000014             DC      X'00'                       REGIONAL TYPE
000015             DC      AL3(IJKTXRP)                ADDRESSING ROUTINE
000018             DC      A(0)
00001C             DC      A(0)
000020             DC      X'0000'                     LOGICAL UNIT
000022   ERRBYTE   DC      X'0000'                     ERROR BYTE
000024             DC      H'0'                        KEYLENGTH
000026             DC      X'00'
000027   SEEKADR   DC      15X'00'
000036             DC      H'10'
000038             DC      6X'0'
00003E             DC      H'10'
000040             DC      6X'0'
000046             DC      H'10'
000048   TABAD     DC      0D'0'
000048             DC      H'0'                        FIRST CCB BYTES
00004A             DC      X'88'
00004B             DC      5X'0'
000050             DC      AL4(CCWAD)                  CCW ADDR IN CCB
000054             DC      F'0'
000058             DC      V(IJIFZZZZ)                 FILE TYPE
00005C             DC      X'22'
00005D             DC      B'10000000'
00005E             DC      CL8'FIDIONR1'
000066             DC      X'0104'
000068             DC      F'0'                        LABEL ROUTINE ADDRESS
00006C             DC      V(IJKTXRM)                  EXTENT ROUTINE ADDRESS
000070             DC      X'0'
000071             DC      AL3(ERRBYTE)
000074             DC      H'0'                        TEST SWITCH
000076             DC      Y(CCWAD-TABAD-32)           POINTER
000078             DC      H'0'                        IJICB2
00007A             DC      X'88'
00007B             DC      5X'0'
00008C             DC      AL4(FILENZ)
000084             DC      4X'0'
000088   FILENZ    CCW     X'07',SEEKADR+1,X'00',6
000090             XI      36(2),C'0'
000094             DC      H'0'                        MAXIMUM DATA LENGTH
000096             DC      YL1(FILEN0-TABAD-1)         PTR TO READ ID STRING
000097             DC      YL1(FILEN1-TABAD-1)         READ KEY
000098             DC      YL1(FILEN2-TABAD-1)         WRITE ID
000099             DC      YL1(FILEN3-TABAD-1)         WRITE KEY
00009A             DC      YL1(FILEN4-TABAD-1)         RZERO
00009B             DC      YL1(FILEN5-TABAD-1)         AFTER
00009C             DC      H'00'
00009E             DC      H'61'                       RIC CONSTANT
0000A0             DC      D'0'
0000A8   FILENC    CCW     X'31',SEEKADR+3,X'40',5
0000B0             DC      1F'0'
0000B4             DC      H'0'
0000B6             DC      H'0'
0000B8             CCW     X'06',IOA1,X'40',0
0000C0             DC      1D'0'
0000C8             CCW     X'39',SEKKADR+3,X'40',4
0000D0             CCW     X'0E',IOA1,X'40',0
000049   FILEN0    EQU     TABAD+1
000049   FILEN1    EQU     TABAD+1
```

```
 0000D8   FILEN2    EQU     *
 0000D8             DC      X'871895'
 000049   FILEN3    EQU     TABAD+1
 000049   FILEN4    EQU     TABAD+1
 000049   FILEN5    EQU     TABAD+1
 0000DB
 0000E0   CCWAD     CCW     X'07',SEEKADR+1,X'40',6
 0000E8             DC      7D'0'
 000120   IOA1      DC      0D'0'
                    EXTRN   IJKTXRP
                    END
```

```
000000   FIDIOVR1  START   0
000000             DC      X'A5'                         OPEN MASK
000001             DC      AL3(TABAD)                    TABLE ADDRESS
000004             DC      X'01'                         FLAG BYTE ONE
000005             DC      AL3(0)                        CHAIN ADDRESS
000008             DC      X'2800'                       FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                          RECORD LENGTH
00000C             DC      X'80'                         DEVICE CODE
00000D             DC      AL3(0)
000010             DC      A(IOA1)                       BUFFER ADDRESS
000014             DC      X'00'                         REGIONAL TYPE
000015             DC      AL3(IJKTXRP)                  ADDRESSING ROUTINE
000018             DC      A(0)
00001C             DC      A(0)
000020             DC      X'0000'                       LOGICAL UNIT
000022   ERRBYTE   DC      X'0000'                       ERROR BYTE
000024             DC      H'0'                          KEYLENGTH
000026             DC      X'00'
000027   SEEKADR   DC      15X'00'
000036             DC      H'10'
000038             DC      6X'0'
00003E             DC      H'10'
000040             DC      6X'0'
000046             DC      H'10'
000048   TABAD     DC      0D'0'
000048             DC      H'0'                          FIRST CCB BYTES
00004A             DC      X'88'
00004B             DC      5X'0'
000050             DC      A14 (CCWAD)                   CCW ADDR IN CCB
000054             DC      F'0'
000058             DC      V(IJIFZZZZ)                   FILE TYPE
00005C             DC      X'22'
00005D             DC      B'11000000'
00005E             DC      C18'FIDIOVR1'
000066             DC      X'0104'
000065             DC      F'0'                          LABEL ROUTINE ADDRESS
00006C             DC      V(IJKTXRM)                    EXTENT ROUTINE ADDRESS
000070             DC      X'0'
000071             DC      A13(ERRBYTE)
000074             DC      H'0'                          TEST SWITCH
000076             DC      Y(CCWAD-TABAD-32)             POINTER
000078             DC      H'0'                          IJICB2
00007A             DC      X'88'
00007B             DC      5X'0'
000080             DC      AL4(FILENZ)
000084             DC      4X'0'
000088   FILENZ    CCW     X'07',SEEKADR+1,X'00',6
000090             XI      36(2),C'0'
000094             DC      H'0'                          MAXIMUM DATA LENGTH
000096             DC      YL1(FILEN0-TABAD-1)           PTR TO READ ID STRING
000097             DC      YL1(FILEN1-TABAD-1)           READ KEY
000098             DC      YL1(FILEN2-TABAD-1)           WRITE ID
000099             DC      YL1(FILEN3-TABAD-1)           WRITE KEY
00009A             DC      YL1(FILEN4-TABAD-1)           RZERO
00009B             DC      YL1(FILEN5-TABAD-1)           AFTER
00009C             DC      H'00'
00009E             DC      H'61'                         RIC CONSTANT
0000A0             DC      D'0'
0000A8   FILENC    CCW     X'31',SEEKADR+3,X'40',5
0000B0             DC      1F'0'
0000B4             DC      H'0'
0000B6             DC      H'0'
0000B8             CCW     X'06',IOA1,X'40',0
0000C0             DC      1D'0'
0000C8             CCW     X'39',SEEKADR+3,X'40',4
0000D0             CCW     X'0E',IOA1,X'40',0
000049   FILEN0    EQU     TABAD+1
000049   FILEN1    EQU     TABAD+1
```

```
0000D8   FILEN2   EQU     *
0000D8            DC      X'871891871815'
000049   FILEN3   EQU     TABAD+1
000049   FILEN4   EQU     TABAD+1
000049   FILEN5   EQU     TABAD+1
0000DE
0000E0   CCWAD    CCW     X'07',SEEKADR+1,X'40',6
0000E8            DC      7D'0'
000120            DC      5D'0'
000148   IOA1     DC      0D'0'
                  EXTRN   IJKTXRP
                  END
```

```
000000   FIDIUNR1  START  0
000000             DC     X'9D'                      OPEN MASK
000001             DC     AL3(TABAD)                 TABLE ADDRESS
000004             DC     X'01'                      FLAG BYTE ONE
000005             DC     AL3(0)                     CHAIN ADDRESS
000008             DC     X'2800'                    FLAG BYTE TWO, COMM. BYTE
00000A             DC     H'0'                       RECORD LENGTH
00000C             DC     X'80'                      DEVICE CODE
00000D             DC     AL3(0)
000010             DC     A(IOA1)                    BUFFER ADDRESS
000014             DC     X'00'                      REGIONAL TYPE
000015             DC     AL3(IJKTXRP)               ADDRESSING ROUTINE
000018             DC     A(0)
00001C             DC     A(0)
000020             DC     X'0000'                    LOGICAL UNIT
000022   ERRBYTE   DC     X'0000'                    ERROR BYTE
000024             DC     H'0'                       KEYLENGTH
000026             DC     X'00'
000027   SEEKADR   DC     15X'00'
000036             DC     H'10'
000038             DC     6X'0'
00003E             DC     H'10'
000040             DC     6X'0'
000046             DC     H'10'
000048   TABAD     DC     0D'0'
000048             DC     H'0'                       FIRST CCB BYTES
00004A             DC     X'88'
00004B             DC     5X'0'
000050             DC     AL4(CCWAD)                 CCW ADDR IN CCB
000054             DC     F'0'
000058             DC     V(IJIFZZZZ)                FILE TYPE
00005C             DC     X'22'
00005D             DC     B'00000000'
00005E             DC     CL8'FIDIUNR1'
000066             DC     X'0104'
000068             DC     F'0'                       LABEL ROUTINE ADDRESS
00006C             DC     V(IJKTXRM)                 EXTENT ROUTINE ADDRESS
000070             DC     X'0'
000071             DC     AL3(ERRBYTE)
000074             DC     H'0'                       TEST SWITCH
000076             DC     Y(CCWAD-TABAD-32)          POINTER
000078             DC     H'0'                       IJICB2
00007A             DC     X'88'
00007B             DC     5X'0'
000080             DC     AL4(FILENZ)
000084             DC     4X'0'
000088   FILENZ    CCW    X'07',SEEKADR+1,X'00',6
000090             XI     36(2),C'0'
000094             DC     H'0'                       MAXIMUM DATA LENGTH
000096             DC     YL1(FILEN0-TABAD-1)        PTR TO READ ID STRING
000097             DC     YL1(FILEN1-TABAD-1)        READ KEY
000098             DC     YL1(FILEN2-TABAD-1)        WRITE ID
000099             DC     YL1(FILEN3-TABAD-1)        WRITE KEY
00009A             DC     YL1(FILEN4-TABAD-1)        RZERO
00009B             DC     YL1(FILEN5-TABAD-1)        AFTER
00009C             DC     H'00'
00009E             DC     H'61'                      RIC CONSTANT
0000A0             DC     D'0'
0000A8   FILENC    CCW    X'31',SEEKADR+3,X'40',5
0000B0             DC     1F'0'
0000B4             DC     H'0'
0000B6             DC     H'0'
0000B8             CCW    X'06',IOA1,X'40',0
0000C0             DC     1D'0'
0000C8             CCW    X'39',SEEKADR+3,X'40',4
0000D0             CCW    X'0E',IOA1,X'40',0
0000D8   FILEN0    EQU    *
0000D8             DC     X'871814'
```

```
000049   FILEN1    EQU     TABAD+1
0000DB   FILEN2    EQU     *
0000DB             DC      X'871895'
000049   FILEN3    EQU     TABAD+1
000049   FILEN4    EQU     TABAD+1
000049   FILEN5    EQU     TABAD+1
0000DE
0000E0   CCWAD     CCW     X'07',SEEKADR+1,X'40',6
0000E8             DC      7D'0'
000120   IOA1      DC      0D'0'
                   EXTRN   IJKTXRP
                   END
```

```
000000   FIDIUVR1  START   0
000000             DC      X'9D'                         OPEN MASK
000001             DC      AL3(TABAD)                    TABLE ADDRESS
000004             DC      X'01'                         FLAG BYTE ONE
000005             DC      AL3(0)                        CHAIN ADDRESS
000008             DC      X'2800'                       FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                          RECORD LENGTH
00000C             DC      X'80'                         DEVICE CODE
00000D             DC      AL3(0)
000010             DC      A(IOA1)                       BUFFER ADDRESS
000014             DC      X'00'                         REGIONAL TYPE
000015             DC      AL3(IJKTXRP)                  ADDRESSING ROUTINE
000018             DC      A(0)
00001C             DC      A(0)
000020             DC      X'0000'                       LOGICAL UNIT
000022   ERRBYTE   DC      X'0000'                       ERROR BYTE
000024             DC      H'0'                          KEYLENGTH
000026             DC      X'00'
000027   SEEKADR   DC      15X'00'
000036             DC      H'10'
000038             DC      6X'0'
00003E             DC      H'10'
000040             DC      6X'0'
000046             DC      H'10'
000048   TABAD     DC      0D'0'
000048             DC      H'0'                          FIRST CCB BYTES
00004A             DC      X'88'
00004B             DC      5X'0'
000050             DC      AL4(CCWAD)                    CCW ADDR IN CCB
000054             DC      F'0'
000058             DC      V(IJIFZZZZ)                   FILE TYPE
00005C             DC      X'22'
00005D             DC      B'01000000'
00005E             DC      CL8'FIDIUVR1'
000066             DC      X'0104'
000068             DC      F'0'                          LABEL ROUTINE ADDRESS
00006C             DC      V(IJKTXRM)                    EXTENT ROUTINE ADDRESS
000070             DC      X'0'
000071             DC      AL3(ERRBYTE)
000074             DC      H'0'                          TEST SWITCH
000076             DC      Y(CCWAD-TABAD-32              POINTER
000078             DC      H'0'                          IJICB2
00007A             DC      X'88'
00007B             DC      5X'0'
000080             DC      AL4(FILENZ)
000084             DC      4X'0'
000088   FILENZ    CCW     X'07',SEEKADR+1,X'00',6
000090             XI      36(2),C'0'
000094             DC      H'0'                          MAXIMUM DATA LENGTH
000096             DC      YL1(FILEN0-TABAD-1)           PTR TO READ ID STRING
000097             DC      YL1(FILEN1-TABAD-1)           READ KEY
000098             DC      YL1(FILEN2-TABAD-1)           WRITE ID
000099             DC      YL1(FILEN3-TABAD-2)           WRITE KEY
00009A             DC      YL1(FILEN4-TABAD-1)           RZERO
00009B             DC      YL1(FILEN5-TABAD-1)           AFTER
00009C             DC      H'00'
00009E             DC      H'61'                         RIC CONSTANT
0000A0             DC      D'0'
0000A8   FILENC    CCW     X'31',SEEKADR+3,'40',5
0000B0             DC      1F'0'
0000B4             DC      H'0'
0000B6             DC      H'0'
0000B8             CCW     X'06',IOA1,X'40',0'
0000C0             DC      1D'0'
0000C8             CCW     X'39',SEEKADR+3,X'40',4
0000D0             CCW     X'0E',IOA1,X'40',0
0000D8   FILEN0    EQU     *
0000D8             DC      X'871814'
```

```
000049   FILEN1    EQU     TABAD+1
0000DB   FILEN2    EQU     *
0000DB             DC      X'871891871815'
000049   FILEN3    EQU     TABAD+1
000049   FILEN4    EQU     TABAD+1
000049   FILEN5    EQU     TABAD+1
0000E1
0000E8   CCWAD     CCW     X'07',SEEKADR+1,'40',6
0000F0             DC      7D'0'
000128             dc      5d'0'
000150   IOA1      DC      0D'0'
                   EXTRN   IJKTXRP
                   END
```

```
000000   FINDIINR3 START   0
000000             DC      X'C5'                        OPEN MASK
000001             DC      AL3(TABAD)                   TABLE ADDRESS
000004             DC      X'01'                        FLAG BYTE ONE
000005             DC      AL3(0)                       CHAIN ADDRESS
000008             DC      X'2800'                      FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                         RECORD LENGTH
00000C             DC      X'80'                        DEVICE CODE
00000D             DC      AL3(0)
000010             DC      A(IOA1)                      BUFFER ADDRESS
000014             DC      X'08'                        REGIONAL TYPE
000015             DC      AL3(IJKTXRP)                 ADDRESSING ROUTINE
000018             DC      A(KEYARG)
00001C             DC      A(0)
000020             DC      X'0000'                      LOGICAL UNIT
000022   ERRBYTE   DC      X'0000'                      ERROR BYTE
000024             DC      H'0'                         KEYLENGTH
000026             DC      X'00'
000027   SEEKADR   DC      15X'00'
000036             DC      H'10'
000038             DC      6X'0'
00003E             DC      H'10'
000040             DC      6X'0'
000046             DC      H'10'
000048   TABAD     DC      0D'0'
000048             DC      H'0'                         FIRST CCB BYTES
00004A             DC      X'88'
00004B             DC      5X'0'
000050             DC      AL4(CCWAD)                   CCW ADDR IN CCB
000054             DC      F'0'
000058             DC      V(IJIFZZZZ)                  FILE TYPE
00005C             DC      X'22'
00005D             DC      B'00000000'
00005E             DC      CL8'FIDIINR3'
000066             DC      X'0104'
000068             DC      F'0'                         LABEL ROUTINE ADDRESS
00006C             DC      V(IJKTXRM)                   EXTENT ROUTINE ADDRESS
000070             DC      X'0'
000071             DC      AL3(ERRBYTE)
000074             DC      H'0'                         TEST SWITCH
000076             DC      Y(CCWAD-TABAD-32)            POINTER
000078             DC      H'0'                         IJICB2
00007A             DC      X'88'
00007B             DC      5X'0'
000080             DC      AL4(FILENZ)
000084             DC      4X'0'
000088   FILENZ    CCW     X'07',SEEKADR+1,X'00',6
000090             XI      36(2),C'0'
000094             DC      H'0'                         MAXIMUM DATA LENGTH
000096             DC      YL1(FILEN0-TABAD-1)          PTR TO READ ID STRING
000097             DC      YL1(FILEN1-TABAD-1)          READ KEY
000098             DC      YL1(FILEN2-TABAD-1)          WRITE ID
000099             DC      YL1(FILEN3-TABAD-1)          WRITE KEY
00009A             DC      YL1(FILEN4-TABAD-1)          RZERO
00009B             DC      YL1(FILEN5-TABAD-1)          AFTER
00009C             DC      H'20'
00009E             DC      H'61'                        RIC CONSTANT
0000A0             DC      D'0'
0000A8   FILENC    CCW     X'31',SEEKADR+3,X'40',5
0000B0             CCW     X'29',KEYARG,X'40',0
0000B8             CCW     X'06',IOA1,X'40',0
0000C0             DC      1D'0'
0000C8             CCW     X'39',SEEKADR+3,X'40',4
0000D0             CCW     X'0E',IOA1,X'40',0
000049   FILEN0    EQU     TABAD+1
0000D8   FILEN1    EQU
0000D8             DC      X'8F1814'
000049   FILEN2    EQU     TABAD+1
```

```
   000049    FILEN3     EQU      TABAD+1
   000049    FILEN4     EQU      TABAD+1
   000049    FILEN5     EQU      TABAD+1
   0000DB
   0000E0    CCWAD      CCW      X'07',SEEKADR+1,X'40',6
   0000E8               DC       7D'0'
   000120    IOA1       DC       0D'0'
   000120    KEYARG     DC       0D'0'
                        EXTRN    IJKTXRP
                        END
```

```
000000  FIDIONR3  START   0
000000            DC      X'A5'                          OPEN MASK
000001            DC      AL3(TABAD)                     TABLE ADDRESS
000004            DC      X'01'                          FLAG BYTE ONE
000005            DC      AL3(0)                         CHAIN ADDRESS
000008            DC      X'2800'                        FLAG BYTE TWO, COMM. BYTE
00000A            DC      H'0'                           RECORD LENGTH
00000C            DC      X'80'                          DEVICE CODE
00000D            DC      AL3(0)
000010            DC      A(IOA1)                        BUFFER ADDRESS
000014            DC      X'08'                          REGIONAL TYPE
000015            DC      AL3(IJKTXRP)                   ADDRESSING ROUTINE
000018            DC      A(KEYARG+8)
00001C            DC      A(0)
000020            DC      X'0000'                        LOGICAL UNIT
000022  ERRBYTE   DC      X'0000'                        ERROR BYTE
000024            DC      H'0'                           KEYLENGTH
000026            DC      X'00'
000027  SEEKADR   DC      15X'00'
000036            DC      H'10'
000038            DC      6X'0'
00003E            DC      H'10'
000040            DC      6X'0'
000046            DC      H'10'
000048  TABAD     DC      0D'0'
000048            DC      H'0'                           FIRST CCB BYTES
00004A            DC      X'88'
00004B            DC      5X'0'
000050            DC      AL4(CCWAD)                     CCW ADDR IN CCB
000054            DC      F'0'
000058            DC      V(IJIFAZZZ)                    FILE TYPE
00005C            DC      X'22'
00005D            DC      B'10010000'
00005E            DC      CL8'FIDIONR3'
000066            DC      X'0104'
000068            DC      F'0'                           LABEL ROUTINE ADDRESS
00006C            DC      V(IJKTXRM)                     EXTENT ROUTINE ADDRESS
000070            DC      X'0'
000071            DC      AL3(ERRBYTE)
000074            DC      H'0'                           TEST SWITCH
000076            DC      Y(CCWAD-TABAD-32)              POINTER
000078            DC      H'0'                           IJICB2
00007A            DC      X'88'
00007B            DC      5X'0'
000080            DC      AL4(FILENZ)
000084            DC      4X'0'
000088  FILENZ    CCW     X'07',SEEKADR+1,X'00',6
000090            XI      36(2),C'0'
000094            DC      H'0'                           MAXIMUM DATA LENGTH
000096            DC      YL1(FILEN0-TABAD-1)            PTR TO READ ID STRING
000097            DC      YL1(FILEN1-TABAD-1)            READ KEY
000098            DC      YL1(FILEN2-TABAD-1)            WRITE ID
000099            DC      YL1(FILEN3-TABAD-1)            WRITE KEY
00009A            DC      YL1(FILEN4-TABAD-1)            RZERO
00009B            DC      YL1(FILEN5-TABAD-1)            AFTER
00009C            DC      H'20'
00009E            DC      H'61'                          RIC CONSTANT
0000A0            DC      D'0'
0000A0  FILENC    CCW     X'31',SEEKADR+3,X'40',5
0000B0            DC      IF'0'
0000B4            DC      H'0'
0000B6            DC      H'0'
0000B8            CCW     X'06',IOA1,X'40',0
0000C0            DC      1D'0'
0000C8            CCW     X'39',SEEKADR+3,X'40',4
0000D0            CCW     X'0E',IOA1+8,X'40',0
0000D8            CCW     X'06',FILENK,X'40',8
0000E0            CCW     X'12',FILENK,X'40',8
```

```
   0000E8              CCW      X'31',FILENF,X'40',5
   0000F0              CCW      X'1E',IOA1,X'40',0
   0000F8              CCW      X'11',CCWAD+32,X'40',3625
   000100              NOPR     0
   000102              NOPR     0
   000104    FILENF    DC       5X'0'
   000109    FILENK    DC       8X'0'
   000049    FILEN0    EQU      TABAD+1
   000049    FILEN1    EQU      TABAD+1
   000049    FILEN2    EQU      TABAD+1
   000049    FILEN3    EQU      TABAD+1
   000111    FILEN4    EQU      *
   000111              DC       X'C718D752C718B5'
   000118    FILEN5    EQU      *
   000118              DC       X'C71834'
   00011B              DC       X'C718B18718CD'
   000121
   000128    CCWAD     CCW      X'07',SEEKADR+1,X'40',6
   000130              DC       7D'0'
   000168    IOA1      DC       0D'0'
   000168    KEYARG    DC       0D'0'
                       EXTRN    IJKTXRP
                       END
```

```
000000    FIDIOVR3  START    0
000000              DC       X'A5'                      OPEN MASK
000001              DC       AL3(TABAD)                 TABLE ADDRESS
000004              DC       X'01'                      FLAG BYTE ONE
000005              DC       AL3(0)                     CHAIN ADDRESS
000008              DC       X'2800'                    FLAG BYTE TWO, COMM. BYTE
00000A              DC       H'0'                       RECORD LENGTH
00000C              DC       X'80'                      DEVICE CODE
00000D              DC       AL3(0)
000010              DC       A(IOA1)                    BUFFER ADDRESS
000014              DC       X'08'                      REGIONAL TYPE
000015              DC       AL3(IJKTXRP)               ADDRESSING ROUTINE
000018              DC       A(KEYARG+8)
00001C              DC       A(0)
000020              DC       X'0000'                    LOGICAL UNIT
000022    ERRBYTE   DC       X'0000'                    ERROR BYTE
000024              DC       H'0'                       KEYLENGTH
000026              DC       X'00'
000027    SEEKADR   DC       15X'00'
000036              DC       H'10'
000038              DC       6X'0'
00003E              DC       H'10'
000040              DC       6X'0'
000046              DC       H'10'
000048    TABAD     DC       0D'0'
000048              DC       H'0'                       FIRST CCB BYTES
00004A              DC       X'88'
00004B              DC       5X'0'
000050              DC       AL4(CCWAD)                 CCW ADDR IN CCB
000054              DC       F'0'
000058              DC       C(IJIFAZZZ)                FILE TYPE
00005C              DC       X'22'
00005D              DC       B'11010000'
00005E              DC       CL8'FIDIOVR3'
000066              DC       X'0104'
000068              DC       F'0'                       LABEL ROUTINE ADDRESS
000060              DC       V(IJKTXRM)                 EXTENT ROUTINE ADDRESS
000070              DC       X'0'
000071              DC       AL3(ERRBYTE)
000074              DC       H'0'                       TEST SWITCH
000076              DC       Y(CCWAD-TABAD-32)          POINTER
000078              DC       H'0'                       IJICB2
00007A              DC       X'88'
00007B              DC       5X'0'
000080              DC       AL4(FILENZ)
000084              DC       4X'0'
000088    FILENZ    CCW      X'07',SEEKADR+1,X'00',6
000090              IX       36(2),C'0'
000094              DC       H'0'                       MAXIMUM DATA LENGTH
000096              DC       YL1(FILEN0-TABAD-1)        PTR TO READ ID STRING
000097              DC       YL1(FILEN1-TABAD-1)        READ KEY
000098              DC       YL1(FILEN2-TABAD-1)        WRITE ID
000099              DC       YL1(FILEN3-TABAD-1)        WRITE KEY
00009A              DC       YL1(FILEN4-TABAD-1)        RZERO
00009B              DC       YL1(FILEN5-TABAD-1)        AFTER
00009C              DC       H'2'
00009E              DC       H'61'                      RIC CONSTANT
0000A0              DC       D'0'
0000A8    FILENC    CCW      X'31',SEEKADR+3,X'40',5
0000B0              DC       1F'0'
0000B4              DC       H'0'
0000B6              DC       H'0'
0000B8              CCW      X'06',IOA1,X'40',0
0000C0              DC       1D'0'
0000C8              CCW      X'39',SEEDADR+3,X'40',4
0000D0              CCW      X'0E',2OA1+8,X'40',0
0000D8              CCW      X'06',FILENK,X'40',8
0000E0              CCW      X'12',FILENK,X'40',8
```

```
0000F8              CCW     X'31',FILENF,X'40',5
0000F0              CCW     X'1E',IOA1,X'40',0
0000E8              CCW     X'11',CCWAD+32,X'40',3625
000100              NOPR    0
000102              NOPR    0
000104    FILENF    DC      5X'0'
000109    FILENK    DC      8X'0'
000049    FILEN0    EQU     TABAD+1
000049    FILEN1    EQU     TABAD+1
000049    FILEN2    EQU     TABAD+1
000049    FILEN3    EQU     TABAD+1
000111    FILEN4    EQU     *
000111              DC      X'C718D752C718B5'
000118    FILEN5    EQU     *
000118              DC      X'C71834'
00011B              DC      X'C718B18718C9C7183187184D'
000127
000128    CCWAD     CCW     X'07',SEEKADR+1,X'40',6
000130              DC      7D'0'
000168              DC      5D'0'
000190    IOA1      DC      0D'0'
000190    KEYARG    DC      0D'0'
                    EXTRN   IJKTXRP
                    END
```

```
000000   FIDIUNR3  START   0
000000             DC      X'9D'                         OPEN MASK
000001             DC      AL3(TABAD)                    TABLE ADDRESS
000004             DC      X'01'                         FLAG BYTE ONE
000005             DC      AL3(0)                        CHAIN ADDRESS
000008             DC      X'2800'                       FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                          RECORD LENGTH
00000C             DC      X'80'                         DEVICE CODE
00000D             DC      AL3(0)
000010             DC      A(IOA1)                       BUFFER ADDRESS
000014             DC      X'08'                         REGIONAL TYPE
000015             DC      AL3(IJKTXRP)                  ADDRESSING ROUTINE
000018             DC      A(KEYARG+8)
00001C             DC      A(0)
000020             DC      X'0000'                       LOGICAL UNIT
000022   ERRBYTE   DC      X'0000'                       ERROR BYTE
000024             DC      H'0'                          KEYLENGTH
000026             DC      X'00'
000027   SEEKADR   DC      15X'00'
000036             DC      H'10'
000038             DC      6X'0'
00003E             DC      H'10'
000040             DC      6X'0'
000046             DC      H'10'
000048   TABAD     DC      0D'0'
000048             DC      H'0'                          FIRST CCB BYTES
00004A             DC      X'88'
00004B             DC      5X'0'
000050             DC      AL4(CCWAD)                    CCW ADDR IN CCB
000054             DC      F'0'
000058             DC      V(IJIFAZZZ)                   FILE TYPE
00005C             DC      X'22'
00005D             DC      B'00010000'
00005E             DC      CL8'FIDIUNR3'
000066             DC      X'0104'
000068             DC      F'0'                          LABEL ROUTINE ADDRESS
00006C             DC      V(IJKTXRM)                    EXTENT ROUTINE ADDRESS
000070             DC      X'0'
000071             DC      AL3(ERRBYTE)
000074             DC      H'0'                          TEST SWITCH
000076             DC      Y(CCWAD-TABAD-32)             POINTER
000078             DC      H'0'                          IJICB2
00007A             DC      X'88'
00007B             DC      5X'0'
000080             DC      AL4(FILENZ)
000084             DC      4X'0'
000088   FILENZ    CCW     X'07',SEEKADR+1,X'00',6
000090             XI      36(2),C'0'
000094             DC      H'0'                          MAXIMUM DATA LENGTH
000096             DC      YL1(FILEN0-TABAD-1)           PTR TO READ ID STRING
000097             DC      YL1(FILEN1-TABAD-1)           READ KEY
000098             DC      YL1(FILEN2-TABAD-1)           WRITE ID
000099             DC      YL1(FILEN3-TABAD-1)           WRITE KEY
00009A             DC      YL1(FILEN4-TABAD-1            RZERO
00009B             DC      YL1(FILEN5-TABAD-1)           AFTER
00009C             DC      H'20'
00009E             DC      H'61'                         RIC CONSTANT
0000A0             DC      D'0'
0000A8   FILENC    CCW     X'31',SEEKADR+3,X'40',5
0000B0             CCW     X'29',KEYARG+8,X'40',0
0000B8             CCW     X'06',IOA1,X'40',0
0000C0             DC      1D'0'
0000C8             CCW     X'39',SEEKADR+3,X'40',4
0000D0             CCW     X'0E',IOA1+8,X'40',0
0000D8             CCW     X'06',FILENK,X'40',8
0000E0             CCW     X'12',FILENK,X'40',8
0000E8             CCW     X'31',FILENF,X'40',5
0000F0             CCW     X'1E',IOA1,X'40',0
```

```
0000F8              CCW      X'11',CCWAD+32,X'40',3625
000100              NOPR     0
000102              NOPR     0
000104    FILENF    DC       5X'0'
000109    FILENK    DC       8X'0'
000049    FILEN0    EQU      TABAD+1
000111    FILEN1    EQU      *
000111              DC       X'8F1814'
000049    FILEN2    EQU      TABAD+1
000114    FILEN3    EQU      *
000114              DC       X'8F1895'
000117    FILEN4    EQU      *
000117              DC       X'C718D752C718B5'
00011E    FILEN5    EQU      *
00011E              DC       X'C71834'
000121              DC       X'C718B18718CD'
000127
000128    CCWAD     CCW      X'7',SEEKADR+1,X'40',6
000130              DC       7D'0'
000168    IOA1      DC       0D'0'
000168    KEYARG    DC       0D'0'
                    EXTRN    IJKTXRP
                    END
```

```
000000   FIDIUVR3  START   0
000000             DC      X'9D'                         OPEN MASK
000001             DC      AL3(TABAD)                    TABLE ADDRESS
000004             DC      X'01'                         FLAG BYTE ONE
000005             DC      AL3(0)                        CHAIN ADDRESS
000008             DC      X'2800'                       FLAG BYTE TWO, COMM. BYTE
00000A             DC      H'0'                          RECORD LENGTH
00000C             DC      X'80'                         DEVICE CODE
00000D             DC      AL3(0)
000010             DC      A(IOA1)                       BUFFER ADDRESS
000014             DC      X'08'                         REGIONAL TYPE
000015             DC      AL3(IJKTXRP)                  ADDRESSING ROUTINE
000018             DC      A(KEYARG+8)
00001C             DC      A(0)
000020             DC      X'0000'                       LOGICAL UNIT
000022   ERRBYTE   DC      X'0000'                       ERROR BYTE
000024             DC      H'0'                          KEYLENGTH
000026             DC      X'00'
000027   SEEKADR   DC      15'00'
000036             DC      H'10'
000038             DC      6X'0'
00003E             DC      H'10'
000040             DC      X'0'
000046             DC      H'10'
000048   TABAD     DC      0D'0'
000048             DC      H'0'                          FIRST CCB BYTES
00004A             DC      X'88'
00004B             DC      5X'0'
000050             DC      AL4(CCWAD)                    CCW ADDR IN CCB
000054             DC      F'0'
000058             DC      V(IJIFAZZZ)                   FILE TYPE
00005C             DC      X'22'
00005D             DC      B'01010000'
00005E             DC      C18'FIDIUVR3'
000066             DC      X'0104'
000068             DC      F'0'                          LABEL ROUTINE ADDRESS
00006C             DC      C(IJKTXRM)                    EXTENT ROUTINE ADDRESS
000070             DC      X'0'
000071             DC      AL3(ERRBYTE)
000074             DC      H'0'                          TEST SWITCH
000076             DC      Y(CCWAD-TABAD-32)             POINTER
000078             DC      H'0'                          IJICB2
00007A             DC      X'88'
00007B             DC      5X'0'
000080             DC      AL4(FILENZ)
000084             DC      4X'0'
000088   FILENZ    CCW     X'07',SEEKARD+1,X'00',6
000090             XI      36(2),C'0'
000094             DC      H'0'                          MAXIMUM DATA LENGTH
000096             DC      YL1(FILEN0-TABAD-1)           PTR TO READ ID STRING
000097             DC      YL(FILEN1-TABAD-1)            READ KEY
000098             DC      YL1(FILEN2-TABAD-1)           WRITE ID
000099             DC      YL1(FILEN3-TABAD-1)           WRITE KEY
00009A             DC      YL1(FILEN4-TABAD-1)           RZERO
00009B             DC      YL1(FILEN5-TABAD-1)           AFTER
00009C             DC      H'20'
00009E             DC      H'61'                         RIC CONSTANT
0000A0             DC      D'0'
0000A8   FILENC    CCW     X'31',SEEKADR+3,X'40',5
0000B0             CCW     X'29',KEYARG+8,X'40',0
0000B8             CCW     X'06',IOA1,X'40',0
0000C0             DC      1D'0'
0000C8             CCW     X'39',SEEKADR+3,X'40',4
0000D0             CCW     X'0E',IOA1+8,X'40',0
0000D8             CCW     X'06',FILENK,X'40',8
0000E0             CCW     X'12',FILENK,X'40',8
0000E8             CCW     X'31',FILENF,X'40',5
0000F0             CCW     X'1E',IOA1,X'40',0
```

```
0000F8                CCW      X'11',CCWAD+32,X'40',
000100                NOPR     0
000102                NOPR     0
000104   FILENF       DC       5X'0'
000109   FILENK       DC       8X'0'
000049   FILEN0       EQU      TABAD+1
000111   FILEN1       EQU      *
000111                DC       X'8F1814'
000049   FILEN2       EQU      TABAD+1
000114   FILEN3       EQU      *
000114                DC       X'8F18918F1815'
00011A   FILEN4       EQU      *
00011A                DC       X'C718D752C718B5'
000121   FILEN5       EQU      *
000121                DC       X'C71834'
000124                DC       X'C718B18718C9C7183187184D'
000130   CCWAD        CCW      X'07',SEEKADR+1,X'40',6
000138                DC       7D'0'
000170                DC       5D'0'
000198   IOA1         DC       0D'0'
000198   KEYARG       DC       0D'0'
                      EXTRN    IJKTXRP
                      END
```

ABS 225
ABS, fixed binary 262
ABS, fixed decimal 265
ABS, long float 269
ABS, short float 267
AC1-AC6 177
ACOMA 194
ACTIO(R5) 196
ACTION0 - 31 180
ACTION0 - 31(D00) 175
ACTION0 - 31(D10) 188
ACTION0 - 31(D11) 198
ACTION2C 183
ADASSI 202
ADASSI1 - 9 192
ADBASA 203
ADBASC 202
ADBISA 203
ADBOIV 202
ADBSIP 203
ADBSOP 203
ADCALL 203
ADCHAP 203
ADCHOP 203
ADCOB3 193
ADCON 293
ADCOST 194
ADD 225,311,316,323
ADDARI 202
ADDEQ 194
ADDIV 202
ADDMU 202
ADDNEG 202
ADDNIG 203
ADDPQL 192
ADDRF 221
ADDSHI 202
ADIF 203
ADIOST 202
ADMACO 198
ADMULI 202
ADMVC 172,197
ADNEG1 202
ADRTLC 203
ADSECO 203
ADTEE6 193
ADUNA 202
AHSTAB 75
ALIGN 297
ALL 225
ALLOC 291
ALLVAR 291
ANDO 158
ANDOST 153
ANEN 158
ANEND 153
ANY 225
ARITH1-2 182
ARGADR 221
Array 70
ARRAY, ARRAY0 178
Array expression begin 279
Array table 205
Array table construction 111
ARRCH 178
ARROUT 178
ARRTAB 209
Array expression end 281
ARYTAB 111
ASGN 226
ASKIP 208,219
ASSCODE 289
Assign label constant 274
Assign label variable 275
Attribute byte 172
Attribute table 107,109
Attribute table compression 109

BANN 51
BAS 310,325
BEBE 120
BFSTR 246
BITSET 223
Bit strings (generated code) 272
BLBL 120
Block description table 229
Block sorting 127
Block structure 57
Block table(PBT) 312
Block table listing 354
BLOCKT 356
BLTAB 298
BLUE 51
BOOLF 221
BOU 326
Branch macro 118
Branch on condition 274
BRAN1 316
BRG 326
BSAC 120,153,158
BUBU 65
Buffer areas 21
Built-in functions, library calls for . 212
Built-in functions II 224
BYBY 159
BYPA 120,153,159

CACTION0-7 193
CALL 277
CALL(3) 282
Call library routine 283
CARFB 65
CARTAB 54,297
CHAIN 178
CHAMO 352
CHAR 353
Character string assignment 275
Character string comparison 271
Character string concatenation 270
Character strings 54
Character string table 54
CHECK3-5 209
CHECKENT 184
CHECKJK 223

CHECKOUT 177
CHECKSP 178,180
CHECKST 135
CKREVA 149
CLOSE 144
CNOP 311,316
Code generation 253,284
COMMON 186
Communication region 7
COMOMA 146
COMOMA0 136
Compiler interface 20
CON 159
CONEND 291
CONLBE 143
CONOUT 294
CONSCR 294
CONSTA 290
Constants in static storage 319
Constants, optimization of 292
Constants, processing of (I) 123
Constants, processing of (II) 237
Constants, sorting of 287
Constant table 123,237
CONTAB 292,297
CONTB 146
CONTBR 150
CONTEB 135
Contextual declarations 92
CONVAR 289
Conversion 180
Conversion CV4-5 270
Conversion (generated code) 270
CONVERT 184,192
CONVN 222
COSC 153
COTAB 190
COUNT 245
COUT 291
CV36 193
CVFISCH 210

D17 219
Data lists of GET/PUT 138
DCA 323
DCAL3 311,316,322
DCF 311,316,323
DCLASS 324
DCSO 324
DCSTA 311,317,323
DCV 323
DCX 311,316,323
DEC 51
Declaration pool 67
Declaration scan 67
Declaration scan II 70
DECLARE statements 67
DED 222
DEDGEN 222
Define label constant 274
Define label macro 118
DEROUT0-7 194
DETERMIN 186
Diagnostic 250
Diagnostic, final 347
DIFL 243
DIMCHK 209
DISPLAY 150

DIVIDE 225
DO branch 275
DODO 120
DOPH 153,158
DO statement 151
DO statement II 155
DPDS 120
DPTE 51
DRIFT 51
DSEND 291
DSGEN 183
DSL 311,316,323
DSPUT 290
DSTAB 298
DTF tables 375
DYNDMP 222

EAACT 208,219
EBACT 208,219
ECAV 51
Edit pattern (pictures) 47
E0ACT 208,219
E0-EF 178
E1GEN 179
E302-E310 183
ELCO 38
ELEL 121
ELO 346
END card generation 342
ENDBL 311,316,323
End-of-statement key 57
ENDX 153
EOP 249
EOPH 136,141,147,150
EOPACT 209,220,225
EOS key 57
EOST 65,121,136,154,159
EQU 294
EQUOUT 294
EQUSR 294
ERCAL 340
ERLI 340
Error 53,66,121,136,154,159
ERROR (D00-D11) 172
Error diagnostic 250
ESD 336
ESD card generation 338
ESFIN 339
ESI 346
ESMO 340
EXCHP 223
EXPA 51
EXPOA 222
EXPOB 222
EXPONENT 183
EXTAB 324
External name table (EXTTAB) 113
External symbol table listing 354
EXTTAB 113

FACTIONY 196
FBFI 125
FBFL 125
FBID 240
FBII 239
FBIL 239
FBLL 239
FBIN 126

FBIS 241
FBLO 82
FBLS 240
FBSL 239
FBSS 241
FBST 125
FBUF 82
FCMB 39
FCOM 38
FCON 125,238
FCSC 124
FCTA 55
FDDI 240
FDFI 125
FDFL 125
FDIB 243
FDIL 242
FDIS 242
FDLB 242
FDLD 243
FDLL 242
FDLS 242
FEFIBL 135
FEND 41
FENV 80
FEOS 83,238
FERR 83,238
FERRUC 238
FEST 124
FETCH 175,180,197
FETCHA, FETCH1 195
FFIL 80
FFIN 56
FFIT 82
FFIX 82
FGSC 237
FIDE 41,55
FILE 70
File attributes and options,
 conflicting 81
File declarations 79
File generation 327
File module 334
File table FILTAB 79
FILL 293
FILTAB 79
Final output 354
FINDKEY 182
FINT 82
FISCH 211
Fixed binary addition 259
Fixed binary assignment with overflow
 check 260
Fixed binary assignment without
 overflow check 261
Fixed binary comparison 261
Fixed binary division 260
Fixed binary exponentiation 261
Fixed binary multiplication with
 overflow check 259
Fixed binary multiplication without
 overflow check 261
Fixed binary negation 260
Fixed binary subtraction 259
Fixed decimal comparison 263
Fixed decimal division 263
Fixed decimal exponentiation 264
Fixed decimal multiplication 263
Fixed decimal negation 263
Fixed decimal subtraction 263
FKBU 39
FKEL 82
FKEW 39
FLBI 242
FLCEILF 222
FLCLDF 222
FLDI 241
Float comparison 269
Float general exponentiation 267
FLTRO 222
FMBU 41,56
FMED 82
FNBI 241
FNCA 39
FNDI 241
FNOP 82
FORMAT 146,246,276
Format label assignment, macros for ... 204
Format list 144
FORMAT statement 144
FORMTAB 209
FOUT 121
FPDT 82
FPER 39
FPFI 126
FPFL 126
FPIN 41,52,56,238
FQUO 38
FREE 195
FREEING 195
FREG 82
FREP 55,126
FROTO 184
FRR 310
FSBI 239
FSCA 55
FSCN 80
FSCS 240
FSDI 240
FSI 52
FSI1 52
FSLA 38
FSPE 82
FSTL 125
FSTN 38
FSTR 55
FSYS 82
FTKW 39
FUNCTA 220
Function procedures 84
FUVN 82
F(Y) 51

GEASS 136
GEASS3 147
GEASSR 150
GECON 291
GEDOST 136
GEEND 136
GENADCO 221
GENCO 221
Generated temporary variables 151
Generator phases 253
GENDEDAD 222
GENLAB 226
GENVAR 290

GEO 324
GEOS 121,154,159
GEPOI 143
GEPUF 147
GEPUII 141
GET 355
GETCON 294
GETEX 339
GETIN 291
GETLA 339
GETLOC 298
GETOP 195
GETPRE 290
GETPUT 246
GETST 195
GETSYM 290
GEVA 143
GEVA0 137
GEVARE 150
GRGR 121
GSN 121,154,159

HAM 52
HDETER 285
HEINS 284
HERH 252
HESUB 31
High 276
HIGHLOW 221
HINDAD 285
HMOCO 284
HMOVE 286
HOPE 285
Housekeeping on work files 25
HTE 52
HUE 286

Identifiers, replacement of 40
IDEXPR 141
IF 275
IF statement 118
IFFALSE statement 118
IFIF 121
IFPH 120
IJKAGI 28
IJKAGINO 28
IJKAGO 28
IJKAGONO 28
IJKANT 27
IJKAMN 29
IKJAPINO 29
IJKAPH 28
IJKAPI 29
IJKAPO 29
IJKAPT 27
IJKAPTR 27
IJKATIN 30
IJKATOUT 30
IJKAWT 27
IJKGI 26,28
IJKGO 26,28
IJKMBL 7
IJKMBS 7
IJKMN 26
IJKMO 26
IJKMTS 7
IJKNT 26
IJKPH 26
IJKPI 26,29
IJKPO 26,29
IJKPONO 29
IJKPT 26
IJKPTR 26
IJKRI 26,28
IJKRO 26,28
IJKTI 26
IJKTO 26
IJKWI 26,29
IJKWO 26,29
IJKWT 26
INC1 159
INCHARF 221
INCR 290
INCRE 219
INIGP 134
INISC2 141
INITIAL 70,276
Initialization for disk versions (A00) . 35
Initialization for tapes (A00) 34
Initialization for tape and disk (A00) . 35
INITSUB 209
INLE 346
INPT 65
INPUT 135,247
INRE 345
INSERT 285
Interface, compiler 20
Interface, new 161
Interface routines 26
Interface structure 25
Internal PL/I code 36
INTREST 220,226
I/O flow during compilation 11
I/O macros I 245
I/O macros II 248
I/O scan I 131
I/O scan II 138
I/O scan III 144
I/O scan IV 148
IPDS 121
ISCR 326
ISU 304

JAHSA1 77
JASSA1 58
JATAA1 76
JATRA1 71
JBEGA1 68
JBETA1 61
JBIPA1 110
JBLT 107
JCATA1 109
JCCBA1 85
JCES 109
JCHAA1 60
JCHEA1 85
JCIR 109
JCPI 110
JCPLA1 68
JCSTA1 84
JCVTA1 71
JCWTA1 78
JDCLA1 75
JDCSA1 68
JDECA1 71
JDEPA1 69

JDLAA1 60,67
JELSA1 59
JENDA1 59,68
JENTA1 58,68
JEOPA1 61
JEOS 83,126
JEOSA1 59,65,69,121,154,159
JEPLA1 57
JERR 126,238
JERRA1 60,66,121,154,159
JFIXA1 61
JGOF 110
JGOTA1 60
JLABA1 75
JLACA1 59
JLEN 110
JMAC 107
JMDCA1 85
JMIB 126
JNAMA1 77
JNSTA1 84
JOPTA1 68
JPCOA1 78
JPCRA1 68
JPDOA1 59
JPIFA1 58
JPREA1 76
JPROA1 58
JPUTA1 78
JQULA1 85
JRPS 107
JRSTA1 84
JSAR 107
JSATA1 76
JSCC 108
JSCNA1 70
JSCOA1 74
JSID 107
JSIPA1 71
JSKPA1 60
JSLC 126
JSLCA1 65,69
JSPOA1 84
JSRTA1 70
JSSAA1 61
JSTAA1 58
JSTBA1 67
JTRA 108
JTRNA1 60
JTRIA1 71
JTRLA1 77
JTRN 83,126,238
JTRNA1 52,65,121,154,159

KBT 310
KCHECK 27
KCONDOUT 236
Keywords, replacement of 36
Keyword table 37
KEY4MOD 187
KGETNOTE 27
KINTER 236
KONLOOK 236
KONSTOUT 236
KOTE 236
KRAFT 159
KREAD 27
KREP 28
KSAVE1 7
KTESCA 235
KTETA 7,25
K2CHECK 27
K5PH 7

LAB 352
LABE 315
LABEL 311,316,323
LABELAS 209
Label declaration list 67
Label handling 312
Label offsets 318
Label table 312
LABS 340
LABTAB 312,336
LAREG 284
LAST 298
LATA 315
LDXEC 340
LEFTH 298
LENGTH 178
LF utility routines 63
LGEN 121,154,159
LIBCALL 223
LIBER 340
Library routines 373
LINCR 297
Linguistic functions 63
Linkage for built-in functions 212
LINKS 356
LIOCS for disk versions 33
LIOCS for tapes 32
LIOCS modules 26
LJX 211
LKW 66
LNK 355
Load 281
Load address of ON block 276
Load array 282
Load DED 282
Load long 282
Load multiple 282
Load scalar 282
Load short 281
Load transmit 277
Load variable 280
LOCAL 297
LOCAT 352
Logical flow of compiler 15
Logical parts of compiler 13
Long float addition 267
Long float assignment 269
Long float division 268
Long float exponentiation 269
Long float general exponentiation 269
Long float multiplication 268
Long float negation 268
Long float subtraction 268
Loop begin 283
Loop end 283
Low 280
LGEN 121
LGEN 154
LGEN 159
LJX 211
LVGE 159
LY 211

Macro generation II 196
Macros and generated code 258
MADM 316
MAIN 180
MAKGEN 210
MAMA 121
MAMIFI 221
Mask table MSKTAB 74
MASURO 287
MASU (X) 198
MATRIX 185
MAX 346
MESID 339
MEX 345
MIO 311
MOC 315
MOD 293
MODDF 222
MODF 221
Model-instruction dictionary 254
MODIF 304
MOK 310,324
MOK1 326
MOKK 346
MOO 304
MOOK 336
MOR 337
MOSC 345
MOV21 209
MOV21-31 220
MOV31 209
MOVE 247
MOVEA1 60
Move address 279
Move character 283
MOVECON 184
MOVEDATA 182
Move immediate 281
MOVEII 173
MOVEIO 173
MOVEIS 173
MOVEIT 173
MOVESO 173
MOVETO 173
MOVGEN 210
MOVO21 209
MOVOUT 220,226
MSKTAB 74
MUDI 243
MUFL 243
MULTI 311,323
MULTIPLY 225
Multiply halfword 283

N1-6 220
NAME 353
NAMTAB 40
New interface 161
NOTOPT 293
NSNS 121

Object code listing 348
OBRA 322
OFFOUT 297
OFFSET 298
Offset handling 306
Offset preparation, final 319
Offset table, building of 299
OFFTAB1 297
OFLIS 318
OGA 324
OGE 310
ON 229
OPCLO 147
OPEN 144
Operation priorities 176
OPT 290
OPTIM 294
OPTIM2 294
OPTIM4 293
OPTIM8 293
OPWSP 247
Or immediate 283
OTSEKI 150
OUT 209,220,290
OUTASS 289
OUTCOND 142
OUTGEV 137
OUTPCI 142
OUTPUT 135,247,291
Output, final 354
Output listing 348
OUTSBST 143
OUTSPV 143
OUTSVC 142
OUTSTR 142
OUTTAB 137
Overlay 281

P80-P183 186
PAD 340
PARA 311,317
Parameter list 67
PARG 222
PAT 52
PBT 312
PCH 355
Phase A25 36
Phase A30 40
Phase A35 42
Phase A45 54
Phase A50 57
Phase B10 67
Phase B15 70
Phase B20 73
Phase B25 79
Phase B30 84
Phase B40 86
Phase B70 92
Phase B90 107
Phase B92 109
Phase B95 111
Phase B97 113
Phase C00 116
Phase C25 118
Phase C30 123
Phase C35 127
Phase C50 131
Phase C55 138
Phase C60 144
Phase C65 148
Phase C85 151
Phase C86 155
Phase C95 161
Phase D00 175
Phase D05 180

Phase D10 188
  part 1 188
  part 2 189
  part 3 194
Phase D11 196
Phase D15 204
Phase D17 212
Phase D20 224
Phase D40 229
Phase D70 237,248
Phase D75 245
Phase E25 250
Phase E50 284
Phase F25 287
Phase F35 292
Phase F75 295
Phase F90 299
Phase F95 306
Phase G00 312
Phase G01 318
Phase G15 319
Phase G17 329
Phase G20 334
Phase G25 338
Phase G30 341
Phase G31 347
Phase G40 348
Phase G55 354
Phases A00, A00D, A10 34
Phases A60-A65 62
Phases D00-D11 (general) 163
Phases E50-E61 (general) 253
Phases E60-E61 285
Phases F95-G55 (general) 304
Phases, list of 10
PICK 298
Pictures 42
PIP1 50
PLENG 222
PLI 355
PMAK0 221
PMAK87 210
P(NA) 186
POB 122
Pointer assignment 274
Pointer comparison 274
PQ 182
PRECSION 184
Precision handling 180
PREMAC 189
Prestatement generation 107
PRET 289
PRETAB 113
PRETEND 290
PRIBLO 356
PRINT 356
PROCE 311,316,323
PROD 225
Prologue 278
PRT 355
Pseudo program (pictures) 47
PUCO 221
PUNCH 356
PUT 294
PUTCO 290
PUTLOC 298
PUTNV 291
PUTPRE 289
PUTOUT 178,294
PUTOUTES 182
PUTOUIFC 182

RAN 318
READIN 220
Record I/O 148
RECT 40
Register usage 7,22
REPE 222
Repeat 281
REPROUN 220
REQUEST 195
RESLEN 183
RETURN 129
Return 277
RETURN(D00-D11) 172
Return function value 280
Return to label constant 274
Return to label variable 275
Reverse polish notation 175
REVERT 229
RFDF 52
RFT 52
RLDCA 346
RLD 336
RLD card generation 341
RLDSYS 337
RERRTEST 297
ROFF 325
ROUNDBF 222
Routine table (D10) 191
RTESTIN 290

SACT30-43 211
SACT50-57 210
SACTION7 196
SACTIONX-Z 196
SAVE 293
SAVER 172
SBO 316
SCADAL1 135
SCADAL2 135
SCAFO 146
SCAL 315
Scope table (SCOTAB) 73
SCOTAB 73
SCSIJT 223
SDLDS 340
SDPRO 339
SEARCH 245,248
SEAV 52
SETERR 220,226
SETIN1-2 246
SETPNT 226
Set true on condition 282
SHIFT 262,284
Shifts (generated code) 272
Short float addition 265
Short float assignment 266
Short float comparison 266
Short float division 265
Short float exponentiation 266
Short float multiplication 265
Short float negation 266
Short float to long float assignment .. 270
Short float subtraction 265
SIGNAL 229

SIGN 225
SIGN, fixed binary 262
SIGN, fixed decimal 264
SIGN, float 267
SITE 52
SKIDLI 146
SKIEX 135
SKIEX3 147
SKILI 135
SKISTA 135
SKISTAT 141
SKIP 209,219,245,292,294,340
SKIPC 297
SKIPF7 219
SKIPIN 289
SKIPRE 290
SKIPV 298
SOURCE 195
SOURCE(I) 194
SPECFUN 221
STATEN 177
Statement decomposition 175
Statement identifier key 57
Statement numbering 316
STEP 122,154,160
STER 52
STOP 229
Storage allocation 295
Storage layout during compilation 20
Storage type, determination of 180
Store 275
Store long 279
Store multiple 279
Store short 275
STR 245
STRI 53
STROUT 178
Structure mapping 86
SUBSCR 209
Subscript evaluation 204
Subscript variable 275
Substring 276
SUBSTT 223
Sum 275
SUM 225
SUP 53
SWITAB 72
Symbol table construction I 73
Symbol table construction II 84
Symbol table listing 116
Symbol table (SYMTAB) 73
SYMTAB 73
SYN1 65
Syntax check 62
Syntax notation, input stream 357
Syntax notation, output stream 365

Table and buffer areas 21
TARGET 195
TARGET(I) 195
TARI1 66
TBSBST 223
TDS 53
TEEL 141
TEPHA 66
TESCON 210
TEST 293
TESTN 179
TESTR 142
Text string during compilation 18
TIMDAT 222
TPEP 53
TRA 353,356
TRACT3-5 210
TRAN 325
TRANLS 353
TRANS 318,352
TRANSL 353
TRANSR 353
TSCSIJ 223
TSUBST 223
TTS1 160
TXT card generation 341
TXTIN 289
TXTSYS 337

UDA 315
UREG 311,317

Variables, sorting of 287
VIRGO 122
VTE 53

WAIT1 289
WAIT2 290
WCTAB 93
WHY 160
WOLA 318
WSLIST 92
WTE 53
WUP 309

Z9 53
ZLEDI 31
ZMO 32
ZPCH 31
ZPRNT 31
ZRCD 31
ZTIN 30
ZTOUT 30

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

Form Y33-9010-0

IBM System/360
DOS/TOS PL/I PLM

- How did you use this publication?

As a reference source	☐
As a classroom text	☐
As a self-study text	☐

- Based on your own experience, rate this publication . . .

As a reference source:	Very Good	Good	Fair	Poor	Very Poor
As a text:	Very Good	Good	Fair	Poor	Very Poor

- What is your occupation? ....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

Y33-9010-0

YOUR COMMENTS PLEASE . . .

This SRL bulletin is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold

Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Department 813

Fold

Fold

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

DOS/TOS PL/I PLM

Form Y33-9010-0

- How did you use this publication?

As a reference source	☐
As a classroom text	☐
As a self-study text	☐

- Based on your own experience, rate this publication . . .

As a reference source:	Very Good	Good	Fair	Poor	Very Poor
As a text:	Very Good	Good	Fair	Poor	Very Poor

- What is your occupation? ....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This SRL bulletin is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Department 813

Fold Fold

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]