3300
3500

COMPUTER SYSTEMS
META/MASTER
REFERENCE MANUAL

CONTROL DATA
CORPORATION

60236400

REVISION RECORD

REVISION	NOTES
(11-20-68)	Original printing.

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Pub. No. 60236400
November 1968

Address comments concerning this manual to:

Control Data Corporation
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

PREFACE

This manual is directed at programmers using the 3300/3500 Meta-Assembler. It discusses the principles, features, methods, rules, and techniques of producing a META language program.

The reader is assumed to be familiar with the CONTROL DATA® 3300 Computer System or the CONTROL DATA® 3500 Computer System. In addition, familiarity with the 3300/3500 MASTER Multiprogramming Executive Operating System and the 3300/3500 COMPASS Assembly Language is helpful.

CONTENTS

		Page
CHAPTER 1	INTRODUCTION	1-1
	1.1 Features	1-1
	1.2 Hardware Configuration	1-2
CHAPTER 2	STATEMENT STRUCTURE	2-1
	2.1 Character Set	2-1
	2.2 Statement Format	2-1
	2.2.1 Label Field	2-2
	2.2.2 Command Field	2-2
	2.2.3 Operand Field	2-3
	2.2.4 Comments Field	2-3
	2.2.5 Statement Continuatior	2-3
	2.2.6 Examples	2-3
	2.3 Elementary Items	2-4
	2.3.1 Delimiters	2-5
	2.3.2 Decimal Integer	2-5
	2.3.3 BCD Decimal Integer	2-6
	2.3.4 Octal Integer	2-6
	2.3.5 Real Number	2-7
	2.3.6 BCD Character String	2-7
	2.3.7 ASCII Character String	2-8
	2.3.8 Operators	2-9
	2.4 Symbols	2-10
	2.5 Locations Counters	2-10
	2.6 Expressions	2-10
	2.6.1 Attributes	2-12
	2.6.2 Modes of Expressions	2-12
	2.6.3 Literals	2-14
	2.7 Sets	2-15
CHAPTER 3	LOCATION CONTROL AND ADDRESSING	3-1
	3.1 Relocatable Addresses	3-1
	3.2 Location Counters	3-2

Page

CHAPTER 4 DIRECTIVES 4-1

4.1 Listing Control 4-1
4.1.1 NOLIST 4-1
4.1.2 LIST 4-2
4.1.3 SPACING 4-2
4.1.4 EJECT 4-2
4.1.5 TITLE 4-3
4.1.6 BRIEF 4-3
4.1.7 DETAIL 4-3
4.2 Object Machine Definition (UNIT) 4-4
4.3 Symbol and Set Definition 4-4
4.3.1 EQU 4-5
4.3.2 RDEF 4-5
4.3.3 NSET 4-6
4.3.4 Forward References 4-7
4.4 Location Control 4-8
4.4.1 SECP 4-9
4.4.2 SECD 4-9
4.4.3 SECA 4-10
4.4.4 ORG 4-11
4.4.5 LIT 4-12
4.4.6 RES 4-12
4.4.7 RESB 4-13
4.5 Data Generation 4-13
4.5.1 GEN 4-13
4.5.2 GEND 4-14
4.5.3 GENB 4-15
4.5.4 FORM 4-15
4.5.5 TEXT 4-18
4.5.6 TEXTC 4-18
4.5.7 TEXTA 4-18
4.6 Program Linking 4-18
4.6.1 ENTRY 4-19
4.6.2 EXT 4-19
4.7 Repeat and Skip 4-19
4.7.1 RPT 4-19
4.7.2 GOTO 4-21
4.7.3 LNID 4-22
4.7.4 RPT and GOTO Processing 4-23
4.8 Assembly Termination 4-24
4.8.1 END 4-24
4.8.2 FINIS 4-24

		Page
CHAPTER 5	FUNCTIONS AND PROCEDURES	5-1
	5.1 Directives	5-2
	5.1.1 PROC	5-2
	5.1.2 FUNC	5-3
	5.1.3 NAME	5-3
	5.1.4 ENDS	5-5
	5.1.5 TREF	5-6
	5.1.6 LIBS	5-7
	5.2 Definition Processing	5-9
	5.3 Referencing	5-10
	5.4 Levels and Local Labels	5-13
CHAPTER 6	ATTRIBUTE FUNCTIONS	6-1
	6.1 Relocation (REL)	6-1
	6.2 Mode (MDE)	6-2
	6.3 Number of Elements (NUM)	6-3
	6.4 Size of Data (SZE)	6-4
	6.5 Symbol(SYM)	6-5
	6.6 Word Address (WRD)	6-7
	6.7 Byte Address (BYT)	6-8
CHAPTER 7	PROGRAM EXECUTION	7-1
	7.1 Control Card Requirements	7-1
	7.1.1 $JOB	7-1
	7.1.2 $SCHED	7-2
	7.1.3 $META	7-3
	7.1.4 Task Name	7-5
	7.1.5 End of File	7-6
	7.2 Sample Deck Structures	7-7
CHAPTER 8	MESSAGES	8-1
	8.1 List Format	8-1
	8.2 Error Codes	8-3
	8.3 Supplementary Information	8-4
	8.4 Cross Reference Table	8-5
	8.5 Messages on Out	8-5
APPENDIX A	CHARACTER SET	A-1
APPENDIX B	3300/3500 MNEMONIC INSTRUCTIONS	B-1
APPENDIX C	BINARY OUTPUT	C-1

Page

APPENDIX D 3300/3500 RELOCATABLE BINARY OBJECT DECK D-1

GLOSSARY Glossary-1

INDEX Index-1

LIST OF TABLES

2-1 Legal Operators 2-9

2-2 Combination of Operators 2-13

A-1 BCD/ASCII Code Conversion Table A-3

B-1 IDENT Procedure References B-9

B-2 MONITOR Procedure References B-21

B-3 BDP Procedure References B-30

B-4 Octal Index to Mnemonics B-39

INTRODUCTION 1

The 3300/3500 META/MASTER, a Meta Assembler executing on a CONTROL DATA® 3300 Computer System or CONTROL DATA® 3500 Computer System, provides a versatile, extensive, and self-extending language for directing the generation of object code.

1.1 FEATURES

Using Meta-Assembler (META), the programmer can choose a relocatable binary output format acceptable for loading and executing under the 3300/3500 MASTER Multiprogramming Executive Operating System, or define output as a byte stream not restricted to a 24-bit object word. Meta-Assembler is an ideal language in which to code compilers and assemblers, or to produce code for an alternate computer system. The object computer, real or simulated, may have a word size up to 48 bits.

Source statements to META include directives that control the assembler in much the same way machine language instructions control the computer, procedure definitions and references, and function definitions and references.

The Meta-Assembler language allows simple, brief notation, nested functions and procedures, and complex expressions involving sets.

Procedures and functions provide extensive parameterization of source statements. For example, META includes standard procedures for the 3300/3500 machine language instructions and for generating equivalent code. While these mnemonics resemble the 3300/3500 COMPASS repertoire, differences in syntax and in notation used for operand fields and modifiers cause incompatibilities between the two languages. In addition, META does not recognize COMPASS macros, most pseudo instructions, or numeric operation codes. For example, the representation of an octal number in the 3300 COMPASS language is a string of octal digits followed by the letter B. The representation of an octal number in the META language is the letter O followed by a string of octal digits enclosed in apostrophes..

A complete list of 3300/3500 mnemonic instructions is given in appendix B.

A group of Meta-Assembler directives makes it possible for the programmer to assign his program and data to as many as 15 relocatable control sections, as well as one absolute control section. The assembler maintains a location counter for each section so that data locations within a control section are relative to the beginning of that section. The programmer can increment these counters by words or bytes. (He can also define the size of words and bytes.)

1.2 HARDWARE CONFIGURATION

The requirements for executing Meta-Assembler on the 3300/3500 are the minimum requirements for executing MASTER.

MASTER minimum core memory (about 16K should be available for META)

One CONTROL DATA® 3304 or 3504 Processor

One CONTROL DATA® 3311 Multiprogramming Module (3300 only)

One CONTROL DATA® 405 Card Reader and buffered controller

One CONTROL DATA® 501, 505, or 3254 Line Printer and buffered controller

One CONTROL DATA® 415 Card Punch and buffered controller

Two CONTROL DATA® 3306 or 3307 (3507) Communication (Data) Channels

2.5 million words (10 million characters) of mass storage.
90K-100K words or about 9 scratch segments of mass storage should be available to META for its temporary files.

STATEMENT STRUCTURE 2

2.1 CHARACTER SET

Programs written for Meta-Assembler may use alphabetic characters A-Z, numeric characters 0-9, blank spaces, and the special characters listed below.

+	plus	'	apostrophe
-	minus	≤	less than or equal
*	multiply	≥	greater than or equal
/	divide	[	left bracket
=	equal	]	right bracket
<	less than	↑	decimal exponent
>	greater than	↓	binary exponent
.	period	¬	NOT
,	comma	;	semicolon
(	left parenthesis	→	right arrow
)	right parenthesis	≡	identity
%	percent	:	colon
$	dollar	∨	OR
		∧	AND

The relationship of these characters to printer graphic characters, internal octal codes, and card codes is shown in appendix A. Characters that have special significance as operators are given in Table 2-1.

2.2 STATEMENT FORMAT

A Meta-Assembler statement consists of a label field, a command field, an operand field, and a comments field. Each field is terminated by two or more consecutive blanks.

Format:

label␣␣command␣␣operand␣␣comments␣␣␣

Statements can begin in character position 1 and continue through character position 71. A semicolon in character position 72 indicates card continuation. Any information beyond character position 72 is not interpreted by Meta-Assembler but does appear on the assembly listing. Thus, columns 72-80 can be used for sequencing.

Within a field, a single blank can separate elementary items, operators, and delimiters. A blank is optional for separating a symbolic operator, such as **, from its operands, but is required for separating a mnemonic operator (AND) from its operands.

2.2.1 LABEL FIELD

The label field begins in character position 1 or 2, and is terminated by two consecutive blanks. If character positions 1 and 2 are blank, the statement has no label.

A label field may contain a symbol, set element reference, or SYM attribute function reference (section 6.5). A set element reference is legal only with an RDEF directive (section 4.3.2).

The definition of a symbol in a label field depends on the content of the command field. Throughout this manual, unless stated otherwise, a label field symbol is optional and, if present, is the value of the control counter prior to processing the command field. This value is either a word address or a byte address, depending on the command (section 3.1).

2.2.2 COMMAND FIELD

The command field of a statement begins with the first nonblank character following the label field and is terminated by two consecutive blanks. If the label field is blank, the command may start in character position 3.

The following are legal command field entries.

An assembler directive

A mnemonic machine instruction code followed by its modifiers which are separated from the instruction code by a comma

The name of a previously defined FORM

The name of a previously defined procedure which may be followed by a set separated from the procedure name by a comma

A SYM attribute function reference

2.2.3 OPERAND FIELD

The command field entry of a Meta-Assembler statement determines if an entry is required in the operand field. If present, the operand field begins with the first nonblank character following the command field. The operand field contains either an expression or a set that supplies information for the command field. For sets, see section 2.7.

Two consecutive blanks terminate the operand field.

2.2.4 COMMENTS FIELD

Comments begin with the first nonblank character after the operand field or, if the statement requires no operand field, with the first nonblank character following the command field. In addition, if the first character of any field is an asterisk, all successive characters of the line are comments. Thus, when the label field entry begins with an asterisk, the line is a comment line. Comments can continue through character position 72 but cannot be continued on the next line.

Any characters are legal as comments. Although META does not process comments, they do appear in the symbolic listing. Comments on lines of procedure or function definitions are not retained in the Meta-Assembler representation of the definition.

2.2.5 STATEMENT CONTINUATION

Normally, character position 71 terminates a source statement that has not yet terminated. However, a line of code that cannot be contained in the first 71 character positions can be continued to the next line by placing a semicolon in character position 72 and continuing the field at character position 2 of the next line; character position 1 is ignored.

Any character other than a semicolon in character position 72 is ignored.

2.2.6 EXAMPLES

The following line contains all four fields.

```
ALPHA LDA LØC LØAD A REGISTER
label | command operand   comments
```

The following line has a blank label field and does not contain comments.

```
  LDA  LØC
```

command operand

The following line is continued.

```
BETA   NSET  L'DØG',L'EASY',L'FØ  L'KIL;
 LØ',L'LIMA',L'MIKE',L'NANCY',L'
```

column 72

The following line contains a command and a comment.

```
  END  *CØMMENTS
```

The following line is a comment line.

```
* PARAMETER LIST FØLLØWS
```

The following line is not continued; character position 71 terminates the operand field.

Ignored; not semicolon

```
GAMMA  NSET    6.13285,5.4E2 ... 8.47PRØG12345
```

Last significant character

2.3 ELEMENTARY ITEMS

The basic representation of data for META are elementary items. An elementary item is self-defining and its meaning is immediately obvious; no additional information is needed for its interpretation. Meta-Assembler recognizes the following as elementary items.

Delimiters

Decimal integers

Binary coded decimal (BCD) integers

Octal integers

Floating-point real numbers

BCD character strings, left or right adjusted

ASCII character strings

Arithmetic and logical operators (symbolic and mnemonic)

2.3.1 DELIMITERS

META recognizes the following characters as delimiters.

Comma	Delimits subfields of a source statement field, elements of a set or subset, and subscripts of a set element reference.
Parentheses	Enclose and delimit function arguments and nested expressions.
Brackets	Enclose and delimit nested subsets and set element references.
Blank	Separates elementary items for visual clarity or delimits them when required.
Two blanks	Terminate fields of a source statement.
Apostrophes	Enclose and delimit character and numeric strings. Within a character string, only the single apostrophe is a delimiter; any other delimiter is accepted as a valid character in the string. A pair of apostrophes signifies a valid BCD or ASCII apostrophe.

2.3.2 DECIMAL INTEGER

A decimal integer is a string of numeric characters from the character set 0-9. Meta-Assembler converts a decimal integer to its binary equivalent. If the resulting binary number exceeds 48 bits, META truncates it with the loss of the most significant bits and sets an error flag. META also sets an error flag and truncates the resulting binary number if it exceeds a specified field size during data generation.

Examples:

429

-3

2.3.3 BCD INTEGER

To specify a BCD integer, write the letter D followed by a string of one to eight numeric characters from the character set 0-9, enclosed in apostrophes.

Examples:

D'078'

D'123'

-D'123'

A BCD integer is not converted to its binary equivalent, but is represented as a string of 6-bit BCD characters (appendix A). If the number of BCD characters is greater than eight, truncation causes loss of the most significant characters and META sets an error flag. During data generation, if the field into which the integer is to be placed is too small, META truncates the most significant characters and sets an error flag (section 4.5).

If during data generation, the field size is greater than required, the BCD integer is right adjusted with leading zeros.

Expressions containing BCD integers are evaluated using 6-bit BCD arithmetic. The sign of a BCD integer is placed in the left bit of the rightmost digit of the number.

Examples:

D'123'	00	01	02	03
-D'123'	00	01	02	43

2.3.4 OCTAL INTEGER

An octal integer is noted with the letter O followed by a string of numeric characters from the character set 0-7 enclosed in apostrophes.

Examples:

O'77'

O'123'

Meta-Assembler converts an octal integer to its binary equivalent. If the resulting binary number exceeds 48 bits, truncation causes loss of the most significant bits and META sets an error flag. If during data generation, the field into which the integer is to be placed is too small, META truncates the most significant digits and sets an error flag (section 4.5).

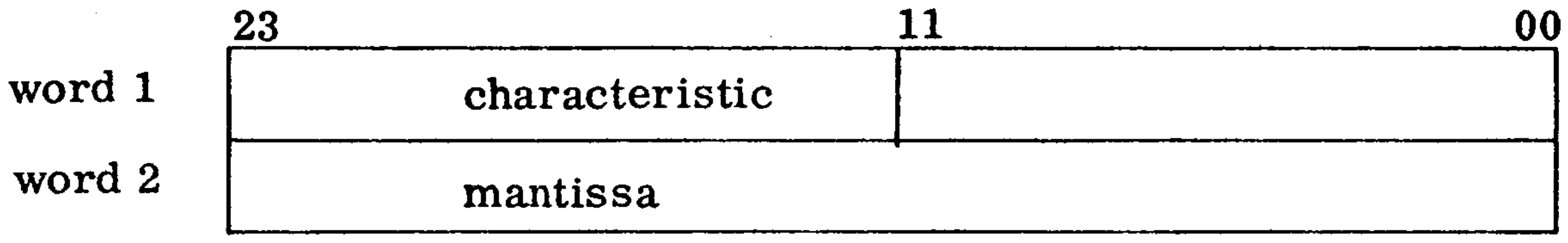
2.3.5 REAL NUMBER

A real or floating-point number is written as a maximum of 14 decimal digits. It must contain a decimal point and may contain an exponent representing a power of 10 designated by the letter E and an optionally signed 1- to 3-digit decimal integer.

Examples:

1.	1.E+2	(1.0×10^{2})
.35	327.7E-2	(327.7×10^{-2})
4.79		

META converts a real number to 48-bit 3300/3500 internal normalized floating-point format. It consists of two 24-bit words made up of a 12-bit characteristic and a 36-bit mantissa.



If during data generation the field size into which the number is to be placed is less than 48 bits, a truncation error is flagged and the rightmost bits of the number are lost. For a negative value the 2-word value, including characteristic, is complemented.

2.3.6 BCD CHARACTER STRING

A programmer specifies that a BCD character string be either right adjusted with leading zeros or left adjusted with trailing blanks.

A right-adjusted character string is written as the letter C followed by a string of not more than eight legal BCD characters enclosed in apostrophes, or simply as a string of BCD characters enclosed in apostrophes.

A left-adjusted character string is written as the letter L followed by a string of not more than eight legal BCD characters, enclosed in apostrophes.

Because an apostrophe is used as a delimiter, the representation of an apostrophe as a character in character strings is two consecutive apostrophes.

Examples:

'ABC'	Right-adjusted string of three characters ABC
C'A''BC'	Right-adjusted string of four characters A'BC
L'A''''BC'	Left-adjusted string of five characters A''BC

If the number of characters in a BCD string exceeds eight, truncation causes loss of the leftmost characters and META sets an error flag. During data generation, if the field into which the character string is to be placed is too small, META truncates the leftmost characters and sets an error flag.

If the field size is greater than that required to hold a left-adjusted string, the string used in data generation is left adjusted with trailing blanks. If the field size is not a multiple of 6 bits, the extraneous bits are on the left and are 0. The remainder of the field is used for characters and is blank filled.

Example:

L'AB' is stored in 21-bit field as

```
20     17            11            05            00
|  0   |      A      |      B      |      Λ      |
   ↑
   3 bits zero
```

A right-adjusted string used in data generation is right adjusted with leading zeros if the field size is greater than that required to hold the string.

2.3.7 ASCII CHARACTER STRING

An ASCII character string is written as the letter A followed by an apostrophe, a string of one to six ASCII characters (appendix A), and an apostrophe. Each ASCII character occupies eight bits.

Because an apostrophe is a delimiter, an apostrophe as a character in the string is represented as two consecutive apostrophes.

Example:

A'AB''CD'	A string of five characters AB'CD
A'''ABC'''	A string of five characters 'ABC'

If the number of characters exceeds six, truncation causes loss of the leftmost characters and META sets an error flag. During data generation, if the field into which the character string is to be placed is too small, META truncates the leftmost characters and sets an error flag.

An ASCII string used in data generation is right adjusted with leading zeros if the field size is greater than that required to hold the string.

2.3.8 OPERATORS

The following table summarizes legal operators and their hierarchies and meanings.

Table 2-1. Legal Operators

Operator	Alternate Mnemonic	Meaning	Hierarchy
+		Unary plus	1
-		Unary minus	
↑	DS	Decimal scaling	2
↓	BS	Binary scaling	
*		Arithmetic product	3
/		Arithmetic quotient	
+		Arithmetic addition	4
-		Arithmetic subtraction	
<	LT	Less than (compare)	5
=	EQ	Equal (compare)	
¬=	NE	Not equal (compare)	
>	GT	Greater than (compare)	
≤	LE	Less than or equal (compare)	
≥	GE	Greater than or equal (compare)	
**	AND	Logical product (AND)	6
--	XOR	Logical difference (exclusive OR)	7
++	OR	Logical addition (inclusive OR)	
=		Unary equal; 1-word literal	8
==		Unary double equal; 2-word literal	

2.4 SYMBOLS

A symbol is an alphabetic character from the set A-Z followed by 0-11 alphabetic or numeric characters from the sets A-Z, 0-9.

Examples:

Legal Symbols	Illegal Symbols
P	5A
R3	ST$RT
PROGRAM	ABC-1

2.5 LOCATION COUNTERS

A unique location counter is associated with each of the 16 control sections available under Meta-Assembler. META interprets a reference to a control section name as a reference to the current value of the location counter (a word address) within that control section.

In addition, META interprets the character $ as the value of the current location counter, a word address, prior to processing the line containing $. Location counters are discussed in detail in section 4.4.

2.6 EXPRESSIONS

A combination of one or more elementary items, symbols, set element references, or function references makes up an expression. The programmer can form subexpressions by using parentheses in the normal role of arithmetic grouping. Thus an expression may contain subexpressions which in turn are made up of operators and other subexpressions or elementary items.

Examples:

```
$
A + 2
(A+ 2)*B
```

Rules for evaluating expressions are:

Expressions are evaluated left to right with lower numbered hierarchies evaluated first.

Parenthetical subexpressions are expanded from the inside and are performed first.

Operators of equal hierarchy are evaluated left to right.

If a mnemonic operator is used in lieu of a special symbol (e.g., DS instead of ↑), it must be preceded and followed by a single blank.

The value of a compare operation is 1 if the expression is true, 0 if it is false.

For the < or LT and the > or GT operators, 0 is greater than -0. For the ≤ or LE, the ¬= or NE, and the = or EQ operators, 0 is equal to -0.

In expressions used for data generation, META performs the arithmetic operation and places the value in the specified field. If the resultant value exceeds the specified field size, META truncates the most significant bits and flags the error.

Examples:

Expression	Evaluation
A + B >C	Add A to B; compare the result to C.
A + B * C	Multiply B by C; add A to the product.
(A + B)*C	Add A to B; multiply the sum by C.
(A< B) + + (C> D)	Compare A to B; compare C to D; perform a logical OR on the two subexpressions. If either or both inequalities are true, the value is 1; if both are false, the value is 0.

If an expression contains relocatable symbolic addresses, its value must be relative to a single location counter, or not related to a location counter and thus nonrelocatable.

Examples:

In the following examples, P_i, D_i, and C_i refer to relocatable addresses in the program, data, and common areas.

The following are relocatable addresses.

P   D + 1   -C

Subtracting one relocatable address from another in the same program control section produces an absolute nonrelocatable result.

$P_1 - P_2$

$-C_1 + C_2$

$D_1 - P_1 + P_2 - D_2 + C_1 - C_2$

The result of an expression cannot be the sum of two or more relocatable addresses in the same or different control sections. The following are illegal.

$P_1 + (P_2 + 5)$	Relocated twice relative to P
$P + D$	Relocated to both P and D
$-P_1 - P_2 + P_3 - P_4$	Relocated twice relative to P

Single relocation or an absolute value can legally result from a complex expression.

$P_1 - P_2 + P_3$	Single positive relocation
$-P_1 + P_2 - P_3$	Single negative relocation
$P + D_1 - (D_2 + 2) - C_1 + (C_2 - 6)$	Result P-8 is single positive relocation
$P_1 + (P_2 + 5) + D_1 - (D_2 + 2) - C_1 + (C_2 + 6)$	Result + 9 is not relocatable relative to any control section.

2.6.1 ATTRIBUTES

An attribute is a property of an expression, such as its mode. Intrinsic attribute functions interpret the properties as values that can be used in expressions. Chapter 6 describes the Meta-Assembler attribute functions.

2.6.2 MODES OF EXPRESSIONS

A mode associated with each elementary item defines how META is to interpret the data when it performs an arithmetic operation on the item. Meta-Assembler recognizes 11 modes accessible through the mode attribute function (section 6.2).

Expressions are evaluated using either integer, real, or binary-coded-decimal arithmetic. META permits mixed-mode arithmetic on real and integer values, converting the integer to a real value and performing the operation in floating-point arithmetic. The mode of the result is real. With any combination other than real and integer, if all elements of the expression are not of the same arithmetic type, META flags an error and sets the value of the expression to 0.

In arithmetic and relational expressions, META treats character strings and addresses that are not external as integers.

META performs logical operations on a bit-by-bit basis without regard to mode. The result of a logical or compare operation is in integer mode.

The following table shows legal combinations of operators and operands. For + , -, *, and /, interchanging the first two columns does not affect the result. The mode of the second value must not be external.

Table 2-2. Combinations of Operators

Operator	Mode 1st Value	Mode 2nd Value	Mode of Result
↑	Integer Real Decimal	Integer Integer Integer	Integer Real Decimal
↓	Integer Real	Integer Integer	Integer Real
+ , -	Integer Integer Integer Real Decimal Word Addr Word Addr Byte Addr Ext Wrd Addr Ext Byte Addr	Integer Real Word Addr Real Decimal Word Addr Byte Addr Byte Addr Integer Integer	Integer Real Word Addr Real Decimal Word Addr Byte Addr Byte Addr Ext Wrd Addr† Ext Byte Addr†
* , /	Integer Integer Real Decimal	Integer Real Real Decimal	Integer Real Real Decimal
**, ++, --	Any	Any	Integer
>, =, ¬=, <, ≤, ≥	Mode 1, 3, 5, 7, 9, 11†† Real Decimal	Mode 1, 3, 5, 7, 9, 11†† Real Decimal	Integer Integer Integer

† External word addresses and external byte addressed cannot be interchanged.

†† Section 6.2

Scale factors, both decimal and binary, must be integer.

Examples:

1.5*3	Legal; value is 4.5 real.
D'15' + D'17'	Legal; both items are decimal integers.
D'15' + 1.5	Illegal; conflicting modes.
1.5 ↓ 2.5	Illegal; scaling factor is not an integer.

2.6.3 LITERALS

A literal is an expression beginning with an equal or a double equal sign depending on whether the value is to occupy one or two words.

Examples:

=O'77700077'	= = 1.2
=A + B - $	= = A'ABCDEF'
=1	= = 1

META places the value of the expression in a literal table. If the value exceeds the specified number of object computer words, META truncates it and flags the error. If the object computer word size is greater than 24 bits, use of a 2-word literal causes truncation because the maximum precision allowed is 48 bits. By using one or more LIT directives (section 4.4.5), the programmer can designate which control sections are to contain literal tables. If the program contains no LIT directive, the literal table is appended to the program section. The address of a literal is the address of the literal table entry relative to the beginning of the control section. Literals with identical expression values are entered into a single literal table only once.

An attempt to place a literal in a numbered common area is flagged as an error; numbered common cannot be preset.

2.7 SET

A set is one or more set elements separated by commas. A set element is an expression, a set name, or a subset. A subset is a set enclosed in brackets.

The NSET directive (section 4.3.3) assigns a set name to a set. Set names can also be assigned through the PROC and FUNC directives (sections 5.1.1 and 5.1.2).

Examples:

`A NSET 1,2`	A is a set of two elements.
`B NSET X+5,A`	B is a set of two elements. The first element is an expression; the second is a set name.

`C NSET [1,2],[[32,33],`
`6],A`
	C is a set of three elements. The first is a subset which is a set of two elements. The second element is a subset which is a set of two elements, the first of which is itself a subset. The third element is a set name.

In the preceding example, the first element of set C could have been written as A.

To refer to a set element, write the name of the set followed by a left bracket and one or more expressions separated by commas and a right bracket. The values of expressions represent the ordinal location of the set element referenced. From left to right, they represent the level of the element in a set containing subsets. To refer to an entire set, write the name of the set.

If the reference is to a nonexistent element, META uses zero.

Example:

The symbol A is defined as the set 5, C, [9, [3,4]]. The set has three elements. The third element [9, [3,4]] contains two elements, the second of which also contains two elements [3,4].

Reference	Element	Value
A	All	5,C, [9, [3,4]]
A[1]	First element of A	5
A[2]	Second element of A	C
A[3]	Third element of A	9, [3,4]
A[3,1]	First element of subset of third element of A	9
A[3,2]	Second element of subset of third element of A	3,4
A[3,2,1]	First element of subset of second element of subset of third element of A	3
A[15,33]	Nonexistent element	0

In the preceding example, if C is a set name for a set consisting of the list elements 7, 8, 6, elements of C could be referred to as follows:

Reference	Element	Value
A[2,1] or C[1]	First element of C	7
A[2,2] or C[2]	Second element of C	8
A[2,3] or C[3]	Third element of C	6

The Meta-Assembler maintains information about a set and its elements together with the symbol defining the set. The programmer can access this information for use by the assembler through attribute function references. For example, the NUM attribute function (section 6.3) supplies the number of elements in the set.

LOCATION CONTROL AND ADDRESSING 3

Meta-Assembler provides location control by making available one absolute and up to 15 relocatable control sections, each with an associated location counter. The counters can be incremented in word or byte increments.

3.1 RELOCATABLE ADDRESSES

A relocatable address is either a word address (mode 9) or a byte address (mode 11). Mode is specified implicitly by the directive. Word-oriented directives cause definition of relocatable word addresses. Byte-oriented directives cause definition of relocatable byte addresses.

A label field symbol is a word address for the following directives.

RES
GEN
GEND
TEXT
TEXTA

Also, literals and control section names are word addresses. A reference to a control section name returns, as a word address, the current value of the location counter in use prior to processing the line. Use of the $ returns the word value of the current location counter prior to processing of the line.

For the following directives, a label field symbol is a byte address.

NOLIST
LIST
SPACING
EJECT
ORG
LIT
RESB
GENB
TITLE
BRIEF
DETAIL
SECP
SECD
SECA
TEXTC
ENTRY
EXT
GOTO
ENDS
TREF
LIBS

A label field symbol on a FORM reference line or a procedure reference line is a byte address. This means that a mnemonic instruction (which is actually a procedure reference) does not cause the counter to be rounded to the nearest word address.

A word-oriented directive that follows a byte-oriented address causes the control counter for the section to be rounded up to the nearest word address. A byte-oriented directive always uses the next available byte.

Use of the $ returns the word value of the current location counter prior to processing the line.

Examples:

```
      UNIT  6,4
A1    RESB  1           Reserve 1 byte
B1    RES   1           Reserve 1 word
      END
```

```
      UNIT  6,4
A2    RESB  1           Reserve 1 byte
B2    RESB  1           Reserve 1 byte
      END
```

RES in the first example causes the control counter to be rounded up to the next word boundary prior to definition of the symbol B1. The control counter is not rounded up in the second example.

```
      UNIT  6,4         Computer word of four 6-bit bytes.
      SECP  PRØG        PROG is control section name.
A     FØRM  6,3,15      FORM defines three fields; 24 bits.
XY    A     1,1,1       Form reference; XY is a byte address.
XM    EQU   PRØG        Control section; XM is a word address.
XK    EQU   $           $ returns XK as a word address.
XZ    EQU   =1          Literal XZ is a word address.
```

RES and RESB are discussed in sections 4.4.6 and 4.4.7.

3.2 LOCATIONS COUNTERS

Location counters are designated 0-15, corresponding to the 16 control sections a programmer can define using SECA, SECD, and SECP directives (chapter 4).

Location counter 0 is reserved for the absolute control section (defined by SECA).

Location counter 1 is reserved for the first program control directive. If the program has no SECP directive defining a program control section name, location counter 1 is still used for the program.

Location counters 2 through 15 are used for either program control sections or data control sections. As META encounters each SECP or SECD directive, it assigns the next available location counter.

DIRECTIVES 4

Directives control the operation of Meta-Assembler much the same as machine mnemonic codes direct the computer. The programmer can use directives to:

- Control the content and format of the Meta-Assembler listing.
- Define word and byte size when the object computer is not a 3300 or 3500.
- Define a symbol and assign it a value or set of values.
- Assign up to 15 relocatable and one absolute location counters for address assignment.
- Generate code to be loaded and executed on the object computer.
- Specify field sizes for the object code.
- Specify that certain symbols are entry points to separately assembled subprograms, or that symbols used within the current subprogram are external to it.
- Repeat or skip source statements conditionally.
- Terminate assembly of a subprogram or group of subprograms.
- Define a procedure and assign it one or more names for subsequent reference.
- Define a function and assign it one or more names.

4.1 LISTING CONTROL

Through listing control directives, the programmer suppresses portions of the output listing, selects spacing, places a title at the top of any page of the listing, and requests the level of detail he wants to appear in the listing. For all listing control directives, a label is optional; if present, it has the current location counter value.

4.1.1 NOLIST

NOLIST suppresses generation of the output listing until the assembler encounters a LIST directive. The NOLIST line is suppressed from the listing.

Format:

```
label NOLIST comments
```

4.1.2 LIST

LIST causes resumption of the normal assembler listing following a NOLIST directive. LIST appears on the output listing.

Format:

label LIST comments

4.1.3 SPACING

SPACING allows the programmer to select single, double, or triple spacing in the output listing.

Format:

label SPACING exp comments

exp Expression evaluated as 1, 2, or 3 corresponding to single, double or triple spacing, respectively. Otherwise, directive is ignored.

The specified spacing remains in effect until another SPACING directive appears. If no SPACING directives appear in a program, the listing is single spaced.

4.1.4 EJECT

EJECT terminates the current page of the output listing and causes listing to resume at the top of the following page. EJECT is printed as the first line of the next page.

Format:

label EJECT comments

If EJECT is already the first line of a page, it is printed but has no other effect.

4.1.5 TITLE

TITLE causes the current page to be ejected and the TITLE directive line itself to be printed on the first line of the new page. Until another TITLE directive is processed, all succeeding pages begin with this title.

Format:

```
label  TITLE  'characterstring'  comments
```

character string   1-56 characters that appear as title at top of each page of output listing (section 8.1)

4.1.6 BRIEF

BRIEF causes listing of source lines and lines of code generated by data generating directives only. BRIEF remains in effect until a DETAIL directive occurs. The default mode of listing is BRIEF.

Format:

```
label  BRIEF  comments
```

4.1.7 DETAIL

DETAIL causes listing of all lines of code other than library procedure definitions in subsequent LIBS directives (section 5.1.6) and causes listing of procedure expansions. DETAIL remains in effect until a BRIEF directive is processed. A NOLIST directive takes precedence over a DETAIL directive.

Format:

```
label  DETAIL  comments
```

4.2 OBJECT MACHINE DEFINITION (UNIT)

The Meta-Assembler running on a Control Data 3300 or 3500 Computer System to assemble programs for other computers must have certain information about the object computer to generate the proper binary information. The UNIT directive defines the byte size and word size of the object computer. Word size of the object computer must not be less than 8 bits nor greater than 48 bits.

Format:

label UNIT exp_1,exp_2 comments

label Optional

exp_1 Evaluatable nonrelocatable expression defining the byte size of the object computer in bits. During assembly, the location counter is incremented by 1 for each exp_1 bits.

exp_2 Evaluatable nonrelocatable expression specifying the number of bytes per word.

In the absence of a UNIT directive, META uses the host computer unit size of 6 bits per byte and 4 bytes per word. Binary output is in the form acceptable to the 3300/3500 MASTER relocatable loader.

UNIT, if used, must precede all lines of code other than listing control directives and comment cards. Use of UNIT causes binary output to be in the alternate form (appendix C).

4.3 SYMBOL AND SET DEFINITION

A symbol that appears in the label field of an EQU or RDEF directive has a defined value. Whenever the symbol is used in an expression, this defined value rather than the address of the symbol is used in evaluating the expression.

A symbol that appears in the label field of an NSET directive or the label field, command field (as a modifier), or the operand field of a PROC or FUNC directive (chapter 5), becomes the set name for a list of set elements. Whenever the subscripted set name is used in an expression, the value of the set element is used in evaluating the expression.

4.3.1 EQU

EQU assigns the value and attributes of the operand field expression to the label field symbol.

Format:

```
label EQU exp comments
```

The label field must contain a symbol. A symbol defined by EQU cannot be redefined later in the program.

Example:

```
BB EQU 7
```

4.3.2 RDEF

RDEF assigns the value and attributes of the operand field expression to the symbol or set element named in the label field.

Format:

```
label RDEF exp comments
```

The label field must contain a symbol or a set element reference. The value and attributes assigned to this symbol or set element remain in effect until an RDEF with an identical label field symbol or set element is processed or until an RPT (section 4.7.1) with an identical label field symbol is processed. If the operand field is blank, the symbol or set element has a value of 0.

Example:

A	EQU	15	A has value of 15
B	RDEF	$	B has value of current location counter
C	RDEF	A+3	C has value A + 3, or 18
A	EQU	16	Illegal; A is doubly defined
B	EQU	B+2	Illegal; B is doubly defined
C	RDEF	C+2	Legal; C changed from 18 to 20
D	EQU	0	D has value 0
D	RDEF	5	Illegal; D may not be redefined
E	NSET	3,5	Define set E
E[2]	RDEF	6	Redefine element two of set E

4.3.3 NSET

NSET assigns the label field symbol as the set name of the operand field set. The label field must contain a symbol which is the name by which the set or set elements can be referenced. If the operand field is blank, the set consists of one element which has a value of 0.

Format:

```
label  NSET  set  comments
```

Example:

```
A      NSET  3,4,$
B      NSET  5,[6,7]
C      NSET  8,A
D      NSET  *DEFINE SET D
A[2]   RDEF  9
```

A is a set of three elements.

B is a set of two elements, the second of which is a set of two elements.

C is a set of two elements, the second of which is a set of three elements.

D is a set of one zero element.

A[2] is redefined to be 9 in the last line of the examples; thus the final set A is defined as though the following had been written.

```
A  NSET  3,9,$
```

An entire set can be redefined through use of NSET.

Example:

```
A  NSET  1,2
A  NSET  2,3,4
```

4.3.4 FORWARD REFERENCES

A forward reference is a reference to a symbol or set element before it is defined. The Meta-Assembler processes forward references in two passes. On the first pass, a reference to a symbol before it is defined is not given a value; a reference to the symbol after it is defined is given the most recently assigned value. On the second pass, the forward reference is given the most recent value assigned.

An expression cannot contain a forward reference if:

1. The value affects location counting.
2. The undefined symbol is defined subsequently by an EQU directive that contains a second forward reference.
3. The undefined symbol or set element is not defined subsequently.
4. The expression is not evaluatable.

A forward reference to a symbol or set element redefined subsequently by RDEF or NSET directives that contain forward references yields the final value assigned to the symbol or set element.

Examples:

Legal use of forward reference	First Pass	Second Pass
`A EQU B`	A undefined	A = 2.5
`B EQU 2.5`	B = 2.5	B = 2.5
`AA EQU XX`	AA undefined	AA = 7
`XX RDEF YY`	XX undefined	XX = 2
`BB EQU XX`	BB undefined	BB = 2
`YY RDEF 2`	YY = 2	YY = 2
`XX RDEF 7`	XX = 7	XX = 7
`GEN AA[2]`	AA[2] undefined	AA[2] = 9
`BB EQU 8`	BB = 8	BB = 8
`AA NSET 1,BB,7`	AA[1] = 1; AA[2] = 8; AA[3] = 7	AA[1] = 1; AA[2] = 8; AA[3] = 7
`GEN AA[2]`		
`AA[2] RDEF 9`	AA[2] = 9	AA[2] = 9

Replacing the last line of the previous example with the following would achieve the same result.

```
AA      NSET  1,9,7
```

Illegal Forward References:

```
        GEN  AA
AA      EQU  BB
BB      EQU  10
```

GEN directive contains forward reference to AA which is defined by an EQU containing a forward reference.

```
AA      RDEF  BB
BB      EQU   7
        RES   AA
```

Value affecting location counting must be defined on first pass.

RES affects location counting.

```
AA      EQU  BB
BB      EQU  10
        RES  AA
```

Value affecting location counting must be defined on first pass.

RES affects location counting.

```
AA      NSET  1,BB,7
BB      EQU   8
        RES   AA[2]
```

Value affecting location counting must be defined on first pass.

RES affects location counting.

4.4 LOCATION CONTROL

Meta-Assembler provides one absolute and 15 relocatable control sections, each of which has an associated location counter. Any program can use one or more control sections.

Meta-Assembler directives described in this section assign names to control sections and address values to location counters.

4.4.1 SECP

The first SECP directive defines a program control section.

Format:

```
label SECP sym comments
```

label — Optional; if present, label has the value of the location counter after the SECP directive is processed.

sym — 1-8 character name of program control section (subprogram name). A reference to sym later in the program returns the current value of the associated location counter.

After the first SECP naming a specific sym, successive SECP directives using this sym indicate that the code that follows is an extension of the previously declared program control section. A programmer coding for MASTER may use only one program control section; any additional SECP directive naming a new sym is flagged with an informative D error.

4.4.2 SECD

The first SECD in a program defines a blank common, numbered common, or labeled common control section.

Format:

```
label SECD sym,exp comments
```

label — Optional; if present, label has the value of the location counter after the SECD directive is processed.

sym — Optional; name of control section defined or referenced.

zero or blank	Control section defined or referenced is zero or blank common.
1-8-character symbol	Control section is labeled common block.
1-4 decimal digits	Control section is numbered common block..

For the 3300/3500 relocatable output, if sym is blank or 0, the block name is 1∧∧∧∧∧∧∧ for chapter 1 and 2∧∧∧∧∧∧∧ for chapter 2. For other than the 3300/3500, the block name depends on the item type (appendix C).

If sym is a symbol, a reference to the symbol later in the program returns the value of the associated location counter.

exp Optional; if execution is under 3300/3500 MASTER, exp is an evaluatable expression with value 1 or 2 designating the chapter to which the section is assigned. If exp is absent, chapter one is assigned.

Each new sym on an SECD directive causes creation of a new control section starting at relative address 0. If a sym appears on a subsequent SECD directive, exp is ignored and code following the subsequent SECD directive down to the next location control directive is an extension of the previously declared control section.

4.4.3 SECA

A program can have an absolute control section declared by a SECA directive.

Format:

```
label SECA sym comments
```

label Optional; if present, label has the value of the location counter after the SECA is processed.

sym 1-8 character name of the absolute control section. A subsequent reference to sym returns the current value of the absolute location counter.

Any SECA directive after the first one in a program indicates that the code following it is an extension of the originally defined absolute control section. If SECA is preceded by an ORG directive setting the absolute location counter, the code following the SECA extends the absolute control section.

SECA cannot be used when coding for MASTER.

4.4.4 ORG

ORG sets the specified control counter to a specified address.

Format:

```
label ØRG exp comments
```

label Optional; if present, the label has the value of the location counter after the ORG directive is processed.

exp Evaluatable expression. The expression indicates the control counter to be selected and the address to which it is to be set. Lines of code following ORG are placed in the control section indicated.

Examples:

`   SECP ALPHA`	Defines program control section ALPHA.
`   .`	
`   .`	
`   SECD CØMM,1`	Specifies labeled common block of COMM in chapter one.
`   .`	
`   .`	
`   SECD 25`	Specifies numbered common block. Chapter one is implied.
`   .`	
`   .`	
`   ØRG ALPHA`	Specifies resumption of program control section. (Here, ORG has the same effect as SECP ALPHA).
`   .`	
`   .`	
`   SECD CØMM`	Specifies resumption of labeled common block COMM.
`   :`	
`D  RES 1`	Location within COMM.
`   ØRG 50`	Selects absolute location counter and sets its value to 50.
`   :`	
`   SECD`	Specifies blank common. For the 3300/3500 the block name is $1_{\wedge\wedge\wedge\wedge\wedge\wedge\wedge}$; otherwise, the block name is 00000000.
`   .`	
`   .`	
`   ØRG D`	Selects the location counter for COMM and sets the location counter value to the value of D.
`   .`	
`   .`	
`   SECP ALPHA`	Specifies resumption of program control section ALPHA.

4.4.5 LIT

LIT designates the control section in which literals are to be placed.

Format:

```
label LIT sym comments
```

label — Optional; if present, label has the value of the location counter.

sym — Name of a previously defined control section.

META places literals (section 2.6.3) in the control section specified by a LIT directive, regardless of which control section contains the reference, until it encounters another LIT directive designating a different control section for literals. In any given literal table, only one entry is made for identical literals. However, a literal table can have entries that duplicate entries in other literal tables. A literal results in the generation of object code.

In the absence of a LIT directive, literals are appended to the first program control section.

4.4.6 RES

RES adds the value of the expression in the operand field to the current location counter value as a word increment to reserve storage.

Format:

```
label RES exp comments
```

exp — Evaluatable nonrelocatable expression (must not contain a forward symbolic reference or reference to an externally defined symbol).

Examples:

`A  RES 2`	Increment location counter by two words.
`   RES 10`	Increment location counter by ten words.
`   RES -5`	Decrement location counter by five words.

4.4.7 RESB

RESB adds the value of the expression in the operand field to the current value of the location counter as a byte increment to reserve storage.

Format:

```
label RESB exp comments
```

exp Evaluatable nonrelocatable expression by which to increment the counter.

Examples:

`B RESB 16`	Increment location counter 16 bytes.
`C RESB $-B`	Increment 16 more bytes.

4.5 DATA GENERATION

Data generating directives define data formats and generate words or bytes of information to be loaded into the computer at execution time.

4.5.1 GEN

GEN places the values of expressions in the operand field set in successive words, one word for each expression.

Format:

```
label GEN set comments
```

set Set of expressions to be generated. A set of sets is not permitted.

Examples:

For the following examples, the object computer word size is 24 bits.

`A GEN 5,C'ABCD'`	Generate two words, the first containing 5, the second containing the internal BCD representation of ABCD.

```
B  NSET  5,C'ABCD'
A  GEN   B
```

Results in the same values as the above.

```
C  NSET  5,[6,7]
D  GEN   C
```

Illegal; the set in the GEN line must not contain sets.

```
C  NSET  5,[6,7]
D  GEN   C[1],C[2,1],C[2,2]
```

Generate three words, containing 5, 6, and 7.

```
E  GEN   2.5
```

Illegal; values must be single precision.

```
   EXT   F
   GEN   F,4
```

Legal; reference to external symbol.

4.5.2 GEND

GEND generates the values of expressions in the operand field set, two object computer words per expression. Maximum precision for a value is 48 bits. If the object computer word size exceeds 24 bits, META truncates the value to 48 bits and flags the error.

Format:

```
label  GEND  set  comments
```

label   Optional

set   Set of expressions to be generated. A set of sets is not permitted.

Example:

In the following example, the object computer word size is 24 bits.

```
A  GEND  2.4,25,C'ABC'
```

The code generates six words. The first two words contain the floating-point representation of 2.4. The next two contain the binary integer representation of 25. The last two words contain the internal BCD representation of ABC right-justified with leading zeros.

4.5.3 GENB

GENB evaluates the values of expressions in the operand field set and places the values in successive bytes. If the value of an expression exceeds the byte size specified in the UNIT directive, META truncates the value to the byte size and flags the error.

Format:

```
label GENB set comments
```

label   Optional

set   Set of expressions to be placed in successive bytes. A set of sets is not permitted.

Example:

For the following example, the object computer byte size is 6 bits.

```
JØE GENB 5,9,63,14,-2
```

The above code generates five 6-bit bytes. The last byte contains the one's complement of -2 truncated to 6 bits (111101).

4.5.4 FORM

FORM defines a data format by specifying field sizes, left to right, in one or more object computer bytes.

Format:

```
label FØRM set comments
```

label   Required; label is the name referring to FORM

set   A set of expressions, each of which defines a field size in bits. A set of sets is not allowed.

Examples:

For the following examples, the object computer byte size is 6 bits and the object computer word size is 24 bits.

```
I       NSET   6,2,1,15
WØRD    FØRM   24            One field, four bytes
WØRD    FØRM   48            One field, eight bytes
CHARS   FØRM   6,6,6,6       Four fields, four bytes
ADR     FØRM   7,17          Two fields, four bytes
INST    FØRM   I             Four fields, four bytes
```

To refer to a format defined by a FORM directive, place the label of the FORM directive line in the command field of a line. Supply a set of expressions, corresponding to the fields, in the operand field of the referencing line. A form reference generates code starting with the next available byte.

A label on the line referring to a FORM directive has the value of the location counter prior to processing the line. If a value exceeds the specified field size or if the field size exceeds 48 bits, high-order bits are truncated and an error flag is generated. For a negative value, the one's complement of the absolute value is used unless the value is in BCD decimal mode. For a BCD decimal value, the sign is inserted in the leftmost bit of the least significant character position of the field.

If the field contains a 6-bit character type value and the field size is not a multiple of 6-bits, the characters are placed in the rightmost bits of the field with the leftmost extraneous bits zero.

References to FORM directives can be circular.

Examples:

```
WØRD    FØRM   24
        WØRD   $+3
```

Generates a single word with value $+3 right justified in the 24-bit field.

```
WØRD    FØRM   48
        WØRD2  1.59
```

Generates the 48-bit floating-point value of 1.59.

```
CHARS  FØRM   6,6,6,6
       CHARS  C'A',63,15,Ø'12'
ADR    FØRM   7,17
```

Generates one word containing the following octal value:

21	77	17	12

```
A      NSET   0,BYT($)
       ADR    A
```

Generates a word with zero in the leftmost 7 bits and the byte value of the location counter in the rightmost 17 bits.

```
I      NSET   6,2,1,15
INST   FØRM   I
       INST   12,0,0,$+2
```

Generates a word with value 12 right adjusted in the leftmost 10 bits, zeros in the next 3 bits, and the current word address plus 2 in the rightmost 15 bits.

The following example illustrates circularity of forms.

```
   UNIT  6,4
F  FØRM  2,3,1
   F     1,1,1,2,2,0      Generates 2 bytes
   F     1,1,1,2,2        Generates 2 bytes identical to
                          last two
G  FØRM  5
   G     2,2,2,2          Generates 4 bytes filling last
                          with zeros
   G     3,3,3,3,3,3      Generates 5 bytes
```

The 4 bytes generated by G are:

04	10	20	40

The 5 bytes generated by G are:

06	14	30	61	43

In the following example, BCD characters XY are to be stored in a 19-bit field.

```
A  FØRM  19
   A     L'XY'
```

BCD characters X, Y, and blank are placed in the rightmost 18 bits of the field. The leftmost bit is 0.

4.5.5 TEXT

TEXT generates an integral number of object computer words containing the specified BCD character string.

Format:

```
label TEXT 'string' comments
```

The last word is padded with blanks as needed. If the object computer word size is not a multiple of 6 bits, as many characters as fit are placed in each word, right adjusted with upper bits zero.

4.5.6 TEXTC

TEXTC is identical to TEXT except that the BCD character string generated is placed in consecutive words without padding the last word.

Format:

```
label TEXTC 'string' comments
```

TEXTC generates code starting with the next available byte.

4.5.7 TEXTA

TEXTA generates 8-bit ASCII characters in the same way TEXT generates BCD characters. Padding of the last word, if needed, is with the internal representation of ASCII blanks.

Format:

```
label TEXTA 'string' comments
```

4.6 PROGRAM LINKING

The directives ENTRY and EXT do not define symbols, but either classify symbols defined within the subprogram as being known outside the subprogram, or classify symbols referenced in a subprogram as being defined outside of the subprogram.

4.6.1 ENTRY

The ENTRY directive specifies which symbols defined may be referenced by subprograms compiled or assembled independently. That is, ENTRY directives list entry points to the current subprogram.

Format:

label ENTRY $sym_1, sym_2, ...sym_n$ comments

sym_i Entry point symbols, 1-8 BCD characters

4.6.2 EXT

The EXT directive lists symbols which are defined as entry points in independently compiled or assembled subprograms, but for which references appear in the subprogram being assembled.

Format:

label EXT $sym_1, sym_2, \}...\{sym_n$ comments

sym_i External symbols, 1-8 BCD characters

4.7 REPEAT AND SKIP

Source statements can be processed repeatedly or skipped conditionally through use of the RPT and GOTO directives.

4.7.1 RPT

RPT specifies processing a portion of code a given number of times.

Format:

label RPT exp, linid comments

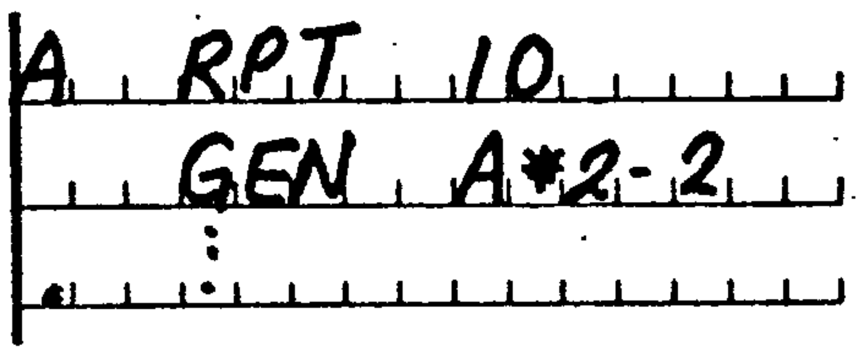
label	Optional; if present, the original value is 0. The value of the label is tested and incremented by 1 prior to each processing of the lines of code, to a final value that is the value of exp.
exp	Absolute evaluatable nonrelocatable expression (contains no forward or external references) indicating the number of times the following lines are to be processed. If exp is less than or equal to 0, the following lines are not processed and the RPT acts as a skip.
linid	Label of the last line to be processed by this RPT. If linid field is missing, one line is processed.

RPTs may be nested to a level of at least six and possibly more depending on available table space. Space not required for processing functions and procedures could be used for additional levels of RPTs. Processing of repeated statements is from innermost to outermost. Every inner RPT range must lie totally within the range of the next outer RPT.

The programmer can redefine the RPT label within the repeated statements to terminate a repetition prematurely.

Examples:

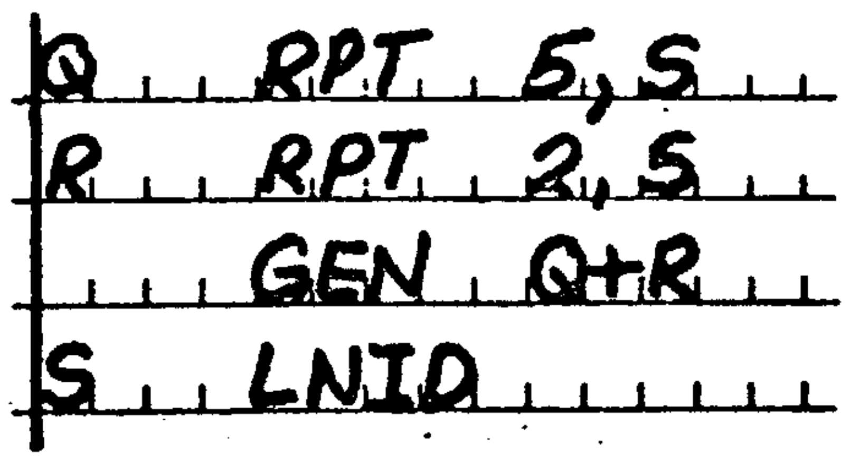
The following sequence generates a 10-word table of even numbers, 0-18. Because linid is absent, only one line is processed.



```
A  RPT  10
   GEN  A*2-2
      :
```

Generates one word for each value 0, 2, 4, 6, 8,...,18

The following example illustrates two levels of repeats; the nested repeats produce 10 words.



```
Q  RPT  5,S
R  RPT  2,S
   GEN  Q+R
S  LNID
```

Q = 1, R = 1	2	Q = 3, R = 2	5
Q = 1, R = 2	3	Q = 4, R = 1	5
Q = 2, R = 1	3	Q = 4, R = 2	6
Q = 2, R = 2	4	Q = 5, R = 1	6
Q = 3, R = 1	4	Q = 5, R = 2	7

In the following example, lines 5-8 are processed three times.

```
S     NSET   0,0,0           1
A     EQU    4               2
B     EQU    5               4
C     RPT    B-A+2,D         5 \
S[C]  RDEF   A+C-1           6 |
      RPT    S[C]=B          7 |  repeat range
      GEN    S[C]*S[C-1]     8 /
D     LNID
```

In this example, the elements of set S are initially zero. On the first processing of lines 5-8, C is 1, and S[1] is redefined as A + C - 1, or 4. On the second RPT directive, the test S[1] = B is not true (0); the GEN line is skipped. When lines 5-8 are repeated, C is 2 and S[2] is redefined as 5. The test S[2] = B is true (1) so the GEN line is processed; it generates one word with a value of 20. On the final iteration, C is 3, S[3] is redefined as 6, and the test S[3] = B is not true (0); the GEN statement is skipped. Without the use of repeats, this example would be:

```
S     NSET   0,0,0
A     EQU    4               A = 4
B     EQU    5               B = 5
S[1]  RDEF   A               S[1] = 4
S[2]  RDEF   A+1             S[2] = 5
      GEN    S[2]*S[1]       |Generate 20
S[3]  RDEF   A+2             S[3] = 6
```

4.7.2 GOTO

GOTO specifies a conditional skip.

Format:

label GØTØ exp,$linid_1,\ldots,linid_n$ comments

exp	Evaluatable nonrelocatable expression
$linid_i$	Line identifiers defined as labels on lines following GOTO.

Expression exp is evaluated and used as an index to the list of line identifiers. The line containing the label identified by the indexed line identifier is the next line assembled. For example, if exp has value 2, the second line identifier is the label of the next line to be assembled. If 0≥ exp> n, where n is the number of line identifiers, assembly continues with the next line.

Example:

For the following lines of code, since (B-A)*B = 2, the next line assembled after GOTO is the line identified by the second line identifier, the line labeled BILL. Lines between GOTO and the line labeled BILL are skipped.

```
A     EQU  1
B     EQU  2
      GØTØ (B-A)*B,JØE,BILL,BØB
          :
JØE   LDA  VAL1
          :
BILL  LDA  VAL2
          :
BØB   LNID
```

4.7.3 LNID

LNID inserts a dummy label for line identification purposes. The label has no value and is not entered in the Meta-Assembler symbol table. As long as no ambiguity exists, the same label may appear on more than one LNID line, or on any non-LNID line, or on both LNID and non-LNID lines.

Format:

```
label  LNID  comments
```

There is no operand; comments can be entered immediately after the command without the use of an asterisk.

LNID is particularly useful for defining the range of an RPT, since the use of normal labels may sometimes result in duplicate symbol definitions.

4.7.4 RPT AND GOTO PROCESSING

When META encounters an RPT directive, it compresses lines of code within the RPT range by removing comments and redundant blanks, and stores the lines in an internal table of definitions.

In the process of saving the lines of code within the RPT range, the assembler examines the command field of each line to ensure that the RPT range does not include an END or FINIS directive. The assembler also recognizes procedure and function definitions (chapter 5) which are within the range of an RPT.

When a procedure or function definition appears within an RPT range, label field symbols within the procedure or function definition are local to the procedure or function definition and are not considered in determining the RPT range.

Example:

```
   RPT    1,A
   PRØC
X  NAME
A  RDEF   1                 Not end of RPT range
   ENDS
A  LNID                     End of RPT range
```

A GOTO directive may appear within the range of an RPT. The object of the GOTO may be either within or outside the range of the RPT. If the object of a GOTO is outside the range of an RPT, the RPT is terminated.

Within a procedure or function definition, the object of a GOTO or an RPT must be within the procedure or function definition, and must be at the same level as the GOTO or RPT directive line. (Level of definition is discussed in section 5.4.)

Examples:

```
   PRØC
A  NAME
     :
   RPT    5,B               B is within the procedure definition and
     :                      is at the same level as the RPT
B  LNID                     directive line.
     :
   ENDS
```

```
CL   PRØC
A    NAME
       :
     GØTØ  1,B
       :
     ENDS
       :
B    LNID
     A        1,2,3
```

Illegal; B is not in the procedure definition. If the procedure is referenced, the GOTO is terminated on encountering ENDS.

4.8 ASSEMBLY TERMINATION

The directives END and FINIS specify the end of a subprogram and of a set of subprograms, respectively.

4.8.1 END

END terminates a subprogram. The symbol in the operand field is optional but, if present, must be a symbol of eight characters or fewer declared as an entry point in some subprogram. The symbol specifies the symbolic location at which execution is to begin.

Format:

```
label  END  symbol  comments
```

4.8.2 FINIS

FINIS causes termination of assembly. Normally, an assembly is a set of subprograms, each of which ends with an END directive. The FINIS directive should follow the END directive for the final subprogram.

Format:

```
label  FINIS  comments
```

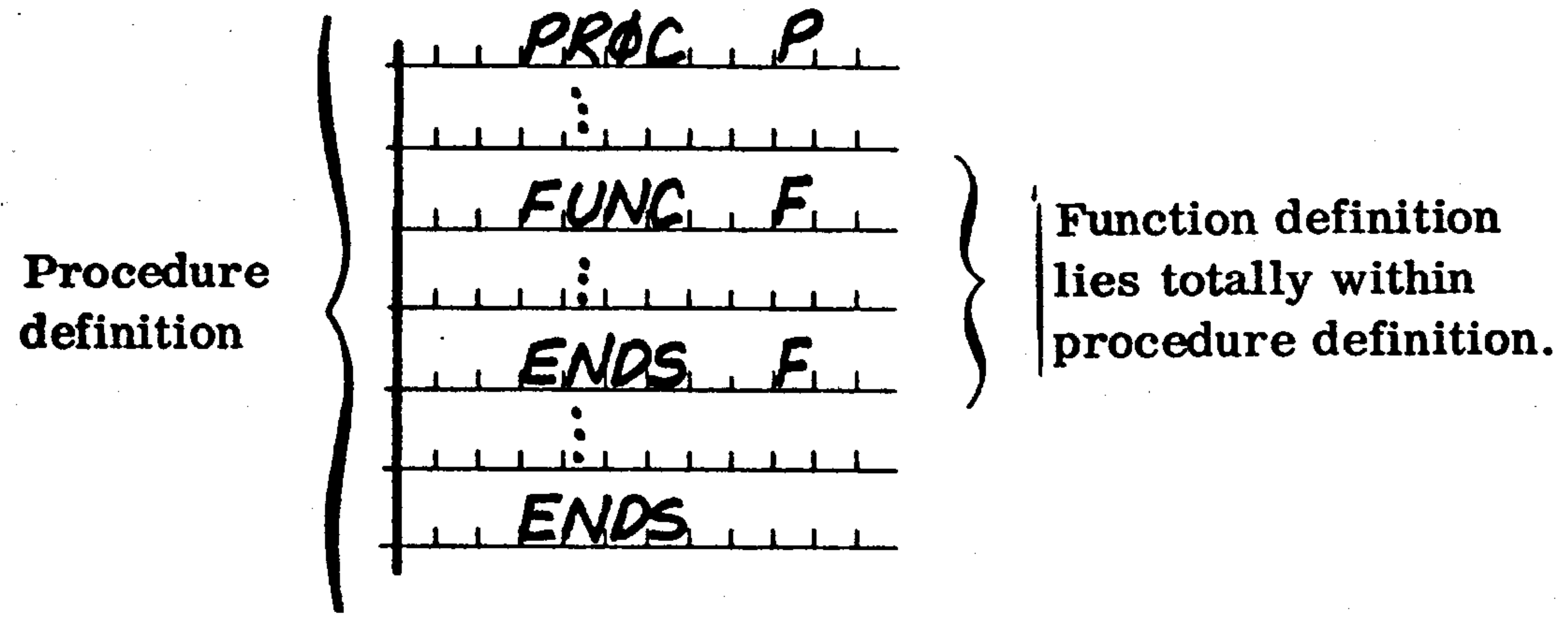
PROCEDURES AND FUNCTIONS 5

Procedure and function definitions are bodies of code resembling subroutines but processed during assembly rather than object-time execution. They provide programmers with a means of conditionally generating sequences of code. A procedure reference consists of the appearance of the procedure name in the command field of a statement; the referenced procedure generates object code each time it is referenced according to parameters supplied with the reference. A function reference consists of the function name and its argument appearing in a statement; the function generates a value or set of values dependent on the argument.

A procedure or function definition begins with a PROC or FUNC directive, respectively, and terminates with an ENDS directive. The definition must precede a reference to it.

A function or procedure definition can wholly contain other definitions and references to yet other definitions. Such definitions are nested. Each nested definition is considered one level higher than the definition that contains it. Nesting can occur to a level of 14. Levels of nesting are discussed more fully in section 5.4.

Examples of nesting:



```
  PRØC  P
   :
  FUNC  F
   :
  ENDS  F
   :
  ENDS
```

Procedure definition
Function definition lies totally within procedure definition.

If the procedure being defined contains a forward reference to a locally defined symbol, proper data generation cannot result in a single pass. An optional parameter on the PROC directive indicates a two-pass procedure to permit local forward references. The Meta-Assembler then makes a preliminary symbol defining pass through the procedure similar to the first assembly pass of a program.

5.1 DIRECTIVES

META provides directives specifically related to use of procedures and functions.

5.1.1 PROC

A PROC directive declares the beginning of a procedure definition.

Format:

```
label  PRØC,setname1  setname2,exp  comments
```

label	Optional; if present, label becomes the name of sets given on NAME lines in the procedure.

$setname_1$	Optional; set name that identifies the set in the command field of the procedure reference. This setname is in the command field and is separated from PROC by a comma.

$setname_2$	Optional; set name that identifies the set in the operand field of the procedure reference.

exp	Optional; if value of expression is nonzero, procedure requires two passes. Note: This option requires core for expression building and causes a reduction in assembly speed. It should not be used unless the procedure contains a forward reference.

When defining a two-pass procedure, the user should take care to prevent the inadvertent doubling of expression values. For the following lines of code, after a reference to procedure TWO, A has value 1 because it was initialized to zero each pass; B has value 2 because it was not initialized and was incremented once each procedure pass.

Example:

```
       PRØC   P,2        Operand field set has name P
TWØ    NAME              TWO is entry name to procedure
A$     RDEF   0
B$     RDEF   B+1
A$     RDEF   A+1
       ENDS
B      RDEF   0
       TWØ               TWO is reference to procedure
```

5.1.2 FUNC

FUNC declares the beginning of a function definition.

Format:

```
label FUNC setname comments
```

label — Optional; if present, label becomes the name of the sets appearing on NAME lines in the function when the function is referenced.

setname — Setname becomes the name of a set of parameters passed to the function.

A function should not include directives that generate code or affect counters.

Example:

```
      FUNC                          Begin function FX
FX    NAME
A     NSET  4,5,6
      :
      :
      ENDS  [A[1],A[2]]             End function FX
      :
      :
B     NSET  FX()                    Set B has two elements,
                                    4 and 5
```

5.1.3 NAME

NAME directives define entry names by which a function or procedure can be referenced. They must be between the PROC or FUNC directive and its associatec ENDS directive. The label field symbol of the NAME directive is used as tbe command field of the statement referencing the function or procedure. Any number of NAME directives can appear within a definition.

Format:

```
label NAME set comments
```

label	Required symbol; an entry name to the procedure or function.

set	Optional set of expressions or sets that are to be associated with this NAME. The name associated with this set is in the label field of the PROC or FUNC directive preceding this NAME. If the PROC or FUNC label field contains a set name and the operand field of the NAME directive is blank, the set consists of one element having a value of 0.

Example:

```
E         PRØC,M   JØE
ENTER1    NAME     12,I
ENTER2    NAME     13,J
:         :        :
```

The procedure can contain references to a set named E. When the procedure is referred to by name ENTER1, elements 12 and I are assigned set name E as if the following line had been written:

```
E  NSET  12,I
```

If, instead, the procedure is referred to by name ENTER2, elements 13 and J are assigned set name E as if the following line had been written:

```
E  NSET  13,J
```

5.1.4 ENDS

ENDS terminates a procedure or function definition.

Format:

```
label ENDS exp comments
```

When ENDS terminates a procedure definition, META expects no operand field entry. However, an asterisk must precede comments.

When ENDS terminates a function definition, exp is either an expression that defines the function value, a set name for a set of values, or set elements enclosed by brackets. A function reference that returns a set or a set name may be used instead of a subset. That is, to return a set, exp must be one of the forms:

(set)

setname

func(set)

Examples:

```
  PRØC  P                       Begin outer procedure.
   :
  PRØC  Q                       Begin inner procedure.
   :
  ENDS                          End inner procedure.
   :
  ENDS                          End outer procedure.
  FUNC  F                       Begin first function.
   :
  ENDS  F[1]+F[2]               End first function.  The
   :                            value of the function is
   :                            the sum of the first two
                                values of the calling set.
  FUNC  FF                      Begin second function.
   :
A NSET  FF[1]*FF[2],FF[3]       End second function.  The
  ENDS  A                       function returns a set of
                                values rather than a single
                                value.
```

5.1.5 TREF

The TREF directive terminates processing of a reference to a procedure or function definition before the ENDS directive.

Format:

```
label  TREF  exp  comments
```

For a function reference, control returns to the statement containing the reference and passes to it the value or set defined by the expression in the operand field of the TREF. Exp is either an expression that defines the function value, a set name for a set of values, or set elements enclosed by brackets. A function reference that returns a set or a set name may be used instead of a subset. That is, to return a set, exp can be one of the forms:

(set)

setname

func(set)

Example:

```
N       PRØC,I  A
IDENT   NAME
I$      EQU     1
F$      FØRM    6,1,2,15
        SECP    SYM(A[1])
        TREF                              Terminate Reference
LDA     NAME    Ø'20'
STA     NAME    Ø'40'
        F       N[1],I[1],A[2],WRD(A[1])
        ENDS
```

A reference to IDENT terminates at the statement before the LDA NAME directive. References to the procedure by names LDA or STA terminate at the ENDS directive.

5.1.6 LIBS

The LIBS directive enables the user to retireve procedure definitions from a file. It must not appear within a procedure or function definition.

Format:

```
label  LIBS  L'dsi', sym1, ..., symn  comments
```

label — Optional symbol.

dsi — Data set identifier of the file containing the procedure definitions. This file, if it is not the system library file, must have been allocated and opened through use of MASTER control cards before META executes (3300/3500 MASTER Reference Manual Pub. No. 60213600). Procedures are searched for by name; they can be in any order on the file. If no dsi is given, META uses ΛΛΛΛ.

sym_i — Label field symbol of each NAME directive line for every outer procedure to be retrieved.

Function definitions can be obtained from a file through nesting of definitions and through externalization (section 5.4).

Procedures are stored on the system library by GLIB, the MASTER library generation program, and can be placed on some other file through use of XFER, the MASTER transfer routine (MASTER Reference Manual). They cannot be on an auxiliary library.

Examples:

The following procedure definition appears in a procedure library on file DSI.

```
      PRØC
P1    NAME
      FUNC
F1$   NAME
        :
      ENDS
      FUNC
F2$   NAME
        :
      ENDS
      ENDS
```

Procedure P1 is obtained by LIBS as follows:

```
  LIBS  L'DSI',P1
```

After P1 has been obtained, function names F1 and F2 are defined by writing P1 as a command field entry.

```
  P1
```

A procedure with names A and B is on the system library, *LIB.

```
   PRØC
A  NAME
    .
    .
    .
B  NAME
    .
    .
    .
   ENDS
```

By using the following LIBS directive, both A and B are defined and may be referenced. The user needs to specify only the first procedure name to obtain the entire definition.

```
   LIBS  L'*LIB',A
```

If a user has no use for the A entry name, he can save core during assembly by obtaining only the portion of the definition following the B entry name.

```
   LIB   L'*LIB',B
```

5.2 DEFINITION PROCESSING

When META encounters a procedure or function definition, it compresses the lines of code representing the procedure by removing comments and redundant blanks, and stores the lines in core.

Meta-Assembler removes the NAME lines of outer level procedures and functions and inserts the labels of these lines into the symbol table. These labels are procedure or function entry names, and contain the location of the definition and the values of any sets associated with the NAMEs.

Entry names of inner definitions are not processed. Meta-Assembler stores these in the procedure definitions area as part of the lines of code comprising the definition. When procedure or function definitions are nested, entry points to the inner definitions are not known until the outer procedure is referenced. META does not save outer level PROC and FUNC lines, but instead creates a PROC or FUNC symbol table entry for each such line.

When an outer procedure or function is referenced, META processes only PROC, FUNC, NAME, and ENDS lines of the next level of procedures or functions. Unless the inner procedure name is externalized (section 5.4) subsequent reference to an inner procedure may occur only within the next outer procedure.

Each procedure and function definition may contain several NAME directive lines. The position of a NAME directive determines the first line of code to be processed when the procedure is referenced.

Example:

```
WL  PRØC  CL
X   NAME
A   RDEF  1
       :
Y   NAME
B   RDEF  2
       :
    ENDS
```

If the procedure is called by name X, the first line of code processed is:

```
A   RDEF  1
```

If the procedure is called by name Y, the first line processed is:

```
B  RDEF  2
```

The position of NAME directive lines within a procedure affects LIBS directive processing. If the following line is written the entire procedure is retrieved from the file.

```
  LIBS  L'dsi',X
```

If LIBS is written as below, the only line preceding the NAME line with label Y retrieved is the PROC directive line.

```
  LIBS  L'dsi',Y
```

5.3 REFERENCING

To refer to a procedure, write the label of any NAME directive line in the definition as a command. The label field can be blank or can contain a symbol that is assigned the value of the current location counter. To supply parameters to the procedure, place a set in the operand field of the procedure call line, append a set to the procedure name in the command field, or do both. Within the procedure definition, the sets are referred to as if they were defined by NSET directives. If set names are provided in the command and operand fields of the PROC directive or the operand field of the FUNC directive and the corresponding field of the procedure or function reference is blank, the set used consists of one zero element.

Example:

```
E       PRØC,M  JØE
ENTER   NAME    12,I
          :
        ENDS
        ENTER,X,Y  A,B,$,[C-3,5]
```

When the procedure is referred to by name ENTER, elements A, B, $, [C-3,5] are associated with name JOE as if the following line had been written.

```
JØE  NSET  A,B,$,[C-3,5]
```

JOE [1] refers to A, JOE [2] refers to B, JOE [3] refers to the value of $ at the time the reference occurs, and JOE [4] consists of a subset of two elements, C-3,5.

Set X, Y has set name M and is referred to as if the following line had been written.

```
M  NSET  X,Y
```

Thus

JOE [3] = $

JOE [4, 1] = C-3

M [2] = Y

The label appearing on the PROC directive line assigns a name to the set in the operand field of the NAME line. In the preceding example, E is the set 12, I.

To refer to a function, write the label of a NAME directive appearing in the function definition followed by an argument enclosed in parentheses. A function reference must include the parentheses.

Example:

```
         FUNC   FU
CQUØT    NAME
EXP      EQU    (FU[1]+FU[2]-1)/FU[2]
         ENDS   EXP
          :
A        EQU    CQUØT(15,4)
```

In the above reference, FU [1] is 15 and FU [2] is 4; A has value (15 + 4 - 1)/4. If the reference had been CQUOT( ), the set FU would have been a single element set with value of zero and would have been illegal because FU [2] is a divisor with value 0.

Parameters are referenced within a function in exactly the same way as they are referenced within a procedure.

A reference to a function that returns a value may appear as an operand in an expression. Reference to a function that returns a set may appear anywhere a set name may appear.

When a procedure is referred to, META forms as many as three sets in the symbol table. The set in the operand field of the procedure reference line, the set appearing in the command field of the procedure reference line, and the NAME directive set associated with the procedure reference.

The set in the operand field of the procedure reference line is evaluated and entered in the assembler symbol table. Its set name is the symbol that appeared in the first operand subfield of the PROC directive line for the procedure. The level of definition of the set is one greater than the level in effect for the procedure reference line.

A set appearing in the command field of the procedure reference line is processed in the same manner as the operand field set of the procedure reference line. The name of this set is the entry in the second subfield of the command field of the PROC directive.

The NAME directive set associated with the procedure reference is treated differently. At the time of procedure reference, the elements of the NAME directive set are already in the assembler symbol table but have no set name. META forms the NAME directive set in the assembler symbol table by copying the elements of the NAME directive set from one point in the symbol table to another and by assigning them the set name (the symbol from the label field of the PROC directive line). The level of definition is the same as for the other two sets previously described.

When META encounters the ENDS line for the procedure, it removes local symbols and sets from the symbol table. Externalized symbols are saved.

Meta-Assembler processes lines of code between a NAME line and a TREF or an ENDS line as if these lines appeared on the source input file. The lines are read from core storage rather than from the source input file. When there is nesting of definition, it also reads PROC, NAME, and ENDS lines from core storage. Again, processing is similar to that for lines on the source input file. Information is extracted from the first encountered PROC line and all associated NAME lines; other lines are skipped until a corresponding ENDS line. Had these lines been on the source input file, the assembler would have saved them. However, since the lines are already in core, it is unnecessary to save them again.

5.4 LEVELS AND LOCAL LABELS

META allows nesting of function and procedure references as well as nesting of definitions. A definition can contain a reference to another procedure and, within that procedure, there can be a reference to still another procedure. Nesting of references, as with nesting of definitions, can continue to 14 levels.

Meta-Assembler recognizes 16 levels of symbol definition. Symbols defined at a given level are always available at the given level and all higher (inner) levels, but cannot be referred to at lower (outer) levels.

Symbols external to the program (i.e., those appearing as operands in an EXT directive) are defined at level 0. Symbols defined in the program but outside of procedures or functions are at level one. Symbols defined within procedures or functions are at level two or higher, the level being raised by one for each nesting of the reference.

Except for labels of NAME directives, which are available to the next outer level, labels within a procedure or function definition are local to the procedure or function; they are not available to outer procedures or to the program.

To make a label defined within a procedure or function available outside that procedure or function, the programmer can append one or more dollar signs to the symbol. Each dollar sign lowers the definition of the symbol one level to a minimum level of 1.

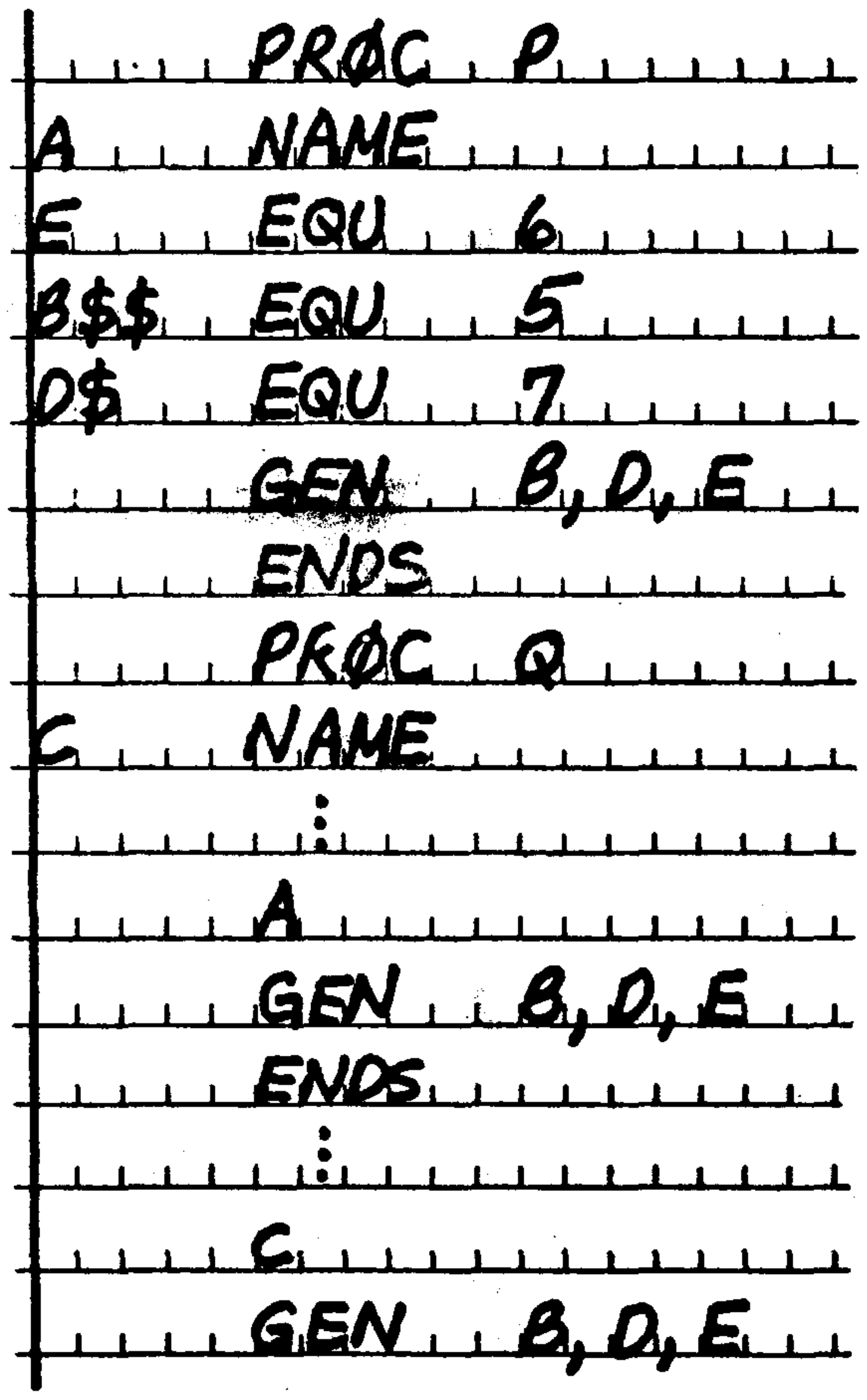
Examples:

A$	EQU	EXP	Define A one level lower.
B$$	EQU	EXP	Define B two levels lower.
SYM$(P[1])	EQU	EXP	Define P[1] one level lower (see section 6.5 for SYM).

Thus, by lowering the procedure level of a symbol definition, the definition is available at a lower level outside the procedure or function.

Example:

In the following example, procedure C is defined at level 2 when referenced by the main program (second line from bottom). Its entry name (C) is known at levels 1 and 2. Within C, a call to procedure A defines A one level higher (level 3) causing its entry name (A) to be known at levels 2 and 3. Label E is local to procedure A. Label B is known at levels 1, 2, and 3. Label D is known at levels 2 and 3. Consequently, labels D and E are not known when they are referenced at level 1 by the GEN directive following the reference to procedure C. If the reference were to A instead of C, A would be defined at level 2 making labels B and D available to the GEN directive.



```
      PRØC  P
A     NAME               A known at levels 3 and 2
E     EQU   6            E local to level 3
B$$   EQU   5            B known down to level 1
D$    EQU   7            D known at levels 3 and 2
      GEN   B,D,E        B,D, and E all known
      ENDS
      PRØC  Q
C     NAME               C known at levels 2 and 1
       :
      A
      GEN   B,D,E        Level 2; E not known
      ENDS
       :
      C                  Level 1; C known
      GEN   B,D,E        Level 1; D and E not known
```

ATTRIBUTE FUNCTIONS 6

In the process of assembling source programs, Meta-Assembler constructs tables of information about elements of the source program. Attribute functions provide the user with information about expressions and sets.

The implicit attribute of a symbol or a set element is its value. Within Meta-Assembler, the value attribute of a symbol is synonymous with the symbol; no further notation is needed to obtain that information.

Example:

Let A and B be defined as follows:

```
A EQU 3
B EQU 4
```

Within META, A*B and 3*4 are identical expressions.

Attribute functions are used to obtain information about attributes other than value. As with a symbolic reference, an attribute function reference results in a value. To refer to an attribute function, write the attribute name followed by an expression or set enclosed in parentheses. An attribute function reference can be an operand in an expression.

6.1 RELOCATION (REL)

The relocation attribute function, REL, returns value zero if the expression within the parentheses is not a value or is an absolute value. If the expression is relocatable relative to a control section origin, REL returns the internal location counter designation (1-15) of the control section containing the expression.

Example:

```
  GOTØ  REL(A)=1,C
```

If A is in control section using location counter 1, go to C.

Assume the program contains only one program control section and that B is an expression in that section.

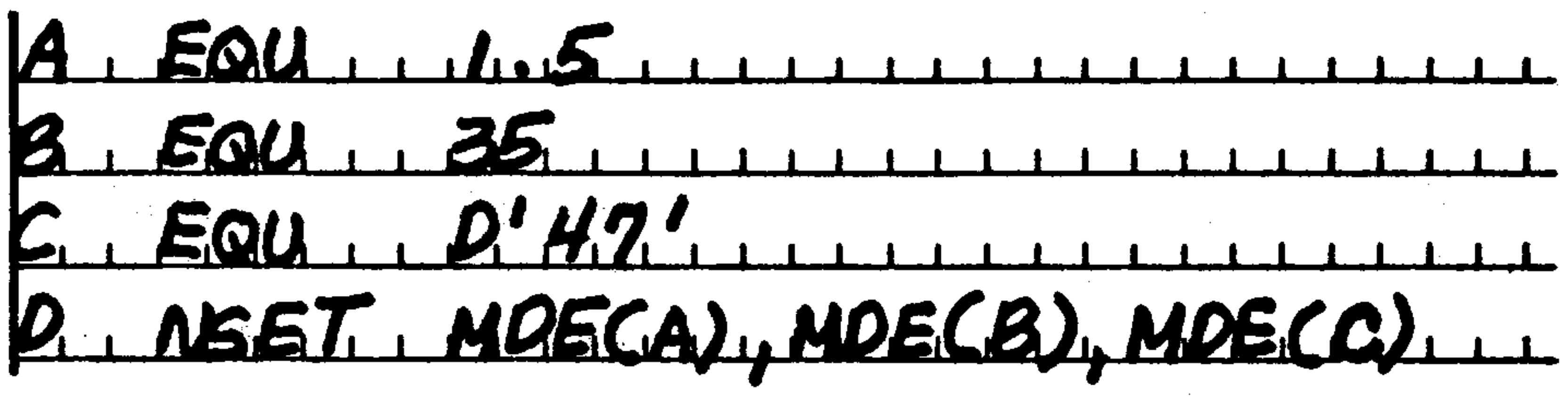
REL(B) = 1	The first program control section is always assigned location counter 1.
REL(15) = 0	The argument is absolute.

6.2 MODE (MDE)

The mode attribute function, MDE, returns the mode of the argument.

Mode	Type of Expression
0	Not a value; for example, a set or function name
1	Integer (decimal or octal) value
2	Real- or floating-point value
3	BCD character string, right adjusted
4	BCD decimal integer
5	BCD character string, left adjusted
7	ASCII character string
9	Relocatable word address (includes literals, control section names, and special character $)
10	External word address
11	Relocatable byte address
12	External byte address

Examples: Let A, B, and C be defined as follows.



```
A EQU  1.5
B EQU  35
C EQU  D'47'
D NSET MDE(A),MDE(B),MDE(C)
```

D[1] = MDE(A) = 2

D[2] = MDE(B) = 1

D[3] = MDE(C) = 4

6.3 NUMBER OF ELEMENTS (NUM)

NUM returns the number of elements in a set. If the symbolic item is not a set, NUM returns value 0.

Examples: Let A and B be defined as shown.

```
A   NSET  4,5,[7,10]
B   EQU   13
```

NUM(A) = 3	Set A has three elements.
NUM(A[1]) = 0	A[1] is a value, not a set.
NUM(A[3]) = 2	A[3] is a set of two elements.
NUM(A[3, 1]) = 0	A[3, 1] is value 7, not a set.
NUM(B) = 0	B is not a set.

```
C   NSET
```

NUM(C) = 1	Set C has one element (zero).

The following example tests for number of elements in a set and tests elements of a set for subsets.

```
A   NSET  4,5,[7,10]
B   RDEF  NUM(A)                B = 3; A has 3 elements.
C   RPT   B,D                   Test each element of A.
    GØTØ  NUM(A[C])>0,E         Exit to E for A[3].
D   LNID
    GØTØ  1,F
E   RDEF  NUM(A[C])             E = 2; Subset A[3] has
H   RPT   E,G                   2 elements.
    GØTØ  NUM(A[C,H])>0,J       Test each element of
     :                          subset for subset.
F   LNID
```

6.4 SIZE OF DATA (SZE)

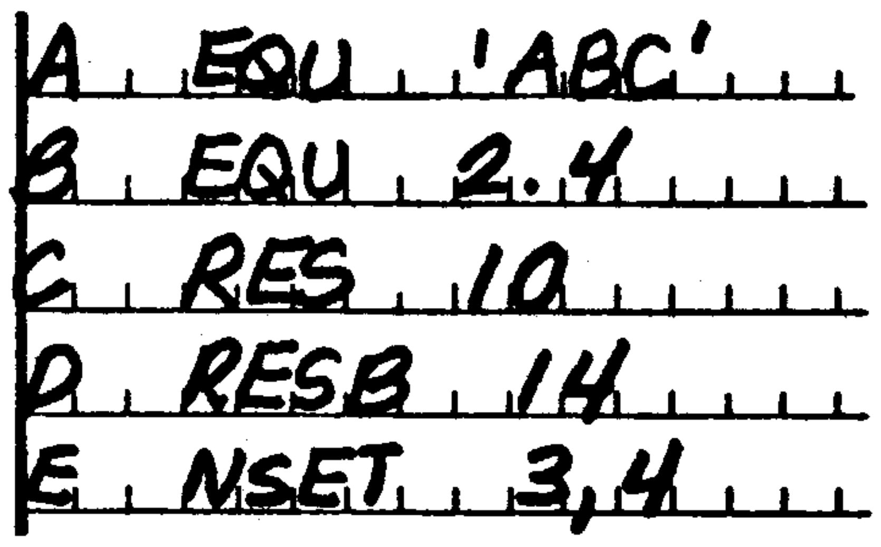
SZE returns either the number of object machine bytes needed to contain the value of an expression or the number of characters, depending on the mode of the expression. If the item is not a value, SZE returns value zero. SZE considers an address to be a one-word value.

SZE returns values depending on mode.

Mode	Size
0	Zero
1 or 2	Number of bytes
3, 4, 5, or 7	Number of characters
9, 10, 11 or 12	One word expressed as a byte count

Examples:

Let A, B, C, D, and E be defined as shown for an object computer word size of 24 bits and byte size of 6 bits.



```
A EQU 'ABC'
B EQU 2.4
C RES 10
D RESB 14
E NSET 3,4
```

SZE(A) = 3	Three characters
SZE(B) = 8	Two words or eight bytes
SZE(C) = 4	One word or four bytes
SZE(D) = 4	One word or four bytes
SZE(E) = 0	E is a set, not a value

6.5 SYMBOL (SYM)

SYM causes the value of the argument expression to be treated as a symbol. A SYM attribute function reference can appear in the label, command, or operand field. By using SYM, the programmer creates a symbol which is the value of the argument expression. The assembler represents the symbol as either 24 or 48 bits.

One use of the SYM attribute function is to refer to a symbol that is otherwise illegal. SYM can also be used for symbol concatenation.

Another use is to move a symbolic parameter into any field of a procedure or function. In this way, symbols supplied as parameters can be defined within a procedure or function.

Examples:

```
SYM(Ø'21212121')    RDEF  1       Defines AAAA
AAAA                RDEF  2     )
SYM(L'AAAA')        RDEF  3     } Redefine
SYM(C'AAAA')        RDEF  4     ) AAAA
SYM(2.5)            RDEF  2.5     Defines 2.5
```

The following example illustrates symbol concatenation. It generates symbol XY by scaling parameters X and Y into appropriate bit positions to form the value of the argument expression.

```
    PRØC  AA
A   NAME
SYM$((AA[1] ↓ 18)+(AA[2] ↓ 12)+'  ')  EQU  1
     .
     .
     .
    ENDS
     .
     .
     .
    A   'X','Y'
```

A reference to the symbol Q8Q.XYZ is ordinarily illegal because of the decimal point. It can, however, be referred to through use of the SYM attribute.

```
     EXT  SYM(L'Q8Q.XYZ')
```

A reference to a procedure can be a SYM-defined name

```
            PRØC  P                  Outer procedure
 JFR        NAME
F           PRØC  Q                  Procedure has
SYM(P[1])   NAME  P[2],P[3]          SYM-defined name
              :
SYM$$(Q[1])  EQU  $
              :
            ENDS                     End inner
            SYM(P[1])  P[4]          procedure
            ENDS                     End definition
            JFR  L'AB',5,9,L'DE'     JFR reference
```

The above code causes the procedure JFR to be interpreted as if it had been written:

```
            PRØC  P                  Begin JFR
JFR         NAME
F           PRØC  Q
AB          NAME  5,9
              :
SYM$$(Q[1]) EQU   $
              :
            ENDS
            AB    P[4]
            ENDS                     End JFR
            JFR L'AB',5,9,L'DE'      Refer to JFR
```

After the inner reference to procedure AB, the EQU line becomes:

```
DE$$  EQU  $
```

6.6 WORD ADDRESS (WRD)

If the mode of the argument expression is either 9 (word) or 11 (byte), WRD returns the value of the argument as a word address. If the mode of the argument expression is 12 (external byte address), WRD changes the mode to 10 (external word address). If the mode of the argument expression is not 9, 10, 11, or 12, WRD returns the argument expression unchanged. If the argument expression is a byte address that does not correspond to a word address, truncation occurs.

Examples:

```
      UNIT   6,4          Computer word 6 bits per byte,
                          four bytes per word.
A     RESB   4            A has mode 11, value 0.
AA    EQU    WRD(A)       AA has mode 9, value 0.
B     RESB   4            B has mode 11, value 4.
BB    EQU    WRD(B)       BB has mode 9, value 1.
C     RES    1            C has mode 9, value 2.
CC    EQU    WRD(C)       CC has mode 9, value 2.
D     EQU    10           D has mode 1, value 10.
DD    EQU    WRD(D)       DD has mode 1, value 10.
E     RESB   1            E has mode 11, value 12.
F     RESB   1            F has mode 11, value 13.
FF    EQU    WRD(F)       FF has mode 9, value 3 truncated.
      EXT    G            G has mode 10, value 0.
GG    EQU    BYT(G)       GG has mode 12, value 0.
GGG   EQU    WRD(GG)      GGG has mode 10, value 0.
```

6.7 BYTE ADDRESS (BYT)

If the mode of the argument expression is 9 or 11 (word or byte), BYT returns the value of the argument expression as a byte address. If the mode of the argument is 10 (external word address), BYT changes the mode to 12 (external byte address). If the mode is not 9, 10, 11 or 12, BYT returns the argument expression unchanged.

Examples:

```
      UNIT   6,4        Computer word 6 bits per
                        byte, four bytes per word.
A     RES    1          A has mode 9, value 0.
AA    EQU    BYT(A)     AA has mode 11, value 0.
B     RES    1          B has mode 9, value 1.
BB    EQU    BYT(B)     BB has mode 11, value 4.
C     RESB   1          C has mode 11, value 8.
CC    EQU    BYT(C)     CC has mode 11, value 8.
D     EQU    10         D has mode 1, value 10.
DD    EQU    BYT(D)     DD has mode 1, value 10.
      EXT    E          E has mode 10, value 0.
EE    EQU    BYT(E)     EE has mode 12, value 0.
```

PROGRAM EXECUTION 7

META can be called either by a MASTER task name control card or by a task already in execution.

When called by control card, META is loaded and placed in multiprogrammed execution as soon as its class, core, and file requirements can be met. When called by a CALL macro, a copy of META is loaded, if the job making the call does not already have a copy of the task. If it has a copy, the call is queued; that is, the caller must wait for the existing copy. Since META reinitializes itself, a job may make multiple calls to the Meta-Assembler. Parameters ordinarily specified on a META control card (including parentheses) are passed as secondary parameters of a CALL macro. For use of CALL macro, see MASTER Reference Manual.

When the object deck is to be executed, it must be called by a task name control card or another task. The job monitor then calls the loader which loads relocatable binary information, links independently assembled subprograms, and loads and links library routines referenced by the loaded program. The program then executes multiprogrammed with all other active tasks.

7.1 CONTROL CARDS

Assembly of META source programs under MASTER and execution of 3300/3500 binary object decks require MASTER control cards identifiable by a $ in column 1 (except for the end-of-file card). The name of the control card followed by any necessary parameters begins in column 2. The name and parameters must be contained on an 80-column card.

MASTER control cards optionally accompanied by source and data decks are read serially from the input card reader. Cards required for META jobs are described in sections 7.1.1 through 7.1.5.

7.1.1 $JOB

A JOB card must appear in a job deck either as the first card or, if a DIRECT card is used, as the second card.

```
$JOB,c,i,tℓ,ℓ,p
```

c	BCD account number; required
i	BCD job identifier; required
tl	Time limit in minutes; optional
l	Printer line limit (1-99999); optional
p	Punched card limit (0-99999); optional

Example:

```
$JOB, 639, DJ, 15, 150, 100, COMMENTS
```

7.1.2 $SCHED

A SCHED card, immediately follows the JOB card in the job deck and provides the system with core and scratch mass storage requirements.

```
$SCHED, CORE=qp, SCR=seg, . . .
```

Other SCHED card parameters, not normally required by the META assembler, are described in the MASTER Reference Manual.

CORE=qp

Estimate of maximum amount of core, in quarter pages, required for assembly or execution, whichever has the higher core requirement. The estimate for the META assembler is a minimum of 32 quarter pages. Add four quarter pages if MASTER mnemonic instruction set is required and allow for any other procedures or functions.

If the loader determines that the estimate is below that required by the job, the job is terminated with a message on the OUT file.

When the CORE field is omitted, qp is set by an installation parameter.

SCR=seg

Number of segments of mass storage scratch area required by the job. The segment size is determined when the operating system is installed.

If the length of a segment is 10,000 words, the file for executable output (usually LGO) requires roughly one segment for each 400 source statements. Normally, LGO needs only one segment.

META uses at least one and sometimes three system scratch files in addition to files indicated on the META card. All are in standard MASTER blocked format with a block size of 1280 characters. META always uses a file with the dsi INT for source card images of the subprogram being assembled. The SCR field must schedule sufficient segments for this file to contain the largest subprogram or a set of subprograms to be assembled.

If the X or F option is requested, META uses a scratch file having the dsi BIN. Normally, one segment is sufficient; the file contains most of the binary output for one subprogram.

If a cross reference table is requested, META writes reference information on a scratch file with the dsi INTP. Normally, one segment is sufficient for INTP.

If the sum of the mass storage requirements indicated by the JOB card line and punch limits and the SCR and ABORT requests exceeds the storage reserved for these files, the job is not initiated.

When the SCR field is omitted, seg is set to an installation parameter.

7.1.3 $META

The MASTER task name control card that causes META to be called, loaded, and executed (multiprogrammed) has the following format.

```
$META(p1,....,pn)
```

The optional parameters, p_i, are separated by commas and may appear in any order within the parentheses. Parameters have the format:

assembly option = dsi

or

assembly option

The assembly options are character strings, beginning with I, L, X, F, P, or R. The dsi's are MASTER data set identifiers of 1-4 alphanumeric characters; 0000 may not be used for a dsi.

The options, and the data set identifier assigned for each if none is given on the META card, are listed below:

Option	Significance	dsi
I	Source input	INP
L	Listable output	OUT
X	Load-and-go output	LGO
F	Load-and-go output with forced execution	LGO
P	Punchable output	PUN
R	Cross reference table (selectable only in conjunction with L)	Same dsi as for L

The X and F options are mutually exclusive. If the X option is used and assembly errors occur, META issues a SUPPRESS request (MASTER Reference Manual) so that the object program is not executed. Under the X option, assembly errors do not prevent generation of the executable output, just its loading and execution in the same job. The F option causes execution of the 3300/3500 object program despite assembly errors.

The Meta-Assembler source deck can be on the standard input card reader (INP) or a file, such as a magnetic tape file, specified by the programmer. If it is on the card reader, the MASTER input preprocessor transfers the deck from the card reader to the INP file. The programmer has the option of bypassing this transfer by placing a DIRECT card in front of his deck.

MASTER either accumulates Meta-Assembler printer output on the mass storage standard output file (OUT) for automatic post-job processing, prints output directly during job execution, or places the output on some other file specified by the user and for which printing is not automatic.

Similarly, MASTER either accumulates Meta-Assembler binary output on a punch file (PUN) for automatic post-job punching, punches output directly during job execution, or places the output on some other file specified for the user and for which punching is not automatic.

For all output options, META assigns a system scratch file if the user does not specify either a standard file (OUT, PUN, or LGO) or a permanent file. All scratch files are automatically released at job end. The SCR parameter on the SCHED card must allow for all scratch files.

Use of permanent files is described in the MASTER Reference Manual.

Example:

```
$META(LIST, XCUTE, PUNCH)
```

META is loaded from MASTER library file *LIB. Source statements are read by META from the INP file. Statements and assembly listings are written on the job OUT file and automatically printed. The punchable output is written on the job PUN file and automatically punched. Executable output is written on the LGO file.

```
$META(IN=SRCE, LIST=OUT, FORSX=GOGO)
```

META is loaded from MASTER library file *LIB. It reads source statements from file SRCE. Printer output goes to the OUT file and is automatically printed. The job does not have any punch output. Executable output goes to user file GOGO. Because of the F option, the program on GOGO can be loaded and executed despite errors. occurring during assembly.

7.1.4 TASK NAME

A task name control card directs MASTER to call and load the object-time program from the specified file and to begin execution of the task.

If the object-time program is to be executed following assembly, a task name card of the following form must follow the source deck (if it is on the standard input file) or the META card (if the source deck is elsewhere).

```
$name, dsi
```

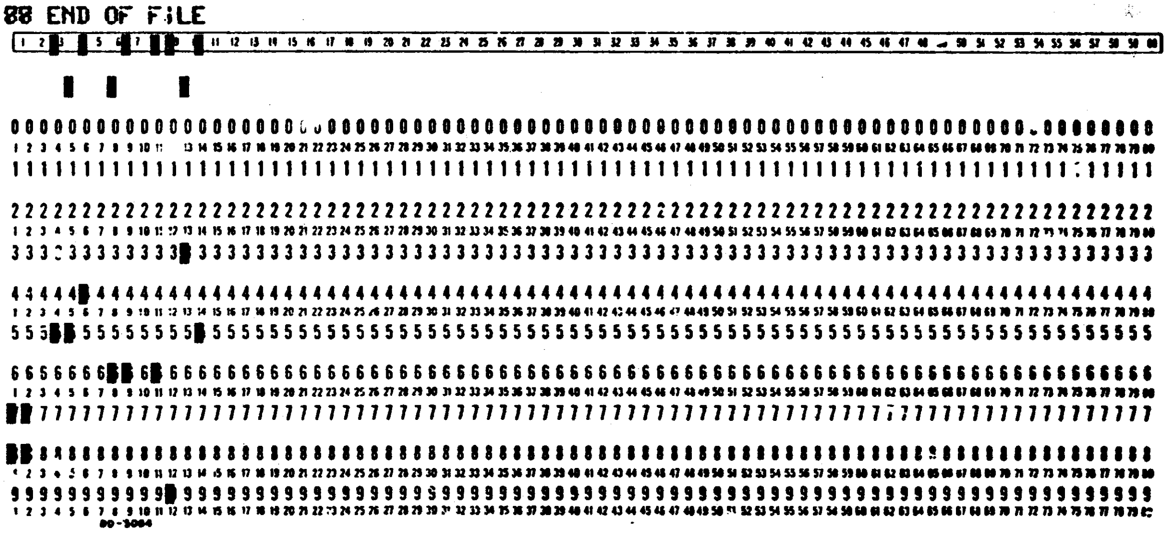
name 1-4 alphanumeric characters; name is required.

dsi dsi of an opened file from which the named task is to be loaded. When the dsi is zero or the field is omitted, MASTER looks for the task on the system library. Normally, dsi is LGO.

For execution of a previously assembled program, the task name card for the object deck immediately follows the SCHED card. The object deck follows the task name card or is on the named file.

7.1.5 END-OF-FILE

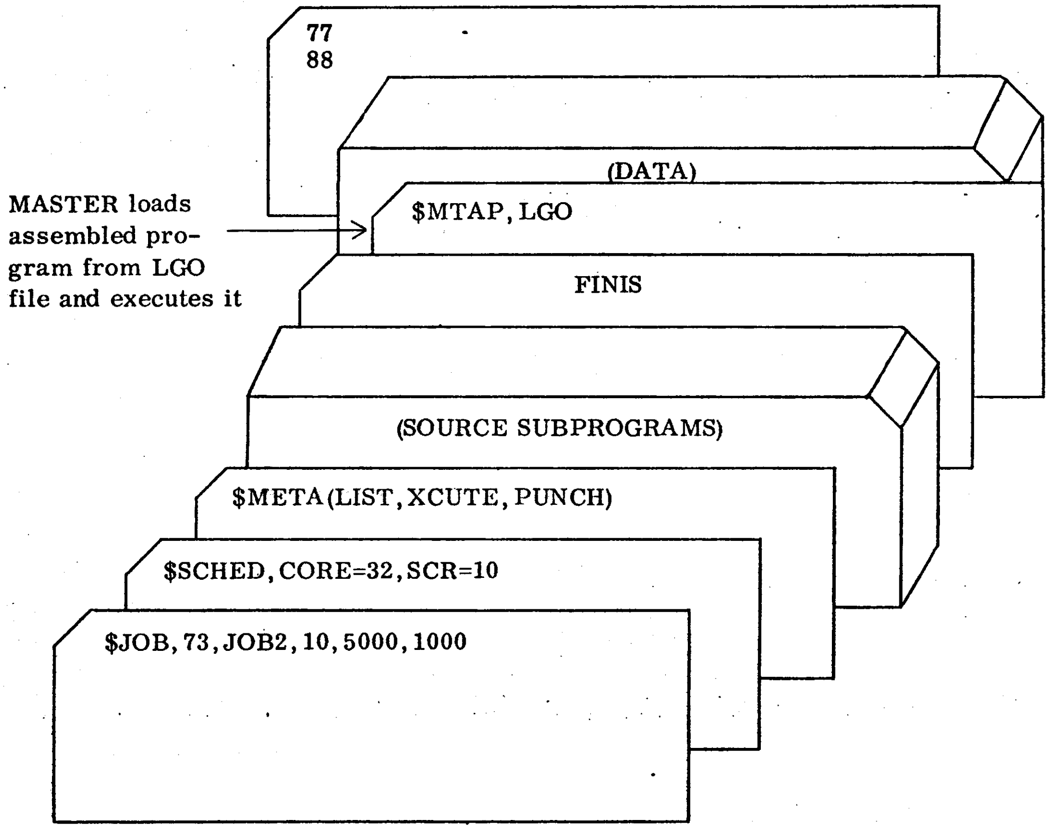
A job is terminated with an end-of-file card characterized by 7,8 punches in columns one and two. Columns 3-80 may contain comments.



7.2 SAMPLE DECKS

The following sample deck structures illustrate the use of MASTER control cards in job decks.

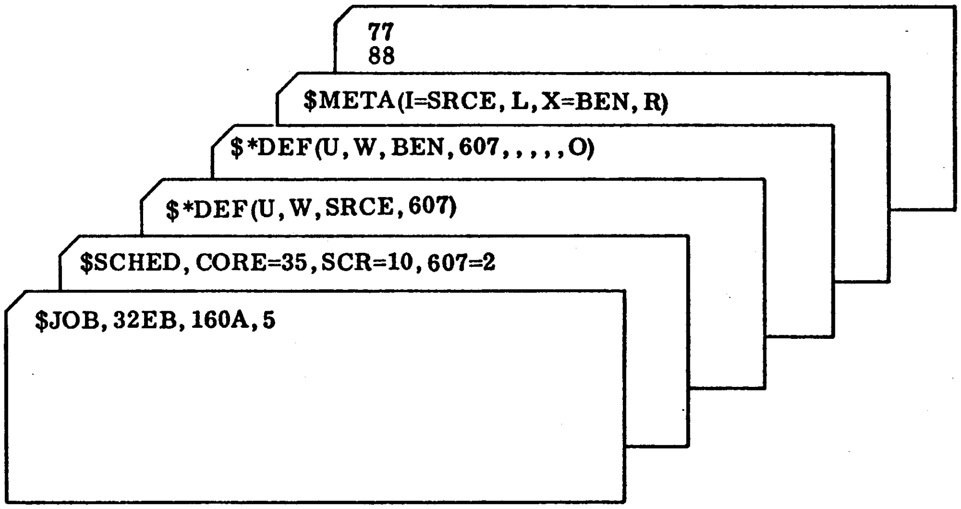
Assemble, list, and execute



```
77
88
(DATA)
$MTAP, LGO
FINIS
(SOURCE SUBPROGRAMS)
$META(LIST, XCUTE, PUNCH)
$SCHED, CORE=32, SCR=10
$JOB, 73, JOB2, 10, 5000, 1000
```

MASTER loads assembled program from LGO file and executes it

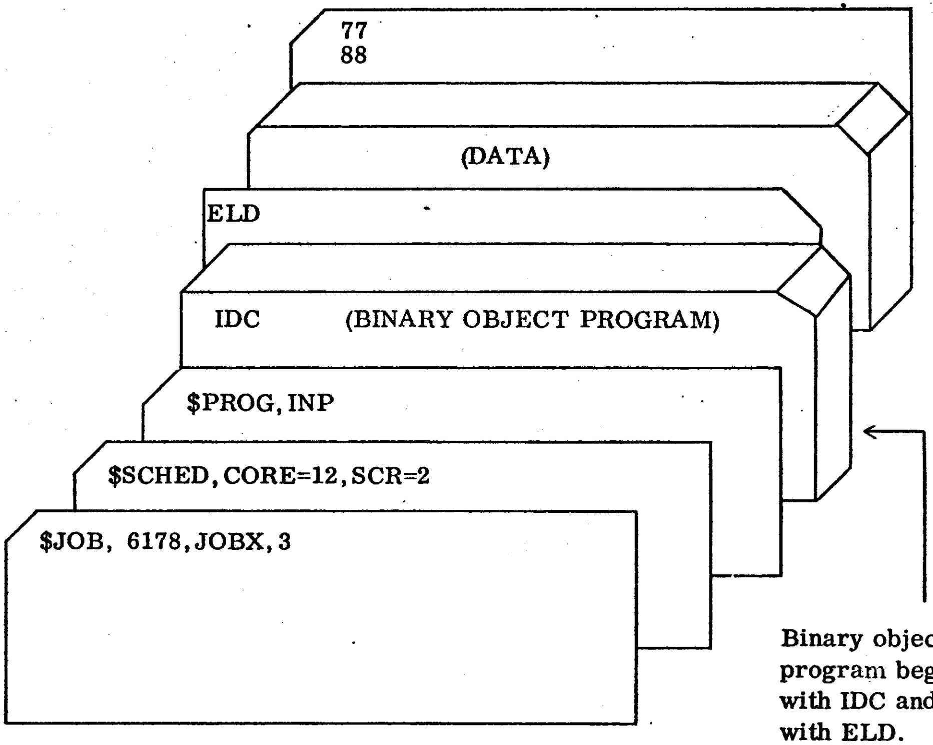
Assemble and list



```
77
88
$META(I=SRCE, L, X=BEN, R)
$*DEF(U, W, BEN, 607, , , , , O)
$*DEF(U, W, SRCE, 607)
$SCHED, CORE=35, SCR=10, 607=2
$JOB, 32EB, 160A, 5
```

This job does not include execution of an object deck because the source program on file SRCE contains a UNIT directive describing a computer system other than the 3300 or 3500. Output is to permanent file BEN. In this example, SRCE and BEN are on magnetic tape. For use of 607 parameter on SCHED card and for use of *DEF cards, refer to the MASTER Reference Manual.

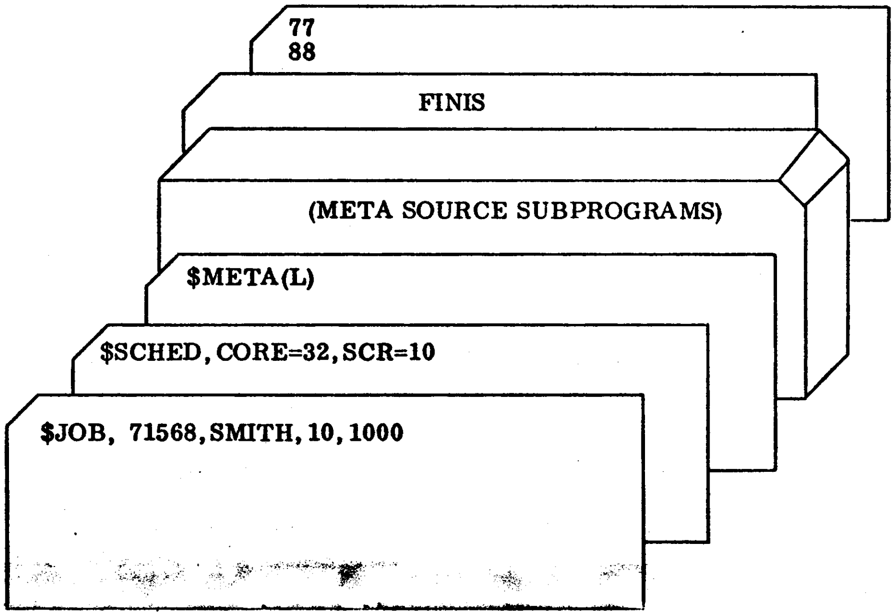
Execute only



Binary object program begins with IDC and ends with ELD.

This example illustrates execution of a 3300/3500 deck assembled previously by META.

List only



```
77
88
FINIS
(META SOURCE SUBPROGRAMS)
$META(L)
$SCHED,CORE=32,SCR=10
$JOB, 71568,SMITH,10,1000
```

This job assembles the source deck but produces only a listing as output.

MESSAGES 8

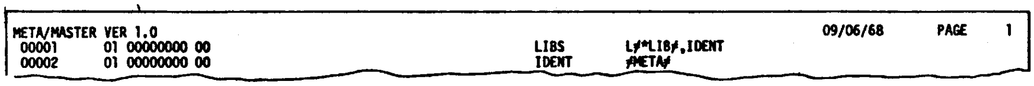
8.1 LIST FORMAT

When the L option is selected on the META control card, META generates list output. Each page of list output is in the following format:

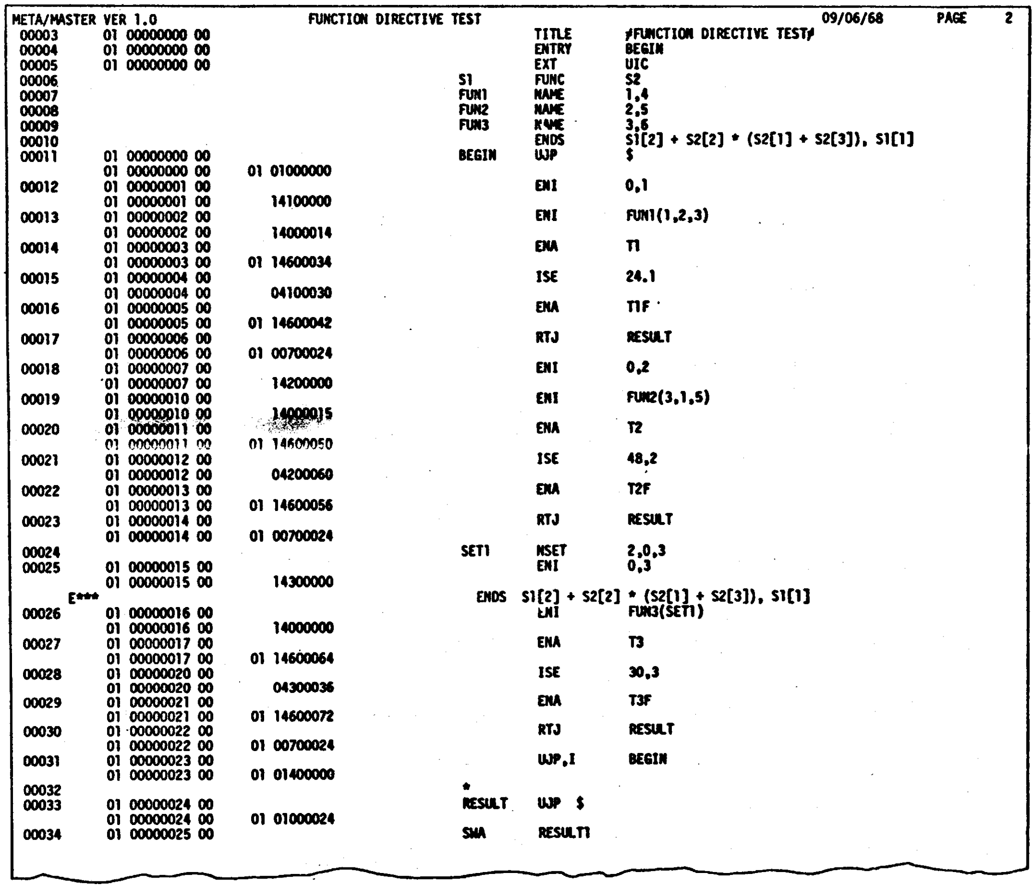
META/MASTER			VER n.n			title (optional)	date PAGE
source statement number	error code	relocation section	word address	byte position	operand relocation	object computer word	source statement

title	Characters supplied by TITLE directive.
date	Date of computer run.
source statement number	Position of source statement in the source deck (00000-99999).
error code	Code if source statement is erroneous (section 8.2).
relocation section	Control section (00-15) containing object computer word.
word address	Address of object computer word.
byte position	On byte-oriented source lines, position of byte in word from left to right. 00-n, respectively, where n is the number of bytes per word.
operand relocation	Control section (00-15) containing operand; X indicates operand is external symbol.
Object computer word	Object computer word generated by META (3-16 octal digits).
source statement	1-80 characters of source input line, including sequence number if provided.

Example:



```
META/MASTER VER 1.0                                                                       09/06/68      PAGE     1
 00001     01 00000000 00                                       LIBS       L≠*LIB≠,IDENT
 00002     01 00000000 00                                       IDENT      ≠META≠
```



```
META/MASTER VER 1.0                     FUNCTION DIRECTIVE TEST                           09/06/68      PAGE     2
 00003     01 00000000 00                                       TITLE      ≠FUNCTION DIRECTIVE TEST≠
 00004     01 00000000 00                                       ENTRY      BEGIN
 00005     01 00000000 00                                       EXT        UIC
 00006                                                S1        FUNC       S2
 00007                                                FUN1      NAME       1,4
 00008                                                FUN2      NAME       2,5
 00009                                                FUN3      K4ME       3,6
 00010                                                          ENDS       S1[2] + S2[2] * (S2[1] + S2[3]), S1[1]
 00011     01 00000000 00                             BEGIN     UJP        $
           01 00000000 00    01 01000000
 00012     01 00000001 00                                       ENI        0,1
           01 00000001 00       14100000
 00013     01 00000002 00                                       ENI        FUN1(1,2,3)
           01 00000002 00       14000014
 00014     01 00000003 00                                       ENA        T1
           01 00000003 00    01 14600034
 00015     01 00000004 00                                       ISE        24.1
           01 00000004 00       04100030
 00016     01 00000005 00                                       ENA        T1F
           01 00000005 00    01 14600042
 00017     01 00000006 00                                       RTJ        RESULT
           01 00000006 00    01 00700024
 00018     01 00000007 00                                       ENI        0,2
           01 00000007 00       14200000
 00019     01 00000010 00                                       ENI        FUN2(3,1,5)
           01 00000010 00       14000015
 00020     01 00000011 00                                       ENA        T2
           01 00000011 00    01 14600050
 00021     01 00000012 00                                       ISE        48,2
           01 00000012 00       04200060
 00022     01 00000013 00                                       ENA        T2F
           01 00000013 00    01 14600056
 00023     01 00000014 00                                       RTJ        RESULT
           01 00000014 00    01 00700024
 00024                                                SET1      NSET       2,0,3
 00025     01 00000015 00                                       ENI        0,3
           01 00000015 00       14300000
        E***                                            ENDS  S1[2] + S2[2] * (S2[1] + S2[3]), S1[1]
 00026     01 00000016 00                                       ENI        FUN3(SET1)
           01 00000016 00       14000000
 00027     01 00000017 00                                       ENA        T3
           01 00000017 00    01 14600064
 00028     01 00000020 00                                       ISE        30,3
           01 00000020 00       04300036
 00029     01 00000021 00                                       ENA        T3F
           01 00000021 00    01 14600072
 00030     01 00000022 00                                       RTJ        RESULT
           01 00000022 00    01 00700024
 00031     01 00000023 00                                       UJP,I      BEGIN
           01 00000023 00    01 01400000
 00032                                                *
 00033     01 00000024 00                             RESULT    UJP  $
           01 00000024 00    01 01000024
 00034     01 00000025 00                             SWA       RESULT1
```

8.2 ERROR CODES

Meta-Assembler flags each detected error with a single-character error code and 3 asterisks on the line of the source statement in error.

Code	Meaning
C***	Common error. An attempt was made to assemble information into numbered common.
D***	Double definition. 1) A symbol has two values at the same level, or 2) A subprogram that does not contain a UNIT directive contains more than one SECP directive.
E***	Expression error. The expression is syntactically correct, but an error, such as an illegal combination of modes, exists.
F***	Forward reference error. A forward reference appeared in an expression which must be evaluatable.
I***	Illegal instruction. The command field contains a symbol that is neither a directive nor the name of a procedure or FORM. The command field contains a misplaced directive.
N***	Nesting error. More than 14 procedure levels or six RPT nests were encountered, or an RPT, procedure, or function is improperly nested.
R***	Relocation error. The relocation associated with an expression is neither absolute, nor singularly positive, nor singularly negative, nor an external plus or minus a constant.
S***	Syntax error. The syntax is unrecognizable or illegal. For example, a symbol has more than 12 characters.
T***	Truncation error caused by 1) A value larger than the receiving field can accept. Note: No error is flagged when all the truncated bits are the same as the most significant bit (sign) of the value placed in the field. 2) A word-oriented statement following a byte-oriented statement. 3) Mixing of word-oriented and byte-oriented operations.
U***	Undefined symbol. An operand contains a reference to a symbol that is neither defined in the program nor declared as external.

8.3 SUPPLEMENTARY INFORMATION

Following the source program listing, META provides supplementary information as a standard part of the Meta-Assembler output listing. The supplementary information is identified as follows:

Message	Meaning
LITERALS	Identifies the list of literals. The location and control section designator (0-15) are given for each literal.
CONTROL SECTIONS	Begins new page. Identifies list of control section names, octal length of section in words, and location counter designator (0-15). Each entry in the list begins with SECA, SECP, or SECD, indicating the type of control section.
EXTERNAL SYMBOLS	Identifies the list of external symbols.
ENTRY-POINT SYMBOLS	Identifies the list of entry-point symbols.
UNDEFINED SYMBOLS	Identifies the list of undefined symbols.
MULTIPLY-DEFINED SYMBOLS	Identifies the list of multiply-defined symbols.
FIRST 25 ERROR LINES	Identifies line numbers of first 25 lines flagged with error codes. If the line in error is not a source input line and thus has no line number, the number of the most recently encountered input line is used.
NUMBER OF LINES WITH DIAGNOSTICS	Identifies count of the number of lines flagged with error codes.

Example:

```
CONTROL SECTIONS
      SECP      REAL               114      1
EXTERNAL SYMBOLS
 UIC
ENTRY-POINT SYMBOLS
 SSSSSSS
UNDEFINED SYMBOLS
MULTIPLY-DEFINED SYMBOLS
FIRST 25 ERROR LINES
         10        11        12        15         16         18

NUMBER OF LINES WITH DIAGNOSTICS    00013
```

8.4 CROSS REFERENCE TABLE

META provides the cross reference table if the R option is selected on the META control card. If both R and L options are selected, the table follows supplementary information. This table is identified by the title:

CROSS REFERENCE TABLE

The first column gives the address of the directive defining the symbol given in the second column. Addresses of references to the symbol are in the remaining columns.

Example:

```
                         CROSS REFERENCE TABLE

15   A

14   B

 1   GENT                        1
```

8.5 MESSAGES ON OUT

After detecting an error, META writes one of the following messages on the OUT file for the job.

Message	Cause
**META request ERROR code DSI dsi LINE line	Input/output error occurred. If other than read error (PICK reject code 04000000 or 050xxxxx), run is abnormally terminated. Message appears as voluntary abort code on accounting information as well as in listing.

request	Blocker/deblocker or system OCARE request name
code	Reject code for request: (Q) for blocker/deblocker (A) for system OCARE
dsi	Data set identifier for request
line	Number of META source input line

**META BAD LIBRARY — The overlays of META are not in task directory. Library generation is incorrect. The run is abnormally terminated and message also appears as voluntary abort code.

**META FINIS GENERATED — FINIS directive generated because of end-of-file condition encountered on source input file. Execution continues.

**META ILLEGAL $META CARD — $META card contains illegal parameter such as illegal option or data set identifier. The run is abnormally terminated and message also appears as voluntary abort code.

**META $SCHED MORE CORE — Request for additional core rejected. The run is abnormally terminated and message also appears as voluntary abort code. Resubmit job with more core specified on $SCHED card.

Examples:

```
**META SEXPAND ERROR 30000000 DSI INT LINE 10422

**META PICK ERROR 05000000 DSI INP LINE 00012
```

APPENDIX SECTION

CHARACTER SET

Type of Character	501 Printer Graphic	Internal Code Octal	Card Code
Alphabetic	A	21	12,1
	B	22	12,2
	C	23	12,3
	D	24	12,4
	E	25	12,5
	F	26	12,6
	G	27	12,7
	H	30	12,8
	I	31	12,9
	J	41	11,1
	K	42	11,2
	L	43	11,3
	M	44	11,4
	N	45	11,5
	O	46	11,6
	P	47	11,7
	Q	50	11,8
	R	51	11,9
	S	62	0,2
	T	63	0,3
	U	64	0,4
	V	65	0,5
	W	66	0,6
	X	67	0,7
	Y	70	0,8
	Z	71	0,9
Numeric	0	00	0
	1	01	1
	2	02	2
	3	03	3
	4	04	4
	5	05	5
	6	06	6
	7	07	7
	8	10	8
	9	11	9

Type of Character	501 Printer Graphic		Internal Code Octal	Card Code
Blank	blank		60	space
Special	+	plus	20	12
	-	minus	40	11
	*	times	54	11,4,8
	/	divide	61	0,1
	=	equals	13	3,8
	<	less than	32	12,0
	>	greater than	57	11,7,8
	.	period	33	12,3,8
	,	comma	73	0,3,8
	(	left parenthesis	74	0,4,8
	)	right parenthesis	34	12,4,8
	%	percent	16	6,8
	$	dollar	53	11,3,8
	≠	not equal (apostrophe on keypunch)	14	4,8
	≤	less or equal	15	5,8
	≥	greater or equal	35	12,5,8
	[	left bracket	17	7,8
	]	right bracket	72	0,8,2
	↑	decimal exponent	55	11,5,8
	↓	binary exponent	56	11,6,8
	¬	NOT	36	12,6,8
	;	semicolon	37	12,7,8
	→	right arrow	75	0,5,8
	≡	identity	76	0,6,8
	:	colon	12	2,8
	∨	OR	52	11,0
	∧	AND	77	0,7,8

TABLE A-1. BCD/ASCII Conversion Table

6-bit BCD Code	8-bit ASCII Character	Binary Status of ASCII Character (bit positions)							
		7*	6	5	4	3	2	1	0
00	0	0	0	1	1	0	0	0	0
01	1	0	0	1	1	0	0	0	1
02	2	0	0	1	1	0	0	1	0
03	3	0	0	1	1	0	0	1	1
04	4	0	0	1	1	0	1	0	0
05	5	0	0	1	1	0	1	0	1
06	6	0	0	1	1	0	1	1	0
07	7	0	0	1	1	0	1	1	1
10	8	0	0	1	1	1	0	0	0
11	9	0	0	1	1	1	0	0	1
12	:	0	0	1	1	1	0	1	0
13	=	0	0	1	1	1	1	0	1
14	'	0	0	1	0	0	1	1	1
15	&	0	0	1	0	0	1	1	0
16	%	0	0	1	0	0	1	0	1
17	[	0	1	0	1	1	0	1	1
20	+	0	0	1	0	1	0	1	1
21	A	0	1	0	0	0	0	0	1
22	B	0	1	0	0	0	0	1	0
23	C	0	1	0	0	0	0	1	1
24	D	0	1	0	0	0	1	0	0
25	E	0	1	0	0	0	1	0	1
26	F	0	1	0	0	0	1	1	0
27	G	0	1	0	0	0	1	1	1
30	H	0	1	0	0	1	0	0	0
31	I	0	1	0	0	1	0	0	1
32	<	0	0	1	1	1	1	0	0
33	.	0	0	1	0	1	1	1	0
34	)	0	0	1	0	1	0	0	1
35	∧	0	1	0	1	1	1	1	0
36	"	0	0	1	0	0	0	1	0
37	;	0	0	1	1	1	0	1	1

*ASCII bit 7 is unassigned and 0 for all codes.

TABLE A-1. BCD/ASCII Conversion Table

6-bit BCD Code	8-bit ASCII Character	Binary Status of ASCII Character (bit positions)							
		7*	6	5	4	3	2	1	0
40	-	0	0	1	0	1	1	0	1
41	J	0	1	0	0	1	0	1	0
42	K	0	1	0	0	1	0	1	1
43	L	0	1	0	0	1	1	0	0
44	M	0	1	0	0	1	1	0	1
45	N	0	1	0	0	1	1	1	0
46	O	0	1	0	0	1	1	1	1
47	P	0	1	0	1	0	0	0	0
50	Q	0	1	0	1	0	0	0	1
51	R	0	1	0	1	0	0	1	0
52	!	0	0	1	0	0	0	0	1
53	$	0	0	1	0	0	1	0	0
54	*	0	0	1	0	1	0	1	0
55	#	0	0	1	0	0	0	1	1
56	\	0	1	0	0	0	0	0	0
57	>	0	0	1	1	1	1	1	0
60	Blank	0	0	1	0	0	0	0	0
61	/	0	0	1	0	1	1	1	1
62	S	0	1	0	1	0	0	1	1
63	T	0	1	0	1	0	1	0	0
64	U	0	1	0	1	0	1	0	1
65	V	0	1	0	1	0	1	1	0
66	W	0	1	0	1	0	1	1	1
67	X	0	1	0	1	1	0	0	0
70	Y	0	1	0	1	1	0	0	1
71	Z	0	1	0	1	1	0	1	0
72	]	0	1	0	1	1	1	0	1
73	, Comma	0	0	1	0	1	1	0	0
74	(	0	0	1	0	1	0	0	0
75	⌐~	0	1	0	1	1	1	0	0
76	_	0	1	0	1	1	1	1	1
77	?	0	0	1	1	1	1	1	1

*ASCII bit 7 is unassigned and 0 for all codes.

3300/3500 MNEMONIC INSTRUCTIONS B

A 3300/3500 META mnemonic instruction is a procedure reference in which the label field optionally contains a symbolic address, the command field contains a mnemonic instruction and modifiers, and the operand field contains operands that depend on the mnemonic.

META assembles 3300/3500 mnemonic instructions through the use of three standard sets of procedures on the system library. The sets are identified through their primary entry names as IDENT, MONITOR, and BDP.

IDENT

IDENT includes procedures for the 3300/3500 mnemonic instructions executable in program state, for the HLT instruction, and for the following 3300/3500 COMPASS/MASTER pseudo instructions.

IDENT
BSS
BSS, C
DEC
DECD

Capabilities paralleling those provided by the following pseudo instructions are available through Meta-Assembler directives (Chapter 4).

END
FINIS
ENTRY
EXT
EQU
NOLIST
LIST
EJECT
TITLE

Of these, TITLE is the only directive that does not correspond to the COMPASS psuedo instruction.

META does not recognize the following 3300/3500 COMPASS/MASTER pseudo instructions.

REM
COMMON
OCT
VFD
IFT
IFZ
PRG
ORGR
BCD
MACRO
IFF
IFN
DATA
EQU, C
BCD, C
ENDM
LIBM
SPACE

MONITOR

MONITOR includes procedures for assembling 3300/3500 mnemonic instructions executable in the monitor state only.

BDP

BDP provides for assembly of 3300/3500 BDP instructions only.

USE OF LIBS

Before they can be used, the 3300/3500 mnemonic instruction procedures must be obtained from the library through use of the LIBS directive.

Examples:

LIBS	'*LIB',IDENT	Program state instructions
LIBS	'*LIB',IDENT,BDP	All but monitor state instructions
LIBS	'*LIB',IDENT,MØNITØR	All but BDP instructions

MASTER PROGRAM TASK

A META-Assembler program to be executed as a task under the MASTER multiprogramming operating system must include a copy of the user interrupt control routine (UIC) that provides the task with an entrance and an exit. Each subprogram must begin with a LIBS directive.

When loading and execution of the assembled output is called for by the task name card (section 7.1.4), the call connects with the UIC routine which contains a return jump to the task primary entry point. The return address is inserted into the operand field for the UJP as a normal function of a return jump execution. To obtain a copy of UIC, the program must declare UIC as an external symbol.

Example:

```
        LIBS    '*LIB',IDENT       Call for library procedures.
        IDENT   L'JØE'             First subprogram named JOE.
        EXT     UIC,XY
        ENTRY   START
START   UJP     0                  START is the task primary entry
        LDA                        point.
        STA
        .
        :
        UJP,I   START
        END     START
        LIBS    '*LIB',IDENT       Begin second subprogram named XY.
        IDENT   L'XY'
        ENTRY   XY
XY      UJP     0
        .
        :
        END
        FINIS
```

PROCEDURE SETS

Three tables present brief descriptions of procedure references and resultant object code assembled by the IDENT, MONITOR, and BDP procedure sets. For a complete description of the actual machine instructions, refer to the 3300 or 3500 Computer System Reference Manual.

Because the 3300/3500 instructions are assembled through procedures, operation code modifiers must be defined as symbols having values. A reference to each of the sets IDENT, MONITOR, and BDP, causes the symbols for operation code modifiers to be defined. No other definition can be given these symbols. Thus, a group of words is reserved for each set of procedures.

The following list of terms defines modifiers, operands, registers, and nonstandard symbols that appear in the tables.

In some instructions, the execution address m or r, or the shift count k may be modified by adding to them the contents of an index register, B^b. The 2-bit designator b specifies which of the three index registers is to be used. Symbols representing the respective modified quantities are M, R, and K.

Term	Meaning
A	MONITOR operation modifier: Conversion (alter the characters transmitted). Other: 24-bit A register or word count control for INAC, and INAW.
b	The b subfield designates an index register. The b subfield may be represented by a digit; a symbol; or an expression with a nonrelocatable value of 1 2, or 3.
B	MONITOR operation modifier: Backward read or write. Other: Index register defined by B^b.
B_m	Index register flag, $M = m+(B_m)$ for these instructions only.
B_r	Index register flag. If $B_r = 1$ or 3, $R = r+(B^1)$. If $B_r = 2$, $R = r+(B^2)$. If $B_r = 0$, $R = r$.
B_s	Index register flag. If $B_s = 1$ or 3, $S = s+(B^1)$. If $B_s = 2$, $S = s+(B^2)$. If $B_s = 0$, $S = s$.
C	IDENT operation modifier: Evaluate address expression modulo $2^{17}-1$
c	00-77_8 BCD code of search character. The c address subfield may contain any symbol, value, or expression, that represents the 6-bit character code of the character for which the search is made, $00 \leq c \leq 77_8$.
ch	Channel designator for input/output instruction. The ch address subfield may contain a symbol, value, or expression that results in a nonrelocatable value $0 \leq ch \leq 7$.
cm	8-bit channel.mask. This address subfield may contain a symbol, constant, or expression that results in a nonrelocatable value $0 \leq cm \leq 2^8-1$.
D	D register
dc	BDP operation modifier: Indicates delimiting character; represented as right-adjusted BCD character string (mode 3).

Examples:

```
        MVZS,'K'              Delimiting character is K.
V       EQU       C'V'        V has mode 3.
        MVZS,V                Delimiting character is V.
```

E	48-bit E register.
EQ	IDENT and BDP operation modifier: Indicates equal.
GE	IDENT operation modifier: Indicates greater than or equal.
H	MONITOR operation modifier: Indicates half assembly or disassembly.
HI	BDP operation modifier: Indicates $(BCR)=01_2$ jump condition.

Term	Meaning
I	IDENT operation modifier: Indicates indirect addressing.
i	Increment or decrement. The i address subfield may contain a symbol, constant, or expression which results in a nonrelocatable value from 0 to 7.
INT	MONITOR operation modifier: Indicates interrupt on completion.
k	Shift count
ℓ	Field length of block. 0-177$_8$. The ℓ address subfield may be a symbol or an expression which results in a nonrelocatable value from 1 to 177$_8$.
LOW	BDP operation modifier: Indicates (BCR)=10$_2$ jump condition.
LR	BDP operation modifier: Indicates left-to-right scan.
LT	IDENT operation modifier: Indicates less than.
ℓ_r	Number of characters in field R.
ℓ_s	Number of characters in field S.
m	15-bit word address, first operand, or jump address. The m address subfield may contain a symbol, $, a constant, an expression, or a literal.
M	Actual operand or jump address as modified; $M = m+(B^b)$.
N	MONITOR operation modifier: Indicates no assembly or disassembly.
n	Same as m, second operand address.
NE	IDENT and BDP operation modifier: Indicates not equal.
P	15 (or 17)-bit P register.
Q	24-bit Q register.
r	17-bit character address. The r address subfield may contain a symbol, literal, constant, external symbol, expression, or $.
R	Actual character address as modified; $R = r+(B^b)$.
RL	BDP operation modifier: Indicates right-to-left scan.
RNI	Abbreviation for read next instruction at. For example, RNI P+1 means read the next instruction at the current location plus 1 of the P register.
s	Same as r, second operand address.
S	IDENT operation modifier: Sign extension if S present; no sign extension if S omitted. Other: Same as R, second operand address; $S = s+(B^b)$.
sc	Scan character
v	6-bit address in register file. The v address subfield may contain a symbol, constant, or expression which results in a nonrelocatable value 0 to 63$_{10}$.

Term	Meaning
w	Page index file address.
x	Connect code or interrupt mask. The x address subfield may contain a symbol, constant, or expression that results in a nonrelocatable value $0 \leq x \leq 2^{12}-1$.
y	15-bit operand. The y address subfield may contain a symbol, * or **, constant, an expression, or a literal.
ZRO	BDP operation modifier: Indicates (BCR)=0 jump condition.
()	Operation analysis symbol indicating the contents of. For example, (A) means the contents of the A register.
→	Operation analysis symbol indicating replace. For example, (M)→(A) means replace the contents of the A register with the contents of the M operand field.

Procedures for COMPASS pseudo instructions precede the tables.

IDENT procedures are grouped according to instruction types as:

Transfers

Arithmetic operations

Character operations

Decisions

Jumps, pauses, and stops

Interrupt operations

No-operation instruction

Shift instructions

Logical instructions

MONITOR procedures are grouped according to instruction types as:

Transfers

Decisions

Jumps, pauses, and stops

Input/output operations

Interrupt operations

BDP procedures are not divided into subgroups.

PROCEDURE REFERENCES FOR COMPASS PSEUDO INSTRUCTIONS

IDENT sym

The IDENT procedure names a subprogram and provides control information for META. The operand field contains a 1-8 character symbol naming the subprogram. The procedure contains a SECP directive that places the name on the IDC card of the relocatable object subprogram deck. The label field is defined as the value of the location counter.

The subprogram name is not an entry point name and cannot be referred to within the source subprogram. Each subprogram must have a SECP directive or IDENT instruction preceding all but the LIBS, UNIT, or list control directives.

Lines of code following IDENT are assembled, using the location control counter, until the next SECP, SECA, SECD, or ORG directive.

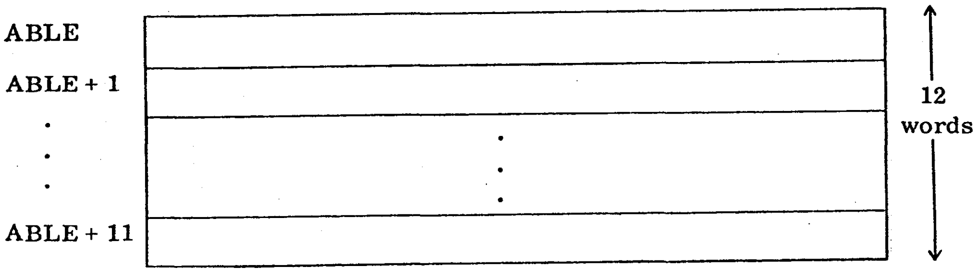
BSS m

BSS reserves and labels a block of words in any area. The label field is blank or contains a symbol defined as the 15-bit relocatable word address of the first word in the block.

The operand field specifies the number of words to be reserved. It must contain a constant, a symbol, or an address expression that results in a nonrelocatable value.

Example:

```
ABLE BSS 12
```



A double asterisk is illegal in the operand field. A symbol in the operand field must be defined in the label field of a preceding instruction.

A negative operand field such as -O '2' is interpreted as O'77777775'. META reserves 77777775_8 words.

If the operand field is in error or is zero, no storage is reserved but the label field symbol is defined. If the operand field is zero, and a byte-oriented instruction immediately precedes the BSS, the next instruction that uses space begins with a new word.

BSS, C m

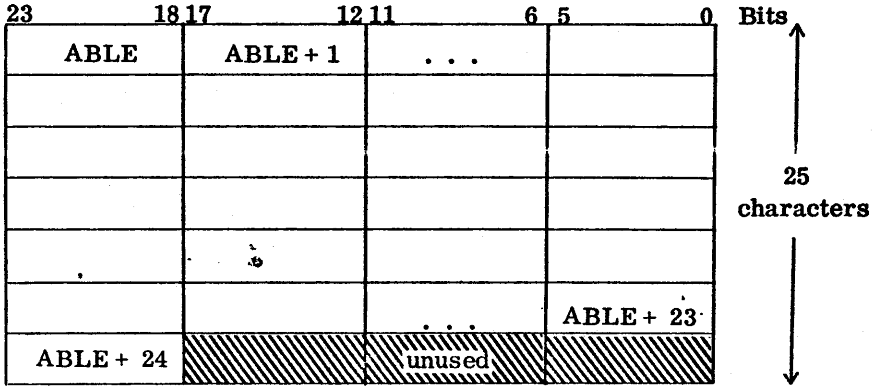
BSS, C reserves and labels a block of bytes. The label field is blank or contains a symbol defined as a 17-bit relocatable address of the first byte (BCD character position) in the block to be reserved. The operand field specifies the number of bytes reserved. It must contain a constant, a symbol, or an address expression that results in a non-relocatable value.

A negative operand field such as -O '2' is interpreted as O '77777775'. META reserves 77777775_8 bytes.

A zero operand does not reserve space but the label field symbol is defined.

Example:

```
ABLE BSS,C 25
```



DEC $d_1, d_2, \ldots, d_n$

DEC generates one computer word for each decimal value in the operand field. The label field is blank or contains a symbol defined as a 15-bit relocatable address of the first word generated. The operand field contains values, symbols, or expressions that result in decimal values.

Example:

```
DEC -38,429,18
```

Generates three words.

DECD $d_1, d_2, \ldots, d_n$

DECD generates two computer words in 48-bit internal floating-point format for each real (floating-point) value in the operand field. The label field is blank or contains a symbol defined as the 15-bit relocatable address of the first word generated. The operand field contains values, symbols, or expressions that result in real or floating-point values.

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Transfer	AEU		$(A) \rightarrow E_{47-24}$	23 17 14 00 55 6 ▨
	AQE		$(A, Q) \rightarrow E_{47-00}$	23 17 14 00 55 7 ▨
	EAQ		$(E_{47-00}) \rightarrow A, Q$	23 17 14 00 55 3 ▨
	ELQ		$(E_{47-24}) \rightarrow Q$	23 17 14 00 55 1 ▨
	ENA	y	$0 \rightarrow A$, then $y \rightarrow A_{14-00}$	23 17 14 00 14 6 y
	ENA, S	y	$0 \rightarrow (A)$, then $y \rightarrow A_{14-00}$, sign extended	23 17 14 00 14 4 y
	ENI	y, b	$0 \rightarrow B^b$; then $y \rightarrow B^b$; becomes a no-operation instruction if b= 0	23 17 14 00 14 0 b y
	ENQ	y	$0 \rightarrow Q$, then $y \rightarrow Q_{14-00}$	23 17 14 00 14 7 y
	ENQ, S	y	$0 \rightarrow Q$, then $y \rightarrow Q_{14-00}$, sign extended	23 17 14 00 14 5 y
	EUA		$(E_{47-24}) \rightarrow A$	23 17 14 00 55 2 ▨
	LCA, I	m, b	Complement of $(M) \rightarrow A$	23 17 14 00 24 a b m

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Transfer	LCAQ, I	m, b	Complement of (M)→A; complement of (M+1)→Q	23 17 14 00 26 a b m
	LDA, I	m, b	(M)→A	23 17 14 00 20 a b m
	LDAQ, I	m, b	(M)→A, (M+1)→Q	23 17 14 00 25 a b m
	LDI, I	m, b	$(M_{14-00}) \rightarrow B^{b}$	23 17 14 00 54 a b m
	LDQ, I	m, b	(M)→Q	23 17 14 00 21 a b m
	QEL		$(Q) \rightarrow E_{23-00}$	23 17 14 00 55 5 ▨
	RIS		Relocate to instruction state	23 17 14 00 55 0 ▨
	ROS		Relocate to operand state	23 17 14 00 55 4 ▨
	STA, I	m, b	(A)→M	23 17 14 00 40 a b m
	STAQ, I	m, b	(A)→M, (Q)→M+1	23 17 14 00 45 a b m
	STI, I	m, b	$(B^{b}) \rightarrow M_{14-00}$	23 17 14 00 47 a b m

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Transfer	STQ,I	m,b	(Q)→M	23 17 14 00 41 a b m
	SWA,I	m,b	(A_{14-00})→M_{14-00}	23 17 14 00 44 a b m
	TAI	b	(A_{14-00})→B^b; becomes a no-operation instruction if b=0	23 17 14 11 00 53 1 b 0 ▨
	TAM	v	(A)→v	23 17 14 11 05 00 53 1 ▨ 2 ▨ v
	TIA	b	0→A, (B)→A_{14-00}; if b=0, 0→(A)	23 17 14 11 00 53 0 b 0 ▨
	TIM	v,b	(B^b)→v_{14-00}	23 17 14 11 05 00 53 1 b 3 ▨ v
	TMA	v	(v_{14-00})→A	23 17 14 11 05 00 53 0 ▨ 2 ▨ v
	TMI	v,b	(v_{14-00})→B^b	23 17 14 11 05 00 53 0 b 3 ▨ v
	TMQ	v	(v)→Q	23 17 14 11 05 00 53 0 ▨ 1 ▨ v
	TQM	v	(Q)→v	23 17 14 11 05 00 53 1 ▨ 1 ▨ v

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Arithmetic	ADA, I	m, b	(A)+(M)→A	23 17 14 00 30 a b m
	ADAQ, I	m, b	(A, Q)+(M, M+1)→A, Q	23 17 14 00 32 a b m
	AIA	b	(A)+(B^b)→A, sign of (B^b) is extended prior to addition	23 17 14 11 00 53 0 b 4 ▨
	AQA		(A)+(Q)→A	23 17 14 11 00 53 0 4 ▨
	DVA, I	m, b	(A, Q)/(M)→A, remainder →Q	23 17 14 00 51 a b m
	DVAQ, I	m, b	(A, Q, E)/(M, M+1)→A, Q, remainder with sign extended →E	23 17 14 00 57 a b m
	FAD, I	m, b	Floating-point addition of (M, M+1) to (A, Q)→A, Q	23 17 14 00 60 a b m
	FDV, I	m, b	Floating-point division of (A, Q) by (M, M+1)→A, Q; remainder with sign extended →(E)	23 17 14 00 63 a b m
	FMU, I	m, b	Floating-point multiplication of (A, Q) and (M, M+1)→A, Q	23 17 14 00 62 a b m
	FSB, I	m, b	Floating-point subtraction of (M, M+1) from (A, Q)→A, Q	23 17 14 00 61 a b m

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Arithmetic	IAI	b	$(A)+(B^b)\rightarrow B^b$, sign of B^b is extended prior to addition	23 17 14 11 00 53 │ 1 │ b │ 4 │ ▨
	INA	y	Increase (A) by y	23 17 14 00 15 │ 6 │ y
	INA,S	y	Increase (A) by y, sign of y is extended	23 17 14 00 15 │ 4 │ y
	INI	y,b	Increase (B^b) by y, signs of y and B^b extended; becomes a no-operation if b=0	23 17 14 00 15 │ 0 │ b │ y
	INQ	y	Increase (Q) by y	23 17 14 00 15 │ 7 │ y
	INQ,S	y	Increase (Q) by y, sign of y extended	23 17 14 00 15 │ 5 │ y
	MUA,I	m,b	(A)*(M)→Q,A	23 17 14 00 50 │ a │ b │ m
	MUAQ,I	m,b	(A,Q)*(M,M+1)→A,Q,E	23 17 14 00 56 │ a │ b │ m
	RAD,I	m,b	(M)+(A)→M	23 17 14 00 34 │ a │ b │ m
	SBA,I	m,b	(A) - (M)→A	23 17 14 00 31 │ a │ b │ m
	SBAQ,I	m,b	(A,Q) - (M,M+1)→A,Q	23 17 14 00 33 │ a │ b │ m

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Character	ECHA	r	0→A, then character address r→A_{16-00}	23 17 00 11 0 z
	ECHA,S	r	0→(A), then character address r→A_{16-00}, sign extended	23 17 00 11 1 z
	LACH	r,1	0→A, character in (R)→A_{05-00}	23 17 00 22 b r
	LQCH	r,2	0→Q, character in (R)→Q_{05-00}	23 17 00 23 b r
	SACH	r,2	Character in (A_{05-00})→R	23 17 00 42 b r
	SCHA,I	m,b	Character address in (A_{16-00}) →M_{16-00}	23 17 14 00 46 a b m
	SQCH	r,1	Character in (Q_{05-00})→R, use (B^1) to index	23 17 00 43 b r
Decision	AQJ,mod	m	If condition is satisfied, RNI m, otherwise, RNI P+1 mod test condition j EQ (A) = (Q) 0 NE (A) ≠ (Q) 1 GE (A) ≥ (Q) 2 LT (A) < (Q) 3	23 17 14 00 03 1 j m

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Decision	ASE	y	If y = (A_{14-00}), RNI P+ 2, otherwise, RNI P+1	23 17 14 00 04 │ 6 │ y
	ASE,S	y	If y = (A_{14-00}), RNI P+2, otherwise, RNI P+1, sign of y is extended	23 17 14 00 04 │ 4 │ y
	ASG	y	If (A) ≥ y, RNI P+2, otherwise, RNI P+1	23 17 14 00 05 │ 6 │ y
	ASG,S	y	If (A) ≥ y, RNI P+2, otherwise, RNI P+1, sign of y is extended	23 17 14 00 05 │ 4 │ y
	AZJ,mod	m	If condition is satisfied, RNI m, otherwise, RNI P+1	23 17 14 00 03 │ 0 │ j │ m
			mod test condition j	
			EQ (A) = 0 0 NE (A) ≠ 0 1	Positive zero = negative zero
			GE (A) ≥ 0 2 LT (A) < 0 3	Negative zero < positive zero
	IJD	m,b	If (B^b) =0, RNI P+1; if (B^b) ≠ 0, $(B^b) - 1 \rightarrow B^b$, RNI m; becomes a no-operation instruction if b=0	23 17 14 00 02 │ 1 │ b │ m
	IJI	m,b	If (B^b) = 0, RNI P+1; if (B^b) ≠ 0, $(B^b) + 1 \rightarrow B^b$, RNI m; becomes no-operation instruction if b=0	23 17 14 00 02 │ 0 │ b │ m
	ISD	y,b	For b ≠ 0, if (B^b) = y, clear B^b and RNI P+2; if (B^b) ≠ y, $(B^b) - 1 \rightarrow B^b$, RNI P+1 For b=0, if y = 0, RNI P+2; if y ≠ 0, RNI P+1	23 17 14 00 10 │ 1 │ b │ y

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Decision	ISE	y,b	For b ≠ 0, if y = (B^b), RNI P+2, otherwise, RNI P+1 For b=0, if y = 0, RNI P+2, otherwise, RNI P+1	23 17 14 00 04 │ 0 │ b │ y
	ISG	y,b	For b ≠ 0, if $(B^b) \geq y$, RNI P+2, otherwise, RNI P+1 For b=0, if $y \geq 0$, RNI P+2, otherwise, RNI P+1	23 17 14 00 05 │ 0 │ b │ y
	ISI	y,b	For b ≠ 0, if $(B^b) = y$, clear B^b and RNI P+2; if $(B^b) \neq y$, $(B^b) + 1 \rightarrow B^b$, RNI P+1 For b=0, if y = 0, RNI P+2; if y ≠ 0, RNI P+1	23 17 14 00 10 │ 0 │ b │ y
	MEQ	m,i	$(B^1) - i \rightarrow B^1$; if (B^1) negative, RNI P+1; if (B^1) positive, test (A) = logical product of (Q) and (M); if true, RNI P+2, if false, repeat sequence Designator i — Decrement Interval 1 — 1 2 — 2 3 — 3 4 — 4 5 — 5 6 — 6 7 — 7 0 — 8	23 17 14 00 06 │ i │ m
	MTH	m,i	$(B^2) - i \rightarrow B^2$, if (B^2) negative, RNI P+1, if (B^2) positive, test (A) ≥ logical product of (Q) and (M); if true, RNI P+2; if false, repeat sequence; designation table same as for MEQ	23 17 14 00 07 │ i │ m

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Decision	QSE	y	If y = (Q_{14-00}), RNI P+2, otherwise, RNI P+1	23 17 14 00 04 7 y
	QSE,S	y	If y = (Q), RNI P+2, otherwise, RNI P+1, sign of y is extended	23 17 14 00 04 5 y
	QSG	y	If $(Q_{14-00}) \geq$ y, RNI P+2, otherwise, RNI P+1	23 17 14 00 05 7 y
	QSG,S	y	If (Q) $\geq$ y, RNI P+2, otherwise, RNI P+1, sign of y is extended	23 17 14 00 05 5 y
Jumps, Pauses, and Stops	HLT	m	Unconditional stop, RNI m upon restarting	23 17 14 00 00 0 m
	RTJ	m	$(P)+1 \rightarrow m_{14-00}$, RNI m+ 1	23 17 14 00 00 7 m
	SJj	m	If SELECT JUMP j (where j = 1-6) is set, jump to m; otherwise, RNI P+ 1	23 17 14 00 00 j m
	UJP,I	m,b	Unconditional jump to M	23 17 14 00 01 a b m
Interrupt Operations	DINT		Disable interrupt control	23 17 11 00 77 73 ▨
	EINT		Interrupt control enabled; allows one more instruction to be executed before interrupt	23 17 11 00 77 74 ▨

TABLE B-1. IDENT PROCEDURE REFERENCES

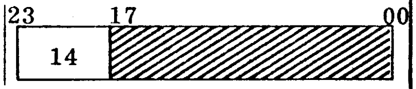
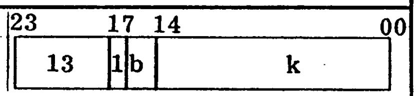
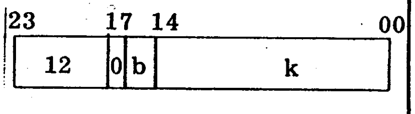
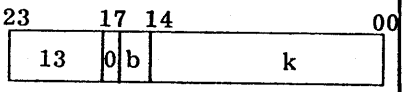
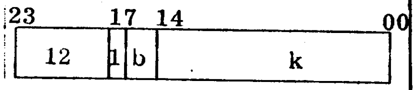
Type	Command Field	Operand Field	Operation	Object Code
No-operation	NOP		No operation (assembled NOP), RNI P+1	
Shift Instructions	SCAQ	k, b	Shift (A, Q) left end around until upper 2 bits of A are unequal; residue K = k- shift count; if b = 1, 2, or 3, K → B^b; if b = 0, K is discarded	
	SHA	k, b	Shift (A); shift count K=k + (B^b) (signs of k and B^b extended); if bit 23 of K=1, shift right; complement of lower 6 bits equals shift magnitude; if bit 23 of K=0, shift left; lower 6 bits equal shift magnitude; left shifts end around; right shifts end off	
	SHAQ	k, b	Shift (A, Q) as one register; shift count K=k + (B^b) (signs of k and B^b extended); if bit 23 of K=1, shift right and complement of lower 6 bits equals shift magnitude; if bit 23 of K = 0, shift left and lower 6 bits equal shift magnitude; left shifts end around; right shifts end off	
	SHQ	k, b	Shift (Q); shift count K=k + (B^b) (signs of k and B^b extended); if bit 23 of K = 1, shift right, complement of lower 6 bits equals shift magnitude; if bit 23 of K = 0, shift left, lower 6 bits equal shift magnitude; left shifts end around; right shifts end off	

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Shift Instructions	SSH	m	Test sign of (m), shift (m) left one place, end around and replace in storage; negative sign, RNI P+2, otherwise RNI P+1	23 17 14 00 10 | 0 | m
Logical Instructions	ANA	y	Logical product (AND) of y and (A)→A	23 17 14 00 17 | 6 | y
	ANA, S	y	Logical product (AND) of y and (A)→A, sign of y extended	23 17 14 00 17 | 4 | y
	ANI	y, b	Logical product (AND) of y and $(B^b) \rightarrow B^b$; becomes no-operation instruction if b=0	23 17 14 00 17 | 0 | b | y
	ANQ	y	Logical product (AND) of y and (Q)→Q	23 17 14 00 17 | 7 | y
	ANQ, S	y	Logical product (AND) of y and (Q)→Q, sign of y extended	23 17 14 00 17 | 5 | y
	LDL, I	m, b	Logical product (AND) of (M) and (Q)→A	23 17 14 00 27 | a | b | m
	LPA, I	m, b	Logical product (AND) of (M) and (A)→A	23 17 14 00 37 | a | b | m
	SCA, I	m, b	Where (M) contains a 1 bit, complement the corresponding bit in A .	23 17 14 00 36 | a | b | m

TABLE B-1. IDENT PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Logical Instructions	SSA,I	m,b	Where (M) contains a 1 bit, set the corresponding bit in A to 1	23 17 14 00 35 a b m
	XOA	y	Selective complement (exclusive OR) of y and (A)→A	23 17 14 00 16 6 y
	XOA,S	y	Selective complement (exclusive OR) of y and (A)→A, sign of y extended	23 17 14 00 16 4 y
	XOI	y,b	Selective complement (exclusive OR) of y and $(B^b) \rightarrow B^b$; becomes no-operation instruction if b = 0	23 17 14 00 16 0 b y
	XOQ	y	Selective complement (exclusive OR) of y and (Q)→Q	23 17 14 00 16 7 y
	XOQ,S	y	Selective compl:ment (exclusive OR) of y and (Q)→Q, sign of y extended	23 17 14 00 16 5 y

TABLE B-2. MONITOR PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Transfer	ACI		$(A_{02-00}) \rightarrow$ channel index register	23 17 11 00 77 | 54 | ▨
	ACR		$(A_{05-00}) \rightarrow$ condition register	23 17 11 08 00 77 | 63 | 4 | 000
	AIS		$(A_{02-00}) \rightarrow$ instruction state register	23 17 11 08 00 77 | 66 | 4 | 000
	AOS		$(A_{02-00}) \rightarrow$ operand state register	23 17 11 00 77 | 66 | 0000
	APF	w, 2	$(A_{11-00}) \underset{2}{\rightarrow}$ page file index w; if b = 1, (B^2) used for indexing	23 17 10 06 00 77 | 64 | b | ▨ | w
	CIA		$0 \rightarrow (A)$, then channel index register $\rightarrow A_{02-00}$	23 17 11 00 77 | 55 | ▨
	CRA		Condition register $\rightarrow A_{05-00}$; clear condition register (CR) Significance 00 Boundary jump 01 Destructive load A 02 OSR in use 03 Program state jump 04 Interrupts enabled 05 Program state	23 17 11 00 77 | 63 | 0000
	ISA		$0 \rightarrow (A)$, instruction state register $\rightarrow A_{02-00}$	23 17 11 08 00 77 | 67 | 4 | 000

TABLE B-2. MONITOR PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Transfer	JAA		Last executed jump address →A_{14-00}	23 17 11 00 77 56 ▨
	LBR	m	Load BCR and restore BDP conditions from data at m	23 17 14 00 70 6 m
	OSA		0→(A); operand state register →A_{02-00}	23 17 11 00 77 67 0000
	PFA	w, 2	0→A, then (page index file w) →A_{11-00}; if b is 1, (B^2) used for indexing	23 17 11 00 77 65 b ▨ w
	RCR		Subcondition register→condition register	23 17 11 08 00 77 63 4 000
	SBJP		Set condition register for boundary jump; system transfers from monitor state to program state when next jump occurs	23 17 11 00 77 62 0000
	SPR	m	Store contents of BCR and BDP conditions at m for interrupt recovery.	23 17 14 00 70 7 m
	SDL		Set 01 in condition register to flag destructive load so that upon next LDA instruction: 1. (M)→A 2. 77777777→M 3. 0→condition register	23 17 11 08 00 77 62 4 000
	SRA		0→A; subcondition register→ A_{02-00}	23 17 11 00 77 63 0000

TABLE B-2. MONITOR PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Decision	CPR,I	m,b	(M) > (A), RNI P+1 (Q) > (M), RNI P+2 (A) ≥ (M) ≥ (Q), RNI P+3 (A) and (Q) are unchanged	23 17 14 00 52 | a | b | m
	TMAV		Initiate memory request; if reply occurs within 5 usec, address exists, RNI P+2; if not, address does not exist, RNI P+1; storage address tested is (B^2) with operand state register) or zero appended	23 17 11 00 77 | 61 | 0000

Pause Sensing Mask

Mask Bits	Mask Codes	Condition	Notes
00	0001	I/O channel 0 busy	Channel read or write operation in progress, the External MC logic within the channel is set, or a Reply or Reject from a previous operation is still present at the channel
01	0002	1	
02	0004	2	
03	0010	3	
04	0020	4	
05	0040	5	
06	0100	6	
07	0200	7	
08	0400	Typewriter busy	Typewriter I/O in progress
09	1000	Typewriter NOT finish	Finish logic not set
10	2000	Typewriter NOT repeat	Repeat logic not set
11	4000	Search/Move control busy	Search or Move operation in progress

TABLE B-2. MONITOR PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Jumps, Pauses, and Stops	PAUS	x	Sense busy lines; if 1 appears on a line corresponding to 1 bits in x, do not advance P; if P is inhibited for longer than 40 ms, read reject instruction from P+1; if no comparison, RNI P+2	23 17 11 00 77 60 x
	PRP	x	Same as PAUS, except real-time clock cannot increment during the pause.	23 17 11 00 77 61 x
	SLS		Program stops if selective stop switch is on; upon restarting RNI P+1	23 17 11 00 77 70 ▨
	UCS		Unconditional stop; upon restarting RNI P+1	23 17 11 00 77 77 ▨
Input/Output	CLCA	cm	Clear the specified channel, but not external equipment	23 17 11 07 00 77 51 ▨ 2 ▨ cm
	CON	x, ch	If channel ch is busy, reject instruction, RNI P+1. If channel ch is not busy, send 12-bit connect code (x) on channel ch with connect enable, RNI P+2	23 17 14 11 00 77 0 ch x
	COPY	ch	External status code from I/O channel ch→A_{11-00}, (interrupt mask register)→A_{23-12}, RNI P+1	23 17 14 11 00 77 2 ch 0000

INTERRUPT MASK REGISTER BIT ASSIGNMENTS

Mask Bit Positions	Mask Codes (x)	Interrupt Conditions Represented
00	0001	I/O Channel 0 (includes interrupts generated within
01	0002	1 the channel and external equipment
02	0004	2 interrupts)
03	0010	3
04	0020	4
05	0040	5
06	0100	6
07	0200	7
08	0400	Real-time clock
09	1000	Exponent overflow/underflow & BCD faults
10	2000	Arithmetic overflow & divide faults
11	4000	Search/Move completion

TABLE B-2. MONITOR PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Input/Output	CTI		Set console typewriter input. Beginning character address must be in location 23 of register file, last character + 1 must be in location 33 of the file	23 17 11 00 77 75 ▨
	CTO		Set console typewriter output Beginning character address must be in location 23 of register file, last character + 1 must be in location 33 of the file	23 17 11 00 77 76 ▨
	EXS	x, ch	Sense external status; if 1 bits occur on status lines in any of the same positions as 1 bits in the mask, RNI P+1; if no comparison, RNI P+2	23 17 14 11 00 77 2 ch x

TABLE B-2. MONITOR PROCEDURE REFERENCES

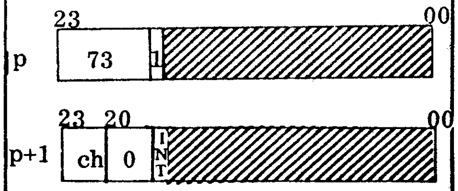
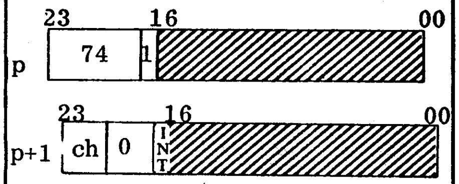
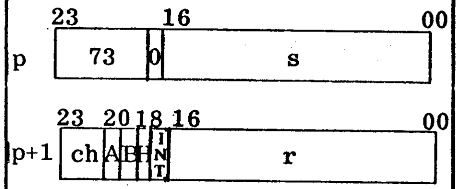
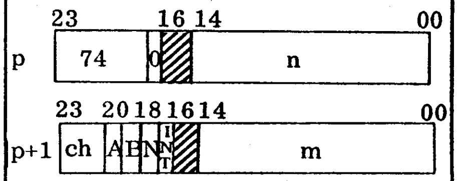
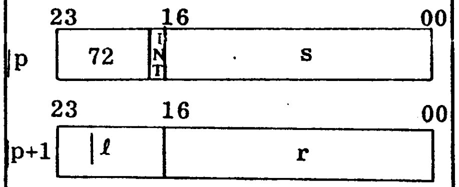
Type	Command Field	Operand Field	Operation	Object Code
Input/Output	INAC, INT	ch	(A) is cleared and a 6-bit character is transferred from a peripheral device to the lower 6 bits of A	 23 00 p 73 1 23 20 00 p+1 ch 0 INT
	INAW, INT	ch	(A) is cleared and a 12- or 24-bit word is read from a peripheral device into the lower 12 bits or all of A (word size depends on I/O channel)	 23 16 00 p 74 1 23 16 00 p+1 ch 0 INT
	INPC, INT, B, H, A	ch, r, s	A 6- or 12-bit character is read from a peripheral device and stored in memory at a given location	 23 16 00 p 73 0 s 23 20 18 16 00 p+1 ch A B H INT r
	INPW, INT, B, N, A	ch, m, n	Word address is placed in bits 14-00; 12- or 24-bit words are read from a peripheral device and stored in memory	 23 16 14 00 p 74 0 n 23 20 18 16 14 00 p+1 ch A B N INT m
	MOVE, INT	ℓ, r, s	Move ℓ characters from r to s; $0 \le \ell \le 127_{10}$	 23 16 00 p 72 INT s 23 16 00 p+1 ℓ r

TABLE B-2. MONITOR PROCEDURE REFERENCES

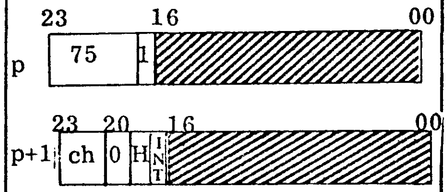
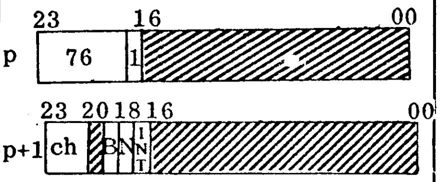
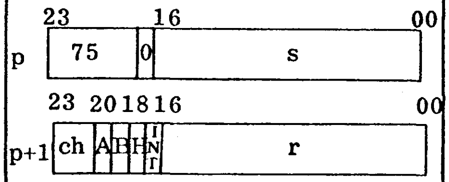
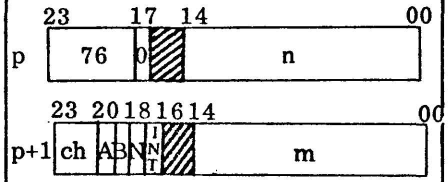
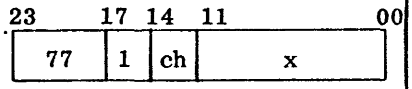
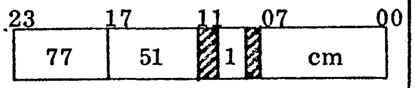
Type	Command Field	Operand Field	Operation	Object Code
Input/Output	OTAC, INT	ch	Character from (A_{05-00}) is sent to peripheral device, (A) retained	 23 16 00 p 75 1 23 20 16 00 p+1 ch 0 H INT
	OTAW, INT	ch	Transfers (A_{11-00}) or A_{23-00}, depending on type of I/O channel, to a peripheral device	 23 16 00 p 76 1 23 20 18 16 00 p+1 ch B N INT
	OUTC, INT, B, H, A	ch, r, s	Storage words assembled into 6- or 12-bit characters and sent to a peripheral device	 23 16 00 p 75 0 s 23 20 18 16 00 p+1 ch A B H INT r
	OUTW, INT, B, N, A	ch, m, n	Transfer 12- or 24-bit words from storage to a peripheral device	 23 17 14 00 p 76 0 n 23 20 18 16 14 00 p+1 ch A B N INT m
	SEL	x, ch	If channel ch is busy, read reject instruction from P+1; if not busy, send a 12-bit function code on channel ch with a function enable, RNI P+2	
Interrupt	CILO	cm	Lockout external interrupt on masked channels, cm, until channel is not busy	

TABLE B-2. MONITOR PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Interrupt	CINS	x, ch	Interrupt mask and internal status→A	23 17 14 11 00 | 77 | 3 | ch | 0000 |
	IAPR		Interrupt associated processor	23 17 11 00 | 77 | 57 | ▨ |
	INCL	x	Interrupt faults defined by x are cleared	23 17 11 00 | 77 | 50 | x |

Internal Status Sensing Mask

Masked Bit Positions	Mask Codes (x)	Interrupt Conditions Represented
00	0001	Parity error on channel ch
01	0002	Channel ch busy reading
02	0004	Channel ch busy writing
03	0010	External reject active on channel ch
04	0020	No-response reject active on channel ch
05	0040	†Illegal write
06	0100	Channel ch preset by CON or SEL, but no reading or writing in progress
07	0200	Internal I/O channel interrupt on channel ch upon: 1) completion of read or write operation, or 2) end-of-record
08	0400	†Exponent overflow/underflow fault (floating-point)
09	1000	†Arithmetic overflow fault (adder)
10	2000	†Divide fault
11	4000	†BCD fault

†Peripheral Equipment Reference Manual, Pub. No. 60108800

TABLE B-2. MONITOR PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
Interrupt	INS	x, ch	Sense internal status†; if 1 bits occur on status lines in any of the same positions as 1 bits in the mask, RNI P+1; if no comparison, RNI P+2	23 17 14 11 00 77 3 ch x
	INTS	x, ch	Sense for interrupt condition; if 1 bits occur simultaneously in interrupt lines and in the interrupt mask, RNI P+1; if not, RNI P+2	23 17 14 11 00 77 4 ch x
	IOCL	x	Clears I/O channel or search/ move control as defined by bits 00-07, 08, and 11 of x	23 17 11 00 77 51 x
	SBCD		Set BCD fault logic	23 17 11 00 77 72 ▨
	SCIM, I	x	Selectively clear interrupt mask register for each 1 bit in x; corresponding bit in the mask register is set to 0	23 17 11 00 77 53 x
	SFPF		Set floating-point fault logic	23 17 11 00 77 71 ▨
	SSIM	x	Selectively set interrupt mask register for each 1 bit in x; corresponding bit in the mask register is set to 1	23 17 11 00 77 52 x

†Internal faults are cleared when sensed.

TABLE B-3. BDP PROCEDURE REFERENCES

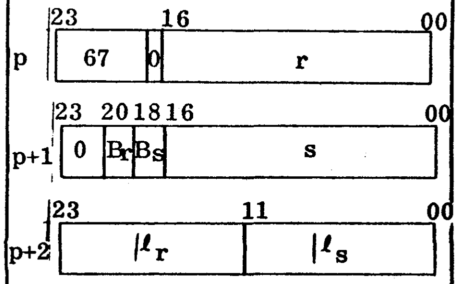
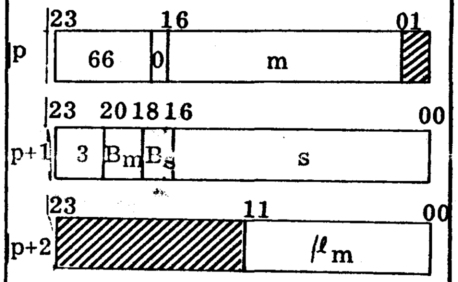
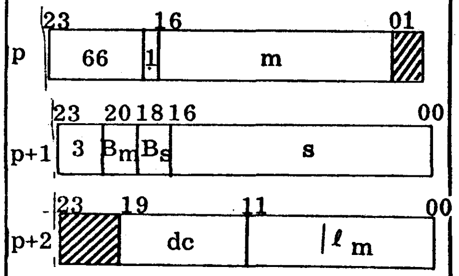
Type	Command Field	Operand Field	Operation	Object Code
BDP	ADM	r, B_r, ℓ_r, s, B_s, ℓ_s	Add field R to field S→ field S	 23 16 00 p 67 0 r 23 20 18 16 00 p+1 0 B_r B_s s 23 11 00 p+2 ℓ_r ℓ_s
	ATD	m, B_m, ℓ_m, s, B_s	Translate ASCII code field M→BCD character field S	 23 16 01 p 66 0 m 23 20 18 16 00 p+1 3 B_m B_s s 23 11 00 p+2 ℓ_m
	ATD, dc	m, B_m, ℓ_m, s, B_s	Translate ASCII code field M→BCD character field S with delimiting character possibility	 23 16 01 p 66 1 m 23 20 18 16 00 p+1 3 B_m B_s s 23 19 11 00 p+2 dc ℓ_m

TABLE B-3. BDP PROCEDURE REFERENCES

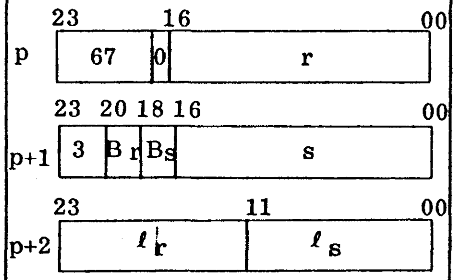
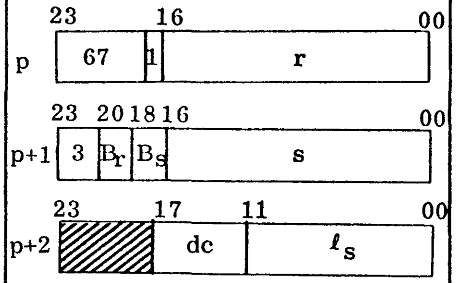
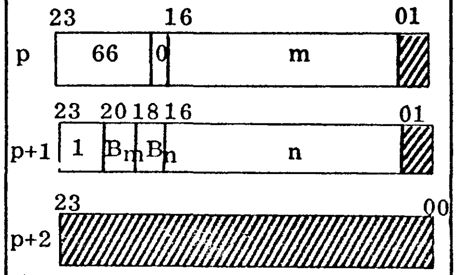
Type	Command Field	Operand Field	Operation	Object Code
BDP	CMP	$r, B_r, \ell_r,$ s, B_s, ℓ_s	Compare field R to field S from left to right, exit upon encountering ≠ characters	 23 16 00 p 67 0 r 23 20 18 16 00 p+1 3 Br Bs s 23 11 00 p+2 ℓr ℓs
	CMP.dc	$r, B_r, s,$ B_s, ℓ_s	Compare field R to field C from left to right, exit upon encountering ≠ characters; delimiting character possibility	 23 16 00 p 67 1 r 23 20 18 16 00 p+1 3 Br Bs s 23 17 11 00 p+2 dc ℓs
	CVBD	$m, B_n,$ n, B_n	Convert binary field M to BCD→field N	 23 16 01 p 66 0 m 23 20 18 16 01 p+1 1 Bm Bn n 23 00 p+2

TABLE B-3. BDP PROCEDURE REFERENCES

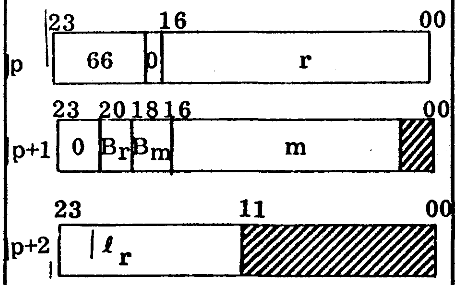
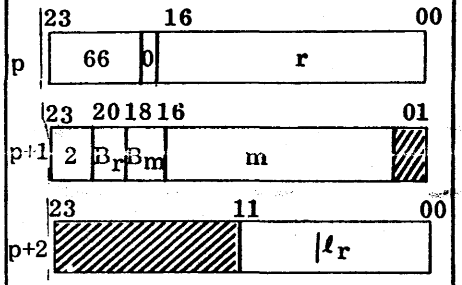
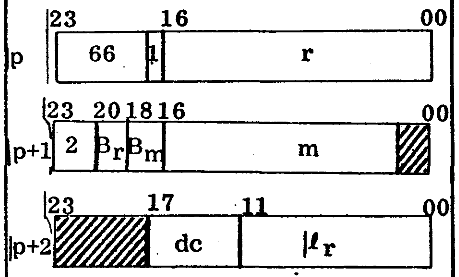
Type	Command Field	Operand Field	Operation	Object Code
BDP	CVDB	r, B_r, ℓ_r, m, B_m	Convert BCD field R to binary→field M	 p 23 16 00 66 0 r p+1 23 20 18 16 00 0 Br Bm m p+2 23 11 00 ℓr
	DTA	r, B_r, ℓ_r, m, B_m	Translate BCD field R to ASCII code→field M	 p 23 16 00 66 0 r p+1 23 20 18 16 01 2 Br Bm m p+2 23 11 00 ℓr
	DTA, dc	r, B_r, ℓ_r, m, B_m	Translate BCD field R to ASCII code→field M; delimiting character possibility	 p 23 16 00 66 1 r p+1 23 20 18 16 00 2 Br Bm m p+2 23 17 11 00 dc ℓr

TABLE B-3. BDP PROCEDURE REFERENCES

Type	Command Field	Operand Field	Operation	Object Code
BDP	EDIT	$r, B_r, \ell_r,$ s, B_s, ℓ_s	Field R→field S with COBOL type of editing specified by picture previously stored in field S	23 16 00 p 64 1 r 23 20 18 16 00 p+1 4 B_r B_s s 23 11 00 p+2 ℓ_r ℓ_s
	FRMT	$r, B_r, \ell_r,$ s, B_s, ℓ_s	Move field R→field S; replace leading zeros with blanks; insert a comma after every three characters moved, insert a decimal point in third lowest order position in S field	23 16 00 p 64 0 r 23 20 18 16 00 p+1 4 B_r B_s s 23 11 00 p+2 ℓ_r ℓ_s
	JMP, mod	m	Test status of BCR = condition specified by mod and jump to m if true; otherwise, RNI P+1 mod (BCR) j HI 01_2 0 ZRO 00 1 LOW 10 2	23 17 14 00 70 j m

TABLE B-3. BDP PROCEDURE REFERENCES

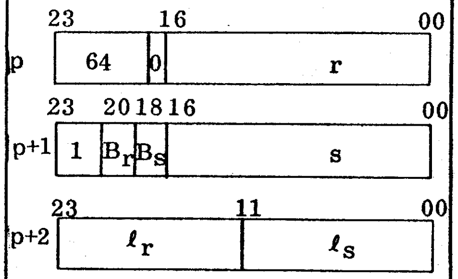
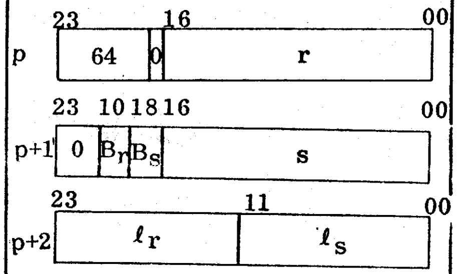
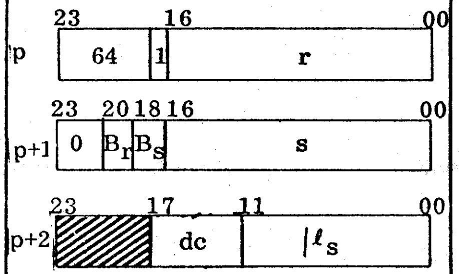
Type	Command Field	Operand Field	Operation	Object Code
BDP	MVBF	r, B_r, ℓ_r, s, B_s, ℓ_s	Move characters from field R →field S; if field S > field R, blank fill	 23 16 00 p 64 0 r 23 20 18 16 00 p+1 1 B_r B_s s 23 11 00 p+2 ℓ_r ℓ_s
	MVE	r, B_r, ℓ_r, s, B_s, ℓ_s	Move characters from field R→field S according to parameters	 23 16 00 p 64 0 r 23 10 18 16 00 p+1 0 B_r B_s s 23 11 00 p+2 ℓ_r ℓ_s
	MVE,dc	r, B_r, s, B_s, ℓ_s	Move characters from field R →field S; delimiting character possibility	 23 16 00 p 64 1 r 23 20 18 16 00 p+1 0 B_r B_s s 23 17 11 00 p+2 dc ℓ_s

TABLE B-3. BDP PROCEDURE REFERENCES

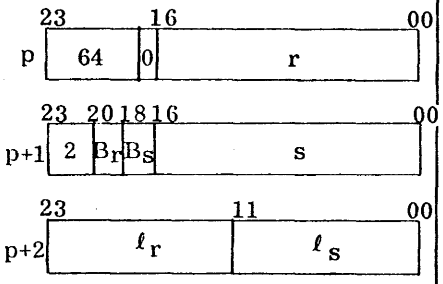
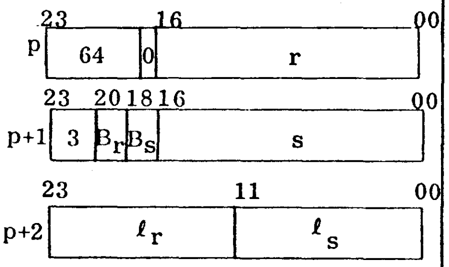
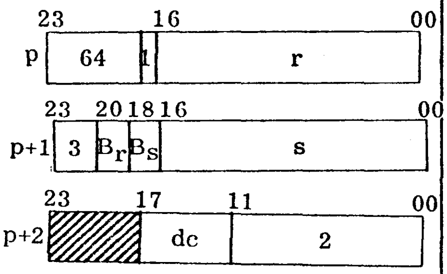
Type	Command Field	Operand Field	Operation	Object Code
BDP	MVZF	$r, B_r, \ell_r, s, B_s, \ell_s$	Move characters from field R → field S; if field S > field R, zero fill	 p 23 16 00 64 0 r p+1 23 20 18 16 00 2 B_r B_s s p+2 23 11 00 ℓr ℓs
	MVZS	$r, B_r, \ell_r, s, B_s, \ell_s$	Move characters from field R → field S; suppress leading zeros	 p 23 16 00 64 0 r p+1 23 20 18 16 00 3 Br Bs s p+2 23 11 00 ℓr ℓs
	MVZS, dc	r, B_r, s, B_s, ℓ_s	Move characters from field R → field S; suppress leading zeros; delimiting character possibility	 p 23 16 00 64 1 r p+1 23 20 18 16 00 3 Br Bs s p+2 23 17 11 00 dc 2

TABLE B-3. BDP PROCEDURE REFERENCES

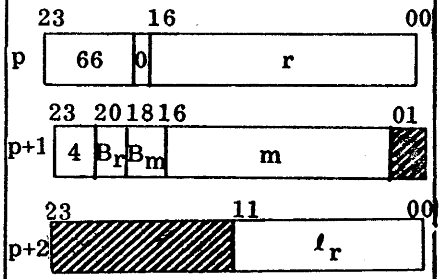
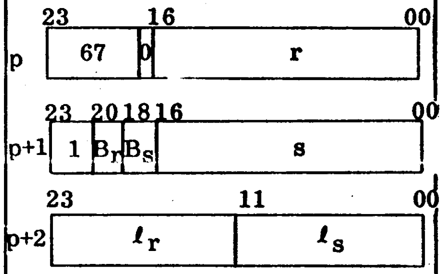
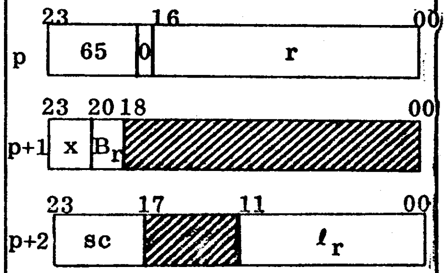
Type	Command Field	Operand Field	Operation	Object Code
BDP	PAK	r, B_r, ℓ_r, m, B_m	Convert and pack a 6-bit numeric BCD field R to a 4-bit numeric BCD field and store the result in field M	 23 16 00 p 66 0 r 23 20 18 16 01 p+1 4 B_r B_m m 23 11 00 p+2 ℓ_r
	SBM	r, B_r, ℓ_r, s, B_s, ℓ_s	Subtract field beginning at R from field beginning at S→ field beginning at S	 23 16 00 p 67 0 r 23 20 18 16 00 p+1 1 B_r B_s s 23 11 00 p+2 ℓ_r ℓ_s
	SCAN, dir, mod	r, B_r, ℓ_r, sc	Scan field beginning at R dir, mod — x LR, EQ — Left to right stop on = — 0 RL, EQ — Right to left stop on = — 1 LR, NE — Left to right stop on ≠ — 2 RL, NE — Right to left stop on ≠ — 3	 23 16 00 p 65 0 r 23 20 18 00 p+1 x B_r 23 17 11 00 p+2 sc ℓ_r

TABLE B-3. BDP PROCEDURE REFERENCES

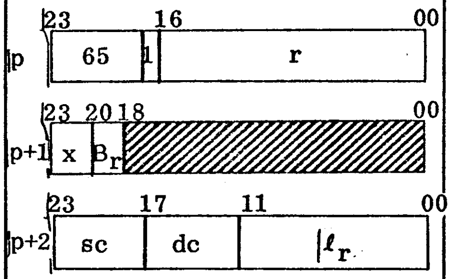
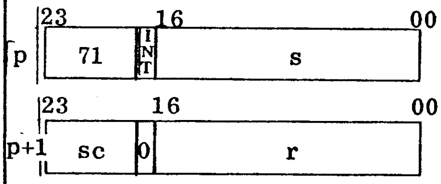
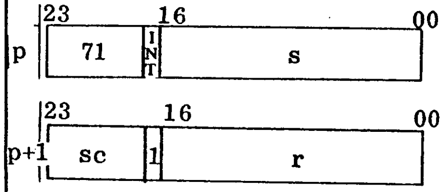
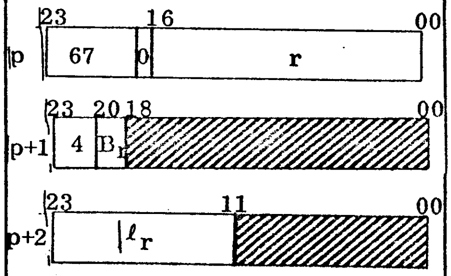
Type	Command Field	Operand Field	Operation	Object Code
BDP	SCAN, dir, mod, dc	r, B_r, ℓ_r, sc	Scan field beginning at R, delimiting possibility dir, mod — x LR, EQ — Left to right, stop on = — 0 RL, EQ — Right to left, stop on = — 1 LR, NE — Left to right, stop on ≠ — 2 RL, NE — Right to left, stop on ≠ — 3	 23 16 00 p 65 1 r 23 20 18 00 p+1 x B_r 23 17 11 00 p+2 sc dc ℓ_r
	SRCE, INT	c, r, s	Search for equality of character c in list beginning at r until an equal character is found, or until character at s is reached; $0 \le c \le 63_{10}$ Operation commences while main control continues at P+3.	 23 16 00 p 71 INT s 23 16 00 p+1 sc 0 r
	SRCN, INT	c, r, s	Inequality search; same as SRCE	 23 16 00 p 71 INT s 23 16 00 p+1 sc 1 r
	TST	r, B_r, ℓ_r	Test field R for −, 0, or +	 23 16 00 p 67 0 r 23 20 18 00 p+1 4 B_r 23 11 00 p+2 ℓ_r

TABLE B-3. BDP PROCEDURE REFERENCES

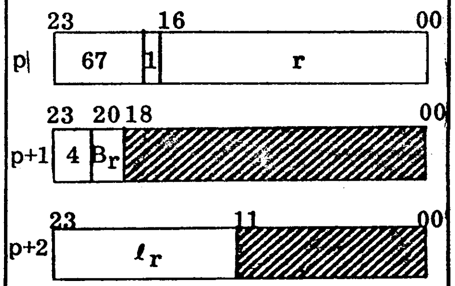
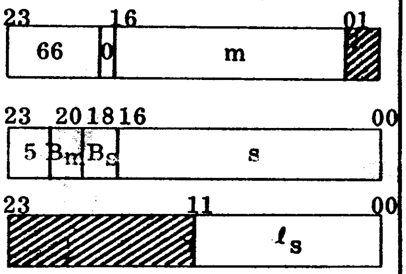
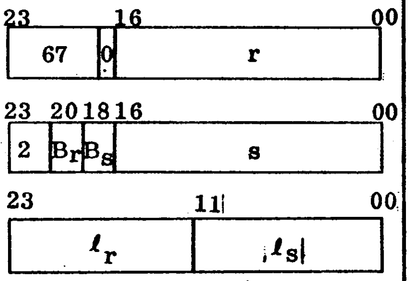
Type	Command Field	Operand Field	Operation	Object Code
BDP	TSTN	r, B_r, ℓ_r	Test field R for numeric	 23 16 00 p 67 1 r 23 20 18 00 p+1 4 Br 23 11 00 p+2 ℓr
	UPAK	m, B_m, ℓ_s B_s, ℓ_s	Unpack 4-bit BCD field M into 6-bit BCD field S	 23 16 01 66 0 m 23 20 18 16 00 5 Bm Bs s 23 11 00 ℓs
	ZADM	$r, B_r, \ell_r,$ s, B_s, ℓ_s	Clear field S; field R→ field S, right justify	 23 16 00 67 0 r 23 20 18 16 00 2 Br Bs s 23 11 00 ℓr ℓs

TABLE B-4. OCTAL CODE INDEX TO MNEMONICS

Octal Code	Mnemonic	Octal Code	Mnemonic	Octal Code	Mnemonic
00.0	HLT	05.5	QSG,S	15.5	INQ,S
00.1-6	SJ1-SJ6	05.6	ASG	15.6	INA
00.7	RTJ	05.7	QSG	15.7	INQ
01	UJP,I	06.0-7	MEQ	16.0	No-op
02.0	No-op	07.0-7	MTH	16.1-3	XOI
02.1-3	IJI	10.0	SSH	16.4	XOA,S
02.4	No-op	10.1-3	ISI	16.5	XOQ,S
02.5-7	IJD	10.4-7	ISD	16.6	XOA
03.0	AZJ,EQ	11.0	ECHA	16.7	XOQ
03.1	AZJ,NE	11.4	ECHA,S	17.0	No-op
03.2	AZJ,GE	12.0-3	SHA	17.1-3	ANI
03.3	AZJ,LT	12.4-7	SHQ	17.4	ANA,S
03.4	AQJ,EQ	13.0-3	SHAQ	17.5	ANQ,S
03.5	AQJ,NE	13.4-7	SCAQ	17.6	ANA
03.6	AQJ,GE	14.0	NOP	17.7	ANQ
03.7	AQJ,LT	14.1-3	ENI	20	LDA,I
04.0-3	ISE	14.4	ENA,S	21	LDQ,I
04.4	ASE,S	14.5	ENQ,S	23	LOCH
04.5	QSE,S	14.6	ENA	24	LCA,I
04.6	ASE	14.7	ENQ	25	LDAQ,I
04.7	QSE	15.0	No-op	26	LCAQ,I
05.0-3	ISG	15.1-3	INI	27	LDL,I
05.4	ASG	15.4	INA,S	30	ADA,I

TABLE B-4. OCTAL CODE INDEX TO MNEMONICS

Octal Code	Mnemonic	Octal Code	Mnemonic	Octal Code	Menmonic
31	SBA, I	53. (0+b)4	AIA	64.4-7	MVE, dc MVZS, dc
32	ADAQ, I	53.41	TQM		
33	SBAQ, I	53.42	TAM	65.0-3	SCAN, LR, EQ SCAN, LR, NE SCAN, RL, EQ SCAN, RL, EQ
34	RAD, I	53. (4+b)0	TAI		
35	SSA, I	53. (4+b)3	TIM	65.4-7	SCAN, RL, EQ, dc SCAN, LR, NE, dc SCAN, RL, EQ, dc SCAN, RL, NE, dc
36	SCA, I	53. (4+b)4	IAI		
37	LPA, I	54	LDI, I		
40	STA, I	55.0	RIS	66.0-3	ATD CVBD CVDB DTA PAK UPAK
41	STQ, I	55.1	ELQ		
42	SACH	55.2	EUA		
43	SQCH	55.3	EAQ	66.4-7	ATD, dc DTA, dc
44	SWA, I	55.4	ROS		
45	STAQ, I	55.5	QEL	67.0-3	ADM CMP SBM TST ZADM
46	SCHA, I	55.6	AEU		
47	STI, I	55.7	AQE		
50	MUA, I	56	MUAQ, I	67.4-7	CMP, dc TSTN
51	DVA, I	57	DVAQ, I	70.0	JMP, HI
52	CPR, I	60	FAD, I	70.1	JMP, ZRO
53.01	TMQ	61	FSB, I	70.2	JMP, LOW
53.02	TMA	62	FMU, I	70.6	LBR
53.04	AQA	63	FDV, I	70.7	SBR
53. (0+b)0	TIA	64.0-3	MVBF MVE MVZF MVZS	71	SRCE, INT SRCN, INT
53. (0+b)3	TMI				

TABLE B-4. OCTAL CODE INDEX TO MNEMONICS

Octal Code	Mnemonic
72	MOVE, INT
73	INPC, INT, B, H, A INAC, INT
74	OUTC, INT, B, H, A INAW, INT INPW, INT, B, N, A
75	OUTC, INT, B, H, A OTAC, INT
76	OUTW, INT, B, N, A OTAW, INT
77.0	CON
77.1	SEL
77.2	EXS COPY
77.3	INS CINS
77.4	INTS
77.50	INCL
77.51	CILO CLCA IOCL
77.52	SSIM
77.53	SCIM
77.54	ACI
77.55	CIA
77.56	JAA
77.57	IAPR
77.60	PAUS
77.61	PRP TMAV
77.62	SBJP
77.624	SDL
77.63	CRA SRA
77.634	ACR RCR
77.64	APF
77.65	PFA
77.66	AOS
77.664	AIS
77.67	OSA
77.674	ISA
77.70	SLS
77.71	SFPF
77.72	SBCD
77.73	DINT
77.74	EINT
77.75	CTI
77.76	CTO
77.77	UCS

BINARY OUTPUT C

When the META source deck contains a UNIT directive, the object computer is not the 3300 or 3500, and binary output (if requested) is in an alternate form. Information is written as binary card images, that is, in 40-word logical records in standard MASTER blocked format (MASTER Reference Manual).

Each 40-word logical record consists of a set of 160 6-bit bytes. Binary output is in the form of a byte stream. The first four bytes of each logical record are:

Byte	Value
1	Unused; 0
2	05_8
3	Unused; 0
4	Unused; 0

The byte stream consists of multibyte items. The first byte of an item is its item type, indicating the class of information. The number and contents of the bytes in the item vary according to item type.

Type	Byte	Information
1	1	Item type
	2-9	Control section name
	10-13	Control section byte length
	14	Chapter number (3 bits) 1 Chapter 1 2 Chapter 2 Control section type (3 bits) 0 Absolute 1 Program 2 Labeled 3 Numbered 4 Blank common
2	1	Item type
	2-9	External symbol
3	1	Item type
	2	Location counter number
	3-6	Load address (byte address)

Type	Byte	Information
4	1	Item type
	2-n+1	Contents of a word (n bytes)
6	1	Item type; item contains relocation information associated with preceding type 4 item
	2-4	Leftmost bit position of field in ward (7 bits)
		Field size (7 bits)
		Positive or negative relocation (1 bit) 0 Positive 1 Negative
		Word or Byte relocation (1 bit) 0 Word 1 Byte
		Unused; 0 (2 bits)
	5	Relocation counter
7	1	Item type; item contains external reference information associated with preceding type 4 item
	2-6	Bit position of field in ward (7 bits)
		Field size (7 bits)
		Positive or negative relocation (1 bit) 0 Positive 1 Negative
		Word or byte relocation (1 bit) 0 Word 1 Byte
		External symbol table ordinal (14 bits)
8	1	Item type
	2-9	Entry point symbol
	10-13	Entry point byte address
	14	Relocation counter
9	1	Item type
	2-9	Transfer symbol
0		Item type; end of stream on a logical record
63		Item type; end of stream

The number of bytes in a type 4 item is a function of the object computer word size. A value is right justified in the number of bytes required. For example, if the object computer word size is 19 bits, n equals 4.

All symbols are left justified and blank filled in eight bytes. The collection of type 2 items forms the external symbol table. Type 7 items refer to this table.

For type 6 and 7 items, bit positions are numbered from right to left in ascending order, beginning with zero. Thus, for a word address reference on the 3300, the following is true.

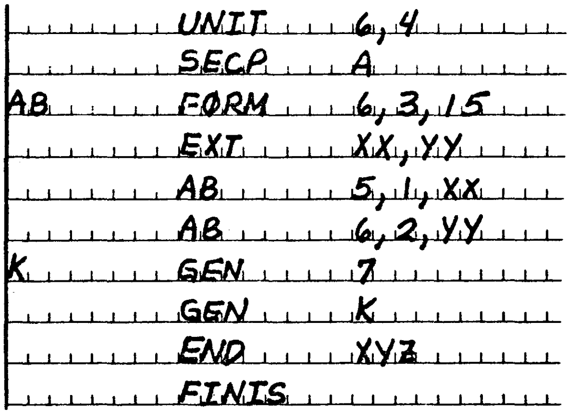
Leftmost bit position of field in word	14
Field size	15

When word size is 12, the leftmost bit position of a 13-bit field is 0.

Example:

The following program results in the binary (byte) stream shown.

Program:



```
          UNIT     6,4
          SECP     A
AB        FØRM     6,3,15
          EXT      XX,YY
          AB       5,1,XX
          AB       6,2,YY
K         GEN      7
          GEN      K
          END      XYZ
          FINIS
```

Binary stream of 6-bit bytes:

Card 1	00	05	00	00										
	01	A	∧	∧	∧	∧	∧	∧	∧	00	00	00	20	11
	02	X	X	∧	∧	∧	∧	∧	∧					
	02	Y	Y	∧	∧	∧	∧	∧	∧					
	03	01	00	00	00	00								
	04	05	10	00	00									
	07	07	03	60	00	01								
	04	06	20	00	00									
	07	07	03	60	00	02								
	04	00	00	00	07									
	04	00	00	00	03									
	06	13	46	00	01									
	11	X	Y	Z	∧	∧	∧	∧	∧					
	00													
Card 2	00	05	00	00										
	77													

The first four bytes cause rows 7 and 9 to be punched in column 1 of a binary card; column 2 is blank. Successive bytes consist of items and their associated information. A space is indicated by ∧.

For a subprogram, all external symbol items (item type 2) form a table of external symbols that immediately follows the table of control section name items.

Normally, a load address item (item type 3) immediately follows the last external symbol item. A load address item appears in the stream as necessary and always precedes the first contents-of-word item (item type 4). If a load address is more than one greater than the address associated with the previous contents-of-word item, META generates a load address item.

Example:

```
     UNIT  6,4
     SECP  LA
     GEN   1            New load address
     RES   5
     GEN   2            New load address
     SECD  NEW
     GEN   3            New load address
     END   MKG
```

The binary output stream for the above is as follows.

Card 1	00	05	00	00										
	01	L	A	∧	∧	∧	∧	∧	∧	00	00	00	34	11
	01	N	E	W	∧	∧	∧	∧	∧	00	00	00	04	12
	03	01	00	00	00	00								
	04	00	00	00	01									
	03	01	00	00	00	30								
	04	00	00	00	02									
	03	02	00	00	00	00								
	04	00	00	00	03									
	11	M	K	G	∧	∧	∧	∧	∧					
	00													
Card 2	00	05	00	00										
	77													

For a subprogram, all control section name items (item type 1) form a table of control section names. This table is first in the binary output stream. The entries in the table are in order according to their associated location counters. The first entry is for counter 0 or 1 depending on whether or not the program uses 0, the absolute location counter.

Example:

First entry in the control section name table is for location counter 0.

UNIT	6,4	
SECD	JØE	Location counter 2
SECA	AB	Location counter 0
SECP	XY	Location counter 1
SECP	KA	Location counter 3
SECD	KB	Location counter 4
END	XYZ	

Binary stream for above program:

Card 1	00	05	00	00										
	01	A	B	∧	∧	∧	∧	∧	∧	00	00	00	00	10
	01	X	Y	∧	∧	∧	∧	∧	∧	00	00	00	00	11
	01	J	O	E	∧	∧	∧	∧	∧	00	00	00	00	12
	01	K	A	∧	∧	∧	∧	∧	∧	00	[illegible]	00	00	11
	01	K	B	∧	∧	∧	∧	∧	∧	00	00	00	00	12
	11	X	Y	Z	∧	∧	∧	∧	∧					
	00													
Card 2	00	05	00	00										
	77													

It is possible to change control sections at points other than at word boundaries and resume the control sections.

Example:

	UNIT	6,4	Causes item 4 to be 5 bytes
	SECP	JØB	Causes item type 1
	GENB	1	Causes item types 3 and 4
A	RESB	1	
	SECD	AKK	Causes item type 1
	GENB	2	Causes item types 3 and 4
	ØRG	A	Returns to address A (counter 1)
	GENB	3	Causes item types 3 and 4
	END	MXT	Causes item type 9 (11_8)

The binary output stream for the above is as follows.

Card 1	00	05	00	00										
	00	J	O	B	∧	∧	∧	∧	∧	00	00	00	02	11
	01	A	K	K	∧	∧	∧	∧	∧	00	00	00	01	12
	03	01	00	00	00	00								
	04	01	00	00	00									
	03	02	00	00	00	00								
	04	02	00	00	00									
	03	01	00	00	00	01								
	04	00	03	00	00									
	11	M	X	T	∧	∧	∧	∧	∧	∧				
	00													
Card 2	00	05	00	00										
	77													

These 2 items must be combined to form 1 object computer word.

For the previous example, note that in the control section named JOB, two contents-of-word items (item type 4) are generated for the contents-of-word location zero.

GENB 1	04	01	00	00	00
GENB 3	04	00	03	00	00
Contents-of-word	01	03	00	00	

A field with a size greater than that of the object computer word may contain a relocatable value or an external symbol plus or minus a constant. More than one contents-of-word item (type 4) result, but they are not consecutive. A relocatable reference item (type 6) or an external reference item (type 7) immediately follows the first contents-of-word item (type 4). The condition can be detected when the leftmost bit in the word and the field size indicate a position beyond the preceding computer word.

Example:

```
         UNIT     12,1          Causes item 4 to be 3 bytes
         SECP     AMT           Causes item 1
A        RES      1
B        RES      1
F1       FØRM     48            48-bit field (four 12-bit words)
         F1       B             Causes items 3, 4, 6, 4, 4, 4
F2       FØRM     11,13
         F2       7,B           Causes items 4, 6, 4
         GEND     A             Causes items 4, 6, 4
         END      XMA           Causes item 9
```

Card 1	00	05	00	00	00										
	01	A	M	T	∧	∧	∧	∧	∧	00	00	00	12	11	
	03	01	00	00	00	02									
	04	00	00												Leftmost 12 bits of 48-bit field; first word.
	06	05	54	00	01										Leftmost bit in word is 11; field size is 48.
	04	00	00												Second word.
	04	00	00												Third word.
	04	00	01												Rightmost 12 bits of 48; fourth word.
	04	00	16												11-bit field containing 7 and leftmost bit of 13-bit field.
	06	00	03	20	01										Leftmost bit is 0; field size is 13.
	04	00	01												13-bit field containing 1.
	04	00	00												First 12 bits of 24 for GEND.
	06	05	46	00	01										Leftmost bit is 11; field size is 24.
	04	00	00												Second 12 bits of 24 for GEND.
	11	X	M	A	∧	∧	∧	∧	∧						
	00														
Card 2	00	05	00	00											
	77														

3300/3500 RELOCATABLE BINARY OBJECT DECK D

The 3300/3500 MASTER relocatable loader accepts relocatable binary object decks produced by the Meta-Assembler when there is no UNIT directive. During assembly, the X or F option on the META control card uses card images of the relocatable deck to be written on the LGO file (or some other file optionally specified). The P option on the META card causes the binary deck to be punched. A binary deck is comprised of the following types of cards.

Subprogram identification card (IDC)

Block common table cards (BCT)

Subprogram entry point cards (EPT)

Relocatable information cards (RIF)

External name and linkage cards (XNL)

Transfer cards (TRA)

End loading card (ELD)

These cards are described in the MASTER Reference Manual. Information on the cards is related to directives as shown in table D-1.

Table D-1 Loader Cards

Card	W	Source of Information
IDC	41_8	Name taken from SECP directive; length of subprogram calculated by META.
BCT	47_8	Names of labeled and numbered common blocks taken from SECD directives.
EPT	42_8	Entry points taken from ENTRY directives.
RIF	$1\text{-}36_8$	Relocatable information generated by mnemonic instructions, GEN, GEND, GENB, LIT, TEXT, TEXTC, TEXTA. RES or RESB causes start of new RIF card. Relocation factor set for character addressing if symbol generated is defined in bytes. Increment/decrement count and base depend on relocation counter used by Meta-Assembler.
XNL	43_8	External symbols taken from EXT directives.
TRA	44_8	Transfer point symbol taken from END directive.
ELD	77_8	Card generated upon encountering FINIS.

GLOSSARY OF TERMS

Absolute program

A program that must be loaded into specific core storage locations.

Assemble

To prepare an object language program for the 3300/3500 Computer System or for some other computer system from a symbolic source language program.

ASCII

American Standard Code for Information Interchange

Attribute

A characteristic of a symbol (value), such as its size in words or bytes and its mode of representation (decimal, octal, character, etc.).

Byte

A subdivision of a word as defined by a UNIT directive, if the source program contains one; otherwise, a byte is 6 bits.

Byte stream

Output from the Meta-Assembler when the source program contains a UNIT directive. Each 40-word record (160 6-bit bytes) consists of 11 types of multibyte items.

Command

The field in the source statement that specifies the operation to be performed by the Meta-Assembler.

Control Section

The portion of object code generated under a single location counter.

Definition

1. A group of source statements comprising a procedure or function. 2. The association of a symbol with a value and its other attributes so that use of the symbol causes its value or the address of its value to be used.

Delimiter

Character or characters that limit a string of characters and therefore cannot be a member of the string.

Directive

A source statement that instructs a Meta-Assembler.

Elementary item

A self defining component of an expression.

Entry point

A label of a source statement at which execution or processing can begin.

Expression

A valid series of values, symbols, and functions that may be connected by mnemonic or symbolic operators as required to cause a desired computation.

External symbol

A label defined in a subprogram other than the subprogram or at a level other than a level currently being assembled and used as an operand in the program or at the level being assembled.

Forward reference

A label that is referenced in the operand field and has not been defined previously.

Function

A series of source statements that, when referenced, provides a single value or a set to be used in the source statement containing the reference.

Label

1. A string of alphanumeric characters used to identify or describe an item or placed at any location for informational and instructional purposes. 2. To assign a symbol as a means of identifying a source statement or a location in an object deck.

Literal

An item of data having a constant value.

Location counter

A counter for the 16 control sections controlled by the assembler.

Meta-Assembler

An assembler that transcends the capabilities of a conventional assembler by allowing extensive programmer control of the assembly process.

Mnemonic instruction

Use of symbolic notation in place of actual machine code. A mnemonic instruction must be translated to actual operation codes by META procedure references.

Operand

A piece of data upon which an operation is performed; the contents of the operand field of a source statement.

Operator

The symbol or mnemonic that tells what to do with two operands, e.g., * is the operator for multiplication of the two operands as in A * B.

Procedure

A subset of source statements meeting a specific purpose that can be repeatedly referenced to cause parameterized code generation.

Processing

The interpretation by the Meta-Assembler of a source statement or group of source statements.

Real number

A value written with a decimal point, using decimal digits. The sign is a unary operator. An integer exponent preceded by E may follow the real number.

Symbolic referencing

The assembler allows mnemonic symbols to be used in place of instruction codes, modifiers, addresses, formats, procedures, and functions. The assembler interprets the symbol and determines where to find specific information.

Set

A collection of elements that bear a relationship to one another and have a common name. An element may be a set; i.e., a subset of a set. A reference to an element consists of the set name followed by one or more integers enclosed in brackets indicating the location of the element.

Source program

A program written in META language that must be translated into machine language before it can be executed.

Statement

An instruction interpreted by an assembler.

Subprogram

A part of a program that can be assembled independently.

Subscript

One or more integers enclosed by brackets used to specify a particular element in a set.

Unary operator

An operator such as the sign of a value (+ or -) that operates on one operand only rather than causing an addition or a subtraction.

Word

A group of bytes as defined by the UNIT directive if the source program contains one; otherwise, 24 bits, the standard 3300/3500 word size.

INDEX

A
  - ASCII string indicator 2-8
  - conversion modifier B-4
- Account number 7-2
- ACI instruction B-21
- ACR instruction B-21
- ADA instruction B-12
- ADAQ instruction B-12
- Addressing
  - absolute 4-10
  - byte-oriented 3-1
  - word-oriented 3-1
- ADM instruction B-30
- AEU instruction B-9
- AIA instruction B-12
- AIS instruction B-21
- ANA instruction B-19
- ANI instruction B-19
- ANQ instruction B-19
- AOS instruction B-21
- APF instruction B-21
- |Apostrophe
  - character in ASCII string 2-8
  - character in BCD string 2-6
  - delimiter, 2-5
  - in TEXT directive 4-18
  - printer character A-2
- AQA instruction B-12
- AQE instruction B-9
- AQJ instruction B-14
- Arithmetic 2-12
- Arithmetic instructions B-13
- ASCII code
  - character set A-3
  - generation of text 4-18
  - instructions (See BDP |procedure references)
  - notation 2-8
- ASE instruction B-15
- ASG instruction B-15
- Asterisk, double 2-9
- Asterisk, single
  - multiplication symbol 2-9
  - example 2-10; 4-21, 22
  - to indicate comments 2-3
- ATD instruction B-30
- Attributes, | description 2-12
- Atribute functions 6-1
- AZJ instruction B-15

B
  - backward read/write modifier B-4
- BCD character notation 2-7
- BCD integer notation 2-6
- BCD pseudo instruction B-7
- BDP procedure references B-30
- Binary scaling
  - example 2-13; 6-5
  - operator 2-9
- Blank, double 2-1, 5
- Blank fill
  - BCD characters 2-8
- Blank, single
  - elementary item separator 2-2, 5
  - expressions 2-11
- Bracket delimiter 2-5, 15
- BRIEF directive
  - byte-oriented directive 3-1
  - description 4-3
- BSS pseudo instruction B-7
- Byte
  - attribute function (BYT) 6-8
  - examples 4-15; 6-7, 8
  - generation 4-15
  - position in object|word 8-1
  - size 4-4
- Byte address, |mode 6-2
- Byte stream
  - file for 7-3
  - format C-1
  - selection as output 4-4

C
- character address modifier B-4
- common error code 8-3
- for right-adjusted BCD string 2-7

Card codes A-1
Chapter, assign data to 4-9
Character set
- BCD/ASCII conversion A-3
- card codes A-1
- internal codes A-1
- legal characters 2-1
- Printer graphics A-1

Character string
- ASCII notation 2-8
- BCD notation 2-7
- Mode 6-2

Character instructions B-14
Characteristic of real number 2-7
CIA instruction B-21
CILO instruction B-27
CINS instruction B-28
Circularity of form 4-17
CLCA instruction B-24
CMP instruction B-31
Comma
- command field delimiter 2-2
- set element delimiter 2-15
- to delimit, general 2-5

Command field, description of 2-2
Comment field, description 2-3
Comment line
- description 2-3
- example 2-4

Common block
- blank 4-9
- illegal assignment of literals to 2-14
- in byte stream C-1
- labeled 4-9, example 4-11
- numbered 4-9, example 4-11
- zero 4-9

Common error 8-3
Compare symbols and mnemonics 2-9
- examples 2-11

COMPASS language
- general differences from META 1-1
- pseudo instructions B-1, 7

CON instruction B-24
Concatenation of symbols 6-5
Configuration, minimum for META 1-2
Continuation of source statement
- description 2-3
- examples 2-4

Control cards required 7-1
Control section
- absolute 4-10
- creation 4-8, 9, 10
- general features 1-1
- list 8-4
- name 4-8, 9, 10
- name as word address 3-1
- relocatable 4-8

COPY instruction B-24
CPR instruction B-23
CRA instruction B-21
Cross reference table
- format 8-5
- selection 7-4

CTI instruction B-25
CTO instruction B-25
CVBD instruction B-31
CVDB instruction B-32
Current address symbol (see dollar sign)

D
- double definition error code 8-3
- for BCD integer 2-6

DEC pseudo instruction B-8
Decimal integer notation
- description 2-6

Decimal point in real number 2-7
Decimal scaling
- of real number 2-7
- operator in expression 2-9

Decision instructions
- IDENT B-15
- MONITOR B-23

Deck structure, examples 7-7
Definition
- form 4-15
- function 5-1, 3
- machine 4-4
- procedure 5-1, 2
- set, 4-4, 6
- symbol 4-4

- Delimiters 2-5
- Delimiting character modifier B-4
- DETAIL directive
  - as byte-oriented directive 3-1
  - description 4-3
- Diagnostic messages 8-5
- DINT instruction B-17
- Directives
  - data generating 4-13
  - definition 4-1
  - location control 4-8
  - machine defining 4-4
  - procedure defining 5-2
  - program linking 4-19
  - repeat 4-20
  - skip 4-21
  - symbol defining 4-4
  - uses 4-1
- Dollar sign
  - as current address symbol 2-10
    - examples 4-5, 13, 17
  - as word address 3-2
  - to change symbol level 5-13
    - examples 5-13, 14; 6-5, 6, 7
- Double definition error 8-3
  - example 4-5
- DTA instruction B-32
- Dummy label 4-22
- DVA instruction B-12
- DVAQ instruction B-12

E

- E
  - exponent indicator 2-7
  - expression error code 8-3
    - example 8-2
- EAQ instruction B-9
- ECHA instruction B-14
- EDIT instruction B-33
- EINT instruction B-17
- EJECT directive
  - as byte-oriented directive 3-1
  - description 4-2
- Elementary items 2-4
- ELQ instruction B-9
- ENA instruction B-9
- END directive
  - description 4-24
  - examples B-3; C-3, 5, 6, 7, 8
  - relationship to TRA card D-1
- End-of-file card 7-6
- ENDS directive
  - as byte-oriented directive 3-1
  - description 5-5
  - examples 4-23; 5-1, 2, 3, 5, 6, 8, 9, 10, 14; 6-5, 6
- ENI instruction B-9
- ENQ instruction B-9
- ENTRY directive
  - as byte-oriented directive 3-1
  - description 4-19
  - example B-3
  - relationship to EPT card D-1
- Entry point symbol
  - declaration 4-19
  - list 8-4
- EQ modifier B-4
- EQU directive
  - description 4-5
  - examples 3-2; 4-5, 7, 8, 21; 5-6, 11, 13, 14; 6-1, 2, 3, 4, 6, 7, 8; B-4
- Errors, assembly
  - codes for 8-3
  - on listing 8-1
  - lines 8-4
- EUA instruction B-9
- Expression
  - arithmetic 2-12
  - definition 2-10
  - evaluation 2-10, 12
  - examples 2-10, 11, 13
  - mixed mode 2-12
  - mode 2-12
  - parenthetical 2-11
  - relational 2-13
  - subexpressions 2-10
- Expression error 8-3
- EXS instruction B-25
- EXT directive
  - as byte-oriented directive 3-1
  - description 4-19
  - examples 4-14; 6-6, 7, 8; C-3
  - relationship to XNL card D-1

External symbol
declaration 4-19
list 8-4
UIC required as B-2

F
forward reference error code 8-3
to force execution 7-4
FAD instruction B-12
FDV instruction B-12
FINIS directive
description 4-24
example B-3, C-3
relationship to ELD card D-1
Floating point
data generation 4-14
in expression 2-12
instructions B-12
mode 6-2
notation 2-7
pseudo instruction B-8
FMU instruction B-12
Forced execution 7-4
FORM directive
description 4-15
example 3-2; 4-16
FORM reference
as byte-oriented statement 3-1
description 4-16
example 3-2; 4-16, 18; 5-6; C-3, 8
Forward reference
discussion 4-7
examples 4-7, 8
in procedure definition 5-1, 2
Forward reference error 8-3
FRMT instruction B-33
FSB instruction B-12
FUNC directive
description 5-3
example 5-1, 3, 5, 7, 11
Function
attribute 6-1
definition 5-1
processing 5-9
Function reference
description 5-11
example 5-3, 11; 6-1, 2, 3, 5, 6, 7, 8

GE as modifier B-8
GEN directive
as word-oriented directive 3-1
description 4-13
example 4-7, 8, 20, 21; 5-14; C-3, 5
relationship to RIF card D-1
GENB directive
as byte-oriented directive 3-1
description 4-15
example C-7
relationship to RIF card D-1
GEND directive
as word-oriented directive 3-1
description 4-14
examples 4-14; C-8
Generate object code
by word 4-13
by byte 4-15
by two words 4-14
Generation of byte stream D-1
GOTO directive
as byte-oriented directive 3-1
description 4-21
example 6-1, 3
processing 4-24

H
half assembly/disassembly modifier B-4
Heading (see TITLE directive)
HLT instruction B-17

I
indirect address modifier B-5
illegal instruction error code 8-3
to select input file 7-4
IAI instruction B-13
IAPR instruction B-28
IDENT procedure set B-1, 9

IDENT pseudo instruction
description B-4
example B-3
IJD instruction B-15
IJI instruction B-15
Illegal instruction error 8-3
INA instruction B-13
INAC instruction B-26
INAW instruction B-26
INCL instruction B-28
INI instruction B-13
INPC instruction B-26
Input files 7-4
Input/output instructions B-24
INPW instruction B-26
INQ instruction B-13
INS instruction B-29
Instructions, machine
arithmetic B-12
BDP B-30
character B-14
decisions B-15, 23
floating-point B-12
input/output B-24
interrupt B-17, 29
jumps, pauses, stops B-17, 24
logical B-19
no-operation B-18
shift B-18
transfer B-9, 21
INT interrupt modifier B-5
interrupt instructions B-17, 29
Integer
BCD notation 2-6
decimal notation 2-5
octal notation 2-6
mode 6-2
Internal octal codes A-1
INTS instruction B-29
IOCL instruction B-29
ISA instruction B-21
ISD instruction B-15
ISE instruction B-16
ISG instruction B-16
ISI instruction B-16

JAA instruction B-22
JMP instruction B-33
JOB control card
description 7-1
examples 7-7
Job identification 7-1
Jump, pause, stop instructions
IDENT B-17
MONITOR B-24

L
to indicate left-adjusted BCD character string 2-7
to select list output 7-4
Label field
description 2-2
LACH instruction B-14
LBR instruction B-22
LCA instruction B-9
LCAQ instruction B-10
LDA instruction B-10
LDAQ instruction B-10
LDI instruction B-10
LDL instruction B-19
LDQ instruction B10
Left-adjusted character strings
description 2-7
example 4-18
Library
procedures B-1
*LIB 5-7; 7-1
LIBS directive
as byte-oriented directive 3-1
description 5-7
example 5-8, 10; B-2, 3
to obtain|instruction set B-2
LIST directive
as byte-oriented directive 3-1
description 4-2
List
brief 4-3
control 4-1
detailed 4-3
format 8-1
of comments 2-3

of sequence columns 2-2
parameter on META card 7-4
resumption 4-2
suppression 4-2

LIT directive
as byte-oriented directive 3-1
description 4-12
relationship to RIF card D-1

Literal
assignment 4-12
symbols 2-9, 14
description 2-14
listing 8-4

LNID directive
description 4-22
examples 4-20, 22, 23, 24; 6-3

Load and go output
scheduling of mass storage 7-3
selection 7-4, 7, 8, 10

Location counter
$ reference to 2-10
absolute 3-3
assignment 3-3
data 3-3; 4-9
description 3-3
general feature 1-1
program 3-3; 4-10
relocatable 3-3
rounding up of 3-2

Logical instructions B-19

Logical operators
symbols, mnemonics 2-9

LOW BCR modifier B-5

LPA instruction B-19

LR left/right modifier B-5

LT less than modifier B-5

LQCH instruction B-14

Machine definition (see UNIT directive)

Mantissa of real number 2-7

Mass storage
minimum required 1-2
scratch 7-3
standard 7-3

Memory, core
examples of scheduling 7-7, 8, 9, 10, 11
minimum required 1-2
scheduling 7-2, 3

MEQ instruction B-16

Messages, error 8-5

Meta-assembler
configuration 1-2
definition Glossary-2
features 1-1
execution 7-1
library task 7-1

META control card
description 7-3
examples 7-5, 7, 8, 10

Mnemonic operators B-3

Mnemonic instructions for 3300/3500
general differences from COMPASS 1-1
list B-1

Mode of expression 2-12

Mode of value
definition 6-2
related to size 6-4

Mode (MDE) attribute 6-2

Modifiers, command field
general format 2-3
3300/3500 instruction B-4, 9

MONITOR procedures B-21

MOVE instruction B-26

MTH instruction B-16

MUA instruction B-13

MUAQ instruction B-13

MVBF instruction B-34

MVE instruction B-34

MVZF instruction B-35

MVZS instruction B-35

N
nesting error code 8-3
no assembly/disassembly modifier B-5

NAME directive
description 5-3
examples 4-23, 24; 5-2, 3, 4, 5, 6, 7, 8, 9,
5-10, 11, 14; 6-5, 6

NE not equal modifier B-5

Nesting level
procedure/function 5-1
repeat 4-20
NOLIST directive
as byte-oriented directive 3-1
description 4-1
NOP instruction B-18
Nesting
expressions 2-12
functions/procedures 5-1, 12
sets 2-15
repeated groups 4-20
NSET directive
description 4-6
example 4-6, 7, 8, 14, 16, 17; 5-3, 4, 5, 11; 6-2, 3, 4
Number of elements attribute (NUM) 6-3
Null set element 2-15

O
to indicate octal integer 2-6
Object computer 1-1; 4-4
Octal instruction index B-39
Octal notation
example 2-6; 4-17
format 2-6
Operand field format 2-3
Operators
examples 2-10, 11, 13
hierarchy 2-9
legal combinations 2-13
list 2-9
ORG directive
as byte-oriented directive 3-1
description 4-11
example 4-11; C-7
OSA instruction B-22
OTAC instruction B-27
OTAW instruction B-27
OUTC instruction B-27
Output files 7-4
OUTW instruction B-27

P
punch selection parameter 7-4
register B-5
PAK instruction B-36
Parentheses
delimiter 2-5
enclosing function arguments 5-11
enclosing nested expressions 2-10
PAUS instruction B-24
PFA instruction B-22
Printer
eject of page 4-2
line limit for job 7-1
list control 4-1
output on 8-1
scheduling mass storage 7-3
selection 4-3; 7-4
spacing 4-2
PROC directive
description 5-2
examples 4-23, 24; 5-1, 2, 4, 5, 6, 7, 8, 9, 10, 5-14; 6-5, 6
on library 5-8, 10
processing 5-9
Procedure reference
as byte-oriented statement 3-2
BDP B-30
description 5-10
examples 5-2, 8, 10, 14; 6-5, 6; B-3, 7, 8
IDENT B-1, 9
MONITOR B-21
processing 5-10, 13
Procedure
definition 5-1
nesting of 5-12
processing 5-9
repetition 4-23
standard library B-1
Processing
forward reference 4-7
function definition 5-9
function reference 5-10, 12
GOTO 4-23
procedure definition 5-9
procedure reference 5-9, 12
RPT 4-23
Program section (see control section)

PRP instruction B-24
Punch output
card limit for job 7-1
scratch needed 7-3
selection 7-4

Q
register B-5
QEL instruction B-10
QSE instruction B-17
QSG instruction B-17

R
relocation error code 8-3
select cross reference 7-4
RAD instruction B-13
RCR instruction B-22
RDEF directive
description 4-5
example 4-5,6,7,21,23; 5-2,9; 6-3
Real notation 2-7
Reference
FORM 4-16
forward 4-7
function 5-11
procedure 5-10
set 2-15
set element 2-15
Relocatable expression, rules 2-11
Relocation attribute (REL) 6-1
Relocation error 8-3
Relocation of operand 8-1
RES directive
as word-oriented directive 3-1
description 4-12
example 3-2; 4-8,11,12; 6-4,7,8
relationship to RIF card D-1
RESB directive
as byte-oriented directive 3-1
description 4-13
example 3-2; 4-13; 6-4,7,8; C-7
relationship to RIF card D-1

Repeat 4-20
Reserve storage
bytes 4-13
words 4-12
Right-adjusted
character strings 2-7,8
examples 2-7; 4-14,17
values in fields 4-16
RIS instruction B-10
RL right/left modifier B-5
ROS instruction B-10
RPT directive
description 4-20
examples 4-20,21; 6-3
processing 4-23
RTJ instruction B-17

S
syntax error code 8-3
SACH instruction B-14
SBA instruction B-13
SBAQ instruction B-13
SBCD instruction B-29
SBJP instruction B-22
SBM instruction B-36
SBR instruction B-22
SCA instruction B-19
Scaling, binary
example 6-5
factor 2-14
operator 2-9
Scaling, decimal
factor 2-14
operator 2-9
of real number 2-7
SCAN instruction B-36
SCAQ instruction B-18
SCHA instruction B-14
SCHED control card
description 7-2
example 7-7,8,9,10
SCIM instruction B-29
SDL instruction B-22

- SECA directive
  - as byte-oriented directive 3-1
  - description 4-10
  - example C-5
- SECD directive
  - as byte-oriented directive 3-1
  - description 4-9
  - example 4-11; C-5, 6, 7
  - relationship to BCT card D-1
- SECP directive
  - as byte-oriented directive 3-1
  - description of 4-9
  - example 3-3;4-11; 5-6; C-3, 5, 6, 7, 8
  - relationship to IDC card D-1
- Segments of mass storage
  - examples of scheduling 7-7, 8, 9, 10
  - minimum for META 1-2
  - scheduling 7-2
- SEL instruction B-27
- Semicolon 2-2, 3
- Set
  - assignment 4-6; 5-2, 3
  - definition 4-6
  - description 2-15
  - element 2-15
  - example 2-15; 4-6
  - function reference 5-10
  - procedure reference 5-10
  - subsets 2-15
- SFPF instruction B-29
- SHA instruction B-18
- SHAQ instruction B-18
- Shift instructions B-18
- SHQ instruction B-18
- SJj instruction B-17
- SLS instruction B-24
- Size attribute (SZE) 6-4
- Space (see blank)
- SPACING directive
  - as byte-oriented directive 3-1
  - description 4-2
- SQCH instruction B-14
- SRA instruction B-22
- SRCE instruction B-37
- SRCN instruction B-37
- SSA instruction B-20
- SSH instruction B-19
- SSIM Instruction B-29
- STA instruction B-10
- STAQ instruction B-10
- Statement
  - format 2-2
  - number 8-1
- STI instruction B-10
- STQ instruction B-11
- SWA instruction B-11
- Symbol
  - attributes 6-1
  - concatenation 6-5
  - definition 4-4
  - entry point 4-19, 24
  - external 4-19
  - illegal 6-6
  - level 5-13
  - reserved B-3
- Symbol attribute (SYM) 6-5
- Syntax error 8-3

- T
  - truncation error code 8-3
- TAI instruction B-11
- TAM instruction B-11
- Task name control card
  - description 7-5
  - example 7-7, 9, 10
- Termination
  - assembly, abnormal 8-5
  - assembly, normal 4-24
  - function definition 5-5
  - procedure definition 5-5
  - reference to definition 5-6
  - repeat 4-20
  - subprogram assembly 4-24
- TEXT directive
  - as word-oriented directive 3-1
  - description 4-18
  - relationship to RIF card D-1
- TEXTA directive
  - as word-oriented directive 3-1
  - description 4-18
  - relationship to RIF card D-1

TEXTC directive
  as byte-oriented directive 3-1
  description 4-18
  relationship to RIF card D-1
TIA instruction B-11
TIM instruction B-11
Time limit for job 7-1
TITLE directive
  as byte-oriented directive 3-1
  description 4-3
TMA instruction B-11
TMAV instruction B-22
TMI instruction B-11
TMQ instruction B-11
TQM instruction B-11
Transfer instructions B-9, 21
TREF directive
  as byte-oriented directive 3-1
  description 5-6
  example 5-6
Truncation
  ASCII character string 2-8
  BCD character string 2-8
  BCD integer 2-6
  decimal integer 2-6
  error code 8-3
  expression value 2-11
  octal integer 2-6
  real number 2-7
  sign bits 8-3
  word size 4-13
TST instruction B-37
TSTN instruction B-38
Two-pass option 5-1, 2

U
  undefined symbol error 8-3
UCS instruction B-24
UIC routine B-3
UJP instruction
  procedure reference B-17
  example B-3
Unary equal 2-9, 14
Unary double equal 2-9, 14
Unary minus 2-9
  examples 2-5, 6; 4-12, 15
Unary plus 2-9
UNIT directive
  description 4-4
  example 3-2; 4-17; 6-7; C-3, 5, 6, 7, 8
UPAK instruction B-38

Word
  attribute function (WRD) 6-7
  generation 4-13
  size 4-4
Word address 3-1
  list 8-1
  mode 6-2

X
  code on listing 8-1
  load and go parameter 7-4
XOA instruction B-20
XOI instruction B-20
XOQ instruction B-20

ZADM instruction B-38
Zero element 4-6
Zero fill
  ASCII character strings 2-8
  BCD character strings 2-8
  odd-sized field 4-18
Zero, negative 2-11
ZRO BDP modifier B-6

CONTROL DATA

COMMENT AND EVALUATION SHEET

3300/3500 META Reference Manual

Pub. No. 60236400 November 1968

THIS FORM IS NOT INTENDED TO BE USED AS AN ORDER BLANK. YOUR EVALUATION OF THIS MANUAL WILL BE WELCOMED BY CONTROL DATA CORPORATION. ANY ERRORS, SUGGESTED ADDITIONS OR DELETIONS, OR GENERAL COMMENTS MAY BE MADE BELOW. PLEASE INCLUDE PAGE NUMBER REFERENCE.

CUT ALONG LINE

PRINTED IN USA

FROM NAME: ____________________

BUSINESS ADDRESS: ____________________

____

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

CUT ALONG LINE

MD248

FOLD FOLD

STAPLE STAPLE

CONTROL DATA

1/2
3/4
1
1-1/4

►►CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

3300/3500 META/MASTER REFERENCE MANUAL

CONTROL DATA

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN, 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60236400

Litho in U.S.A.