File Number 705/7080-22
Form C28-6224-1

IBM Systems Reference Library

IBM 705/7080 Programming Systems
7058 Processor: Autocoder III Language

Autocoder III is the basic programming language for programs to be assembled by the 7058 Processor for the IBM 705 I, II, III, and 7080. This manual is one of two publications which describe Autocoder III. It supplies detailed specifications for using all parts of the language except the general-purpose macro-instructions. This information is prerequisite to the second manual, which supplies detailed information on using each general-purpose macro-instruction in the 7058 Processor library. The second manual is "7058 Processor: General Purpose Macro-Instructions," Form C28-6130.

MAJOR REVISION (April, 1964)

This publication is a major revision of the reference manual "IBM 705/7080 Programming Systems 7058 Processor: Autocoder III Language," Form C28-6224. The revision incorporates changes and additions throughout the text, and the format is changed from that of a Reference Manual to that of a Systems Reference Library (SRL) publication. Installations using the 705 Processor should continue to use the publication "Programming the IBM 705 Using the Autocoder III System," Form C28-6057.

This introduction provides an outline of basic program requirements, symbolic programming languages, and the program assembly process for the IBM 705/7080 Data Processing and Programming Systems. Readers already familiar with these systems may wish to go directly to Chapter 1. Information on these systems may be found in the publications "7080 Data Processing System," Form A22-6775; "705 Data Processing System," Form A22-6506; and "7080 Data Processing System," Form A22-6560. Although the 7058 Processor cannot operate on the 705 I, it can assemble 705 I programs. All comments in this manual that pertain to the 705 II are also applicable to the 705 I.

BASIC ASPECTS OF PROGRAMMING

A program is written in order to process data in a specified manner. In commercial data processing, most of the data is in the form of business records, e.g., accounts receivable, sales records, inventories, payrolls, etc. Although the main function of a program is to process these records as specified, the program does not consist solely of record-processing routines. These may be considered the body of the program and are often called the main-line routines or the main-line coding.

Any program must include routines for bringing the records to be processed into core storage and for taking the processed records out of storage. The routines which handle this data movement are called input/output or I/O routines. Although records and programs may be stored on magnetic tape or punched cards, magnetic tape is generally used with large-scale data processing systems.

A program must also contain actual storage locations for each instruction as well as locations for the area or areas the records will occupy. Records are usually grouped in blocks: consequently, an entire block enters storage. Similarly, the processed records are reblocked in storage before being placed on tape. Programs dealing with blocked records generally reserve space for separate input and output areas, the areas being equal to the size of the record block. In this case, a work area equal to the size of one record must also be reserved so that each record can be taken from the input area, moved to the work area for processing, and then placed in the output area. The processing instructions can then be addressed to the work area and do not have to be modified. If the records were to be processed in the input area, the instructions would have to be modified to operate on each record in turn. Consequently, most programs must reserve space for input, output, and work areas.

Certainly, a program must also provide routines for detecting and handling error conditions resulting from I/O operations. Such routines may reread or rewrite the records in error, place the invalid records on a special tape, attempt to determine whether or not the error is in the tape itself, etc. Error detection routines may include the procedure to be executed when an error condition prevents the continuation of processing.

Finally, there are supplementary procedures which must be performed by all programs but which are not directly connected with the main-line processing. They fall into no specific category, although they might be described as procedures which implement the operation of the program. Those which are executed before any main-line processing begins are called housekeeping routines; those which are executed after all main-line processing is completed are called end-of-job routines. Housekeeping operations include such procedures as readying input/output units, setting ASUs, checking and writing tape identifications, and bringing the first block of records into storage. End-of-job routines include such procedures as moving the last block of records from storage to tape, writing tape identifications, rewinding tapes, and writing messages.

To sum up, a program must incorporate at least the following procedures:

1. Data processing
2. Input/output
3. Storage assignments
4. Error detection and correction
5. Housekeeping and end-of-job

SYMBOLIC PROGRAMMING SYSTEMS

A program may be written in the actual, i.e., machine language of the computer on which it will run, or it may be written in a symbolic language. If it is written in machine language, it can be executed by the computer directly, but if it is written in symbolic language, it must first be translated into machine language before it can be executed. The length and complexity of programs today makes programming in machine language extremely difficult and results in programs which are increasingly liable to error. However, powerful symbolic programming systems have been developed to relieve the programmer of the many burdens involved in machine language programming. A symbolic programming system consists of a symbolic language and a processor. The language provides a method of representing program functions as a series of meaningful statements rather than as a collection of alphameric codes and actual storage locations. The processor converts the symbolic language program into a machine language program, assigns storage locations to the program, and performs various other functions. The symbolic language program is generally called the source program; the machine language program is called the object program. In other words, the source program is the input to the processor, and the object program is the output of the processor.

Thus, processing the data for which a program is written becomes the second of two data processing applications. The first application is the processing or conversion of the source program itself, with the object program as output. The second application is the processing of the actual data by the object program; the output of the second is the solution of the problem for which the program was written. Once the object program is produced, it is used in subsequent data processing applications until it is obsolete or is modified to such an extent that a reassembly is advisable.

Since the programs written in symbolic language need not make location assignments, the order of the statements which compose the program may be changed and the program reassembled without modification. For the same reason, it is easy to insert or delete statements in a symbolic language program. When it is reassembled, a new object program is produced.

The Symbolic Language

Instructions form a major portion of the statements in a symbolic language program just as they do in a machine language program. A symbolic one-for-one instruction contains a mnemonic code representing a machine operation and a symbolic address representing the storage location of data or an instruction. Such instructions are called one-for-one because the processor replaces each one with one machine instruction. An important development in symbolic programming is the macro-instruction, a source program statement which is eventually replaced by more than one machine instruction. Essentially, it is a request for several one-for-one instructions, each of which is subsequently replaced by one machine instruction. A macro-instruction also contains a mnemonic code, but the code does not represent any one machine operation. A macro-instruction usually contains more than one symbolic address; each address represents the storage location of data or of an instruction.

Symbolic languages enable the user to write program statements describing the storage areas which will be occupied by program data. On the basis of the information the processor obtains from these statements, it assigns actual storage locations to the data areas. It also uses this information when generating one-for-one instructions to replace macro-instructions which reference these areas. If the data is to be supplied to the area by input records, the statement indicates the size of the area and the type of data which will occupy it. If not, the statement itself supplies the data, which is placed in storage as a constant.

The programmer is also able to create a symbolic address for each data area or instruction. The symbolic address represents the actual storage location to be assigned by the processor, and it provides the means of referencing an area or an instruction. This is done by using the symbolic address as the operand of the instruction which makes the reference. Usually, it is desirable to create symbolic addresses which describe the areas or instructions to which they are assigned. For instance, an address such as "master file" might be assigned to a data area which will be filled by records from the master tape; an address such as "start" might be assigned to the first instruction to be executed, etc. In converting the source program to machine language, the processor replaces each symbolic address with an actual storage location, just as it replaces each mnemonic code with an actual operation code.

The Processor

The processor of a programming system is a machine language program which converts a symbolic language program into machine language. The process of converting is called assembling the program. In other words, a processor assembles a source program into its object program form. During the assembly, the processor makes an analysis of the source program, generates one-for-one instructions to replace each macro-instruction it encounters, inserts any subroutines requested by the program, substitutes machine language instructions for all one-for-one instructions, and assigns storage locations to the object program.

The processor contains a library of macro-instructions and subroutines. Every macro-instruction contains a set of incomplete one-for-one instructions. When a source program macro-instruction is encountered during assembly, the processor determines which of the one-for-one instructions are appropriate, completes those which it selects, and inserts them into the object program. Selection and completion of the appropriate instructions are done on the basis of information from the program analysis made by the processor. The same macro-instruction may be used many times in a program, but the one-for-one instructions generated from it will not necessarily be the same. The variation results from differences in program requirements or data format.

Library subroutines differ substantially from macro-instructions. A subroutine is a fixed set of instructions; these may be one-for-one instructions or one-for-one instructions and macro-instructions. When a request for a subroutine is encountered during assembly, the set of instructions is taken from the library and inserted in the program. The instructions will not vary from program to program unless the subroutine itself contains macro-instructions. The programmer may write macro-instructions and subroutines and add them to the processor library.

The object program is not the only output of the processor. A sequential listing of the source program is also produced. Each program step in the listing is assigned an index number for reference purposes. The one-for-one instructions in the source program are shown with the corresponding machine language instructions and the storage locations assigned to them. The source program macro-instructions are followed by the one-for-one instructions generated from them, the machine language instructions corresponding to the one-for-one instructions, and the storage locations assigned to the instructions. Location assignments are also shown for all record areas and subroutines.

THE BASIC 705/7080 PROGRAMMING SYSTEM

A programming system has been defined as a symbolic language and a processor. The basic programming system for the 705 and the 7080 Data Processing Systems is composed of Autocoder III language and the 7058 Processor.

The 7058 Processor

The 7058 Processor, hereafter called "the Processor," is a machine language program which assembles programs written in Autocoder III for the 705 I, II, III, and the 7080. The Processor operates on the 705 II and III and the 7080 when it is in 705 II or III mode. The Processor itself is so large that it must operate through a number of inter-related sections or phases. Each phase is a program which performs one or more of the various assembly functions. The phases may be classified as belonging to one of the two portions of the Processor: the compiler and the assembler. The compiler phases analyze the source program in detail, generate Autocoder III statements from higher language statements (explained on pages 5-6), and generate one-for-one instructions from macro-instructions. The assembler phases assign storage locations, replace one-for-one instructions with machine language instructions, and create the Processor output.

The output of the Processor consists of the object program in card form and the program listing with related messages. Both are produced on tape, but the Processor can be modified so that the object program is produced on line as a punched card deck and the program listing is printed by an on-line printer. The listing and messages are the minimum assembly documentation. Additional documentation consisting of the Operator's Notebook and/or the Symbolic Analyzer can be requested.

The Operator's Notebook lists the following:

1. Programmed halts
2. Titles of the various portions of the program
3. A list of special 7080 program statements
4. Specific location assignments requested by the program
5. Program switches set up by the Processor at the request of the program

The Notebook is useful to the programmer in debugging the object program and to the console operator during the object program run. The Symbolic Analyzer is an alphabetical list of the symbolic addresses used in the program. Each symbolic address is followed by a list of the instructions which reference it. All may be easily located in the listing because their index numbers are shown. Referencing a field or an instruction, as used in this manual, means specifying the data to be operated on or specifying an instruction to be executed. An Autocoder III statement which calls for data movement to a work area references the data and the work area. A statement which causes the program to transfer to an instruction references that instruction.

The Processor library contains a set of general purpose macro-instructions which cover most commercial data processing functions. Programmers may write their own macro-instructions and subroutines and may insert them in the library. However, the preparation of macro-instructions is a complicated procedure requiring a thorough knowledge of Autocoder III and the Processor.

Autocoder III Language

Autocoder III is the basic symbolic language for programs to be assembled by the Processor. Statements written in the higher languages may be inserted in Autocoder III programs. During the assembly, certain phases of the Processor translate these statements into a series of Autocoder III statements. Program steps written in Autocoder III language are called statements rather than instructions, because the language contains more than a set of processing instructions. There are six types of Autocoder III statements:

1. Area definitions
2. Switch definitions
3. One-for-one instructions
4. Macro-instructions
5. Address constants
6. Instructions to the Processor

AREA DEFINITIONS. Area definitions reserve storage space for data which is supplied either by records or by the programmer. If the space will be occupied by data from records, the area definitions also describe the nature of the data. If not, the area definitions specify the constant data to be placed in storage. The storage space reserved by each area definition is generally called a data field. Area definitions may also be used to indicate that a series of adjacent data fields are to be treated as the interior portions of a single unit.

For input/output areas, it is usually necessary to define a data field for a block of records without making any attempt to distinguish one record from another or to identify portions of a record. However, in defining the work area, the opposite is true. Since an individual record will be moved into the work area, it is usually defined as a series of data fields which correspond to the various portions of the record.

Suppose that each record in a file contains the name and yearly salary of an employee and that these records are on tape in blocks of ten. Processing consists of updating the yearly salary. The input (and the output) area is defined as one data field, although it will contain ten records. However, the work area to which each record is moved for processing is defined as two data fields, one for the employee's name, and one for the employee's yearly salary. Only the salary field is referenced by processing instructions, but the entire record is referenced as a unit when it is moved to or from the work area. Consequently, the work area must actually be defined as a data field consisting of two interior fields.

SWITCH DEFINITIONS. Switch definitions describe three types of switches: data, program, and console. All three may be used to control the path of the program, e.g., to determine whether or not all the routines in the program will be executed, to determine the sequence in which routines will be executed, etc.

Data Switch. A data switch is a data field in which alphameric codes are placed. The definition of the switch allows a meaning to be associated with each code. When a data switch is defined as a portion of a record area, the records supply the codes for the switch.

When a data switch is defined independently of a record area, the program itself supplies the codes.

In the employee records used as an example in the section on area definitions, suppose now that each record consists of three fields: name, yearly salary, and number of exemptions of the employee. The work area is defined by area definitions for the name and yearly salary fields and a switch definition for the exemption field. In this case, the codes in the data switch would be numeric characters. The manner in which each record is processed depends on the number of exemptions; therefore, the program contains a number of processing routines. As each record is placed in the work area, the data switch becomes whatever character the exemption field contains. The program tests the switch to determine what code is present and then transfers to the processing routine appropriate for that code.

Program Switch. A program switch is an instruction which causes the program either to continue sequentially or to transfer. When a program switch is ON, the program transfers to an out-of-line instruction. When a switch is OFF, the program executes the next in-line instruction.

Suppose that it is desired to type a message if a certain error condition is detected. The program switch is defined so that when it is OFF, the program proceeds to the next instruction, but when it is ON, the program transfers to the message-writing routine. Initially, the switch is set OFF; as long as it remains OFF, the program continues through the switch to the following instruction. If the error-detection routine encounters the error condition, it sets the switch ON; then, when the program reaches the switch, it transfers to the message-writing routine.

Console Switch. A console switch is one of the six alteration switches on the console. They are numbered 0911-0916, and they must be set manually by the console operator. Console switches are useful when it is desired to execute a routine only for certain object runs. For example, a program which is run each week may include a routine which should be executed only at the end of the month. If a console switch is defined, the program may test the switch and transfer to the end-of-month routine when the switch is ON. The console operator must, of course, set the switch ON prior to each end-of-month run.

ONE-FOR-ONE INSTRUCTIONS. One-for-one instructions are the symbolic equivalents of machine instructions. Coding any portion of a program in one-for-one instructions means much more hand-coding for the programmer than coding the same portion in macro-instructions. This also increases the possibility of error. One-for-one instructions should be used only when it is inadvisable to use macro-instructions.

MACRO-INSTRUCTIONS. A macro-instruction is a powerful programming device; essentially it is a request for those one-for-one instructions which will accomplish the function stated by the macro-instruction. These instructions are selected to suit the characteristics of the data fields and/or the other hand-coded instructions referenced by the macro-instruction. The field characteristics are obtained from the field definition analysis made by the Processor. Whenever a choice exists among the one-for-one instructions to be generated, the Processor selects the most efficient coding.

As an example of the scope of a macro-instruction, the basic coding generated from the ADDX macro-instruction adds the contents of two numeric fields and stores the result in a field designated as the result field. But, if the result contains more decimal positions than the number specified in the result field definition, the generated coding includes instructions either to round or to truncate the excess positions before the result is stored. The choice depends on which process the programmer specifies in the macro-instruction. Also, if the result contains more integer positions than the number specified in the result field definition, the generated coding includes instructions to truncate the excess high-order positions before the result is stored. However, the programmer may request an option which generates instructions to do the following: truncate the excess positions if they contain zeros and store the result; transfer to a routine designated by the programmer if they do not contain zeros. This entire procedure, which obviously involves many one-for-one instructions, is generated from one macro-instruction.

ADDRESS CONSTANTS. An address constant contains the symbolic address of a data field or an instruction. During the program assembly, a constant is created from the actual location assigned to the field or instruction. Address constants are used to initialize an instruction. Initialization is the process of supplying a reference to an instruction which lacks one or replacing the reference made by an instruction. An instruction makes a reference by designating the symbolic address of a data field or an instruction. The symbolic address designated by an address constant is used to initialize the instruction.

Suppose that an input area contains a block of records, each of which must be moved from the area in succession. The input area is given a symbolic address so that the area can be referenced by the instruction which moves the records. Initially, the instruction has as its address portion the symbolic address of the area, thus referencing the first record in the area. However, the instruction's address portion must be modified before it can reference successive records; the modification is generally an increment equal to the size of one record. Eventually, the input area is emptied, and a new block of records is placed in it. But the modified instruction no longer references the first record. At this point, it is necessary to initialize the instruction, that is, to return the instruction to its original form, by means of an address constant. Assume that the address constant has been coded and that it consists of the symbolic address of the input area. Now the address constant can be placed in the address portion of the modified instruction. Once the instruction is initialized, it references the first record in the area again.

INSTRUCTIONS TO THE PROCESSOR. Instructions to the Processor allow the programmer to control certain aspects of the assembly process and to take advantage of the special features of the Processor. The Processor instructions are written as Autocoder III statements in the program. When they are encountered during assembly, the Processor performs the operations they request. Instructions to the Processor concern the following aspects of the assembly:

1. The listing of the program
2. Location assignments made by the Processor
3. Coding generated by the Processor

INPUT/OUTPUT SYSTEMS FOR USE WITH AUTOCODER III PROGRAMS

Input/Output Control Systems (IOCS) have been developed for the IBM 705 III and 7080. IOCS consists of a group of routines which handle all input/output functions. These routines are made available to an Autocoder III program when IOCS macro-instructions in the Processor library are used in the program. The following IOCS publications are available:

1. "Input/Output Control System for the IBM 705 III," Form C28-6109. Reference manual.
2. "IBM 7080 Input/Output Control System for Use with 729 Magnetic Tape Units," Form C28-6237. SRL publication.

HIGHER LANGUAGES FOR USE WITH AUTOCODER III PROGRAMS

As mentioned earlier, the 7058 Processor accepts program statements written in several higher languages. The languages are: FORTRAN; Report/File Writing; Decision; Arithmetic; Table-Creating. Various Processor phases translate each of these statements into one or more Autocoder III statements.

FORTRAN is the name for Formula Translation language. As the name implies, complex problems can be stated in formula form using FORTRAN. Both fixed point and floating point calculations are possible.

Report/File Writing language is a set of statements which may be used to describe the format and contents of a report or file. The routine generated from these statements will create the report or file.

Decision language is one statement. It requests a logical decision to be made on the basis of a test of the various conditions supplied in the statement.

Arithmetic language, also one statement, requests a series of mathematical computations to be performed on the elements supplied in the statement.

Table-Creating language consists of a statement which requests the creation of a table from a set of data. The data itself must accompany the Table statement.

The following higher language publications are available:

1. "FORTRAN," Form F28-8074. General information manual.
2. "705 FORTRAN Programming System," Form J28-6122. Bulletin.
3. "7058 Processor: Decision, Arithmetic, and Table-Creating Languages," Form C28-6226. Reference manual.
4. "7058 Processor: Report/File Language," Form J28-6234. Reference manual.
5. "705/7080 Programming Systems: Cobol-Additional Specifications," Form J28-6177. Reference manual.

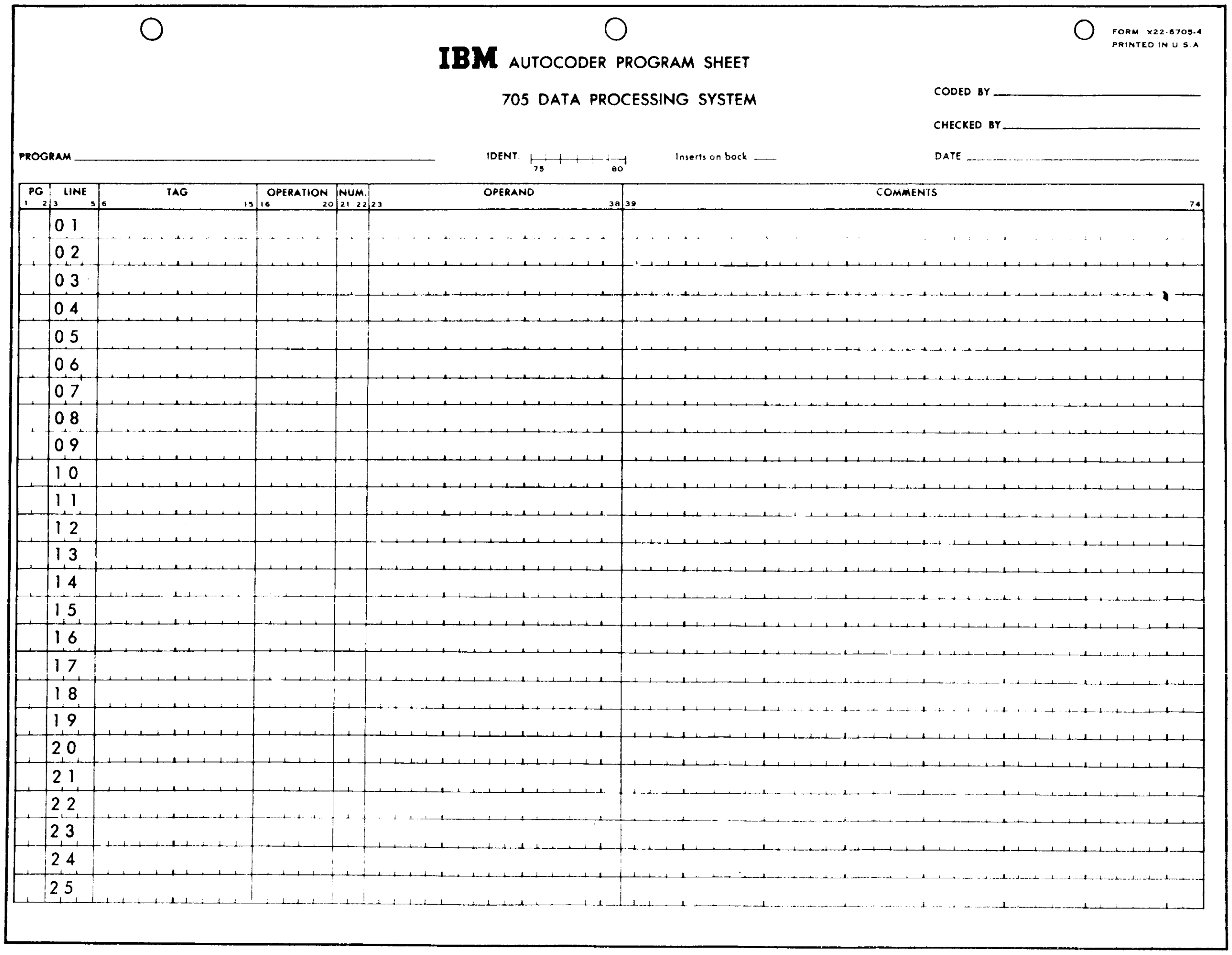
An Autocoder III program is written on the IBM Autocoder Program Sheet, Form 22-6705-4, shown in Figure 1. One card is punched for each line of the coding sheet. The card designed for Autocoder III programs is the IBM Autocoder System Card, Electro 893094. An Autocoder III statement is formed by filling out the appropriate fields on the sheet according to the specifications for the type of statement being written. Some statements may occupy more than one line. The term "field" applies to the character positions included under each heading on the program sheet. The position numbers listed with the field headings correspond to the columns on the card.

PROGRAM IDENTIFICATION (COLUMNS 75-80)

The identification is filled in at the top of the coding sheet. It should appear in columns 75-80 of every card punched for an Autocoder III statement.

PAGE (COLUMNS 1-2)

The sequence of the coding sheets is designated by a two-position page number. Any alphameric character may be used in the number. Normally, however, special characters are not used. The IBM 705 collating sequence, shown in Figure 2, is used to determine the order of the pages.



FORM X22-6705-4
PRINTED IN U.S.A.

IBM AUTOCODER PROGRAM SHEET

705 DATA PROCESSING SYSTEM

CODED BY

CHECKED BY

PROGRAM

IDENT. 75 80

Inserts on back

DATE

PG 1 2	LINE 3 5	TAG 6 15	OPERATION 16 20	NUM. 21 22	OPERAND 23 38	COMMENTS 39 74
	01					
	02					
	03					
	04					
	05					
	06					
	07					
	08					
	09					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					

Figure 1

Blank . ⌑ ⧧ & $ * - / , % # @ $\overset{+}{0}$ A through I $\overset{-}{0}$ J through R ⧧ S through Z 0 through 9

Figure 2. IBM 705 Collating Sequence

LINE (COLUMNS 3-5)

The sequence of the statements on each coding sheet is designated by a three-position line number. On the front of each sheet, the first two positions are pre-numbered; any alphameric character may be used in the last position, although special characters are not used normally. Ordering should be done according to the 705 collating sequence. It is recommended that column 5 be left blank except when designating the sequence of insertions.

The back of each sheet may be used for insertions. The insertion page number should be the page number of the statement the insertion is to follow. The insertion line number should be higher than that of the statement preceding the insertion and lower than that of the statement following the insertion. For example, a three-line insertion may be required between two statements numbered 03b and 04b (b represents a blank). The insertions might be numbered 031, 032, and 033, or they might be numbered 03A, 03B, and 03C.

TAG (COLUMNS 6-15)

A tag is the symbolic address which represents the actual location of a data field or an instruction. The field is filled in starting in column 6. When an Autocoder III statement references a tag, it refers to the data field or the instruction at the storage location represented by the tag. During assembly, all fields and instructions are assigned storage locations, and all references to tags are replaced with the locations assigned to the tags.

A tag may contain up to ten characters; these may be alphabetic and/or numeric and blanks. A tag may not contain special characters. If composed of numeric characters only, a tag must consist of five or more characters. It is recommended that tags not start with one or more blanks, because the Processor must left-justify them, a time-consuming operation. It is also recommended that pure numeric tags not be used. It is best to create tags which describe the data fields or the instructions to which they are assigned. Tags should not be assigned unless they are referenced by program statements; because unnecessary tags slow the assembly process and produce needless messages.

OPERATION (COLUMNS 16-20)

The mnemonic code of the Autocoder III statement is placed in the operation field, starting in column 16. No machine operation code should be used.

NUMERIC (COLUMNS 21-22)

The use of the numeric field varies according to the type of Autocoder III statement being written. A one-position entry is placed in column 22.

OPERAND (COLUMNS 23-38)

The use of the operand field varies according to the type of Autocoder III statement being written. The field is filled in starting in column 23, and the entry may be continued into the comments field. Macro-instruction operands may be continued from the comments field of one line into the operand and comments fields of succeeding lines of the coding sheet.

COMMENTS (COLUMNS 39-74)

Additional information about an Autocoder III statement may be written in the comments field and will appear in the program listing. Comments are useful for explaining the purpose of program statements. The field does not have to be filled in starting in column 39. The comments may be continued in the comments field on subsequent lines of the coding sheet; there is no limitation on the number of comments continuation lines.

The rules governing comments and comments continuations vary according to whether or not the comments accompany a macro-instruction. If they do, they must be separated from the operand by a minimum of two blank spaces whether the operand terminates in the operand field or continues into the comments field. The comments continuation lines for macro-instructions may not contain entries in any fields except page, line, and comments.

If the comments do not accompany a macro-instruction, they do not have to be separated from the operand by blank spaces, and comments continuation lines may contain entries in any columns except 16-17 (first two positions of the operation field) and 21-22 (numeric field). However, to make the comments easier to read, it is recommended that the continuation lines be restricted to entries in the page, line, and comments fields.

CHAPTER 2. AREA DEFINITIONS

Area definition statements describe data fields; the data may be variable data supplied by records or constant data supplied by the area definition statement. The programmer must know the length and composition of the records so that each field may be defined correctly. The Processor uses the information provided by area definitions when it reserves storage space for the fields and when it encounters instructions which reference the fields.

There are five types of area definitions:

1. Definition of a Record - RCD
2. Definition of a Constant Factor - CON
3. Definition of a Floating Decimal Point Number - FPN
4. Definition of a Report Format Field - RPT
5. Definition of a Continuous Portion of Memory - NAME

An area definition statement must contain a tag if the field is to be referenced. The reference is made by using this same tag in the operand of the Autocoder III statement making the reference. Since the tag requirement applies to all area definitions, the tag field will not be discussed separately in the remainder of this chapter.

DEFINITION OF A RECORD - RCD

The function of an RCD statement is to define a data field in which a record block, an individual record, or a portion of a record will be placed. The definition specifies the size of the field and the nature of data it will contain. The RCD statement is written as follows:

OPERATION FIELD. The mnemonic code RCD is placed here. In a continuous series of RCD statements, only the first need contain the mnemonic code. The Processor assumes that each immediately subsequent statement with a blank operation field is an RCD and treats it accordingly. This assumption makes it possible in subsequent statements to use columns 18-20 of the operation field as an expansion of the numeric field. (The operation field is assumed to be blank if columns 16 and 17 are blank.)

NUMERIC FIELD. The size of the data field is entered here. A one-digit entry is placed in column 22 and need not be preceded by a zero. When the operation field contains the RCD code, the numeric field is limited to a two-digit entry. However, when the operation field is blank and the statement has been preceded by another RCD statement, columns 18-20 of the operation field may be used as an expansion of the numeric field. Under these conditions, in effect, the numeric field consists of five positions. Thus, data fields which exceed 99 positions may be defined, but they may not be the first in a series of RCD statements.

OPERAND FIELD. The operand field contains one of the following:

1. A descriptive code. This is used to define alphameric fields or numeric fields containing integers only.
2. A description of an integer and decimal format. This is used to define numeric fields containing mixed or pure decimals.
3. A layout of group marks and/or record marks. This is used to describe the position of group marks and/or record marks in a field.

The alphameric and numeric fields of integers are:

Code	Contents of Field
+	Signed numeric data consisting of integers.
N	Unsigned numeric data consisting of integers.
F	Signed numeric data in floating point form. The field must consist of ten positions: a two-character exponent, signed in the low-order position, followed by an eight-character mantissa, also signed in the low-order position. This is the form in which a floating decimal point constant appears in storage. See page 17 for further explanation.
A	Alphameric data which may or may not provide left protection for the immediately subsequent field.
A+	Alphameric data which always provides left protection for the immediately subsequent field.

Zoning in the low-order position of a field constitutes left protection for the subsequent field. Left protection must be provided when the subsequent field contains signed numeric data and is referenced by an instruction having an arithmetic function. The low-order position of the field providing left protection must be occupied by one of the following: an alphabetic character, a signed numeric character, a blank, or any special character.

Figure 3 shows fields defined with descriptive codes. Notice that the final field cannot be referenced, because it is not tagged.

TAG 6-15	OPERATION 16-20	NUM 21-22	OPERAND 23-38	39
UNSIGNED	RCD	8	N	
ALPHAFIELD	1	25	A+	
SIGNED		13	+	
FLOAT		10	F	
	12	00	A	

Figure 3

Numeric Fields Containing Mixed or Pure Decimals. The operand must indicate the number of integer and decimal positions in the field and whether the field is signed or unsigned. This may be done in either of the following ways, although the first method is the preferred use:

1. Enumerating the number of integer and decimal positions. Signed numeric fields are represented as #+xx.yy, and unsigned numeric fields as #bxx.yy, where xx and yy represent the number of integer and decimal positions respectively (b represents a blank position). If there are no integer positions, xx is written as 00. If there are less than ten positions on either side of the decimal point, the numeric digit is preceded by a zero. The sum of xx and yy must equal the entry in the numeric field. The maximum size data field which can be defined consists of 99 integer and 99 decimal positions.

2. Showing a layout of the integer and decimal positions. Each integer and decimal position is indicated by an X, with a decimal point placed in the appropriate position. The layout of a pure decimal starts with the decimal point and is followed by the necessary number of Xs to the right of it. When defining signed numeric fields, a plus sign is placed in the first position of the operand and is followed by the layout. The operand defining an unsigned numeric field starts with the layout itself. A blank position is not used to indicate unsigned numeric data.

The total number of Xs must equal the entry in the numeric field. Although both the decimal point and the sign occupy positions in the layout, neither is included in the count for the numeric field entry. The point itself does not exist in the record nor does the sign exist in the record as a separate position. However, the Processor needs this information for various purposes, such as selecting the proper coding to replace macro-instructions.

The definitions in Figure 4 are paired to show how the same numeric fields would be defined by each of these methods. Note that SIGNED3 is too large to be defined by a layout.

TAG 6-15	OPERATION 16-20	NUM. 21-22	OPERAND 23-38	39
SIGNED1	RCD	8	#+05.03	
SIGNED1	RCD	8	+XXXXX.XXX	
	≀			
UNSIGNED1	RCD	12	# 11.01	
UNSIGNED1	RCD	12	XXXXXXXXXXX.X	
	≀			
SIGNED2	RCD	13	#+00.13	
SIGNED2	RCD	13	+.XXXXXXXXXXXXX	
	≀			
UNSIGNED2	RCD	2	# 00.02	
UNSIGNED2	RCD	2	.XX	
	≀			
SIGNED3	RCD	73	#+47.26	

Figure 4

Indicating the Position of Record Marks and/or Group Marks. This information should be supplied if the record which contains such characters is referenced by a macro-instruction. The position or positions the characters occupy must be defined as one field of the record, unless no other information is to be given about the record. The operand must be a layout of the record portion which contains the characters and may indicate one of the following: a terminal group mark, a terminal record mark, or an internal group mark followed by a terminal record mark. The operand may contain the following symbols only:

‡	record mark
⯒	group mark
b	blank

Figure 5 shows two ways in which the position of a terminal group mark could be indicated in defining a record consisting of 31 positions of data, three blanks, and a group mark.

TAG 6-15	OPERATION 16-20	NUM. 21-22	OPERAND 23-38	39
FIRSTWAY	RCD	31	A	
		4	⯒	
	≀			
SECONDWAY	RCD	34	A	
		1	⯒	

Figure 5

If the three blanks had been data, the definition for SECONDWAY would have been used. If the blanks had been group marks, the definitions in Figure 6 would have been used.

TAG 6-15	OPERATION 16-20	NUM. 21-22	OPERAND 23-38	39
NEWWAY	RCD	31	A	
		4	⯒⯒⯒⯒	

Figure 6

If one or more group marks appear within a record, they may be made terminal by defining them as a separate field and giving the field a tag. Figure 7 shows how the four group marks within a 90-position record may be made terminal by being defined as a separate field.

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
FIRSTPART	RCD	30	A+	
GROUPMARK		4	≢≢≢≢	
SECONDPART		56	A+	

Figure 7

Figure 8 shows two ways in which a record terminated by three blanks and a record mark could be defined.

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
FIRSTWAY	RCD	21	A	
		4	‡	
	≀			
SECONDWAY	RCD	24	A	
		1	‡	

Figure 8

If the final blank had been a group mark, the record could have been defined in either of the ways shown in Figure 9.

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
FIRSTWAY	RCD	21	A	
		4	≢‡	
	≀			
SECONDWAY	RCD	23	A	
		2	≢‡	

Figure 9

If all the blanks had been group marks, the record would have been defined as shown in Figure 10.

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
FIRSTWAY	RCD	21	A	
		4	≢≢≢‡	

Figure 10

If a record of less than 52 positions is being defined and it is not desired to give any information about the contents other than the location of group marks and/or record marks, the entire record may be defined by a layout operand. Figure 11 shows the definition of a 20-position record which contains a group mark in the fifteenth position and a terminal record mark.

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
MARKSONLY	RCD	20	≢	‡

Figure 11

COMMENTS FIELD. Comments may be started here. If comments continuation lines are written and the statement following the last continuation is blank in columns 16 and 17 of the operation field, the Processor will assume it is another RCD statement.

Using an RCD of Zero Length

If the first data field in a record exceeds 99 positions, its RCD definition may be preceded by an RCD of zero length. In this way, the definition becomes the second in a series of RCD statements, and the mnemonic code RCD may be omitted for the second. Columns 18-20 of the operation field may then be used as an extension of the numeric field. No space will be reserved for an RCD of zero length.

Restrictions on RCD Statements

The size of a data field may not exceed 80,000 positions. If a single RCD statement specifies a larger field size, the Processor will subtract 80,000 from the specified size and use the remainder as the size of the field when reserving storage space. A message to this effect is provided at assembly time.

Definitions of one or more terminal group marks may not indicate internal record marks or internal group marks. Definitions of a terminal record mark may not indicate internal record marks.

DEFINITION OF A CONSTANT FACTOR - CON

The function of a CON statement is to define a data field which will contain constant data and to provide the constant itself. The data may consist of any combination of alphameric characters and/or blanks. The CON statement is written as follows:

OPERATION FIELD. The mnemonic code CON is placed here. In a continuous series of CON statements, only the first need contain the code in the operation field. The Processor assumes that each immediately subsequent statement which is blank in columns 16-17 of the operation field is a CON and treats it accordingly. This assumption makes it possible in subsequent statements to use columns 18-20 of the operation field as an expansion of the numeric field.

NUMERIC FIELD. The size of the constant is entered here. A one-digit entry is placed in column 22 and need not be preceded by a zero. When the operation field contains the CON code, the numeric field is limited to two positions. However, when the operation field is blank and the statement has been preceded by another CON statement, columns 18-20 of the operation field may be used as an expansion of the numeric field. Under these conditions, in effect, the numeric field consists of five positions. Thus, constants which exceed 99 positions may be defined, but they may not be the first in a series of CON statements.

OPERAND FIELD. The constant is entered here. If the entry in the numeric field is not equal to the number of positions specified in the operand, the Processor will do one of the following:

1. Truncate the excess low-order positions when the numeric field entry specifies fewer positions than those contained in the operand.
2. Supply low-order zeros or blanks when the numeric field entry specifies more positions than those contained in the operand. Blanks will be supplied for alphameric fields; zeros will be supplied for signed numeric fields.

In Figure 12, the numeric field for TAG2 indicates that the constant contains nine low-order blanks.

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
TAG1	CON	5	ABCDE	
TAG2		20	THE DATE IS	
TAG3		4	A3‡Z	

Figure 12

Defining a Numeric Constant. A constant consisting of signed numeric data must contain a plus or minus sign in column 23 of the operand field. If the data is a mixed or pure decimal, the decimal point should be placed in the appropriate position. In storage, the low-order position of the field is signed accordingly. However, neither the sign nor the decimal point is included in the count of field positions for the numeric field entry. A signed numeric constant that exceeds 99 integer or 99 decimal positions should not be referenced by a general purpose macro-instruction.

Unsigned numeric data consisting of integers only is written starting in column 23 of the operand field. Unsigned numeric data consisting of mixed or pure decimals should not be specified as a constant if it is to be referenced by an Automatic Decimal Point macro-instruction, because it will be treated as alphameric data containing a period.

In Figure 13, note the following: the TAG3 constant will appear in storage as 8bbb, the TAG4 constant will appear as 64000 with a plus sign over the low-order zero, and the TAG5 constant will appear as 365 with a minus sign over the 5.

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
TAG1	CON	4	+75.25	
TAG2		3	845	
TAG3		4	8	
TAG4		5	+64	
TAG5		3	-3.65	

Figure 13

Defining a Constant of Record Marks and/or Group Marks. It may be desired to supply a constant of record marks and/or group marks as the terminal field of a record. For example, to follow a 33-position data field with a blank and a record mark, the definition would be written as shown in Figure 14.

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
	RCD	33	A	
CONSTANT	CON	2	‡	

Figure 14

If a data field containing a 42-position record is to be followed by a constant of two group marks and a record mark, the definitions in Figure 15 would be used:

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
	RCD	42	A	
CONSTANT	CON	3	⯒⯒‡	

Figure 15

COMMENTS FIELD. Comments may be started here. If comments continuation lines are written and the last continuation line is to be followed by another CON statement, one of the following things must also be done:

1. Place a zero in column 22 of each comments continuation line.
2. Use the mnemonic code CON in the statement which follows the last continuation line.

If neither is done, the Processor will interpret the statement following the last comments continuation as an RCD.

Restrictions on CON Statements

A one-position CON statement should be used to supply a plus sign or a minus sign as an alphameric constant. If an alphameric constant consisting of a plus or minus sign followed by numeric characters is desired, a one-position CON statement should be used to define the sign, and another CON should be used to define the numeric characters as an unsigned numeric constant.

DEFINITION OF A FLOATING POINT NUMBER - FPN

The function of an FPN statement is to define a data field for constant numeric data and to provide the data in floating point form. Numeric data should be defined in floating point form when there is a possibility that the limits of the accumulator might be exceeded during arithmetic operations with the data if it were defined in fixed point form.

Floating point form consists of a mantissa and an exponent. The mantissa is a pure decimal with a non-zero high-order digit; the exponent is a number specifying a power of ten. When the mantissa is multiplied by the power of ten that the exponent specifies, the data is produced in fixed point form. The following lists show the same data expressed in both forms.

Fixed	Floating
+9427.38	$+.942738 \times 10^4$
-.3264	$-.3264 \times 10^0$
+.0035	$+.35 \times 10^{-2}$
-623	$-.623 \times 10^3$

The FPN statement is written as follows:

OPERATION FIELD. The mnemonic code FPN is placed here. In a continuous series of FPN statements, only the first need contain the code in the operation field. The Processor assumes that each immediately subsequent statement which is blank in columns 16-17 of the operation field is an FPN statement and treats it accordingly.

NUMERIC FIELD. This is left blank; the Processor assumes 10 positions.

OPERAND FIELD. The exponent and the mantissa, each preceded by a plus or minus sign, are placed here in the following format: ±EE±DDDDDDDD.

The exponent must be a two-position number, as specified by EE. The sign which precedes the exponent indicates the direction in which the decimal has been moved in order to convert the data from fixed point to floating point form. The plus sign indicates the decimal has been moved to the left; the minus sign indicates the decimal has been moved to the right.

As indicated by DDDDDDDD, the mantissa may consist of up to eight digits and is preceded by the sign of the number itself. If fewer than eight digits are specified, the Processor will supply low-order zeros to complete the mantissa; if more than eight are specified, the Processor will truncate the excess low-order digits. When the data is placed in storage, the signs are placed over the low-order positions of the exponent and the mantissa.

Figure 16 shows a list of fixed point numbers, their corresponding FPN definitions, and the constants that would be created from them.

COMMENTS FIELD. Comments may be started here. If comments continuation lines are written and the last continuation line is to be followed by another FPN definition, one of the following things must also be done:

1. Place a zero in column 22 for each comments continuation line.
2. Use the mnemonic code FPN in the statement immediately following the final continuation line.

If neither is done, the Processor will interpret the statement following the last continuation line as an RCD.

Restrictions on FPN Statements

The absolute value of the exponent may not exceed 99. An exponent of 00 is signed +.

FPN definitions may not be referenced by any Automatic Decimal Point macro-instructions. The programmer must provide his own macro-instructions and/or subroutines in order to calculate with floating point numbers, because the Automatic Decimal Point macro-instructions calculate with numeric data in fixed point form only.

DEFINITION OF A REPORT FORMAT - RPT

The function of an RPT statement is to define a data field for numeric data which will be printed in a report and to specify the print format for the data. The RPT field may be referenced by macro-instruc-

Fixed Point Form	TAG	OPERATION	NUM.	OPERAND		Constants Placed in Storage
	6 15	16 20	21 22	23 38	39	
1. +589.46782		FPN		+03+58946782		1. $0\overset{+}{3}5894678\overset{+}{2}$
2. +.0025				-02+25		2. $0\overset{-}{2}2500000\overset{+}{0}$
3. -4327.9				+04-43279		3. $0\overset{+}{4}4327900\overset{-}{0}$
4. -.063				-01-63		4. $0\overset{-}{1}6300000\overset{-}{0}$
5. -.4792				+00-4792		5. $0\overset{+}{0}4792000\overset{-}{0}$
6. +17482.18936				+05+1748218936		6. $0\overset{+}{5}1748218\overset{+}{9}$

Figure 16

tions which place the numeric data in the field and supply the elements of the desired format. The following elements may be specified in the definition:

1. Commas and/or a decimal point
2. Fixed or floating dollar sign
3. The printing or suppressing of leading zeros
4. Asterisk protection
5. Indication of the numeric field sign
6. The blanking of a field of zeros

The RPT statement is written as follows:

OPERATION FIELD. The mnemonic code RPT is placed here. In a continuous series of RPT definitions, only the first need contain the code. The Processor assumes that each immediately subsequent statement which is blank in columns 16-17 of the operation field is an RPT statement and treats it accordingly.

NUMERIC FIELD. The size of the RPT field is entered here. All positions of the format, as shown by a layout in the operand field, must be counted. The count consists of the positions for the numeric data and any commas, decimal points, dollar signs, and positions reserved for printing the sign of the field.

OPERAND FIELD. The layout of the report format is started here; it consists of the symbols used to define the numeric characters, and the symbols for a dollar sign, a comma, and a decimal point if any are used. The layout may also contain one or two blank positions reserved for printing the sign of the field. Usually, the layout is followed by a set of indicators which provide the macro-instructions with additional information about the desired print format. In explaining the method of laying out the format, three sets of data will be used as examples throughout this section: the first consists of four integer and two decimal positions; the second consists of three decimal positions; the third consists of five integer positions.

Indicating Numeric Characters, Commas, Decimal Point. Xs and Zs are used to indicate the position of each numeric character in the format. If commas and/or a decimal point are desired, the symbols for them are placed in the appropriate positions. The numeric positions of the format are defined as follows:

1. Decimal positions. Zs must be used to define all decimal positions. Any trailing, i.e., significant, zeros in the data entering these positions will be retained and printed.

2. Integer positions. Xs and/or Zs may be used to define integer positions. The treatment of any leading, i.e., insignificant, zeros in the data entering these positions depends on whether the position in which the zero occurs is defined by a Z or an X. If the position is defined by a Z, the zero will be retained and printed; if it is defined by an X, the zero will be converted to a blank. Xs may be used to the left of Zs but not to the right of them. If the format layout does not contain a decimal point, the Processor assumes that a field of integers is being defined.

In Figure 17, the MIXED and INTEGER definitions indicate that any leading zeros are to be replaced by blanks. Notice that no decimal point is specified in the INTEGER field.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
MIXED	RPT	8	X,XXX.ZZ	
DECIMAL		4	.ZZZ	
INTEGER		5	XXXXX	

Figure 17

If 004320 were placed in the MIXED field defined in Figure 17, it would be printed as bbb43.20 (the comma having been replaced by a blank).

The MIXED and INTEGER fields are redefined in Figure 18 so that leading zeros will be retained. The MIXED definition requests that leading zeros which occur in the two low-order integer positions be printed. The INTEGER definition requests that leading zeros be printed in all but the high-order position.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
MIXED	RPT	8	X,XZZ.ZZ	
INTEGER		5	XZZZZ	

Figure 18

If 000120 were placed in the MIXED field defined in Figure 18, it would be printed as bbb01.20, and if 00089 were placed in the INTEGER field, it would be printed as b0089.

Leading zeros may also be replaced by asterisks. This is called asterisk protection and is requested by an indicator which is placed immediately after the format layout. The indicator consists of a lozenge, an asterisk, and a lozenge (¤*¤) and is not included in the count for the numeric field. In Figure 19, the INTEGER field is defined for complete asterisk protection. The MIXED field, however, is defined for asterisk protection only in the positions defined by Xs.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
INTEGER	RPT	5	XXXXX¤*¤	
MIXED		8	X,XXX.ZZ¤*¤	

Figure 19

The position of the decimal point can be indicated to macro-instructions which handle numeric data without having the point appear in the printed report. This is done by placing the symbol D in the appropriate position of the layout. The D is not included in the count of positions for the numeric field. This may be seen in Figure 20.

TAG 6 15	OPERATION 16 20	NUM. 21 22	OPERAND 23 38	39
MIXED	RPT	7	X,XXXDZZ	
DECIMAL		3	DZZZ	

Figure 20

Indicating the Position and Treatment of Dollar Signs. The dollar sign, if desired in the printed report, is written to the left of the high-order position of the format layout and is included in the count for the numeric field. A fixed or floating dollar sign can be specified as part of the print format through indicators which are placed to the right of the format layout. The indicators are surrounded by lozenge symbols (¤) and are not included in the count for the numeric field, because they are not part of the format layout. A fixed dollar sign is printed in the same position for each use of the data in the report.

If a fixed dollar sign with asterisk protection is desired, the format layout is immediately followed by an indicator consisting of a lozenge, an asterisk, and a lozenge (¤*¤). If a fixed dollar sign without asterisk protection is desired, the format layout is not followed by any dollar sign indicators. If any leading zeros occur in the data, they will be maintained or replaced by blanks, depending on whether Zs or Xs are used in the integer positions of the format layout.

A floating dollar sign is shifted so that it is printed to the left of the first numeric character in each set of data. It is requested by an indicator consisting of a lozenge, a dollar sign, and a lozenge (¤$¤) placed to the immediate right of the format layout.

Figure 21 shows one field as it would be defined to request each of the following: a floating dollar sign; a fixed dollar sign with asterisk protection; a fixed dollar sign without asterisk protection and with leading zeros converted to blanks; a fixed dollar sign without asterisk protection and with up to three leading zeros retained; no dollar sign but asterisk protection.

TAG 6 15	OPERATION 16 20	NUM. 21 22	OPERAND 23 38	39
MIXED1	RPT	9	$X,XXX.ZZ¤$¤	
MIXED2		9	$X,XXX.ZZ¤*¤	
MIXED3		9	$X,XXX.ZZ	
MIXED4		9	$X,ZZZ.ZZ	
MIXED5		8	X,XXX.ZZ¤*¤	

Figure 21

Assume that 003418 and 000570 are placed in each of the fields defined in Figure 21. The definitions would cause the data to be printed as follows:

MIXED1	$34.18	$5.70
MIXED2	$***34.18	$****5.70
MIXED3	$ 34.18	$ 5.70
MIXED4	$ 034.18	$ 005.70
MIXED5	***34.18	****5.70

Note that the commas in MIXED2 and MIXED3 are converted to an asterisk and a blank respectively. In MIXED4, and MIXED5, the comma is converted to a blank.

Indicating Field Signs and Zero Fields. Sets of characters which occupy one or two positions are available for printing either or both of the following in the report:

1. An indication of the sign of the field supplying data to be placed in the RPT field.
2. An indication that the field supplying data consists of zeros.

The requested characters will be printed to the right of the data.

One or two blank positions, depending on which set of characters is requested, must be added to the low-order portion of the format layout and must be included in the count for the numeric field entry. These blank positions are considered part of the layout. The special characters, called field sign indicators, are written to the right of the dollar sign indicator and its accompanying lozenges. Each character is also followed by a lozenge.

At this point, it is necessary to discuss the lozenges which separate the indicators in the RPT operand. Not only are the indicators significant to the Processor, but the presence or absence of the associated lozenges is also significant. When an option is not desired, the indicator which requests it must be omitted. If no subsequent options are to be requested in the same operand, the lozenge associated with the omitted indicator is also omitted. However, the lozenge is retained and placed back-to-back with the preceding lozenge if subsequent options are requested in the operand. The lozenge placement indicates to the Processor which option or options are not desired. A lozenge which may be omitted when its associated indicator and all subsequent indicators are omitted is called a conditional lozenge.

The lozenges associated with the dollar sign indicator are conditional. When a dollar sign is not included in the format layout or when a fixed dollar sign without asterisk protection is desired, no dollar sign indicator is required. The associated lozenges may be omitted unless a field sign is being requested. In this case, the dollar sign lozenges must be placed back-to-back and must precede all field sign indicators and their associated lozenges.

The field sign lozenges are not conditional. If any field sign indicators are used, the lozenge associated with each indicator must be placed after the indicator itself, or must be placed back-to-back with the preceding lozenge when the indicator is omitted.

The full dollar sign and field sign indicator structure is: ¤X_1¤X_2¤X_3¤X_4¤

X_1 is the dollar sign indicator or is omitted. The lozenges are conditional.

X_2 is the negative field sign indicator or is omitted.

X_3 is the zero field indicator or is omitted.

X_4 is the positive field sign indicator or is omitted.

The field sign indicators are as follows (b designates a blank):

1. One-position indicators: b - * +
2. Two-position indicators: b- b* ** CR DR DB

If indicators from the first set are used, one blank position must appear as the final position of the format layout; if indicators from the second set are used, two blank positions must appear as the final positions of the format layout.

The symbols CR, DB, -, and b- may be used for the negative indicator only. The symbols DR and + may be used for the positive indicator only. The other symbols are interchangeable. A blank is generated in the sign position when the condition associated with an omitted indicator is encountered.

It is possible to leave one blank position as the final position of the format layout, use the dollar sign indicator and its lozenges, but omit all field sign indicators and their associated lozenges. In this case, a blank will be generated in the sign position for both zero and positive fields, and a minus sign will be generated for negative fields. If a dollar sign indicator is not desired, the format layout can be terminated with the blank position, which must be included in the count for the numeric field entry.

The definition in Figure 22 requests a floating dollar sign. It also specifies that the minus, asterisk, and plus symbols are to be printed after negative, zero, and positive fields, respectively. One blank position for sign indication terminates the layout.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
MIXED1	RPT	10	$X,XXX.ZZ ¤$¤-¤*	¤+¤

Figure 22

Assume that the definition in Figure 22 defines the RPT field for the data shown below:

Data Entering RPT Field	RPT Field Printed
$03257\bar{0}$	$325.70-
$00000\overset{+}{0}$	$.00*
$45763\overset{+}{8}$	$4,576.38+

Figure 23 shows a request for a fixed dollar sign with asterisk protection, with the symbol CR printed after negative fields and the symbol DR printed after positive fields. Two blank positions for sign indication terminate the format layout.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
MIXED2	RPT	11	$X,XXX.ZZ ¤**¤CR	¤¤DR¤

Figure 23

Assume that the definition in Figure 23 defines the RPT field for the data shown below:

Data Entering RPT Field	RPT Field Printed
$00395\bar{5}$	$***39.55CR
$00000\overset{+}{0}$	$*****.00
413675	$4,136.75DR

Note that the symbol D for the decimal point is not included in the count of the format positions in Figure 24. Only the three numeric character positions and the two blank positions for field sign indication are counted. The sign indicators specify that the dollar sign is omitted and that a negative field is to be indicated by two asterisks.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
DECIMAL	RPT	5	DZZZ ¤¤**¤¤¤	

Figure 24

The definition in Figure 25 allows one position for field sign indication but does not contain a dollar sign or any sign indicators. Consequently, a minus sign will be generated for a negative field, and a blank will be generated for zero and positive fields. The Zs specify that leading zeros are not to be converted to blanks.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
INTEGER1	RPT	6	ZZZZZ	

Figure 25

Assume that the definition in Figure 25 defines the RPT field for the data shown below:

Data Entering RPT Field	RPT Field Printed
$0027\bar{8}$	00278-
$0000\overset{+}{0}$	00000
$3462\overset{+}{8}$	34628

Figure 26 specifies a floating dollar sign and two asterisks printed to the right of zero fields. All positions of a zero field except the sign positions will be blanked; this includes the dollar sign, comma, and decimal point positions.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
INTEGER1	RPT	9	$XX,XXX ¤$¤¤**¤	¤

Figure 26

Blank-If-Zero Option. If this is requested, any defined commas, the decimal point, and a floating dollar sign will be blanked along with the numeric positions when the field contains all zeros. Only a fixed dollar sign will not be blanked. To request the option, the symbol BZ is used as the zero field indicator. All five lozenges must be included whether or not BZ is the only indicator used. This option is independent of the other sign options; consequently, when BZ is the only indicator used, it is not necessary to terminate the format layout with any blank positions.

The definition for MIXED1 in Figure 27 specifies only that the field is to be blanked when it contains all zeros. The definition for MIXED2 calls for a fixed dollar sign with asterisk protection, a minus sign following a negative field, and the Blank-If-Zero option. A positive field will be printed without any field sign indication, and the fixed dollar sign will be retained when a zero field is blanked.

TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
MIXED1	RPT	7	XXXX.ZZ⌑⌑⌑BZ⌑⌑	
MIXED2	RPT	10	$X,XXX.ZZ ⌑*⌑-⌑B	Z⌑⌑

Figure 27

COMMENTS FIELD. Comments may be started here. If comments continuation lines are written and the last continuation line is to be followed by another RPT definition, one of the following things must also be done:

1. Place a zero in column 22 of each comments continuation line.
2. Use the mnemonic code RPT in the statement which follows the final comments continuation line.

If neither is done, the Processor will interpret the statement following the last continuation line as an RCD.

Restrictions on RPT Statements

The format layout of an RPT operand may not exceed 52 positions. One and two-position field sign indicators may not be mixed in the same statement.

The number of positions in the format layout must be identical to the entry in the numeric field. If blank positions for sign indication are included in the layout, it is important to see that no more than two blank positions are allocated. The number of commas in the format layout may not exceed nine.

DEFINITION OF A CONTINUOUS PORTION OF MEMORY - NAME

A NAME statement has two functions which may be used independently of or in conjunction with each other.

1. To identify a series of adjacent data fields as the interior fields of an area so that they may be treated as a unit.
2. To specify the final digit or digits of the starting location to which a data field is assigned.

ENCLOSING ADJACENT FIELDS. A NAME statement which identifies fields as interior to an area may be said to enclose the fields. The following Autocoder III statements define fields that may be enclosed by a NAME statement:

1. Area definitions: RCD,CON,FPN,RPT,NAME
2. Switch definitions: CHRCD, BITCD
3. Address constants: ACON4,ACON5,ACON6, ADCON

The interior fields of the NAME area may be referenced individually by their tags or referenced as a unit by the tag of the NAME area. For example, a work area may be defined as a NAME area consisting of four interior fields. Each field may be operated on individually, but the fields may also be moved to and from the work area as a unit rather than one at a time.

SPECIFYING A LOCATION. The location requested by the NAME statement is assigned to the high-order position of the immediately subsequent field. The NAME statement specifies what the final digit or digits of the address may be. The next available location which ends in the requested digit or digits is then assigned to the high-order position of the field defined immediately after the NAME statement. Suppose that a 4/9 location is requested, i.e., that the high-order position of the field should be assigned a location ending in 4 or 9, whichever is available first. If 00012 is the last location assigned prior to the request, location 00014 will be assigned; and if 00017 is the last assignment, then 00019 will be assigned. In either case, if a 00 assignment had been requested, 00100 would have been assigned.

The NAME statement is written as follows:

OPERATION FIELD. The mnemonic code NAME is placed here.

NUMERIC FIELD - This field is left blank if Processor is to assign the next available location to the name. If a specific address ending is desired for the starting location, one of these codes is placed in column 22:

Code	Requests Location Ending In	Code	Requests Location Ending In
0 or 5	0 or 5	4 or 9	4 or 9
1 or 6	1 or 6	A	0
2 or 7	2 or 7	B	00
3 or 8	3 or 8	C	000

For purposes of location assignment, an X in column 22 has the same effect as a blank. However, if an X is used, the Processor will not make the terminal location of the field available for the macro-generation phase. (The X is used for generation of higher languages; preferably, it should not be used in Autocoder.)

OPERAND FIELD. This field is left blank when NAME is used only to request a location assignment. When NAME is used to enclose a series of interior fields, the tag of the interior data field which terminates the NAME is placed in the operand field. If an operand is used, the NAME statement itself must be tagged.

The NAME statement in Figure 28 requests the positioning of FIELD1 starting at the first available address ending in 0. The statement also makes four fields interior to STARTNAME by designating the ENDNAME field as the terminal field.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
STARTNAME	NAME	A	ENDNAME	
FIELD1	RCD	4	N	
FIELD2	1	25	A+	
FIELD3		5	#+03.02	
ENDNAME	CON	1	‡	

Figure 28

Figure 29 shows NAME used to position the RPT field ANYTAG in the next available address ending in 2 or 7.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
	NAME	2		
ANYTAG	RPT	7	$ZZZ.ZZ	

Figure 29

NAME is used in Figure 30 to identify the interior fields of the area tagged BEGIN.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
BEGIN	NAME		END	
FIELD1	FPN		+03+4.38	
END			+02+67845	

Figure 30

Figure 31 shows a way of creating the constant +12345 in such a way that it will not appear in storage as 1234E ($12\overset{+}{5}$).

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
ALPHA	NAME		ENDALPHA	
	CON	1	+	
ENDALPHA		5	12345	

Figure 31

COMMENTS FIELD. Comments may be started here.

Information Provided by the Processor

The Processor counts the total number of positions occupied by the interior fields of a NAME area. A message indicating the total will appear in the listing immediately following the entry specified as the terminal field definition.

Internal NAMEs

One or more NAME areas may be made internal to another NAME. The operand of each internal and outer NAME statement must contain the tag of the field which terminates it. Internal NAMEs may be terminated by the same field which terminates the outer NAME, or they may be terminated by fields which are internal to the outer NAME.

In Figure 32, the OUTERNAME is terminated by the CON field ENDOUTER, while INNERNAME is terminated by the RCD field ENDINNER.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
OUTERNAME	NAME	0	ENDOUTER	
FIELD1	RCD	5	+	
FIELD2	1	50	A+	
INNERNAME	NAME		ENDINNER	
FIELD3	RCD	12	A+	
FIELD4		7	#+04.03	
ENDINNER		1	‡	
FIELD5	RPT	10	$XX,XXX.ZZ¤¤	
FIELD6	RCD	35	A	
ENDOUTER	CON	5	‡‡‡‡‡	

Figure 32

In Figure 33, both FIRSTNAME and SECONDNAME are terminated by the RCD field ENDFIRST.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
FIRSTNAME	NAME	0	ENDFIRST	
FIELD1	RCD	25	A+	
		5	+	
SECONDNAME	NAME		ENDFIRST	
	RPT	9	$ZZ,ZZZ ¤¤CR¤¤D	R¤
	RCD	5	N	
ENDFIRST		1	‡	

Figure 33

Restrictions on NAME Statements

The number of positions enclosed in a NAME may not exceed 80,000. If the cumulative limit is exceeded, the Processor will subtract 80,000 from the total and use the remainder when developing the message which specifies the size of the NAME area.

Internal NAME statements should not specify location assignments. The operand of one NAME statement, i.e., the tag of the termination field, cannot be the tag of another NAME entry.

The NAME statement itself must be tagged if the operand contains a tag.

No more than 32 NAME areas may be defined concurrently.

CHAPTER 3. SWITCH DEFINITIONS

Switches are programming or hardware devices used to control the path of a program. Three types of switches may be defined: data switches, program switches, and console switches. The statements used for each type are as follows:

1. Data Switches
   a. Character Code - CHRCD
   b. Bit Code - BITCD
2. Program Switches
   a. Switch Set to Transfer - SWT
   b. Switch Set to No Operation - SWN
3. Console Switches
   a. Alteration Switch - ALTSW

With one exception, the format of switch definition statements varies according to the type of switch being defined. The exception is the comments field. Comments about any switch may be started in the comments field of the definition statement. For those switches which must be defined by a set of statements, comments continuation lines may intervene between the first statement and the remaining statements, or the continuations may be placed in the comments fields of the remaining statements.

DATA SWITCHES

A data switch is a data field. There are two types of data switches: character code and bit code. The character code switch provides a method of relating alphameric codes to various meanings or conditions. The bit code switch provides a method of relating the bits which form a storage position to various meanings or conditions. Both character code and bit code switches are described by a set of statements, the first of which is the switch definition statement. It indicates whether a character code or bit code is being defined. The rest of the character code switch statements specify the alphameric codes which may occupy the switch and the condition which each code represents. The rest of the bit code switch statements designate the various bits of the storage position and the condition each bit represents. A character code switch may occupy one or two positions; a bit code switch may occupy only one position.

A record field may be defined as a data switch, and the switch may be interior to a record area defined by a NAME statement. The switch will be set each time a record is placed in the area. If the data switch is not defined as part of a record area, the program itself must set the switch. The way in which the switch is initially set depends on its use in the program. If the switch definition statement follows an RCD, the statement should not specify the initial setting. The Processor reserves storage space for the switch but does not set it to any code. If an initial setting has been specified, the Processor ignores it. However, the switch definition statement that does not follow an RCD should specify an initial setting. The Processor reserves space for the switch and sets it as specified. If the initial setting has been omitted, the Processor sets the switch to a blank.

Program Branch Control macro-instructions are normally used to set the switches ON or OFF or to test their settings. A character code switch is set ON by placing one of the defined codes in it and is set OFF by placing a blank in it. When a character code switch is tested, it is examined to see whether or not a given code is present. If it is, the switch is ON. If the switch contains anything other than the code designated in the test, the switch is OFF. A bit code switch is set ON by setting the designated bits ON and is set OFF by setting the designated bits OFF. When a bit code switch is tested, it is examined to see whether or not the bit designated in the test is ON. If it is, the switch is ON; otherwise, the switch is OFF.

Suppose that statements for a character code switch specify that codes A and B represent the conditions of Surplus and Deficit, respectively. If the switch is tested for the Surplus condition and A is present, the switch is ON. On the other hand, suppose the switch is tested for the Deficit condition. Now, if B is present, the switch is ON. In other words, the data switch must be tested for a condition which has been specified in its definition. If the code which represents the specified condition is present, the switch is ON. Otherwise, it is OFF.

Now suppose that the switch is a bit code switch and that the Surplus condition is represented by turning ON the 1-bit, while the Deficit condition is represented by turning ON the 2-bit. If the switch is tested for the Surplus condition and the 1-bit is ON, the switch is ON. It does not matter whether the 2-bit is ON or OFF, because the test does not specify the Deficit condition. It is possible, although not logical in this example, that the switch be ON for both conditions.

A character code switch may represent only one condition at any time, whereas a bit code switch may represent multiple conditions simultaneously. In each case, the number of ON states for a data switch is equal to the number of codes or bits specified in the switch definition.

Character Code - CHRCD

A character code switch is defined by a series of statements. The first is the CHRCD statement; its function is to define the switch as a character code switch and to specify the size and initial contents of the switch. The statements which follow the CHRCD statement specify the codes and the conditions they represent. The format of the set of statements is as follows:

Tag	Operation	Num	Operand
	CHRCD	n	X_1
T_1			C_1
T_2			C_2
T_3			C_3
etc.			etc.

n	is blank when defining a one-position switch. is 2 when defining a two-position switch.
X_1	is the initial contents of the switch or is blank.
T_1, T_2, T_3,...	are the tags of the codes. They specify the conditions the codes represent.
C_1, C_2, C_3,...	are the codes; any alphameric characters may be used. The codes may be composed of one or two characters, depending on what is specified in the numeric field.

If the CHRCD statement immediately follows an RCD statement, the CHRCD operand should be left blank. If the switch does not follow an RCD field, the operand of the CHRCD statement should specify the initial setting; otherwise, a blank will be placed in the switch.

Figure 34 shows a one-position character code switch defined as a portion of a record area. Notice that the switch is enclosed by a NAME statement. The NAME operand indicates that the statement tagged CANCELED terminates the NAME.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
RECORDAREA	NAME		CANCELED	
COMPANY	RCD	25	A	
	CHRCD			
NEW			N	
REGULAR			R	
CANCELED			C	

Figure 34

In Figure 35, the operand of the CHRCD statement specifies the initial switch setting, i.e., that the switch contains the code 18.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
	CHRCD	2	18	
NEWYORK			10	
BOSTON			06	
CHICAGO			18	
ATLANTA			27	

Figure 35

During the program assembly, the tag of each code is assigned to the storage position occupied by the switch. Suppose that the switch defined in Figure 34 is assigned location 000315. When instructions which reference NEW, REGULAR, and CANCELED are translated into machine language, 000315 will appear as the address portion of each one.

Figure 36 is part of a listing. Notice the machine language portions for both the switch definitions and the instructions which reference the switch.

Tag	Oper.	Nu	Operand	Loc	Op	Su	Address
	CHRCD			000343			
BLUE			A	000343			
GREEN			B	000343			
RED			C	000343			
Instructions that reference the switch:							
	CMP	1	GREEN	002129	4	1	000343
	CMP	1	RED	002624	4	1	000343
	CMP	1	BLUE	002679	4	1	000343

Figure 36

RESTRICTIONS ON A CHRCD SWITCH. A code should not be represented as a signed numeric character but as the alphabetic character equivalent to the signed numeric character. For example, A should be used to represent +1, J should be used to represent -1, etc.

The CHRCD statement should not be tagged, since the switch is referenced by the tags of the codes.

Bit Code - BITCD

A bit code switch is defined by a series of statements. The first is the BITCD statement; its function is to define the switch as a bit code switch and to specify the initial setting of the switch. The statements which follow the BITCD statement specify the bits and the conditions they represent. The format of the set of statements is as follows:

Tag	Operation	Num	Operand
	BITCD		X_1
T_1		B_1	
T_2		B_2	
T_3		B_3	
T_4		B_4	

X_1	is the initial setting of the switch or is blank.
$T_1 \ldots T_4$	are the tags of the bits. They specify the conditions which the bits represent when they are ON.
$B_1 \ldots B_4$	are the bit codes 1, 2, 4, and A.

If the BITCD statement immediately follows an RCD statement, the operand should be left blank. If the switch does not follow an RCD field, the operand of the BITCD statement should specify the initial setting. The setting is indicated by the alphameric character created when the desired bits are set ON.

A bit that contains zero (0) is defined as ON; a bit that contains one (1) is defined as OFF. For instance, if the 4-bit should be set ON initially, the operand may be any character that contains a zero in the 4-bit. If the 1, 4, and A bits should be ON, the operand may be any character that contains zeros in those bits. It is recommended that the selected character contain a zero in the 8-bit and a one in the B-bit so that the character in the switch will always be valid for printing purposes.

The bit code switch in Figure 37 indicates various types of payroll deductions and is defined as a portion of a record area. The maximum number of bits has been used.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
RECORDAREA	NAME		OTHER	
EMPLOYEE	RCD	25	A+	
	BITCD			
IRS		1		
FICA		2		
STATE		4		
OTHER		A		

Figure 37

The BITCD definition in Figure 38 specifies that GROSSTOTAL is to be set ON initially. The switch will contain B (12-2), thus setting the 1-bit to zero.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
	BITCD		B	
GROSSTOTAL		1		
NETTOTAL		2		

Figure 38

During the program assembly, the tag of each defined bit is assigned to the storage position occupied by the switch. Suppose that the switch defined in Figure 38 is assigned location 000100. When instructions which reference GROSSTOTAL and NETTOTAL are translated into machine language, 000100 will appear as the address portion of each one.

Figure 39 is taken from a listing. Notice the machine language portions for both the switch definition and the instructions which reference the switch.

Tag	Oper.	Nu	Operand	Loc	Op	Su	Address
	BITCD			000237			
EAST		1		000237			
WEST		2		000237			
NORTH		4		000237			
Instructions that reference the switch:							
	RCVS		EAST	002319	U		000237
	RCVS		WEST	002464	U		000237
	RCVS		NORTH	002739	U		000237

Figure 39

RESTRICTIONS ON A BITCD SWITCH. A bit code switch may not be used in a program for a 705 II or the 705 II portion of a 7080 program.

The BITCD statement should not be tagged, since the switch is referenced by the tags of the bits.

PROGRAM SWITCHES

A program switch is an instruction. Each time the switch is encountered, it causes the program to do one of two things:

1. To transfer to a designated instruction when the switch is ON.
2. To execute the next in-line instruction when the switch is OFF.

A program switch is defined by a single statement which specifies the initial switch setting. If the initial setting is ON, the switch statement becomes a Transfer instruction in the object program. If the initial setting is OFF, the statement becomes a No-Operation instruction in the object program.

Program Branch Control macro-instructions are used to set the switches ON or OFF and to test their settings. Setting the switch ON or OFF involves modifying the operation portion of the generated instruction to Transfer or No-Operation, respectively. Testing the switch involves determining whether or not it will cause the program to transfer. All program switch definition statements must be tagged so that the switches can be referenced by macro-instructions.

Switch Set to Transfer - SWT

The function of an SWT statement is to define a program switch which will be ON initially. The format of the SWT statement is as follows:

Tag	Operation	Num	Operand
T_1	SWT		X_1

T_1	is the tag of the switch.
X_1	is the tag of the instruction to which a transfer is to be made when the switch is ON.

As long as the switch is ON, a transfer occurs each time the switch is encountered. When the switch is encountered after it is set OFF, the transfer does not occur; the program proceeds instead to the next in-line instruction.

The SWT statement in Figure 40 indicates that LOOPSWITCH is to be set ON initially and that the transfer point is the instruction tagged STARTLOOP.

TAG 6-15	OPERATION 16-20	NUM. 21-22	OPERAND 23-38	39
LOOPSWITCH	SWT		STARTLOOP	

Figure 40.

RESTRICTIONS ON AN SWT SWITCH. A hand-coded Transfer instruction may not be referenced as a program switch with Program Branch Control macro-instructions. Since the hand-coded instruction will not be recognized as a switch, the proper coding will not be generated from any macro-instructions referencing it.

Switch Set to No Operation - SWN

The function of an SWN statement is to define a program switch which will be OFF initially. The format of the SWN statement is as follows:

Tag	Operation	Num	Operand
T_1	SWN		X_1

T_1	is the tag of the switch.
X_1	is the tag of the instruction to which a transfer is to be made after the switch is turned ON.

As long as the switch is OFF, no transfer occurs when the switch is encountered. The program proceeds instead to the next in-line instruction. After the switch is set ON, a transfer occurs each time the switch is encountered.

The SWN statement in Figure 41 indicates that LOOPSWITCH is to be set OFF initially and that when the switch is set ON, the transfer point is the instruction tagged STARTLOOP.

TAG 6-15	OPERATION 16-20	NUM. 21-22	OPERAND 23-38	39
LOOPSWITCH	SWN		STARTLOOP	

Figure 41

RESTRICTIONS ON AN SWN STATEMENT. A hand-coded No-Operation instruction may not be referenced as a program switch with Program Branch Control macro-instructions. Since the hand-coded instruction will not be recognized as a switch, the proper coding will not be generated from any macro-instructions referencing it.

CONSOLE SWITCHES

Console switches are the console alteration switches 0911-0916. Each is identified by one console switch statement. The switches themselves must be set ON or OFF manually by the console operator, either before or during the execution of the program. A console switch statement does not specify the initial switch setting. It merely provides a method of assigning a tag to an alteration switch so that it can be referenced by a Program Branch Control macro-instruction. The switch statement is not translated into a machine language instruction.

Alteration Switches - ALTSW

The function of the ALTSW statement is to designate a console alteration switch. The format of the statement is as follows:

Tag	Operation	Num	Operand
T_1	ALTSW	X_1	

T_1	is the tag of the switch statement.
X_1	is a code identifying the console switch. The codes are as follows:

Code	Switch Being Identified
A	0911
B	0912
C	0913
D	0914
E	0915
F	0916

Figure 42 shows switches 0911 and 0912 being identified.

TAG 6-15	OPERATION 16-20	NUM. 21-22	OPERAND 23-38	39
WEEKLYRUN	ALTSW	A		
MONTHLYRUN	ALTSW	B		

Figure 42

A one-for-one instruction is a symbolic instruction which is replaced by one machine instruction. It consists of a 705 or 7080 operation code and an Autocoder III operand. Figure 44 lists the 705 and 7080 operation codes. The basic Autocoder III operands are as follows:

1. tag
2. literal
3. actual
4. location counter
5. blank

A prefix, a suffix, or both may be added to some of the basic operands:

Prefix	Suffix
operand modifier	character adjustment
indirect address	

The format of an Autocoder III one-for-one instruction is summarized in the next section, "One-For-One Instruction Format." The balance of the chapter describes the basic operands and the prefix and/or suffix that may be added to each operand. Chapter 6, entitled "Address Constants," describes a specialized form of Autocoder III operand called an address constant literal.

The details of each 705 or 7080 operation are supplied in the following reference manuals:

1. "705 Data Processing System", Form A22-6506
2. "7080 Data Processing System", Form A22-6560

ONE-FOR-ONE INSTRUCTION FORMAT

Like other Autocoder III statements, a one-for-one instruction is tagged if it is to be referenced. The mnemonic operation code is placed in the operation field. No actual operation codes may be used. If the operation requires designation of the accumulator, an ASU, or a bit, the appropriate entry is placed in the numeric field. A one-for-one instruction has a single entry in the operand field; if necessary, the operand may be continued from the operand field into the comments field. The operand may not, however, be continued onto the next line of the coding sheet. Comments about the instruction may be started in the comments field.

BASIC OPERANDS

A description of the basic Autocoder III operands follows.

Tag

The tag may be that of the data field or the source program instruction involved in the operation.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
FIELD	RCD	13	#+07.06	
	≀			
INSTR	RAD		FIELD	

Figure 43.

Literal

A literal is actual data enclosed by literal signs (#). It may be any combination of alphameric characters and/or blanks, e.g., #A#, #bb3C#, #0500#, #GO TO END#, #+345#, #-.67#, #1234#, #+9.876#. The Processor creates a constant from a literal operand. The term "literal" is frequently used to refer to the literal operand or to the constant created from the literal.

As an example of the use of a literal operand, it may be necessary to calculate with a constant of +30. The constant could be defined by a CON statement, and the appropriate arithmetic instruction could reference the constant by having the tag of the CON as an operand. On the other hand, it might be desired to omit the CON and supply the constant directly by writing it as the literal operand of the arithmetic instruction. While a literal is a convenient way of supplying an occasional constant, those constants that are used repeatedly throughout the program should be supplied by CON statements.

If a signed numeric constant is desired, the first character following the literal sign must be a plus or minus sign. In storage, the low-order position of the constant will be signed. If the numeric data is a mixed or pure decimal, the decimal point will not appear in the constant. If an unsigned numeric constant is desired, the first character following the literal sign must be the first character of the numeric data. In storage, the constant will appear exactly as it is written in the literal. Thus, the constant created from an unsigned mixed or pure decimal will contain a decimal point. For this reason, unsigned mixed or pure decimals should not be written as the literal operands of arithmetic instructions, e.g., ADD, SUB.

A literal may also supply the floating point form of a signed numeric constant. It must be written in the format of an FPN operand: #±EE±XXXXXXXX#.

Name of Instruction	Mnemonic Code	Use in Programs For		
		705 II	705 III	7080
Add	ADD	x	x	x
Add Address to Memory	AAM		x	x
Add to Memory	ADM	x	x	x
Backspace	BSP	x	x	x
Backspace File	BSF		x	x
Blank Memory	BLM		x	x
Blank Memory Serial	BLMS		x	x
Channel Reset	CHR			x
Comma, No Operation	CNO			x
Compare	CMP	x	x	x
Control Read (Read 04-CRD)	RD 04‡			x
Control Write (Write 04-CWR)	WR 04‡			x
Divide	DIV	x	x	x
Dump Memory (Write 01)	DMP	x	x	x
Enable Compare Backward (ECB)	CTL 12‡			x
Enable Indirect Address	EIA			x
Enter Interrupt Mode	EIM			x
Enter 7080 Mode	EEM			x
Forward Space (Read 01)	FSP	x	x	x
Leave Interrupt Mode	LIM			x
Leave Interrupt Program	LIP			x
Leave 7080 Mode	LEM			x
Lengthen	LNG	x	x	x
Load	LOD	x	x	x
Load Address	LDA		x	x
Load Four Characters	LFC			x
Load Store Bank	LSB			x
Multiply	MPY	x	x	x
No Operation	NOP	x	x	x
No Operation, Comma	CNO			x
Normalize and Transfer	NTR	x	x	x
Read 00	RD	x	x	x
Read 01 (Forward Space)	FSP	x	x	x
Read 02 (Read Memory Address)	RMA		x	x
Read 03 (Sense Status Trigger)	SST			x
Read 04 (Control Read - CRD)	RD 04‡			x
Read 05 (Read Memory Block -RMB)	RD 05‡			x
Read Memory Address (Read 02)	RMA		x	x
Read Memory Block (Read 05 - RMB)	RD 05‡			x
Read While Writing	RWW	x	x	x
Receive	RCV**	x	x	x
Receive Serial	RCVS**	x	x	x
Receive Ten Characters	RCVT**			x
Reset and Add	RAD	x	x	x
Reset and Subtract	RSU	x	x	x
Rewind	RWD	x	x	x
Rewind and Unload	RUN			x
Round	RND	x	x	x
Select	SEL	x	x	x
Send	SND		x	x
Sense Status Trigger (Read 03)	SST			x
Set Bit Alternate	SBA		x	x
Set Bit 1	SBN*		x	x
Set Bit Redundant	SBR		x	x
Set Bit 0	SBZ*		x	x
Set Control Condition (Write 03)	SCC			x
Set Density High	SDH			x
Set Density Low	SDL			x
Set Left	SET	x	x	x
Set Record Counter (Write 02)	SRC		x	x
Set Starting Point Counter	SPC			x
Shorten	SHR	x	x	x
Sign	SGN	x	x	x
Skip Tape	SKP		x	x
Stop	HLT	x	x	x
Store	ST	x	x	x
Store for Print	SPR	x	x	x
Subtract	SUB	x	x	x
Suppress Print or Punch	SUP	x	x	x
Ten Character Transmit	TCT			x
Transfer	TR	x	x	x
Transfer Any	TRA	x	x	x
Transfer Auto Restart	TAR			x
Transfer Echo Check	TEC		x	x
Transfer on Equal	TRE	x	x	x
Transfer on High	TRH	x	x	x
Transfer to Interrupt Program	TIP			x
Transfer Instruction Check	TIC		x	x
Transfer Machine Check	TMC		x	x
Transfer Nonstop (Transfer 07 - TNS)	TRA 07‡			x
Transfer Overflow Check	TOC		x	x
Transfer on Plus	TRP	x	x	x
Transfer Read-Write Check	TRC		x	x
Transfer Ready	TRR		x	x
Transfer Sign Check	TSC		x	x
Transfer on Signal	TRS	x	x	x
Transfer and Store Location	TSL		x	x
Transfer Switch A On (0911)	TAA		x	x
Transfer Switch B On (0912)	TAB		x	x
Transfer Switch C On (0913)	TAC		x	x
Transfer Switch D On (0914)	TAD		x	x
Transfer Switch E On (0915)	TAE		x	x
Transfer Switch F On (0916)	TAF		x	x
Transfer Synchronizer Any	TSA		x	x
Transfer Transmission Check	TTC		x	x
Transfer on Zero	TRZ	x	x	x
Transfer on Zero Bit	TZB*		x	x
Transmit	TMT	x	x	x
Transmit Serial	TMTS	x	x	x
Turn off I-O Indicator	IOF	x	x	x
Turn on I-O Indicator	ION	x	x	x
Unload	UNL	x	x	x
Unload Address	ULA		x	x
Unload Four Characters	UFC			x
Unload Storage Bank	USB			x
Write 00	WR	x	x	x
Write 01 (Dump Memory)	DMP	x	x	x
Write 02 (Set Record Counter)	SRC		x	x
Write 03 (Set Control Condition)	SCC			x
Write 04 (Control Write - CWR)	WR 04‡			x
Write 05 (Write Multiple Control -WMC)	WR 05‡			x
Write and Erase 00	WRE	x	x	x
Write and Erase 01	WRE 01	x	x	x
Write Multiple Control (Write 05-WMC)	WR 05‡			x
Write Tape Mark	WTM	x	x	x
IBM 760 Operations				
Read or Write Tape, Early Start	RWT	x	x	x
Read or Write Tape, Write on Printer	RWS	x	x	x
Reset 760 Counter	RST	x	x	x
Write on Printer and Magnetic Tape	PTW	x	x	x
IBM 777 Operations				
Bypass TRC	BPC	x	x	x
Prepare to Read While Writing	PRW	x	x	x
Read Tape to TRC	RTS	x	x	x
Write TRC to Tape	WST	x	x	x

NOTES:

* Place a 1, 2, 4, 8, A, or B in column 22 to designate the bit (TZB can have a C also). If column 21 is not blank, the Processor assumes that ASU zoning, valid or invalid, has been designated.

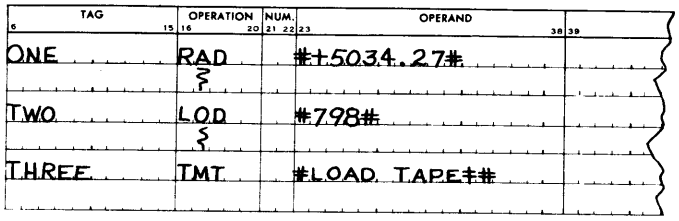
**The three different mnemonics for the receive instruction (RCVS, RCV, and RCVT) indicate to the Processor the type of address to be assigned. If the mnemonic is RCV, the location assigned is the high-order address of the field specified in the operand of the instruction. For an RCV, 4 is added to the high-order address of the field. Since an RCV is generally used when a 4/9 ending is desired (as with a TMT or SND), the high-order address of the field should end in a 0 or 5. An RCVT is assigned the high-order address of the field plus 9. Since RCVT is used when a 9 ending is desired (as with a TCT), the high-order address of the field should end in 0. If the generated address does not end in a 4 or 9 (RCV) or 9 (RCVT), a 4/9 check message is prepared.

‡The 7058 Processor will not accept the mnemonics CRD, CWR, ECB, RMB, TNS, and WMC.

Figure 44. 705 and 7080 Mnemonic Codes for One-For-One Instructions.

Trailing zeros will be supplied when the literal contains fewer than eight mantissa positions. For example, the literal #+03-7# will appear in storage as $0\overset{+}{3}7000000\bar{0}$.

The length of a literal must be a multiple of five when used with an operation which requires a 4 or 9 location. The literal must also contain a record mark in the low-order position if it is used with a TMT operation. Such literals are positioned in the literal table so that the high-order character occupies a 0 or 5 location.



TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
ONE	RAD		#+5034.27#	
	≀			
TWO	LOD		#798#	
	≀			
THREE	TMT		#LOAD TAPE‡#	

Figure 45

The Processor places all constants that it creates from literal operands in an area of storage called the literal table. Although the same literal may be used in several statements, it will appear only once in the table. The Processor classifies literals and assigns them to the table according to whether they are signed or unsigned:

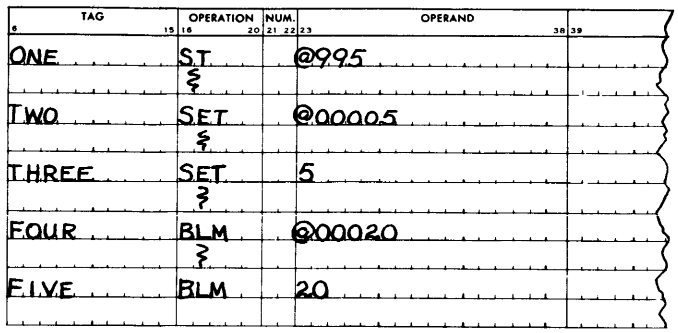
1. Any literal containing a sign in the first position is automatically classified as signed. If the signed literal supplies numeric data, it appears in storage as previously described. If the literal supplies alphameric data, the sign is placed in the low-order position of the constant if it is occupied by a numeric character. Otherwise, the sign is ignored.
2. Any literal that does not contain a sign in the first position is automatically classified as unsigned. As previously indicated, the constant appears in storage in exactly the same form in which it is written on the coding sheet.
3. A literal symbol may not appear within a literal unless it is the first character of the literal.

Actual

An actual operand is a set of numeric characters, usually preceded by the actual address symbol (@), which designates one of the following:

1. An actual storage location
2. A setting for the accumulator or an ASU
3. The size of a block of storage positions

The @ symbol need not be used when the operand contains less than five numeric characters and is used with one of the following operations: BLM, BLMS, CTL, HLT, LIP, LNG, RND, SEL, SET, SHR, SPC, SRC, TRANS. Notice in Figure 46 that the SET and BLM instructions have been written two ways.



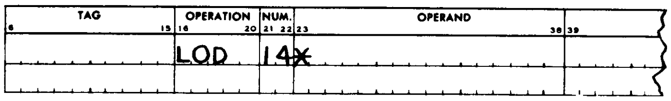
TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
ONE	ST		@995	
	≀			
TWO	SET		@00005	
	≀			
THREE	SET		5	
	≀			
FOUR	BLM		@00020	
	≀			
FIVE	BLM		20	

Figure 46

RESTRICTIONS. If the program is operating in 7080 mode with 160K memory, the highest actual operand that may be used is 159999. Otherwise, 79999 is the highest that may be used. (This includes 7080 mode with 80K memory.) If a higher one is encountered, 80000 will be subtracted from it and the remainder used as the operand. A message to this effect is provided at assembly time. If a six-position actual operand is encountered outside the 7080 mode portion, the low-order digit will be ignored. (See Chapter 7 for a further definition of the "7080 mode portion of a program.")

Location Counter

A location counter is represented by the asterisk symbol (*), which designates the low-order position of the instruction in which it appears. Since each instruction occupies five positions in the object program, an instruction containing a location counter references its own low-order position. The effect of the instruction in Figure 47 is to cause the 4 or 9 location assigned to the instruction to be placed in ASU 14.



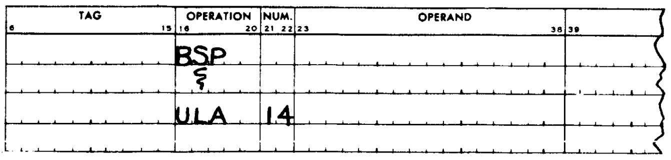
TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
	LOD	14	*	

Figure 47

Note: The versatility of a location counter is more fully utilized when the counter is character-adjusted. This use is explained in the following section, "Additions to Basic Operands."

Blank

A blank operand is one which has blanks in the first 10 columns of the operand field. Blank operands should be used if the instruction is initialized by the program or if the operation itself does not require an address. In the object program, a blank operand is replaced by zeros.



TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
	BSP			
	≀			
	ULA	14		

Figure 48

ADDITIONS TO BASIC OPERANDS

A description of the suffix and the prefixes that may be added to an Autocoder III operand follows.

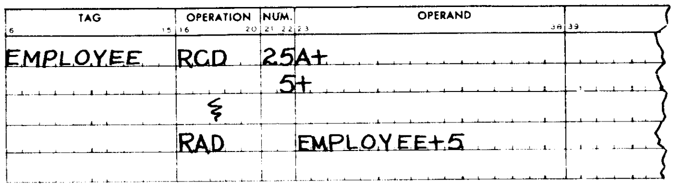
Character Adjustment

Character adjustment is designated by a suffix to the basic operand. A reference to an untagged field, an untagged instruction, or a particular position within a field or an instruction can be made by using character adjustment. The suffix consists of an arithmetic operator that specifies the type of operation and one or more numeric characters that specify the size of the adjustment. The operators are as follows:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division

Character adjustment may be used with all basic operands except the blank operand. The operator should appear immediately after the operand; it may not appear beyond column 33 unless the operand itself continues into column 33 or beyond.

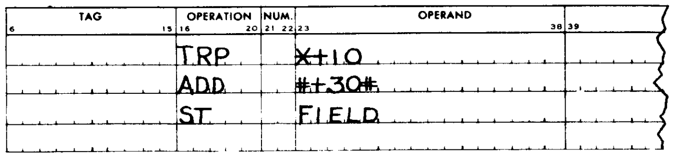
In Figure 49, the character-adjusted operand of the RAD instruction references the field that follows EMPLOYEE.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
EMPLOYEE	RCD	25	A+	
		5	+	
	≀			
	RAD		EMPLOYEE+5	

Figure 49

A character-adjusted location counter may be used to bypass in-line instructions. In Figure 50, *+10 references the low-order (4 or 9) position of the ST instruction.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
	TRP		*+10	
	ADD		#+30#	
	ST		FIELD	

Figure 50

RESTRICTIONS. The numeric portion of a character adjustment cannot exceed five positions nor may its absolute value be greater than 79,999. If it is greater, 80,000 will be subtracted until the absolute value is less than 80,000. If the numeric portion of the adjustment is less than five positions, the position immediately following must be nonnumeric.

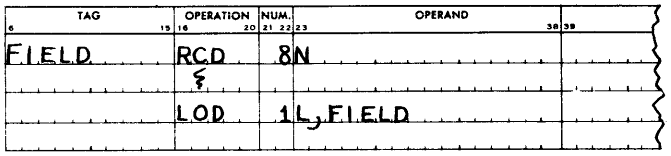
Further restrictions apply to location-counter operands and actual operands. These operands can have only + and - operators. If any other is used, it will be treated as the + operator.

Operand Modifier

An operand modifier is a two-character prefix which may be used with a tag or a literal operand. It enables the user to reference a particular position of a field or an instruction or to reference the size of a field. The operand modifiers are as follows:

Modifier	Modifier Designates
L,	Left-hand position
R,	Right-hand position
H,	High speed position
S,	Size

In Figure 51, the LOD instruction references the left-hand position of FIELD. When the instruction is executed, the contents of that position, rather than the entire contents of FIELD, are placed in ASU 01.



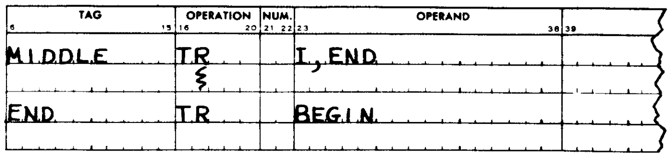
TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
FIELD	RCD	8	N	
	≀			
	LOD	1	L,FIELD	

Figure 51

Note: If the modifier "S", had been used in the preceding example, the LOD instruction would reference the contents of storage location 00008.

Indirect Address

An indirect address is an indirect reference; that is, it is a reference to an operand that references some other operand. It is designated by a two-character prefix to the basic operand. The prefix consists of an I followed by a comma (I,). An indirect address may be used with the following operands: tag, blank, actual, character-adjusted location counter. In Figure 52, BEGIN is the effective transfer point of the first instruction.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
MIDDLE	TR		I,END	
	≀			
END	TR		BEGIN	

Figure 52

When the Processor encounters an instruction containing "I," in the 7080 mode portion of the program, it generates two instructions: The first is an EIA (Enable Indirect Address). If the one-for-one instruction containing the indirect address is tagged, the Processor transfers the tag to the EIA instruction. The second instruction is the same one-for-one instruction without the hand-coded "I," and without the hand-coded tag. If the first instruction in Figure 52 had been written in the 7080 portion of the program, it would have been followed by the generated instructions, as shown in Figure 53.

Tag	Operation	Num	Operand
MIDDLE	TR		I, END
MIDDLE	EIA		END
	TR		END

Figure 53

MULTIPLE ADDITIONS TO A BASIC OPERAND

The following pairs of additions may be used with either a tag or a literal operand:

1. Operand modifier and character adjustment.
2. Indirect address and character adjustment.

The second pair may also be used with a location counter.

In Figure 54, the operand of the LOD instruction references the second position in FIELD, i.e., the position to the right of the high-order position.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
FIELD	RCD	10	A	
	≀			
	LOD		L, FIELD+1	

Figure 54

In Figure 55, COMPUTE is the effective transfer point of the first transfer instruction.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
ONE	RAD		RECORD1	
	TR		I, *+10	
TWO	RAD		RECORD2	
	TR		COMPUTE	

Figure 55

CHAPTER 5. GENERAL PURPOSE MACRO-INSTRUCTIONS

A macro-instruction is a source program statement which represents multiple operations. When the program is assembled, each macro-instruction is replaced by a number of one-for-one instructions; the number varies according to what the macro-instruction is and how it is used. The general purpose macro-instructions in the 7058 Processor library are shown in Figure 56. The purpose of this chapter is to present them as a part of the Autocoder III language; consequently, the chapter is limited to an explanation of their basic coding format and a few examples of individual macro-instructions. The specifications for using each general purpose macro-instruction are provided in the reference manual, "7058 Processor: General Purpose Macro-Instructions," Form C28-6130. Hereafter, the aforementioned will be called the macro-instruction manual. (Input/output macro-instructions are a part of the Input/Output Control System, IOCS, and are described in the IOCS reference manuals for the 705 III and 7080.)

In addition to individual specifications and examples of generated coding, the macro-instruction manual provides detailed explanations of the conventions and restrictions governing the use of all the general purpose macro-instructions. It also explains restrictions that may apply to only one type of macro-instruction. It has been necessary to establish certain conventions and restrictions in creating a macro-instruction library to serve a large number of users with a variety of program needs. However, it is possible for programmers to prepare their own macro-instructions and insert them into the library.

Because of the flexibility of the Processor, programmers need not observe most of the restrictions described in the macro-instruction manual when creating macro-instructions to meet their particular requirements. Specifically, they may designate as acceptable operands any of the basic operands and additions to basic operands described in Chapter 4. Programmers writing their own macro-instructions may also designate an entry in the numeric field as the method of supplying an ASU reference or other special information. The process of creating a macro-instruction requires a thorough knowledge of a special language which is described in the reference manual, "Preparation of Macro-Instructions for Use with Autocoder III," Form C28-6056-1.

The remainder of this chapter is an introduction to the general purpose macro-instructions in the 7058 Processor library; the discussion is based on the conventions and restrictions that apply to these macro-instructions.

ADDRESS MODIFICATION	
Add Address	(ADDA)
Compare Address	(COMPA)
Decrement Address	(DECRA)
Increment Address	(INCRA)
Initialize Address	(INITA)
Move Address	(MOVEA)
Subtract Address	(SUBA)
ASSEMBLY CONTROL	
Enter 80 Mode	(ENT80)
Leave 80 Mode	(LEV80)
Speed or Space	(SPEED)
AUTOMATIC DECIMAL POINT	
Absolute Value	(ABSX)
Add	(ADDX)
Decrement	(DECRX)
Diminish	(DIMX)
Divide	(DIVX)
Divide or Halt	(DVHX)
Increment	(INCRX)
Multiply	(MPYX)
Negative Absolute Value	(NABSX)
Negative Divide	(NDIVX)
Negative Divide or Halt	(NDVHX)
Negative Multiply	(NMPYX)
Subtract	(SUBX)
Sign and Zero Test	(TESTX)
DATA TESTING	
Compare	(COMP)
Test for Numeric Field	(IFNUM)
Test if in Range	(RANGE)
DATA TRANSMISSION	
Blank Memory	(BLANK)
Define ASU	(ASU)
Move	(MOVE)
Restore Decimal	(DEC)
Zero Memory	(ZERO)
PROGRAM BRANCH CONTROL	
Alternating NOP	(ALTNP)
Alternating Transfer	(ALTTR)
First Time NOP	(FTNOP)
First Time NOP on a Bit	(FTNPB)
First Time Transfer	(FTTR)
First Time Transfer on a Bit	(FTTRB)
Set Switches OFF	(SETOF)
Set Switches ON	(SETON)
Test Switch	(IFON)
TABLE	
Add an Item	(ADITM)
Delete an Item	(DLITM)
Replace an Item	(RPITM)
Search a Table	(SERCH)
Table Control	(TBCTL)
MISCELLANEOUS	
Address Modification Diagnostic	(DIAGM)
Dead-End Halt	(STOP)
Link to Subroutine	(LINK)
Non-Address-Modification Diagnostic	(DIAG)
Transfer Indirect	(TRIN)
Type a Message	(TYPE)

Figure 56. 7058 Processor General Purpose Macro-Instructions for Use in Autocoder III Programs

GENERAL PURPOSE MACRO-HEADER FORMAT

The portion of a macro-instruction that is written as a source program statement is called a macro-header. As with other Autocoder III statements, a macro-header is tagged if it is to be referenced. The mnemonic code is placed in the operation field. Entries in the numeric field are rarely permitted; those which are permitted do not relate to an ASU number or a bit as they do in a one-for-one instruction. Most macro-headers have two or more entries in the operand field; some may contain up to twenty entries, and a few may have only one. The entries will be called operands throughout this chapter and in the macro-instruction manual. Each operand is terminated by a lozenge (⌑), the same symbol which was previously explained as part of an RPT statement.

Operands may be placed in the operand and comments fields of the line on which the macro-header starts and may be continued in the operand and comments fields of the next five lines on the coding sheet. However, an operand may not be written on two lines, i.e., it may not be started in the comments field of one line and continued in the operand field of the next line. Similarly, the lozenge which terminates an operand may not be separated from it. If the positions at the end of a line are insufficient for both an operand and its lozenge, the positions must be left blank and the operand started in column 23 of the next line on the coding sheet. Operand continuation lines must be blank in the tag, operation, and numeric fields.

Comments may be started in the comments field of the line on which the operands terminate, but the comments must be separated from the final lozenge by a minimum of two spaces. Comments may also be continued in the comments field of succeeding lines of the coding sheet.

TYPES OF OPERANDS

The operands of a macro-header designate the data and/or the instructions involved in the operations the macro-instruction represents. Most operands are either tags or literals.

Tag Operands

The tags may be those of defined data fields, switches, source program instructions, macro-instructions, and address constants. Other tags that may be used as operands are those of Class A subroutine items. Characteristics of items within Class B subroutines are not available to macro-instructions.

For instance, the function of the IFON macro-instruction is to test a switch and to transfer to one of two specified instructions, depending on the status of the switch. The operands of the IFON macro-header are the tags of the switch to be tested and the tags of the transfer points, i.e., the instructions to which the transfer is made if the switch is ON or OFF. In the generated coding, the tags appear as the operands of the appropriate one-for-one instructions.

In most cases, the tag of an instruction is used as an operand in order to designate the instruction as a transfer point. This is not true of the operands of Address Modification macro-headers. Such operands designate the operands or other instructions rather than the instructions themselves. When an Address Modification macro-header must designate the operand of another macro-header, it may not reference the macro-header by its tag alone. The tag must be written as a special form of operand called the macro suffix tag. This consists of a tag to which a suffix is added. The suffix is of the form #x or #xx where x or xx are numbers that designate one of the operands of the macro-header being referenced. For example, a macro suffix tag designating the first operand of a macro-header tagged MACRO would be written as MACRO#1 or MACRO#01. Similarly, a macro suffix tag designating the third operand would be written as MACRO #3 or MACRO#03. The use of the macro suffix tag is illustrated at the end of this chapter and in the macro-instruction manual. No adjustments are permitted on a macro suffix tag.

Secondary Field Definitions in Tag Operands

A secondary field definition is a description of the characteristics of a data field. It is written as part of a macro-header operand that references the field, i.e., the operand is the tag of the field, and it causes the macro-instructions to treat the field as having the characteristics that the secondary field definition provides. Depending on the reason for which a secondary definition is used, it may supply characteristics identical with those previously defined for the field or it may supply a different set of characteristics. When a data field is referenced indirectly in a macro-header operand, the characteristics of the data field are not available to the Processor. Therefore, a secondary definition must be supplied in the operand that contains the indirect reference.

A secondary field definition may be supplied by the tag of a field, a literal, or either of the RCD forms, #+xx. yy or #bxx. yy. The macro-header operand containing the definition is written as follows: the tag of the data field, a comma, the secondary definition.

Using the Tag of a Field. A macro-header operand containing the tag of a field as a secondary definition would be one such as TAGA, TAGB ⌑. The field specified by TAGA will be treated as having the characteristics of the field specified by TAGB.

If a field with the desired characteristics has been defined, its tag may be used to supply the secondary field definition. Otherwise, two fields must be defined with different tags and overlapped by use of a location assignment (LASN). Reference to the field should be made by using the tag of the definition which is appropriate at the time the reference is made.

Using a Literal. A macro-header operand containing literal secondary definition would be one such as TAG, #+XXXX# ⌑. Regardless of the defined characteristics of the field TAG, it is now defined as a signed fraction consisting of three integer positions and one decimal position. This method can be used to define only numeric fields other than unsigned fractions.

Note that the letter X is the only character that can be used in defining integer and decimal positions.

Using the RCD Form. With the RCD form of secondary definition, the example given above would be written as TAG, #+03.01⌑. This form is fully discussed in Chapter 2 of this manual. This method can be used to define signed or unsigned numeric fields only.

Literal Operands

A literal is actual data enclosed by literal signs (#) and is explained in Chapter 4. In the coding generated from macro-headers containing literal operands, the literals appear as the operands of the appropriate one-for-one instructions, just as tags appear as one-for-one operands. Whenever the macro-instruction manual designates the tag of a field as an operand, a literal may be used instead. An unsigned numeric literal supplying a mixed or pure decimal should not be used as the operand of an Automatic Decimal Point macro-header, because the constant created from the literal will contain a special character (the decimal point). Floating point literals may not be used as the operands of Automatic Decimal Point macro-headers for the reason given in the explanation of FPN (Chapter 2).

TYPES OF LOZENGES

Lozenges indicate to the Processor the termination of each operand and the position which an omitted operand would normally occupy in relation to the other operands. There are two types of lozenges:

Fixed. A fixed lozenge may never be omitted. If the operand it terminates is omitted, the fixed lozenge is placed back-to-back with the lozenge which terminates the preceding operand.

Conditional. A conditional lozenge may be omitted only if the operand it terminates is omitted and no additional operands are written. If other operands follow an omitted operand, its conditional lozenge must be placed back-to-back with the lozenge which terminates the preceding operand.

OMITTED OPERANDS

The specifications in the macro-instruction manual indicate that certain operands may be omitted. The associated lozenge is assumed to be fixed unless the specifications state that it is conditional.

When the omitted operand is a transfer point, the generated coding provides a transfer to the next in-line source program instruction. This may be most readily seen in those macro-instructions which make some sort of test and then transfer according to the results of the test. The IFON macro-header should be written with two transfer points, one to be used if a tested switch is ON, and the other if it is OFF. The second transfer point may be omitted; if it is, the generated instruction for the OFF condition is a transfer to the next in-line source program instruction.

THE IMPORTANCE OF PROPERLY DEFINED DATA FIELDS

A macro-header makes a field reference when it has the tag of a field as an operand. In other words, it references a field which is defined by either an area definition or a switch definition. In order to generate coding which is proper for the field, the Processor must know the characteristics of the data which will occupy the field. Obviously, it is not possible for the Processor to examine the actual data at assembly time. Consequently, the Processor obtains the characteristics from the definition and generates coding which is proper for the field according to its definition. If the data does not conform to these characteristics, it may be improperly processed. However, the generated coding itself is not improper.

The importance of field definitions may be seen in a macro-instruction which is used to compare the contents of two fields. The fields may be alphameric or numeric. The one-for-one instructions which should be used to compare alphameric data differ from those which should be used to compare numeric data. By using the macro-instruction, the programmer is relieved of having to select the proper instructions, but the Processor cannot assume this burden unless the characteristics of the field are available to it. Similarly, if literals are used instead of the tags of fields, the literals must be written in accordance with the standards previously specified. For instance, an unsigned decimal written as a literal will not be treated as numeric data but as alphameric data.

EXAMPLES OF MACRO-INSTRUCTIONS AND THEIR USE

The balance of this chapter contains examples of a few general purpose macro-instructions in the Processor library. The function and coding format of each macro-instruction is followed by an example which illustrates how it might be used and what instructions would be generated for that usage. In Figures 57-60, the macro-headers are overlaid with a band of grey to distinguish them from generated instructions. The explanations should not be considered as the specifications for the macro-instructions. In some examples, certain options which are available have been omitted entirely. Complete specifications are provided only by the macro-instruction manual.

1. Blank Memory: BLANK

The function of BLANK is to place blanks in a field. The basic format of the BLANK macro-header is as follows:

Tag	Operation	Num	Operand
T_1	BLANK		X_1¤X_2¤X_3¤.....X_{20}¤

T_1	is the tag of the macro-header or is omitted.
$X_1 \ldots X_{20}$	are the tags of the field in which blanks are to be placed. The lozenges are conditional.

In Figure 57, TAG1 indicates that the contents of fields ONE and TWO are to be replaced by blanks.

Tag	Operation	Num	Operand
	NAME	0	
ONE	RCD	5	+
TWO	RPT	8	XXXX. ZZ
	≀		
TAG1	BLANK		ONE¤TWO¤
TAG1	RCV		ONE
	BLM		@00001
	RCVS		TWO
	BLMS		@00008

Figure 57

2. Test Switch: IFON

The function of IFON is to test a switch and to transfer according to the results of the test. The basic format of the IFON macro-header is as follows:

Tag	Operation	Num	Operand
T_1	IFON		X_1¤X_2¤X_3¤

T_1	is the tag of the macro-header or is omitted.
X_1	is the tag of the switch to be tested.
X_2	is the tag of the ON transfer point, i.e., the instruction to which a transfer should be made if the switch is ON.
X_3	is the tag of the OFF transfer point. The operand may be omitted, in which case a transfer will be made to the next in-line instruction. The lozenge is conditional.

In Figure 58, ONPOINT and OFFPOINT must be assumed to be the tags of instructions. If OFFPOINT and its associated lozenge had been omitted, the final instruction would not have been generated.

3. Add: ADDX

The function of ADDX is to add the data in two numeric fields and place the result in a numeric field or an RPT field. The numeric fields may be signed or unsigned. The basic format of the ADDX macro-header is as follows:

Tag	Operation	Num	Operand
T_1	ADDX		X_1¤X_2¤X_3¤

T_1	is the tag of the macro-header or is omitted.
X_1	is the tag of one numeric source field, i.e., the field which is the source of one set of data to be added.
X_2	is the tag of the other numeric source field.
X_3	is the tag of the numeric or RPT result field, i.e., the field in which the result is to be placed.

4. Increment Address: INCRA

INCRA is an Address Modification macro-instruction; the function of this type of macro-instruction is to modify other instructions, either macro-instructions or one-for-one instructions. The function of INCRA

Tag	Operation	Num	Operand
	CHRCD		
NEWYORK			A
CHICAGO			B
	≀		
TAG2	**IFON**		**NEWYORK¤ONPOINT¤OFFPOINT¤**
TAG2	LOD	1	#A#
	CMP	1	NEWYORK
	TRE		ONPOINT
	TR		OFFPOINT

Figure 58.

Tag	Operation	Num	Operand
SOURCE	RCD	5	#+02.03
RESULT		6	#+03.03
	≀		
TAG3	**ADDX**		**SOURCE¤#+75.000#¤RESULT¤**
TAG3	RAD		SOURCE
	SET		@00006
	ADD		#+75.000#
	ST		RESULT

Figure 59

is to increment a field reference made by another instruction, thus modifying the instruction so that it makes a different field reference. An instruction makes a field reference by having the tag of a field as an operand. INCRA designates the instruction which makes the field reference and the amount by which the reference is to be increased. The basic format of the INCRA macro-header is as follows:

Tag	Operation	Num	Operand
T_1	INCRA		X_1¤X_2¤

T_1	is the tag of the macro-header or is omitted.
X_1	is the tag of an instruction which makes the field reference to be incremented.
X_2	is the increment.

In Figure 60, the first operand of INCRA is a macro suffix tag, designating the second operand of MACRO. Initially, MACRO references FIELD. However, INCRA modifies MACRO so that it subsequently references whatever is located 500 positions above FIELD. For instance, assume that FIELD occupies locations 001000-001002. When MACRO is executed initially, it will cause these locations to be blanked. Once modified by INCRA, it will cause locations 001500-001502 to be blanked. (Note: MAC0017#02 is a tag generated by the Processor.)

Tag	Operation	Num	Operand
OTHER	RCD	8	A
FIELD	≀	3	A
MACRO	**BLANK**		**OTHER¤FIELD¤**
MACRO	RCVS		OTHER
	BLMS		@00008
MAC0017#02	RCVS		FIELD
	BLMS		@00003
	≀		
TAG3	**INCRA**		**MACRO#2¤#+500#¤**
TAG3	RAD	15	#+500#
	AAM	15	MAC0017#02

Figure 60

An address constant is a numeric constant consisting of a storage location. An address constant statement designates the storage location by specifying one of four operands: tag, literal, actual, location counter. At assembly time, the location assigned to the tag, literal, or location counter or the location designated by the actual operand is used to create the constant. In effect, the function of an address constant statement is to define a data field that will contain a constant and to designate the constant to be placed in the field. The actual constant is generated by the Processor and placed in the field created for it. Thus, an address constant enables the user to reference a constant which is not created until the program is assembled.

Address constants are used to initialize instructions, a procedure which alters the reference made by an instruction or supplies a reference to an instruction which lacks one. For example, suppose that an instruction must reference two record areas alternately, areas tagged FIRST and SECOND. This means that the operand of the instruction must contain FIRST at certain points in the program and SECOND at other points. To initialize the instruction, i.e., to modify the reference, address constants must be created from each of these tags so that one or the other of them can be placed in the instruction as required. In the assembled program, the address portion of the instruction will alternate between the actual locations assigned to FIRST and SECOND. Note the difference between an instruction which references FIRST and an instruction which references an address constant created from FIRST. In the former case, the instruction references the contents of a record area; in the latter case, the instruction references a constant consisting of the storage location of the record area.

The basic operand of an address constant statement may be a tag, literal, actual, or location counter. Operand modifiers may be used with a tag or literal to request a generated constant:

Modifier	Address Constant Generated From
Right-hand	storage location of the low-order position of a field, instruction, or literal.
Left-hand	storage location of the high-order position of a field, instruction, or literal.
High speed	a left-hand address plus four.
Size	the number of positions occupied by a field or literal.

If no operand modifier is used, a right-hand address will be generated as the constant. As the preceding list indicates, a right-hand operand modifier may be written, but it is not necessary.

Character adjustments to the basic operand cause numerical adjustment of the address constant. Addition, subtraction, multiplication, or division by a specified amount may be requested. For example, a character adjustment of plus five would cause the constant to be five greater than the storage location referenced.

An address constant may be both operand-modified and character-adjusted. (Such an operand may have to continue into the comments field.) The operand modifier is a prefix to the basic operand; it consists of the appropriate modifier symbol followed by a comma. The character adjustment is a suffix to the basic operand; it consists of the arithmetic operator followed by a number designating the amount of adjustment. The amount may not exceed 80000. The symbols are as follows:

Operand Modifier		Character Adjustment	
R,	Right-hand	+	Add
L,	Left-hand	-	Subtract
H,	High speed	*	Multiply
S,	Size	/	Divide

Assume that FIELD, a data field, is assigned to locations 001300-001309. An address constant statement having L, FIELD as its operand will cause 001300 to be created as the address constant. The operand R, FIELD+6 will cause 001315 to be created as an address constant. The same constant would be created from FIELD+6. Since the field occupies 10 positions, the operand S, FIELD will cause a constant of 10 to be created; the operand S, FIELD*5 will create a constant of 50.

Comments about an address constant may be started in the comments field of the address constant statement.

ADCON Address Constant

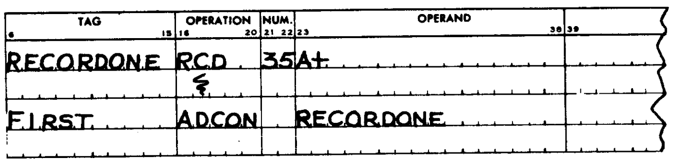
The function of an ADCON statement is to create an instruction which consists of a four-character, unsigned address constant preceded by the actual code for No Operation. The instruction is positioned in a 4 or 9 location. The ADCON statement is written as follows:

Tag	Operation	Num	Operand
T_1	ADCON	nn	X_1

T_1	is the tag of the address constant.
nn	is ASU zoning or is blank.
X_1	is a tag, literal, actual, or location counter.

The ADCON statement creates an instruction of the form Axxxx. A is the actual code for No Operation; xxxx is the address constant. The instruction Axxxx will be positioned so that the low-order character occupies a 4 or 9 location. Any ASU zoning will be properly generated as part of the constant.

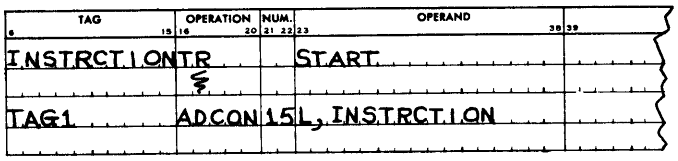
The ADCON statement in Figure 61 will cause an address constant to consist of the storage location of the right-hand position of the RECORDONE data field. Instructions referencing the constant do so by referencing its tag, FIRST.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
RECORDONE	RCD	35	A+	
FIRST	ADCON		RECORDONE	

Figure 61

Figure 62 specifies that the left-hand address constant consisting of the location of INSTRCTION is to be zoned for ASU 15.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
INSTRCTION	TR		START	
TAG1	ADCON	15	L, INSTRCTION	

Figure 62

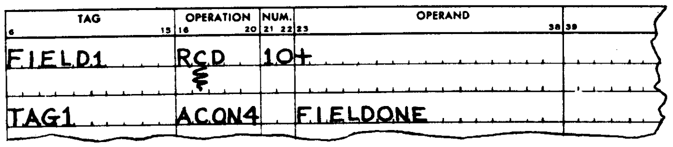
ACON4 Address Constant

The function of an ACON4 statement is to create a four-character, unsigned address constant. The constant is placed in the next four available storage locations without regard to the positioning of its low-order character. ASU zoning, if specified, is properly generated as part of the constant. The format of the ACON4 statement is as follows:

Tag	Operation	Num	Operand
T_1	ACON4	nn	X_1

T_1	is the tag of the address constant.
nn	is an ASU number or is blank.
X_1	is a tag, literal, actual, or location counter.

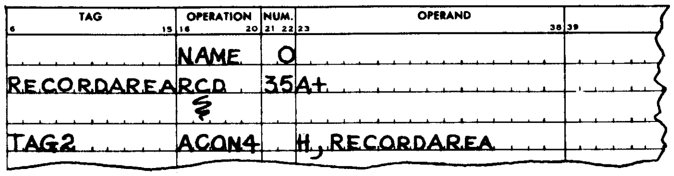
In Figure 63, the ACON4 statement is a request for an address constant consisting of the storage location assigned to FIELD1. Since no operand modifier is specified, the right-hand address will be generated. The constant may be referenced by its tag, TAG1.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
FIELD1	RCD	10	+	
TAG1	ACON4		FIELDONE	

Figure 63

Figure 64 shows that the constant will consist of the location assigned to the RECORDAREA field. Since the operand modifier "H," is used, the high speed address will be generated.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
	NAME	0		
RECORDAREA	RCD	35	A+	
TAG2	ACON4		H, RECORDAREA	

Figure 64

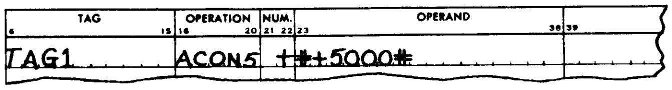
ACON5 Address Constant

The function of an ACON5 statement is to create a five-character address constant, either signed or unsigned. The constant is placed in the next five available storage locations without regard to the positioning of its low-order character. The sign, if specified, is placed over the low-order character. The format of the ACON5 statement is as follows:

Tag	Operation	Num	Operand
T_1	ACON5	s	X_1

T_1	is the tag of the address constant.
s	is + for a positive constant.
	is - for a negative constant.
	is blank for an unsigned constant.
X_1	is a tag, literal, actual, or location counter.

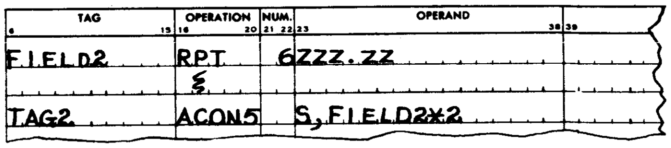
The ACON5 statement in Figure 65 specifies that the location of the literal is to be made an address constant. Notice that the address constant will be signed. The sign of the address constant is not related to the sign of the literal.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
TAG1	ACON5	+	#+5000#	

Figure 65

Figure 66 shows a request for an unsigned constant twice the size of FIELD2. The constant 00012 will be generated.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
FIELD2	RPT	6	ZZZ.ZZ	
TAG2	ACON5		S, FIELD2×2	

Figure 66

Restrictions on an ACON5 Statement. ASU zoning may not be specified in an ACON5 statement.

A signed constant should not be specified if there is a possibility that the address from which the constant is created will exceed 79999. In the event that a signed constant is requested for such an address, the constant is developed without a sign. A message to the effect that the constant exceeds the address limit is provided at assembly time.

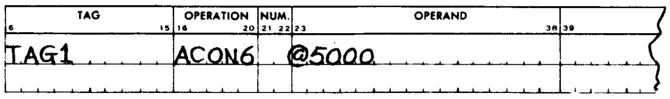
ACON6 Address Constant

The function of an ACON6 statement is to create a six-character, unsigned address constant. The constant is placed in the next six available storage locations without regard to the positioning of its low-order character. The format of the ACON6 statement is as follows:

Tag	Operation	Num	Operand
T_1	ACON6		X_1

T_1	is the tag of the address constant.
X_1	is a tag, literal, actual, or location counter.

In Figure 67, the ACON6 statement requests that 5000 be generated as a constant.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
TAG1	ACON6		@5000	

Figure 67

Restrictions on an ACON6 Statement. ASU zoning may not be specified in an ACON6 statement.

ADDRESS CONSTANT LITERAL

An address constant literal is an operand with a double function; it is a request for an address constant and an operand that references the constant. The generated address constant is placed in the literal table. For example, when an instruction references a tag as part of an address constant literal, a constant consisting of the location assigned to the tag will be created and placed in the literal table. When the program is assembled, the operand (address constant literal) of the instruction will be replaced by the location assigned to the generated constant. If a program requires many address constants, they should be created with address constant statements. The address constant literal operand is useful in a program that requires an occasional address constant.

Writing an Address Constant Literal Operand

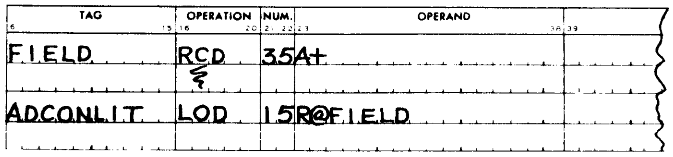
The operand may contain a tag or a literal; operand modifiers must be used with either one to specify the type of address being requested. If ASU zoning is to be generated as part of the constant, the ASU number is placed directly after the operand modifier and is followed by a comma. The basic format of the entire operand is either of the following:

1. Operand modifier plus a tag or literal.
2. Operand modifier plus ASU zoning plus a tag or literal.

The symbols for the operand modifiers and ASU zoning are shown in the following list (nn represents an ASU number):

Address Type	Operand Modifier	Modifier and ASU Zoning
Right-hand	R@	R@nn,
Left-hand	L@	L@nn,
High speed	H@	H@nn,
Size	S@	S@nn,

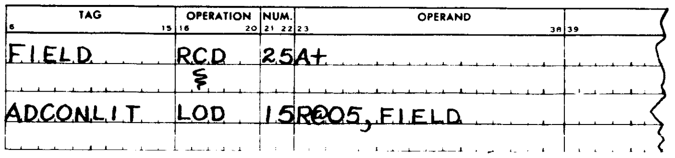
In Figure 68, an address constant is requested for the right-hand address of FIELD. The instruction specifies that the address constant is to be loaded into ASU 15. When the instruction is executed, the right-hand address of FIELD rather than the contents of FIELD will be placed in ASU 15.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
FIELD	RCD	35	A+	
ADCONLIT	LOD	15	R@FIELD	

Figure 68

Figure 69 specifies that the address constant consisting of the right-hand address of FIELD be zoned for ASU 5. As in the preceding example, when the instruction is executed, the address constant will be placed in ASU 15.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
FIELD	RCD	25	A+	
ADCONLIT	LOD	15	R@05,FIELD	

Figure 69

Arithmetic instructions, such as ADD, SUB, etc., cause a five-position signed constant to be created; the constant is signed plus. In the 7080 mode, the maximum constant is 79999. Instructions requiring a 4 or 9 address, such as LDA, AAM, etc., cause a four-position unsigned constant to be created and properly positioned in a 4 or 9 location. In 7080 mode, the maximum constant is 159999. All other instructions cause a four-position unsigned constant to be created for a 705 II program and a five-position unsigned constant to be created for 705 III and 7080 programs. (See Chapter 7 for further definition of "7080 mode.")

Restrictions on an Address Constant Literal Operand. Character adjustment may not be used for the purpose of modifying the constant itself. If character adjustment is written in an address constant literal operand, it will be applied to the location of the constant.

If an address constant literal operand is used in a macro-header, it may not designate ASU zoning.

CHAPTER 7. INSTRUCTIONS TO THE PROCESSOR

Instructions to the Processor concern the assembly process; they are executed by the Processor at assembly time. Consequently, they do not appear in the object programs, although they are written in the source program wherever they are required. Through these statements, the programmer is able to communicate with the Processor. The instructions to the Processor are listed below according to the aspect of the assembly process that they concern:

1. Standard Assembly Procedures
   Location Assignment - LASN
   Special Assignment - SASN
   Assignment of Library Subroutines - SUBOR
   Assignment of Literals - LITOR
   Transfer Card - TCD
2. Object Program Content
   Include Subroutine - INCL
   Translation - TRANS
   Source Program Language - MODE
3. Object Program Listing
   Skip to New Page - EJECT
   Title for Routine or Comment - TITLE

INSTRUCTIONS TO THE PROCESSOR THAT CONCERN STANDARD ASSEMBLY PROCEDURES

Certain instructions to the Processor may be used to alter standard assembly procedures. To understand how these instructions may be used, it is first necessary to know what the procedures are:

1. Location assignments. The Processor assigns storage locations in ascending order to the object program. In making the assignments, it uses a location counter that is set initially to location 00160. The parts of the object program are assigned in the following sequence: the machine language equivalent of the source program, the library subroutines, the literal table. If no subroutines have been requested by either the source program or the Processor itself, the literal table is placed after the source program.

2. Standard "00" transfer control card. The Processor produces this as the terminal card of the object program deck. (Chapter 8 contains additional information on the object deck.) The standard "00" card contains instructions to set various ASUs. The final instruction on the card is a transfer to the first instruction in the object program. At the time the object program is to be executed (object time), it is placed in storage by a loading program. When the loading program encounters the standard "00" transfer card, it executes the instructions the card contains, thereby transferring control to the object program itself.

The instructions to the Processor explained in this section enable the programmer to direct the Processor to do one or more of the following:

1. To use more than one location counter in making assignments.
2. To assign specific locations designated by the programmer.
3. To alter the order of the object program parts.
4. To provide additional "00" cards and to place them within the object program.

It is often necessary to modify the standard assembly procedure. For example, it must be done when using IOCS (Input/Output Control System), because the IOCS routines occupy a large storage area starting in location 00160. The object program, therefore, must be positioned beyond the IOCS area. The positioning is accomplished by starting the source program with an instruction to the Processor to set the location counter to a location above the IOCS area.

The ability to specify storage assignments allows the programmer to conserve storage space by overlapping assignments, i.e., by assigning the same area of storage to more than one routine or block of data. A housekeeping routine is frequently overlapped with another routine, since the housekeeping routine is only executed once.

With the use of instructions to the Processor, the programmer is able to cause the housekeeping routine to be placed in storage and executed before the other routine is placed in the same area. Another example of overlapping is the assignment of two or more NAME definitions to the same area. This is often desirable when the program is to process sets of records that possess different characteristics but require the same amount of storage space. As long as all the records need not be in storage simultaneously, the same location assignment may be specified for the various NAMEs.

Location Assignment - LASN

The function of a LASN statement is to set a location counter to a specified location; 10 counters are available. A LASN statement may set the designated counter to one of the following:

1. An actual location specified by the programmer.
2. An actual location, unknown to the programmer, that has already been assigned by the Processor to a field or an instruction.
3. One location beyond the highest location assigned from the counter at any point in the assignment process.
4. Location 00160, the initial location assignment.
5. One location beyond the highest location

assigned from a point in the assignment process specified by the programmer.

Each time the Processor encounters a LASN, it sets the designated counter and makes subsequent assignments from that counter. This continues until another LASN is encountered or until the assignment process is completed. Multiple counters are useful when specifying location assignments in a program of many sections, because one counter can be allocated to each section.

The LASN is written as follows:

TAG FIELD. This field must be left blank.

OPERATION FIELD. The mnemonic code LASN is placed here.

NUMERIC FIELD. The counter to be set is designated in column 22 of this field. The column is left blank when designating the Blank counter; each of the other counters is designated by one of the digits 1-9. The Blank counter may be considered the primary counter, since it is used by the Processor in the absence of any LASN statements. Additional information on the Blank counter is supplied in the section "Location Assignments from the Blank Counter."

OPERAND FIELD. To set the counter designated in the numeric field, the entry in this field may be one of the following:

1. An actual operand. The counter is set to the location specified by the operand.
2. The tag of a statement appearing anywhere in the program before the LASN. The counter is set to the location previously assigned to the instruction or field identified by the tag. The tag may be character-adjusted.
3. A blank operand. The counter is set to one location beyond the highest location previously assigned from it.

To reset the counter to location 00160, from which the standard assignment process starts, leave columns 23-73 blank and place the character R in column 74. When used in column 74 of a LASN statement, this character may be considered the Reset character. (For additional information on the Reset character see the section entitled "Special Uses of the Reset Character.")

COMMENTS FIELD. When a tag or an actual operand is used, comments about the statement may be placed in this field. When writing comments, column 74 should be examined to make sure it does not contain R. If it does, subsequent use of the counter is affected as described in the section entitled "Special Uses of the Reset Character."

In Figure 70, storage assignments are shown to the right of the hand-coded Autocoder III statements. Notice that the assignments made after the LASN statements are consistent with the requirement of a 4 or 9 location for instructions and with NAME statements that specify a location through an entry in the numeric field.

Tag	Operation	Num	Operand 74	Assignments
	LASN		@2000	002000
	≀			≀
				003007
START	NAME	0	END	003010
ONE	RCD	4	+	003013
TWO		7	#+04.03	003020
END	CON	4	‡	003024
	≀			≀
	LASN	1	@50000	050000
TAG	ADCON		START	050004
	≀			≀
				069994
	LASN	1	TWO	003014
EXTRA	RCD	7	#+05.02	003020
	≀			≀
				004000
	LASN	1		069995
	≀			≀
	LASN	1	R	000160
	≀			≀
	LASN			003025

Figure 70

SPECIAL USES OF THE RESET CHARACTER. Placing the Reset character (R) in column 74 of a LASN statement containing an actual or a tag operand does not modify the setting designated by the operand. However, it may affect a subsequent setting designated by a blank operand for the same counter, because the Processor will ignore any assignments it made before encountering the statement containing the Reset character.

This may best be seen with an illustration. Suppose that the highest assignment made from counter 1 is location 59999. The Processor then encounters a LASN for counter 1 to location 2000. After setting the counter, the Processor assigns a block of 500 positions, bringing counter 1 to 2499. Now a LASN with a blank operand is encountered for counter 1. The counter is set to location 60000, one location beyond the highest assignment made from the counter up to this point in the assignment process. To return to the beginning of this example, when location counter 1 contains 59999, suppose that the Processor encounters a LASN for counter 1 to location 2000, but the statement also contains R in column 74. As before, the counter is set to 2000, a block of 500 positions is assigned, and the counter is again at 2499. Now a LASN with a blank operand is encountered for counter 1. Because the Reset character destroyed the previous high location (59999), the counter is set to 2500. This is one location beyond the highest assignment made by the Processor after it encountered the Reset character.

LOCATION ASSIGNMENTS FROM THE BLANK COUNTER. The Processor uses the Blank counter unless directed by a LASN statement to do otherwise. When the assignment of the machine language version of the source program is completed, the library subroutines must be assigned. The Processor uses the Blank counter to make the assignments. It first sets the Blank counter to one location beyond the highest location previously assigned, no matter what counter was used to make assignment. After it completes the subroutine assignments, it repeats the same process in assigning the literal table, i.e., it sets the Blank counter to one location beyond the highest location previously assigned. If no LASNs have been encountered within a subroutine, the Blank counter itself contains the highest location previously assigned at the time the literal table is to be positioned. The programmer should keep this use of the Blank counter in mind when placing LASN statements in subroutines. (The entire assignment of library subroutines and the literal table may be altered by LITOR and SUBOR. Both are instructions to the Processor and are explained on subsequent pages.)

RESTRICTIONS ON A LASN STATEMENT. The operand may not be the tag of a field or an instruction if its high-order position occupies location 000000. A LASN may, however, contain an actual operand of zero (@000000).

A LASN statement may not be referenced by another Autocoder III statement.

Special Assignment - SASN

The function of a SASN statement is to set the Blank counter as follows:

1. To an actual assignment specified by the programmer.

2. To an actual location, unknown to the programmer, that has already been assigned by the Processor to a field or an instruction.

SASN is a limited form of LASN. Like LASN, it may be used in library subroutines as well as in programs. However, it differs substantially from LASN in the following respect. The highest location assignment resulting from a SASN is ignored when the Processor sets the Blank counter to one location beyond the highest location previously assigned from the counter. (Such a setting is specified by a LASN with a blank operand.) In effect, location assignments resulting from a SASN are no longer significant once the SASN is terminated. Termination of a SASN results when a LASN is encountered, no matter what counter the LASN designates or what type of operand it contains.

Because the SASN is a limited form of LASN, it does not require a detailed explanation. It is written as follows:

Tag	Operation	Num	Operand
	SASN		X_1

X_1 is an actual operand.
is the tag of a statement appearing anywhere in the program before the SASN. The tag may be character-adjusted.

Notice that the tag and numeric fields must be left blank. Comments may be placed in the comments field, and the character R in column 74 has no effect.

Figure 71 illustrates the fact that SASN assignments are ignored during subsequent LASN assignments.

Tag	Operation	Num	Operand 74	Location Assigned
	LASN		@2000	002000
				002499
	SASN		@3000	003000
				004000
	LASN			002500

Figure 71

RESTRICTIONS ON A SASN STATEMENT. The operand may not be the tag of a field or an instruction if location 000000 is occupied by the high-order position of the field or instruction. A SASN may, however, contain an actual operand of zero (@000000).

A SASN statement may not be referenced by another Autocoder III statement.

Assignment of Library Subroutines - SUBOR

The function of a SUBOR statement is to specify the starting location for the assignment of library subroutines. The SUBOR assignment supersedes the standard subroutine placement, i.e., after the last instruction in the program. SUBOR enables the user to position the block of subroutines anywhere in storage, and the statement itself may be written at any point in the program. For a program written in two modes, it may be necessary to place the subroutines below the storage limit of the secondary mode. For example, the primary mode of a program may be 7080, and the secondary mode 705 III. If the 705 III portion of the program must have access to the subroutines, and it is anticipated that the final instruction will occupy a location close to or beyond the storage size of the 705 III, a SUBOR must be used to position the subroutines in the lower portion of storage. This would alter the order of the object program parts so that the block of subroutines would be placed within the machine language equivalent of the source program. It may even be desirable to place the subroutines at the beginning of the object program.

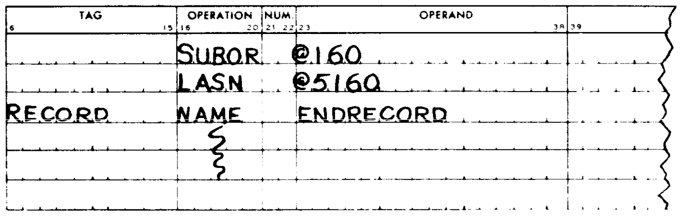
The SUBOR statement is written as follows:

Tag	Operation	Num	Operand
	SUBOR		X_1

X_1 is an actual operand.
is the tag of an Autocoder III statement. The tag may be character-adjusted. The tagged statement must precede the SUBOR statement.

Comments may be placed in the comments field.

Figure 72 indicates that the programmer assumes the subroutines cannot possibly occupy more than 5,000 positions.



TAG	OPERATION	NUM.	OPERAND	
	SUBOR		@160	
	LASN		@5160	
RECORD	NAME		ENDRECORD	

Figure 72

RESTRICTIONS ON THE SUBOR STATEMENT. The operand may not be the tag of a field or an instruction if location 000000 is occupied by the high-order position of the field or the instruction. A SUBOR may, however, contain an actual operand of zero (@000000).

A SUBOR statement may not be referenced by another Autocoder III statement.

Assignment of Literals - LITOR

The function of a LITOR statement is to specify the starting location for the assignment of the literal table. The LITOR assignment supersedes the standard literal table placement, i.e., after the subroutine block or after the last instruction of the program if no subroutines are used. LITOR enables the user to position the literal table anywhere in storage, and the statement itself may be written at any point in the program. (The previous discussion on the use of SUBOR applies as well to LITOR.)

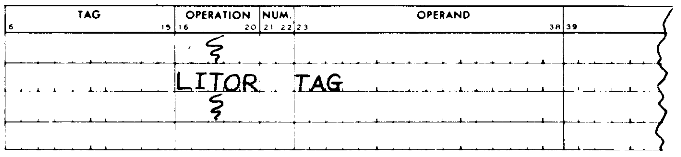
The LITOR statement is written as follows:

Tag	Operation	Num	Operand
	LITOR		X_1

X_1 is an actual operand.
is the tag of an Autocoder III statement. The tag may be character-adjusted. The tagged statement must precede the SUBOR statement.

Comments may be placed in the comments field.

In Figure 73, the Processor is instructed to start the literal table assignment at the same location already assigned to TAG. It must be assumed either that the contents of TAG are no longer needed when the literal table is actually placed in storage or that the contents of TAG are placed in storage after the literal table is no longer needed.



TAG	OPERATION	NUM.	OPERAND	
	LITOR		TAG	

Figure 73

RESTRICTIONS ON THE LITOR STATEMENT. The operand may not be the tag of a field or an instruction if location 000000 is occupied by the high-order position of the field or instruction. A LITOR may, however, contain an actual operand of zero (@000000).

A LITOR statement may not be referenced by another Autocoder III statement.

Transfer Card - TCD

The function of a TCD statement is to create a "00" transfer control card in addition to the standard "00" card that terminates the object program deck. The additional "00" card will be internal to the object program, occupying the same relative position in it that the TCD statement occupies in the source program.

The TCD statement must be followed by Autocoder III statements that specify the contents of the card, i.e., the instructions or the instructions and data the card will contain. The last of these Autocoder III statements must be a transfer back to the loading program or to another object program instruction that is already in storage. A LASN (or SASN) statement must be used after the final statement supplying the contents of the "00" card. A program may contain more than one TCD statement. Multiple TCDs may be written consecutively or interspersed throughout the program.

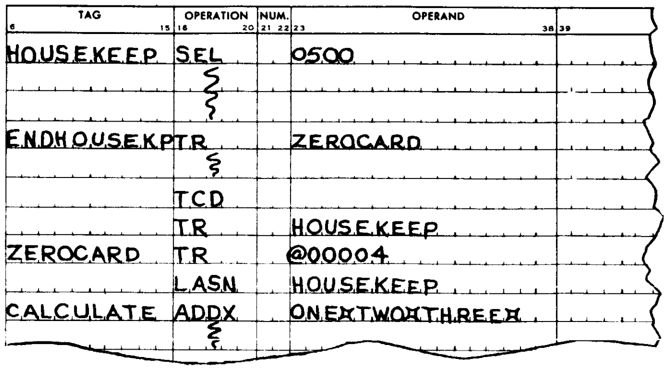
The format of the TCD statement is as follows:

Tag	Operation	Num	Operand
T_1	TCD		

T_1 is the tag of the statement or is omitted.
Comments about the "00" card may be written in the comments field.

THE EFFECT OF THE "00" CARD ON THE LOADING PROCESS. As previously explained, a "00" card causes the loading program to interrupt the loading procedure and to execute the instructions on the card as soon as it is loaded into storage. The area of storage assigned to the contents of any "00" card is the input area used by the loading program, i. e., locations 000080-000159. On the standard "00" card that the Processor automatically produces, the final instruction is a transfer to the first instruction in the object program. A return is not made to the loading program, because the standard "00" card is the final card of the object program deck. In contrast, the "00" card created by a TCD statement is followed by additional object program cards. Consequently, this "00" card must contain as its final instruction a transfer back to the loading program or to some other routine, already in storage, that will ultimately return control to the loading program.

A "00" card is often used to execute an overlapped routine, as shown in Figure 74. As soon as the "00" card is placed in the loading input area, a transfer is made to the HOUSEKEEP routine, which is already in storage. The last instruction of the routine is a transfer back to the "00" card, which transfers in turn to the loading program. When loading is resumed, the HOUSEKEEP routine will be overlapped by the CALCULATE routine.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
HOUSEKEEP	SEL		0500	
	≀			
	≀			
ENDHOUSEKP	TR		ZEROCARD	
	≀			
	TCD			
	TR		HOUSEKEEP	
ZEROCARD	TR		@00004	
	LASN		HOUSEKEEP	
CALCULATE	ADDX		ONE⌑TWO⌑THREE⌑	
	≀			

Figure 74

RESTRICTIONS ON THE TCD STATEMENT. The machine language version of the Autocoder III statement specifying the "00" card content may not exceed 60 positions. (A machine language instruction occupies five positions.)

If an object program contains "00" cards created from TCD statements, the input area of the loading program used with the object program must start at location 000080.

INSTRUCTIONS TO THE PROCESSOR THAT CONCERN OBJECT PROGRAM CONTENT

Include Subroutine - INCL

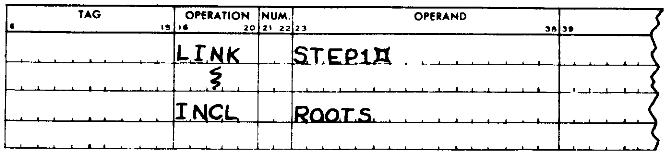
The function of an INCL statement is to designate a library subroutine that the Processor is to insert in the object program. The source program must also contain an instruction or a routine that supplies the linkage to the subroutine designated by an INCL statement. The format of the INCL statement is as follows:

Tag	Operation	Num	Operand
	INCL		X_1

X_1 is the five-character mnemonic identification code of the subroutine to be included.

Comments about the subroutine may be written in the comments field.

The function of the macro-instruction LINK, used in Figure 75, is to provide linkage to a subroutine. The subroutine is ROOTS; the tag of its entry point is STEP1.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
	LINK		STEP1⌑	
	≀			
	INCL		ROOTS	

Figure 75

TYPES OF LIBRARY SUBROUTINES. Programmers may write subroutines in Autocoder III language and add them to the standard Processor library. Such a subroutine will be included in a program assembly only if it is designated by an INCL statement. The standard library also contains subroutines that are required by macro-instructions, but the Processor automatically supplies these subroutines, and the details of their inclusion are not relevant to the use of INCL.

Two types of subroutines may be written in Autocoder III language:

1. Class A. These may contain any Autocoder III statement.

2. Class B. These may contain any Autocoder III statement except the following: NAME used to enclose fields, macro-instructions other than ENT80 and LEV80, an INCL that designates a Class A subroutine.

RESTRICTIONS ON THE INCL STATEMENT. An INCL statement may not be referenced by another Autocoder III statement.

Translation - TRANS

The function of a TRANS statement is to translate the operand of a one-for-one instruction into an actual location.

The TRANS statement designates an actual location and equates it to the reference made by the operand of a one-for-one instruction. More than one instruction may reference the same TRANS statement. In this case, each reference will be equated to the location designated by the TRANS.

The TRANS statement is written as follows:

TAG FIELD. The entry in this field must be the tag that appears as the operand of the one-for-one instruction making the reference.

OPERATION FIELD. The mnemonic code TRANS is placed here.

NUMERIC FIELD. This field must be left blank.

OPERAND FIELD. The entry in this field may be one of the following operands:

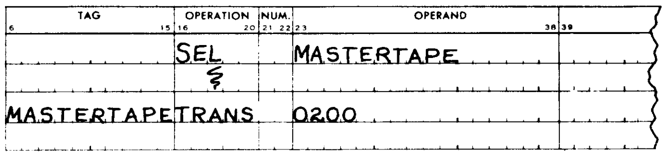
1. An actual operand. This location will appear in an object program instruction wherever the tag of the TRANS appears as a source program operand.

2. A location counter without character adjustment (*). The location of the instruction following the TRANS will appear in an object program instruction wherever the tag of the TRANS appears as a source program operand.

3. A location counter with a character adjustment of minus five (*-5). The location of the instruction immediately preceding the TRANS will appear in an object program instruction wherever the tag of the TRANS appears as a source program operand.

COMMENTS FIELD. Comments may be placed here.

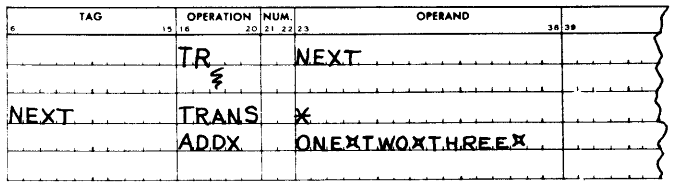
In Figure 76, the TRANS statement equates MASTERTAPE to an actual tape address. In the object program listing, the machine language version of the SEL instruction will contain the address 0200.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
	SEL		MASTERTAPE	
	≷			
MASTERTAPE	TRANS		0200	

Figure 76

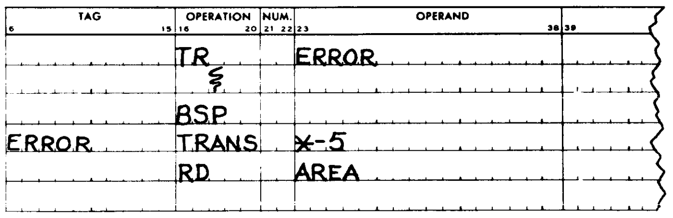
Assume that location 05009 is assigned to the first instruction generated from the ADDX macro-instruction in Figure 77. The operand of the TR instruction is also translated to 05009, because the TRANS statement does not exist in the object program. The * operand of a TRANS statement is, in effect, *+5.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
	TR		NEXT	
	≷			
NEXT	TRANS		*	
	ADDX		ONE*TWO*THREE*	

Figure 77

If the RD instruction in Figure 78 is assigned to location 03059, the operand of the TR instruction will be translated to 03054. This results from the fact that the TRANS statement does not appear in the object program. Consequently the BSP instruction is the instruction actually preceding the RD instruction and is assigned to location 03054.



TAG 6–15	OPERATION 16–20	NUM. 21–22	OPERAND 23–38	39
	TR		ERROR	
	≷			
	BSP			
ERROR	TRANS		*-5	
	RD		AREA	

Figure 78

RESTRICTIONS ON THE TRANS STATEMENT. The only character adjustment that may be used with a location counter operand is minus five (*-5). If other adjustments appear, they will be ignored.

The TRANS statement must be tagged, because it is always referenced.

Source Program Language - MODE

An Autocoder III program may contain statements written in the following languages:

1. FORTRAN
2. Report/File Writing
3. Decision
4. Arithmetic
5. Table-Creating

The term "higher languages of the 7058 Processor" includes all of the above-listed languages except FORTRAN. MODE statements are instructions to the Processor that indicate a change in the language of the source program, and they must be used in Autocoder III programs that contain Report/File Writing statements and/or FORTRAN statements. MODE statements may not be tagged, but comments may be written in the comments field.

FORTRAN MODE STATEMENT. The statement in Figure 79 must precede each Fortran portion of an Autocoder III program.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
	MODE		FORTRAN4	

Figure 79

The operand FORTRAN4 indicates that the subsequent statements are in standard FORTRAN format, i.e., 704 FORTRAN format. For further information on mixing FORTRAN and Autocoder III, see the IBM bulletin, "705 FORTRAN Programming System," Form J28-6122.

REPORT/FILE WRITING MODE STATEMENT. The statement shown in Figure 80 must precede each Report/File Writing portion of an Autocoder III program.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
	MODE		REPORT	

Figure 80

AUTOCODER MODE STATEMENT. The statement shown in Figure 81 must precede each Autocoder III portion of a program if that portion follows Report/File Writing or FORTRAN statements. The statement is used whether or not the Autocoder III portion also contains Decision, Arithmetic, and Table-Creating statements.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
	MODE		AUTOCODER	

Figure 81

NOTE: This MODE statement is not used when the entire program consists of Autocoder III statements alone or in combination with Decision, Arithmetic, and/or Table-Creating statements.

CODING GENERATED IN 7080 MODE

The terms "7080 mode" and "7080 mode portion of the program" are used throughout this manual. They refer to the object machine for which the Processor produces coding, makes location assignments, etc. The program mode is communicated to the Processor by using the macro-instructions Enter Eighty Mode (ENT80) and Leave Eighty Mode (LEV80), both of which are described in the macro-instruction manual. The 7080 mode portion of a program consists of any set of statements preceded by an ENT80 and followed by a LEV80. Of course, if the entire program is in 7080 mode, the LEV80 is not necessary. Since these macro-instructions are Assembly Control macro-instructions, they should be considered along with other instructions to the Processor.

ENT80 and LEV80 affect the coding generated from the statements in the portion of the program that each of them precedes. After encountering an ENT80, the Processor subsequently generates 7080 instructions whenever a choice exists in the type of coding to be generated. This continues until the Processor encounters a LEV80. It then generates 705 II or 705 III coding, depending on which is designated as the secondary mode for the assembly. The program mode is a consideration in using address constants, macro-instructions, one-for-one instructions, and instructions to the Processor. For example, the Processor generates an FIA instruction when it encounters an indirect address in the operand of an instruction in the 7080 mode portion of a program. This is true whether the indirect address appears in a hand-coded one-for-one instruction or a generated instruction. As another example, an ACON6 should not be referenced by an instruction outside the 7080 mode portion of a program.

INSTRUCTIONS TO THE PROCESSOR THAT CONCERN THE PROGRAM LISTING

Skip to New Page - EJECT

The function of an EJECT statement is to advance the listing to a new page. The program statement that follows EJECT will be the first statement on the new page. Unless the listing is controlled by EJECT statements, each page will contain 55 lines of print. The statement is written as shown in Figure 82. It may not be tagged, and it may contain only one line of comments.

TAG	OPERATION	NUM.	OPERAND	
6 15	16 20	21 22	23 38	39
	EJECT			

Figure 82

EJECT does not appear on the listing page. However, it is assigned an index number, and the number is one greater than the index number of the statement that precedes the EJECT. (Index numbers are explained in Chapter 8.)

Title for Routine or Comment - TITLE

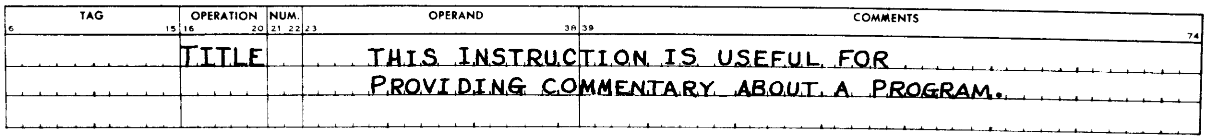
The function of a TITLE statement is to place lines or paragraphs of descriptive information in the program listing. TITLE may be used in any way the programmer desires; some of the more common uses will be discussed following the specifications for writing the statement.

The TITLE statement is written as follows:

TAG FIELD, OPERAND FIELD, COMMENTS FIELD. Any or all of these fields may be used for the descriptive information. The commentary does not have to start in the first column of any of the fields, and it does not have to extend to the end of the comments field before a continuation line is started.

OPERATION FIELD. The mnemonic code TITLE is placed here. If the information is continued into subsequent lines of the coding sheet, i.e., is written as a paragraph, only the first line must contain TITLE. If a series of paragraphs is written, and each is separated by one or more blank lines on the coding sheet, the lines of the paragraphs will be treated as TITLE continuation lines.

NUMERIC FIELD. This field may contain an entry in the first TITLE line. However, it must be left blank in the continuation lines. It is recommended that the numeric field be left blank at all times.



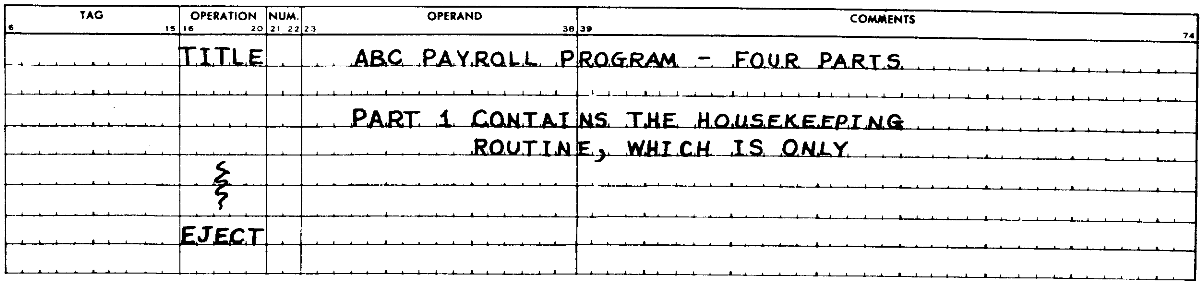
TAG 6 15	OPERATION 16 20	NUM. 21 22	OPERAND 23 38	COMMENTS 39 74
	TITLE		THIS INSTRUC	TION IS USEFUL FOR
			PROVIDING CO	MMENTARY ABOUT A PROGRAM.

Figure 83

COMMON USES OF TITLE. Describing the function of each program portion, summarizing program procedures, and providing a table of contents for the program listing are a few of the uses for TITLE. In addition to appearing in the program listing, all TITLES are also printed in a special section of the Operator's Notebook, an optional feature of the assembly documentation provided by the Processor. This special page shows each TITLE and its location in the listing. This TITLE page is useful as an index for the program listing. It is often desirable to have information about the program at the start of the listing and/or before each major program portion. TITLE can be combined with EJECT, as in Figures 84 and 85, to provide a page of commentary only.

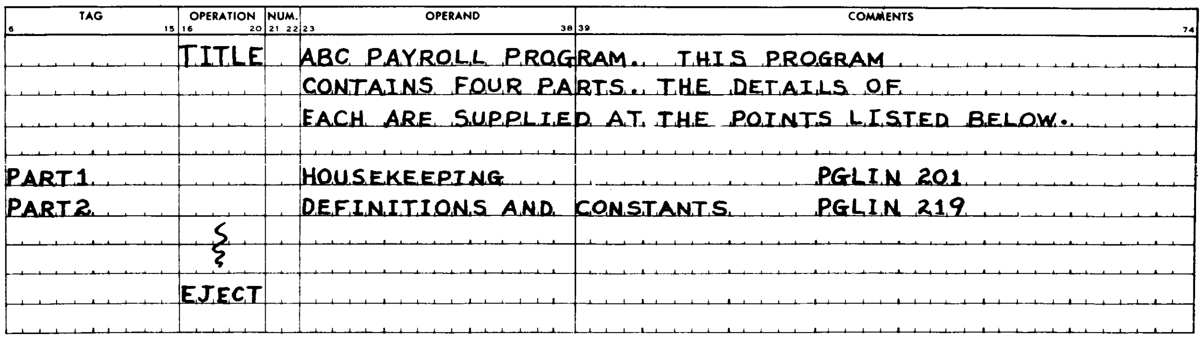
When planning pages of commentary or describing program parts, it should be remembered that an EJECT statement before each part will cause that part to appear on a new page of the listing. Thus, EJECT and TITLE may be used to separate each program portion, to describe it, and to provide a table of contents or an index. The standard listing page contains 55 lines unless EJECT is used. In Figure 84, it must be assumed that TITLEs designating the four program parts have been used elsewhere in the program and that this TITLE page is to be the introductory page of the listing.

In Figure 85, it must be assumed that the listing page containing each of the parts is headed by a TITLE describing that part of the program.



TAG 6 15	OPERATION 16 20	NUM. 21 22	OPERAND 23 38	COMMENTS 39 74
	TITLE		ABC PAYROLL P	ROGRAM - FOUR PARTS.
			PART 1 CONTAI	NS THE HOUSEKEEPING
			ROUTIN	E, WHICH IS ONLY
	EJECT			

Figure 84



TAG	OPERATION	NUM.	OPERAND	COMMENTS
6 15	16 20	21 22	23 38	39 74
	TITLE		ABC PAYROLL PROG	RAM.. THIS PROGRAM
			CONTAINS FOUR PA	RTS.. THE DETAILS OF
			EACH ARE SUPPLIE	D AT THE POINTS LISTED BELOW.
PART1			HOUSEKEEPING	PGLIN 201
PART2			DEFINITIONS AND	CONSTANTS PGLIN 219
	EJECT			

Figure 85

One card is punched for each line of the coding sheet, as explained in Chapter 1. A card-image tape produced from the source program deck is the input to the Processor. The assembly output consists of the object program deck and program documentation. Although the object program deck is normally produced on a card-image tape, it will be referred to as a deck.

OBJECT PROGRAM DECK

The sequence and contents of the deck is shown in the following list:

1. load card
2. literal table
3. machine language equivalent of source program
4. Class A subroutines
5. Class B subroutines
6. standard "00" transfer control card.

Note that the literal table, although assigned to storage locations above those of the object program instructions, precedes the instructions into storage.

The format of the object program card is as follows:

1. Program identification (6 positions). This is the source program identification (ident field on coding sheet).
2. Serial number (3 positions). This is the number of the object program card. It is assigned by the Processor and bears no relation to the number of a source program statement (pg/line fields on coding sheet).
3. Initial address (4 positions). This indicates the storage location at which the first instruction or the first constant on the card is to be placed.
4. Number of columns (2 positions). This is the amount of data being supplied by the card. A maximum of 60 positions may be indicated; this is the space required by 12 instructions. The "00" card contains zeros in these positions.
5. Instructions and/or constants (1-60 positions). This is the actual portion of the object program being supplied by the card. It is placed at the storage location specified by number 3 above.

ASSEMBLY DOCUMENTATION

A listing of the object program itself and diagnostic messages is the minimum assembly documentation; optional documentation consisting of the Operator's Notebook and the Symbolic Analyzer may be requested as additions to the listing. A column-by-column explanation of the listing format appears in a subsequent section of this chapter, "Details of the Listing."

Program Listing

The contents of the listing are as follows:

1. Literal Table. The Processor classifies a literal as signed when the first position after the literal symbol (#) is occupied by a plus or minus sign. The literal table is separated into five parts: signed literals, length not a multiple of five; signed literals, length a multiple of five; unsigned literals, length a multiple of five; unsigned literals, length not a multiple of five; address constant literals.
2. Source Program With Generated Coding. This may be considered the main portion of the program listing. The source program statements appear in their original sequence; any generated coding appears directly after the statement that caused the generation.
3. Class A Subroutines. The subroutines are inserted alphabetically, i.e., according to the mnemonic identification code of each subroutine. Any generated coding appears directly after the statement that caused the generation.
4. Class B Subroutines. The subroutines are inserted alphabetically.
5. Diagnostic Messages. These messages are produced by the Processor and indicate errors, or possible errors, in source program statements. When the Processor detects a possible error conditioı , it often makes certain assumptions and generates coding based on them. It also supplies a warning message on the nature of the possible error or the action taken to correct an error. The reference manual, "Autocoder III - System Operation," Form C28-6062, describes such messages.

Operator's Notebook - Optional Documentation

This is an index to the location of certain types of Autocoder III statements, both hand-coded and generated, that appear in the program listing. The pages that comprise the Notebook are as follows:

1. HALTS
2. TITLES
3. 80 SPEC
4. ASSGNS (LASN and SASN)
5. SWITCHES (SWN and SWT)

The TITLE page also lists all ENT80 and LEV80 macro-instructions. The 80 SPEC page lists each statement containing an "I," prefix (indirect address) in the 7080 mode portion of the program

Symbolic Analyzer - Optional Documentation

This is an index of every hand-coded and generated tag in the program. The tags are listed in alphabetical order, and each is followed by a list of every instruction, either hand-coded or generated, that references the tag.

DETAILS OF THE PROGRAM LISTING

The listing page consists of 14 fields. The entries in the TAG through the COMMENTS fields comprise an Autocoder III statement. The machine language translation of the statement, i.e., an object program instruction or constant, appears in the OP through the ADDRESS fields. Other fields contain information on storage locations, statement sequence, and references to other statements. The fields of the listing are as follows:

INDEX. This is a number that the Processor creates for each line of the listing. A hand-coded statement is assigned a number of the form xxbyy; a generated statement is assigned a number of the form bxxyy. In each case, xx is the listing page number and yy is the line number. On a reassembly, a number of the form xx*yy is assigned to a statement that has been replaced, added, or that follows a deleted statement. The INDEX number is not identical to the pg/line number on the coding sheet.

TAG. Any hand-coded or generated tag appears in this field, which corresponds to the tag field on the coding sheet.

OP. Any mnemonic code appears in this field, which corresponds to the operation field on the coding sheet.

NU. The entry in this field varies just as it does when hand-coded. The field corresponds to the num field on the coding sheet.

AT. An entry in this field is either an operand modifier or an indirect address. On the coding sheet, such entries are written in columns 23-24 of the operand field.

OPERAND. The entry in this field varies just as it does when hand-coded. The field corresponds to the operand field on the coding sheet with two exceptions. The first is the placement of a prefix to the basic operand; this appears in the AT field explained in the preceding paragraph. The second is the placement of generated character adjustment; in the listing, hand-coded character adjustment appears as it was written, but generated character adjustment appears in the right-hand portion of the COMMENTS field.

COMMENTS. Any source program comments appear in this field, which corresponds to the comments field on the coding sheet.

LOC. The entry in this field is a six-character number designating the location assigned to the object program instruction or constant.

OP. The entry in this field is the actual operation code of the instruction.

SU. The entry in this field is an ASU number. It does not necessarily correspond to the num field, which is used for other purposes besides ASU assignments.

ADDRESS. This field contains two entries. One shows the actual address portion of an instruction as six positions, and the other shows it as four positions with ASU zoning included. The latter, when combined with the operation code, forms the machine language instruction.

PGLIN. The entry in this field corresponds to the pg/line entry on the coding sheet.

SER. An entry in this field is the three-character serial number of an object program card. The number appears only in the line containing the first instruction on the object program card. Subsequent lines with blanks in the SER field contain instructions that appear on the same card.

REF. An entry in this field is the INDEX number of some other line in the listing and serves as a cross-reference. The entry appears when the instruction references a field or another instruction.

CHAPTER 9. THE COMMUNICATION WORD

A portion of the Processor called the communication word is described in the system operation manual previously mentioned. The Processor obtains certain information from the communication word during the assembly. The purpose of this chapter is to indicate those Autocoder III symbols which are defined in the communication word of the standard 7058 Processor as distributed by IBM. If the communication word has been modified to suit the needs of an individual 705 or 7080 installation, the modifications may affect the detailed information on the use of the symbols that this manual supplies.

The following are defined in the communication word as Autocoder III symbols:

1. plus sign (+)
2. minus sign (-)
3. multiplication sign (*)
4. division sign (/)
5. decimal point and period (.)
6. left literal (#)
7. right literal (#)
8. actual address (@)
9. location counter (*)
10. macro break (¤)

Another portion of the communication word supplies information about each ASU, i.e., how it is set and whether the Processor may assign it to coding that the Processor is generating. ASU settings and availability are described in the macro-instruction manual. It is sufficient here to note that certain ASUs are selected by the Processor for assignment to a generated instruction, and the programmer should be aware of this when writing one-for-one instructions that require an ASU assignment.

The terms that follow are explained in relation to their use in this manual. No attempt has been made to supply a glossary of basic programming terms. Definitions that appear in the text of the manual are not repeated on this page. The Index supplies page references to such definitions.

Address. Something that designates a storage location. The term "address of an instruction" and the term "address portion" both refer to the portion of a machine language instruction that identifies a storage location.

Alphabetic characters. The letters A-Z. Alphabetic data consists of alphabetic characters.

Alphameric characters. A set of characters comprising the following: alphabetic, numeric, special, blank. Alphameric data consists of any of these characters or any combination of them.

Blank character. The absence of a character. May be designated on the coding sheet by the symbol b.

Coding. Program statements that may or may not form a routine.

Data field. A unit of information consisting of an alphameric character or a set of adjacent alphameric characters.

Decimal positions. The positions to the right of the decimal point in numeric data.

Format layout. A graphic representation on the coding sheet of a specific arrangement of characters. Also referred to as a "layout."

Generated. An adjective describing coding provided by the Processor.

Hand-coded. An adjective describing coding written by the programmer.

Integer positions. The positions to the left of the decimal point in numeric data.

Initialization. A procedure that places an instruction or a switch in an initial condition or restores either one to a previously defined condition. Initialization is a type of modification.

Location. A place in storage. The term may refer to one storage position or the positions occupied by a field or an instruction. Also referred to as "storage location."

Machine language. A language that is intelligible to the computer. Also referred to as "actual language."

Machine language instruction. A 705 or 7080 machine instruction consisting of an actual operation code and an address portion.

Mixed decimal. A term used to designate a number containing integer and decimal positions.

Modification. A procedure that alters an instruction or a switch setting. Address modification is the procedure of altering the address portion of an instruction.

Numeric characters. The digits 0-9. Numeric data consists of a combination of digits representing a signed or unsigned integer, pure decimal, or mixed decimal.

Processor library. The portion of the 7058 Processor System tape that contains the elements of each macro-instruction and each subroutine.

Pure decimal. A term used to designate a number containing decimal positions only.

Record. A set of adjacent data fields.

Special characters. The following group of characters: .⌑‡&$*-/,%#@+‡

APPENDIX A. SAMPLE ASSEMBLY

```
INDEX    TAG      OP    NU AT    OPERAND    PROGRAM SAMPLE 05-22-62    COMMENTS       LOC  OP SU    ADDRESS    PGLIN SER REF
AA 01             TITLE            THIS SAMPLE ASSEMBLY HAS BEEN PROVIDED  AS AN                             0101
AA 02                              EXAMPLE OF THE ASSEMBLY DOCUMENTATION DESCRIBED                           0102
AA 03                              IN CHAPTER 8 AND THE ASSEMBLING OF SOURCE                                 0103
AA 04                              PROGRAM STATEMENTS. THE OPERATORS NOTEBOOK                                0104
AA 05                              AND THE SYMBOLIC ANALYZER, WHICH COMPRISE THE                             0105
AA 06                              OPTIONAL DOCUMENTATION, APPEAR AFTER THE                                  0106
AA 07                              ASSEMBLY LISTING. TITLES AND COMMENTS INDICATE                            0107
AA 08                              THE FUNCTIONS OF THE VARIOUS STATEMENTS. SINCE                            0108
AA 09                              THIS ASSEMBLY IS NOT INTENDED TO ILLUSTRATE                               0109
AA 10                              A PROGRAM, THE SOURCE STATEMENTS DO NOT FORM                              0110
AA 11                              A ROUTINE.                                                                0111
AA 12                                                                                                        0112
AA 13                              THIS PAGE CONTAINS THE INITIAL SOURCE                                     0113
AA 14                              STATEMENT,A TITLE STATEMENT, AND IT WOULD                                 0114
AA 15                              NORMALLY APPEAR AFTER THE PAGE CONTAINING                                 0115
AA 16                              THE LITERAL TABLE. IT HAS BEEN TAKEN OUT                                  0116
AA 17                              OF SEQUENCE TO ILLUSTRATE TITLE AS A CONVENIENT                           0117
AA 18                              MEANS OF SUPPLYING INTRODUCTORY COMMENTS TO                               0118
AA 19                              PRECEDE A PROGRAM LISTING.                                                0119
AA 20                                                                                                        0120
AA 21                              IN THE LISTING, THE HAND-CODED STATEMENTS ARE                             0121
AA 22                              DISTINGUISHED FROM GENERATED STATEMENTS BY THE                            0122
AA 23                              POSITION OF THE INDEX NUMBERS. THOSE OF THE                               0123
AA 24                              HAND-CODED STATEMENTS ARE LEFT-JUSTIFIED,                                 0124
AA 25                              WHILE THOSE OF THE GENERATED STATEMENTS ARE                               0125
AA 26                              INDENTED ONE POSITION. MACRO-HEADERS ARE                                  0201
AA 27                              DISTINGUISHED FROM THE GENERATED CODING                                   0202
AA 28                              BY OVER-PRINTING. BECAUSE THIS ASSEMBLY IS                                0203
AA 29                              SHORT AND CONSISTS MAINLY OF SOURCE STATEMENTS,                           0204
AA 30                              A SEPARATE LISTING OF THE SOURCE PROGRAM                                  0205
AA 31                              DOES NOT APPEAR.                                                          0206
AA 32                                                                                                        0207
AA 33                              THE SUBROUTINE THAT APPEARS AT THE END OF                                 0208
AA 34                              THE PROGRAM IS AN IBM SUBROUTINE. IT IS                                   0209
AA 35                              AUTOMATICALLY PROVIDED ON EVERY ASSEMBLY.                                 0210
```

```
INDEX    TAG        OP     NU AT   OPERAND      PROGRAM SAMPLE 05-22-62    COMMENTS        LOC   OP SU    ADDRESS    PGLIN SER REF

¤A00  SIGNED LITERAL    1                                                                  001350                                 001
¤A01  SIGNED LITERAL    1   &                                                              001351
¤A02  SIGNED LITERAL    1   A                                                              001352
¤A03  SIGNED LITERAL    2   0C                                                             001354
¤A04  SIGNED LITERAL    3   12C                                                            001357
¤A05  SIGNED LITERAL    4   000A                                                           001361

¤A06  UNSIGNED LITERAL  1                                                                  001362
¤A07  UNSIGNED LITERAL  1   .                                                              001363
¤A08  UNSIGNED LITERAL  1   ,                                                              001364
¤A09  UNSIGNED LITERAL  1   @                                                              001365
¤A10  UNSIGNED LITERAL  1   A                                                              001366
¤A11  UNSIGNED LITERAL  1   1                                                              001367
¤A12  UNSIGNED LITERAL  2   AB                                                             001369
¤A13  UNSIGNED LITERAL  2   00                                                             001371
¤A14  UNSIGNED LITERAL  4   0002                                                           001375

*A01 TAG2        HI-SP  5    0659                                                          001384
*A02 TAG2        HI-SP  5    06N9                                                          001389
*A03 XXPRTSTHLD RIGHT   5    1324                                                          001394
```

```
INDEX    TAG        OP     NU AT   OPERAND      PROGRAM SAMPLE 05-22-62    COMMENTS        LOC   OP SU    ADDRESS    PGLIN SER REF

AB 01 SECTION 1  TITLE       THIS SECTION CONTAINS THE DEFINITION OF A RECORD                                            0301

AB 02                        AREA AND MACRO-HEADERS THAT REFERENCE INTERIOR                                              0302

AB 03                        FIELDS OF THE AREA. THE GENERATED CODING FOR EACH                                           0303

AB 04                        MACRO- INSTRUCTION CONTAINS LINKAGE TO CODING                                               0304

AB 05                        GENERATED FROM THE SUBROUTINE AT THE END OF THE                                             0305

AB 06                        PROGRAM                                                                                     0306

AB 07            LASN        @500                                               000500                                   0307

AB 08 RECORD     NAME    0   RECORDEND          TO DEFINE THE INTERIOR FIELDS AND TO 000500                              0308       AB15

AB 09                                           POSITION FIELD1 TO START IN A 0/5                                        0309

AB 10                                           LOCATION                                                                 0310
AB 11 FIELD1     RCD     2   A&                 LEFT PROTECTION FOR FOLLOWING FIELD  000501                              0311          2
AB 12 FIELD2             5   #&03.02            SIGNED NUMERIC FIELD                 000506                              0312          7
AB 13 FIELD3     RPT     7   ZZZ.ZZ ¤¤-¤¤&¤     LOW-ORDER POSITION OF LAYOUT IS A    000513                              0313   002   14
AB 14                                           BLANK FOR SIGN INDICATION                                                0314
AB 15 RECORDEND  CON     1   #                  TO SUPPLY TERMINAL RECORD MARK       000514                              0315         15
 AB16 RECORD                 THIS DEFINITION OCCUPIES    15 CHARACTER POSITIONS  000514                                  0315

AB 17 ROUTINE1   MOVE        #AB#¤FIELD1¤       TO PLACE THE LITERAL AB,WHICH                                            0316

AB 18                                           PROVIDES LEFT PROTECTION, IN FIELD1                                      0317
 AB19 ROUTINE1   RCVS        FIELD1                                              000519 U    000500 0500                          AB11
 AB20            TMTS   02   #AB#                                                000524 9  2 001368 1308                          ¤A12

AB 21 ROUTINE2   ADDX        FIELD2¤#&123#¤FIELD3¤  TO ADD THE CONTENTS OF FIELD1                                        0318

AB 22                                               TO THE LITERAL &123 AND PLACE                                        0319

AB 23                                               THE SUM IN FIELD3                                                    0320
 AB24 ROUTINE2   RAD         #&123#                                              000529 H    001357 1357                          ¤A04
 AB25            LNG         @00002                                              000534 D    000002 0002
 AB26            ADD         FIELD2                                              000539 G    000506 0506                          AB12
 AB27            RCVS        XXPRTST                                      &00006 000544 U    000771 0771                          AE05
 AB28            TSL         XXPRTST                                             000549 1  1 000769 07W9                          AE05

 AB29            TR          *&00030                                             000554 1    000584 0584

 AB30            CON     1   9                                                   000555
 AB31            ACON4     L,FIELD3                                              000559      000507 0507                          AB13
 AB32            CON     1                                                       000560
 AB33                    2   &00                                                 000562
 AB34                    2   &03                                                 000564
 AB35                    2   &05                                                 000566
 AB36                    2   &03                                                 000568                                  003
 AB37                    2   &02                                                 000570
 AB38                    1   &1                                                  000571
 AB39                    1   &1                                                  000572
 AB40                    1   &                                                   000573
 AB41                    1                                                       000574
 AB42                    1                                                       000575
 AB43                    1                                                       000576
 AB44                    1   -                                                   000577
 AB45                    1                                                       000578
 AB46                    1   #                                                   000579
```

```
INDEX  TAG       OP    NU AT  OPERAND   PROGRAM SAMPLE 05-22-62  COMMENTS       LOC  OP SU   ADDRESS   PGLIN SER REF
AB 47 SECTION 2  TITLE        THIS SECTION CONTAINS DEFINITIONS OF SWITCHES AND                              0321

AB 48                         MACRO-HEADERS THAT REFERENCE THE SWITCHES•                                     0322
AB 49            RCD   0                   TO CAUSE THE PROCESSOR TO ASSUME     000579                       0323
AB 50                                      THE SUBSEQUENT SWITCH IS PART OF                                  0324
AB 51                                      A RECORD AREA                                                     0325
AB 52            CHRCD                     ONE-POSITION CHARACTER CODE SWITCH•  000580                       0301
AB 53 CHICAGO            A                 IT WILL NOT BE SET INITIALLY• SINCE  000580                       0302
AB 54 NEWYORK            B                 THE DEFINITION FOLLOWS AN RCD•       000580                       0303
AB 55 LOSANGELES         C                                                      000580                       0304
AB 56 SWITCHOFF  SWN     ROUTINE1          PROGRAM SWITCH THAT IS A             000589 A    000519 0519 0305  004AB19

AB 57                                      NO-OPERATION INSTRUCTION INITIALLY                                0306

AB 58 SWITCHON   SWT     ROUTINE2          PROGRAM SWITCH THAT IS A TRANSFER    000594 1    000529 0529 0307     AB24

AB 59                                      INSTRUCTION INITIALLY• ROUTINE2 IS                                0308

AB 60                                      THE TRANSFER POINT                                                0309

AB 61            BITCD   5                 ONE-POSITION BIT CODE SWITCH• THE    000595                       0310
AB 62 DAILY        1                       INITIAL SETTING IS DESIGNATED•       000595                       0311

AC 01 WEEKLY       2                       BECAUSE THE SWITCH DOES NOT FOLLOW   000595                       0312
AC 02 MONTHLY      4                       AN RCD• SINCE THE 2-BIT IS TO BE SET 000595                       0313
AC 03                                      TO ZERO•WEEKLY IS SET ON INITIALLY•                               0314

AC 04            IFON    CHICAGO¤¤OUT¤     TO DETERMINE IF THE CHARACTER CODE                                0315

AC 05                                      SWITCH CONTAINS THE CODE A•TO                                     0316

AC 06                                      TRANSFER TO THE NEXT IN-LINE                                      0317

AC 07                                      INSTRUCTION IF IT DOES• AND TO                                    0318

AC 08                                      TRANSFER TO OUT IF IT DOES NOT•                                   0319
 AC09            LOD   01 #A#                                                   000604 8  1 001366 13W6       005¤A10
 AC10            CMP   01 CHICAGO                                               000609 4  1 000580 05Y0          AB53
 AC11            TRE      NEXT¤00004                                            000614 L    000624 0624          AC13
 AC12            TR       OUT                                                   000619 1    000639 0639          AC21

 AC13 NEXT¤00004 TRANS    *

AC 14            SETON    DAILY¤SWITCHOFF¤   TO SET THE 1-BIT OF THE BIT CODE                                0320

AC 15                                        SWITCH ON AND TO MODIFY SWITCHOFF                               0321

AC 16                                        SO THAT IT BECOMES A TRANSFER                                   0322

AC 17                                        TO ROUTINE1                                                     0323
 AC18            SBZ    1 DAILY                                                 000624 %  1 000595 05Z5          AB62
 AC19            RCVS     SWITCHOFF                                             000629 U    000585 0585          AB56
 AC20            TMTS   1 #1#                                                   000634 9  1 001367 13W7          ¤A11
AC 21 OUT        HLT      9                TO PROVIDE A TRANSFER POINT          000639 J    000009 0009 0324


AD 01 SECTION 3  TITLE    THIS SECTION CONTAINS AREA DEFINITIONS•ONE-FOR-ONE                                 0401

AD 02                     INSTRUCTIONS•AND ADDRESS CONSTANTS• ASSUME THAT                                    0402

AD 03                     ASU 01 IS SET TO ONE POSITION•                                                     0403

AD 04            NAME   4                  TO POSITION THE SUBSEQUENT FIELD TO  000644                       0404

AD 05                                      START IN A 4/9 LOCATION                                           0405
AD 06            CON    1                  TO PROVIDE LEFT-PROTECTION FOR THE   000644                       0406  006
AD 07                                      FOLLOWING FIELD                                                   0407
AD 08 TAG1       RCD   10  &               SIGNED NUMERIC FIELD                 000654                       0408
AD 09 TAG2             10  &               SIGNED NUMERIC FIELD                 000664                       0409
AD 10                   5  &               SIGNED NUMERIC FIELD                 000669                       0410
AD 11            TR         *&15           TO BYPASS TWO INSTRUCTIONS           000674 1    000689 0689 0411  007

AD 12 STEP1      RAD        TAG1           TO PLACE CONTENTS OF TAG1 IN         000679 H    000654 0654 0412     AD08
AD 13                                      ACCUMULATOR                                                       0413
AD 14            LOD    1 L•TAG1           TO PLACE CONTENTS OF HIGH-ORDER      000684 8  1 000645 06U5 0414     AD08
AD 15                                      POSITION OF TAG1 IN ASU 01                                        0415
AD 16            LOD    1 L•TAG1&1         TO PLACE CONTENTS OF POSITION TO     000689 8  1 000646 06U6 0416     AD08
AD 17                                      RIGHT OF HIGH-ORDER POSITION OF                                   0417
AD 18                                      TAG1 IN ASU 01                                                    0418
AD 19            RCVS       TAG1           TO RECEIVE INTO HIGH-ORDER POSITION  000694 U    000645 0645 0419     AD08
AD 20                                      OF TAG1                                                           0420
AD 21            TMTS   1   TAG2           TO TRANSMIT CONTENTS OF HIGH-ORDER   000699 9  1 000655 06V5 0421     AD09
AD 22                                      POSITION OF TAG2 THROUGH ASU 01 INTO                              0422
AD 23                                      TAG1                                                              0423
AD 24            RAD        TAG2&5         TO PLACE THE CONTENTS OF THE FIELD   000704 H    000669 0669 0424     AD09
AD 25                                      FOLLOWING TAG2 IN THE ACCUMULATOR                                 0425
AD 26            RAD      I•STEP1          THE INDIRECT ADDRESS DESIGNATES THE  000709 H    000679 067Z 0501     AD12
AD 27                                      OPERAND OF THE INSTRUCTION TAGGED                                 0502
AD 28                                      STEP1                                                             0503
```

```
INDEX TAG          OP      NU AT OPERAND          PROGRAM SAMPLE 05-22-62   COMMENTS   LOC    OP SU  ADDRESS    PGLIN SER REF

AD 29              ENT80                                                                                       0504

AD 30              RAD        I,STEP1             SAME INSTRUCTION IN 7080 MODE                                0505
AD 31              EIA          STEP1             SAME INSTRUCTION IN 7080 MODE        000714 , 10 000679 00P9 0505    AD12
AD 32              RAD          STEP1             SAME INSTRUCTION IN 7080 MODE        000719 H    000679 0679 0505    AD12

AD 33              LEV80                                                                                       0506
AD 34 STEP2        ADCON        TAG1              TO MAKE THE LOCATION OF THE LOW-     000724 A    000654 0654 0507    AD08
AD 35                                             ORDER POSITION OF TAG1 A CONSTANT.                           0508
AD 36              CON      1                     TO POSITION THE FOLLOWING ACON4 IN A 000725                  0509
AD 37                                             4/9 LOCATION BY INSERTING A BLANK                            0510
AD 38              ACON4      L,TAG1              TO MAKE THE LOCATION OF THE HIGH-    000729      000645 0645 0511    AD08
AD 39                                             ORDER POSITION OF TAG1 A CONSTANT                            0512 008
AD 40              ACON4    3 L,TAG1              TO ZONE THE SAME CONSTANT FOR ASU 03 000733    3 000645 06D5 0513    AD08
AD 41 STEP3        ACON5    &   @1000             TO MAKE LOCATION 1000 A 5-POSITION   000738      001000      0514
AD 42                                             CONSTANT,SIGNED PLUS                                         0515
AD 43              ACON5      S,TAG1              TO MAKE THE SIZE OF TAG1 A           000743      000010      0516    AD08
AD 44                                             5-POSITION UNSIGNED CONSTANT                                 0517
AD 45              ACON6        TAG1              TO MAKE THE LOCATION OCCUPIED BY THE 000749      000654      0518    AD08
AD 46                                             LOW-ORDER POSITION OF TAG1 A                                 0519
AD 47                                             6-POSITION CONSTANT                                          0520
AD 48              RAD          STEP3             TO PLACE THE CONSTANT TAGGED STEP3   000754 H    000738 0738 0521    AD41
AD 49                                             IN THE ACCUMULATOR                                           0522
AD 50              LDA     15 H@TAG2              TO MAKE THE HIGH SPEED ADDRESS OF    000759 # 15 001384 1CH4 0523    *A01
AD 51                                             TAG2 A CONSTANT, TO PLACE THE                                0524
AD 52                                             CONSTANT IN THE LITERAL TABLE, AND                           0525
AD 53                                             TO MAKE THE LOCATION OF THE CONSTANT                         0601
AD 54                                             THE ADDRESS PORTION OF THE LDA                               0602
AD 55                                             INSTRUCTION. AT OBJECT TIME, THE                             0603
AD 56                                             LOCATION OF TAG2 IS PLACED IN ASU15                          0604
AD 57              LDA     15 H@TAG2              TO ZONE THE SAME CONSTANT.    ASU 02 000764 # 15 001389 1CH9 0606    *A02


AE01                          THE FOLLOWING ENTRIES ARE CLASS A SUBROUTINES

AE02 XXPRTST       PRTST      XXPRTST01□XXPRTST02□XXPRTST03□XXPRTST04□XXPRTST05□                               001

AE03                          XXPRTST06□XXPRTSTFLS□XXPRTSTWRK□XXPRTSTHLD□                                      002

AE04                          XXPRTSTEND□                                                                      003
AE05 XXPRTST       RCV          XXPRTSTWRK                                             000769 U    001244 1244         AF29
AE06               ADCON                                                               000774 A           0000
AE07               SET     15   @00006                                                 000779 B 15 000006 0&&6
AE08               SND     15 I,XXPRTST                                         &00005 000784 / 15 000774 0GGU         AE05
AE09               RCVS       I,XXPRTSTWRK                                      -00020 000789 U    001249 124Z         AF29
AE10               SGN     01   XXPRTST01                                       -00004 000794 T  1 001055 10V5      009AE63
AE11               ADM     01   XXPRTST01                                       -00004 000799 6  1 001055 10V5         AE63
AE12               ADM     01   XXPRTST02                                       -00004 000804 6  1 000960 09W0         AE44
AE13               ADM     01   XXPRTST03                                       -00004 000809 6  1 001105 11#5         AF02
AE14               ADM     01   XXPRTST04                                       -00004 000814 6  1 001025 10S5         AE57
AE15               ADM     01   XXPRTST05                                       -00004 000819 6  1 001080 10Y0         AE68
AE16               UNL          XXPRTSTHLD                                             000824 7    001324 1324         AF30
AE17               LDA     14 R@XXPRTSTHLD                                             000829 # 14 001394 1CR4         *A03
AE18               SUB     14   XXPRTSTWRK                                      -00013 000834 P 14 001256 1BN6         AF29
AE19               ULA     14   XXPRTST02                                       &00005 000839 * 14 000969 0I09         AE44
AE20               UNL      2   XXPRTSTHLD                                      -00053 000844 7  2 001271 12P1         AF30
AE21               UNL      3   XXPRTSTHLD                                      -00050 000849 7  3 001274 12G4         AF30
AE22               RAD      3   #&0#                                                   000854 H  3 001351 13E1      010□A01
AE23               RAD     14   #&0001#                                                000859 H 14 001361 1C01         □A05
AE24               LOD      1   #@#                                                    000864 8  1 001365 13W5         □A09
AE25               RAD      2   XXPRTSTWRK                                      -00017 000869 H  2 001252 12N2         AF29
AE26               RAD     15   XXPRTSTWRK                                      -00015 000874 H 15 001254 1BE4         AF29
AE27               CMP      1   XXPRTSTWRK                                      -00024 000879 4  1 001245 12U5         AF29
AE28               TRH          *&00010                                                000884 K    000894 0894
AE29               TR           *&00050                                                000889 1    000939 0939

AE30               TMTS     1   XXPRTSTWRK                                      &00005 000894 9  1 001245 12U5         AF29
AE31               CMP     15   #00#                                                   000899 4 15 001371 1CG1         □A13
AE32               TRE          *&00015                                                000904 L    000919 0919
AE33               SUB     15   #&1#                                                   000909 P 15 001352 1CE2         □A02
AE34               TR           *&00025                                                000914 1    000939 0939      011

AE35               RAD     15   #&03#                                                  000919 H 15 001354 1CE4         □A03
AE36               TR           *&00015                                                000924 1    000939 0939

AE37 XXPRTST06     SUB     15   #&1#                                                   000929 P 15 001352 1CE2         □A02
AE38               SUB      2   #&1#                                                   000934 P  2 001352 13N2         □A02
AE39               CMP      2   #00#                                                   000939 4  2 001371 13P1         □A13
AE40               TRE          XXPRTST01                                              000944 L    001059 1059         AE63
AE41               CMP     15   #00#                                                   000949 4 15 001371 1CG1         □A13
AE42               TRE          XXPRTST04                                              000954 L    001029 1029         AE57
AE43               AAM     14   *&00010                                                000959 @ 14 000969 0I09
AE44 XXPRTST02     SWN          *&00025                                                000964 A    000989 0989
```

```
INDEX   TAG         OP    NU AT   OPERAND     PROGRAM SAMPLE 05-22-62   COMMENTS        LOC   OP SU    ADDRESS    PGLIN SER REF

 AE45               CMP    3                                                         000969 4  3        0060
 AE46               TRE           *600045                                            000974 L    001019 1019          012
 AE47               SGN    1      XXPRTST02                                   -00004 000979 T  1 000960 09W0             AE44
 AE48               SGN    1      XXPRTST04                                   -00004 000984 T  1 001025 10S5             AE57
 AE49               TMTS   1 I.XXPRTST02                                      600005 000989 9  1 000969 09WZ             AE44
 AE50               CMP    2      #00#                                               000994 4  2 001371 13P1             ⌑A13
 AE51               TRE           *600010                                            000999 L    001009 1009
 AE52               TR            XXPRTST06                                          001004 1    000929 0929             AE37

 AE53               SGN    1      XXPRTST05                                   -00004 001009 T  1 001080 10Y0             AE68
 AE54               TR            XXPRTST01                                   600005 001014 1    001064 1064             AE63

 AE55               TMTS   1      XXPRTSTWRK                                  600010 001019 9  1 001250 12V0             AF29
 AE56               TR            XXPRTST06                                          001024 1    000929 0929             AE37

 AE57 XXPRTST04     SWN           *600020                                            001029 A    001049 1049

 AE58               TMTS   1      XXPRTSTWRK                                  600010 001034 9  1 001250 12V0          013AF29
 AE59               RAD   15      #603#                                              001039 H 15 001354 1CE4             ⌑A03
 AE60               TR            XXPRTST06                                   600005 001044 1    000934 0934             AE37

 AE61               TMTS   1      #,#                                                001049 9  1 001364 13W4             ⌑A08
 AE62               TR            *-00015                                            001054 1    001039 1039

 AE63 XXPRTST01     SWN           XXPRTST03                                   -00015 001059 A    001094 1094             AF02

 AE64               RAD    2      XXPRTSTWRK                                  -00011 001064 H  2 001258 12N8             AF29
 AE65               SGN    1      XXPRTST01                                   -00004 001069 T  1 001055 10V5             AE63
 AE66               SGN    1      XXPRTST04                                   -00004 001074 T  1 001025 10S5             AE57
 AE67               UNL    3      XXPRTSTWRK                                  -00019 001079 7  3 001250 12E0             AF29
 AE68 XXPRTST05     SWN           XXPRTST06                                          001084 A    000929 0929             AE37

 AE69               TR            XXPRTST06                                   600010 001089 1    000939 0939             AE37

 AE70               RAD    1      XXPRTSTWRK                                  -00008 001094 H  1 001261 12W1          014AF29
 AE71               CMP    1      #1#                                                001099 4  1 001367 13W7             ⌑A11

 AF01               TRE           *600030                                            001104 L    001134 1134
 AF02 XXPRTST03     SWN           *600045                                            001109 A    001154 1154

 AF03               RAD    2      XXPRTSTWRK                                  -00009 001114 H  2 001260 1200             AF29
 AF04               RAD   15      XXPRTSTWRK                                  -00009 001119 H 15 001260 1BF0             AF29
 AF05               SGN    1      XXPRTST03                                   -00004 001124 T  1 001105 11#5             AF02
 AF06               TR            XXPRTST06                                   600010 001129 1    000939 0939             AE37

 AF07               TMTS   1      #.#                                                001134 9  1 001363 13W3             ⌑A07
 AF08               ADD    1      #61#                                               001139 G  1 001352 13V2             ⌑A02
 AF09               ST     1      XXPRTSTWRK                                  -00008 001144 F  1 001261 12W1             AF29
 AF10               TR            XXPRTST03                                          001149 1    001109 1109             AF02

 AF11               RAD    1      XXPRTSTWRK                                  -00007 001154 H  1 001262 12W2          015AF29
 AF12               TRZ    1      XXPRTSTEND                                         001159 N  1 001229 12S9             AF26
 AF13               LOD   14      *600005                                            001164 8 14 001169 1A09
 AF14               ADCON 01      XXPRTSTWRK                                  -00006 001169 A  1 001263 12W3             AF29
 AF15               UNL   14      *600050                                            001174 7 14 001224 1BK4
 AF16               CMP    1      #1#                                                001179 4  1 001367 13W7             ⌑A11
 AF17               TRE           *615                                               001184 L    001199 1199
 AF18               LOD    1      # #                                                001189 8  1 001362 13W2             ⌑A06
 AF19               ADM    1      *600029                                            001194 6  1 001223 12S3
 AF20               LOD   14      #0002#                                             001199 8 14 001375 1CP5             ⌑A14
 AF21               TRZ           *600015                                            001204 N    001219 1219
 AF22               TRP           *600015                                            001209 M    001224 1224
 AF23               AAM   14      *600010                                            001214 @ 14 001224 1BK4          016
 AF24               AAM   14      *600005                                            001219 @ 14 001224 1BK4
 AF25               TMTS   1                                                         001224 9  1        00#0
 AF26 XXPRTSTEND LOD       2      XXPRTSTHLD                                  -00053 001229 8  2 001271 12P1             AF30
 AF27               SET    3      3                                                  001234 B  3 000003 0063
 AF28               LOD    3      XXPRTSTHLD                                  -00050 001239 8  3 001274 12G4             AF30
 AF29 XXPRTSTWRK CON      30                                                         001269
 AF30 XXPRTSTHLD CON      55                                                         001324                       017

 AF31 IOSAS         IORAS                                                                                   004

 AF32               XXACC         XAC⌑XAC1⌑XAC2⌑XAC3⌑XAC4⌑XACB⌑XACA⌑                                        005
 AF33               CON    1                                                         001325
 AF34 XAC                  5      600000                                             001330                       018

 AF35               XXACD         XACC⌑XACC1⌑                                                               006

 AF36               NAME   3                                                         001333
 AF37               CON    1                                                         001333                       019
 AF38 XACC                 6      6000000                                            001339
 AF39                      4                                                         001343
 AF40 XACC1                6      6000000                                            001349

 AF41 XXBSRCH       BSRCH         XXBSRCH01⌑XXBSRCH02⌑XXBSRCH03⌑XXBSRCH04⌑XXBSRCH05⌑                          007

 AF42                             XXBSRCH06⌑XXBSRCH07⌑XXBSRCH08⌑XXBSRCH09⌑XXBSRCH10⌑                          008

 AF43                             XXBSRCHEXT⌑                                                               009
```

```
INDEX    TAG        OP    NU AT    OPERAND     PROGRAM SAMPLE 05-22-62    COMMENTS        LOC  OP SU    ADDRESS    PGLIN SER REF

AF44 XXSSRCH     SSRCH        XXSSRCH01⌑XXSSRCH02⌑XXSSRCH03⌑XXSSRCH04⌑XXSSRCH05⌑                              010

AF45                          XXSSRCH06⌑XXSSRCHEXT⌑                                                           011

AF46 XXFIX       FIX          XXFIX01⌑                                                                        012

AF47 XXFLOAT     FLOAT        XXFLOAT01⌑                                                                      013

AF48 IOTYPENTRY TYPIO         IOTYPEXIT⌑                                                                      014

AF49 XXADDENTRY AITEM         XXADDEXIT⌑XXDELET02⌑XXDELET04⌑XXDELET03⌑XXDELET07⌑                              015

AF50                          XXDELET05⌑XXDELET06⌑                                                            016

AF51 XXDELENTRY DITEM         XXDELEXIT⌑XXDELET02⌑XXDELET03⌑XXDELET04⌑XXDELET05⌑                              017

AF52                          XXDELET06⌑XXDELET07⌑                                                            018
```

```
INDEX    TAG        OP    NU AT    OPERAND     PROGRAM SAMPLE 05-22-62    MESSAGES        LOC  OP SU    ADDRESS    PGLIN SER REF

 AB52               CHRCD          AFTER  0325                                                           0301
AB 54 NEWYORK                   TAG NOT REQUIRED                                                         0303
AB 55 LOSANGELES                TAG NOT REQUIRED                                                         0304
AB 58 SWITCHON    SWT           TAG NOT REQUIRED                                                         0307
AC 01 WEEKLY             2      TAG NOT REQUIRED                                                         0312
AC 02 MONTHLY            4      TAG NOT REQUIRED                                                         0313
AD 32             RAD           SGN CHK                                                                  0505
AD 34 STEP2       ADCON         TAG NOT REQUIRED                                                         0507
```

```
INDEX    TAG        OP    NU AT    OPERAND     PROGRAM SAMPLE 05-22-62    HALTS           LOC  OP SU    ADDRESS    PGLIN SER REF

AC 21 OUT         HLT        9                 TO PROVIDE A TRANSFER POINT           000639 J     000009 0009 0324
```

```
INDEX    TAG        OP    NU AT    OPERAND     PROGRAM SAMPLE 05-22-62    TITLES          LOC  OP SU    ADDRESS    PGLIN SER REF

 AA 01              TITLE                THIS SAMPLE ASSEMBLY HAS BEEN PROVIDED  AS AN           T                0101
 AA 02                                   EXAMPLE OF THE ASSEMBLY DOCUMENTATION DESCRIBED         T                0102
 AA 03                                   IN CHAPTER 8 AND THE ASSEMBLING OF SOURCE               T                0103
 AA 04                                   PROGRAM STATEMENTS. THE OPERATORS NOTEBOOK              T                0104
 AA 05                                   AND THE SYMBOLIC ANALYZER, WHICH COMPRISE THE           T                0105
 AA 06                                   OPTIONAL DOCUMENTATION, APPEAR AFTER THE                T                0106
 AA 07                                   ASSEMBLY LISTING. TITLES AND COMMENTS INDICATE          T                0107
 AA 08                                   THE FUNCTIONS OF THE VARIOUS STATEMENTS. SINCE          T                0108
 AA 09                                   THIS ASSEMBLY IS NOT INTENDED TO ILLUSTRATE             T                0109
 AA 10                                   A PROGRAM, THE SOURCE STATEMENTS DO NOT FORM            T                0110
 AA 11                                   A ROUTINE.                                              T                0111
 AA 12                                                                                           T                0112
 AA 13                                   THIS PAGE CONTAINS THE INITIAL SOURCE                   T                0113
 AA 14                                   STATEMENT,A TITLE STATEMENT, AND IT WOULD               T                0114
 AA 15                                   NORMALLY APPEAR AFTER THE PAGE CONTAINING               T                0115
 AA 16                                   THE LITERAL TABLE. IT HAS BEEN TAKEN OUT                T                0116
 AA 17                                   OF SEQUENCE TO ILLUSTRATE TITLE AS A CONVENIENT         T                0117
 AA 18                                   MEANS OF SUPPLYING INTRODUCTORY COMMENTS TO             T                0118
 AA 19                                   PRECEDE A PROGRAM LISTING.                              T                0119
 AA 20                                                                                           T                0120
 AA 21                                   IN THE LISTING, THE HAND-CODED STATEMENTS ARE           T                0121
 AA 22                                   DISTINGUISHED FROM GENERATED STATEMENTS BY THE          T                0122
 AA 23                                   POSITION OF THE INDEX NUMBERS. THOSE OF THE             T                0123
 AA 24                                   HAND-CODED STATEMENTS ARE LEFT-JUSTIFIED,               T                0124
 AA 25                                   WHILE THOSE OF THE GENERATED STATEMENTS ARE             T                0125
 AA 26                                   INDENTED ONE POSITION. MACRO-HEADERS ARE                T                0201
 AA 27                                   DISTINGUISHED FROM THE GENERATED CODING                 T                0202
 AA 28                                   BY OVER-PRINTING. BECAUSE THIS ASSEMBLY IS              T                0203
 AA 29                                   SHORT AND CONSISTS MAINLY OF SOURCE STATEMENTS,         T                0204
 AA 30                                   A SEPARATE LISTING OF THE SOURCE PROGRAM                T                0205
 AA 31                                   DOES NOT APPEAR.                                        T                0206
 AA 32                                                                                           T                0207
 AA 33                                   THE SUBROUTINE THAT APPEARS AT THE END OF               T                0208
 AA 34                                   THE PROGRAM IS AN IBM SUBROUTINE. IT IS                 T                0209
 AA 35                                   AUTOMATICALLY PROVIDED ON EVERY ASSEMBLY.               T                0210
 AB 01 SECTION 1    TITLE          THIS SECTION CONTAINS THE DEFINITION OF A RECORD              T                0301
 AB 02                             AREA AND MACRO-HEADERS THAT REFERENCE INTERIOR                T                0302
 AB 03                             FIELDS OF THE AREA. THE GENERATED CODING FOR EACH             T                0303
 AB 04                             MACRO- INSTRUCTION CONTAINS LINKAGE TO CODING                 T                0304
 AB 05                             GENERATED FROM THE SUBROUTINE AT THE END OF THE               T                0305
 AB 06                             PROGRAM                                                       T                0306
 AB 47 SECTION 2    TITLE          THIS SECTION CONTAINS DEFINITIONS OF SWITCHES AND             T                0321
 AB 48                             MACRO-HEADERS THAT REFERENCE THE SWITCHES.                    T                0322
 AD 01 SECTION 3    TITLE          THIS SECTION CONTAINS AREA DEFINITIONS,ONE-FOR-ONE            T                0401
 AD 02                             INSTRUCTIONS,AND ADDRESS CONSTANTS. ASSUME THAT               T                0402
 AD 03                             ASU 01 IS SET TO ONE POSITION.                                T                0403
 AD 29              ENT80                                                                        T                0504
 AD 33              LEV80                                                                        T                0506
```

```
INDEX    TAG       OP    NU AT   OPERAND     PROGRAM SAMPLE 05-22-62    80 SPEC         LOC  OP SU    ADDRESS    PGLIN SER REF

AD 30              RAD        I,STEP1          SAME INSTRUCTION IN 7080 MODE               T                     0505
```

```
INDEX    TAG       OP    NU AT   OPERAND     PROGRAM SAMPLE 05-22-62    ASSGNS          LOC  OP SU    ADDRESS    PGLIN SER REF

AB 07              LASN       @500                                                  000500 L      001349       0307
```

```
INDEX    TAG       OP    NU AT   OPERAND     PROGRAM SAMPLE 05-22-62    SWITCHES        LOC  OP SU    ADDRESS    PGLIN SER REF

AB 56 SWITCHOFF   SWN         ROUTINE1         PROGRAM SWITCH THAT IS A                000589 A      000519 0519 0305   004AB19
AB 58 SWITCHON    SWT         ROUTINE2         PROGRAM SWITCH THAT IS A TRANSFER       000594 1      000529 0529 0307      AB24
 AE44 XXPRTST02   SWN         *&00025                                                  000964 A      000989 0989
 AE57 XXPRTST04   SWN         *&00020                                                  001029 A      001049 1049
 AE63 XXPRTST01   SWN         XXPRTST03                                         -00015 001059 A      001094 1094           AF02
 AE68 XXPRTST05   SWN         XXPRTST06                                                001084 A      000929 0929           AE37
 AF02 XXPRTST03   SWN         *&00045                                                  001109 A      001154 1154
```

```
INDEX    TAG       OP    NU AT   OPERAND     PROGRAM SAMPLE 05-22-62 LOCATION LENGTH   SYMBOLIC ANALYZER

AE22  SIGNED LITERAL   01   &
AE33  SIGNED LITERAL   01   A
AE37  SIGNED LITERAL   01   A
AE38  SIGNED LITERAL   01   A
AF08  SIGNED LITERAL   01   A
AE35  SIGNED LITERAL   02   0C
AE59  SIGNED LITERAL   02   0C
AB24  SIGNED LITERAL   03   12C
AE23  SIGNED LITERAL   04   000A
AF18  UNSIGNED LITERAL 01
AF07  UNSIGNED LITERAL 01   .
AE61  UNSIGNED LITERAL 01   ,
AE24  UNSIGNED LITERAL 01   @
AC09  UNSIGNED LITERAL 01   A
AC20  UNSIGNED LITERAL 01   1
AE71  UNSIGNED LITERAL 01   1
AF16  UNSIGNED LITERAL 01   1
AB20  UNSIGNED LITERAL 02   AB
AE31  UNSIGNED LITERAL 02   00
AE39  UNSIGNED LITERAL 02   00
AE41  UNSIGNED LITERAL 02   00
AE50  UNSIGNED LITERAL 02   00
AF20  UNSIGNED LITERAL 04   0002
AB25             LNG        @000002
AF27             SET    3   @000003
AE07             SET   15   @000006
AC21  OUT        HLT        @000009
AB07             LASN       @000500
AD41  STEP3      ACON5  &   @001000

AB53  CHICAGO                                                000580    00001
AC10             CMP   01   CHICAGO

AB62  DAILY                                                  000595    00001
AC18             SBZ    1   DAILY

AB11  FIELD1                                                 000501    00002
AB19  ROUTINE1   RCVS       FIELD1

AB12  FIELD2                                                 000506    00005
AB26             ADD        FIELD2

AB13  FIELD3                                                 000513    00007
AB31             ACON4      FIELD3

AB55  LOSANGELES                                             000580    00001

AC02  MONTHLY                                                000595    00001
```

```
INDEX    TAG        OP     NU AT   OPERAND     PROGRAM SAMPLE 05-22-62 LOCATION LENGTH   SYMBOLIC ANALYZER

AB54   NEWYORK                                                 000580    00001

AC13   NEXT¤00004                                              000624    00000
AC11              TRE            NEXT¤00004

AC21   OUT                                                     000639    00005
AC12              TR             OUT

AB16   RECORD                                                  000514    00015

AB15   RECORDEND                                               000514    00001
AB08   RECORD     NAME    0      RECORDEND

AB19   ROUTINE1                                                000519    00005
AB56   SWITCHOFF  SWN            ROUTINE1

AB24   ROUTINE2                                                000529    00005
AB58   SWITCHON   SWT            ROUTINE2

AD12   STEP1                                                   000679    00005
AD26              RAD            STEP1
AD31              EIA            STEP1
AD32              RAD            STEP1

AD34   STEP2                                                   000724    00005

AD41   STEP3                                                   000738    00005
AD48              RAD            STEP3

AB56   SWITCHOFF                                               000589    00005
AC19              RCVS           SWITCHOFF

AB58   SWITCHON                                                000594    00005

AD08   TAG1                                                    000654    00010
AD14              LOD     1      TAG1
AD16              LOD     1      TAG1         &
AD38              ACON4          TAG1

AD40              ACON4   3      TAG1
AD43              ACON5          TAG1
AD12   STEP1      RAD            TAG1
AD19              RCVS           TAG1
AD34   STEP2      ADCON          TAG1
AD45              ACON6          TAG1

AD09   TAG2                                                    000664    00010
AD50              LDA    15 H@TAG2                     ASU 00
AD57              LDA    15 H@TAG2                     ASU 02
AD21              TMTS    1      TAG2
AD24              RAD            TAG2         &

AC01   WEEKLY                                                  000595    00001

AF34   XAC                                                     001330    00005

AF38   XACC                                                    001339    00006

AF40   XACC1                                                   001349    00006

AE05   XXPRTST                                                 000769    00005
AE08              SND    15      XXPRTST      &
AB27              RCVS           XXPRTST      &
AB28              TSL            XXPRTST

AF26   XXPRTSTEND                                              001229    00005
AF12              TRZ     1      XXPRTSTEND

AF30   XXPRTSTHLD                                              001324    00055
AE17              LDA    14 R@XXPRTSTHLD                ASU 00
AE16              UNL            XXPRTSTHLD
AE20              UNL     2      XXPRTSTHLD-
AE21              UNL     3      XXPRTSTHLD-
AF26   XXPRTSTEND LOD     2      XXPRTSTHLD-
AF28              LOD     3      XXPRTSTHLD-
```

```
INDEX    TAG        OP    NU AT   OPERAND      PROGRAM SAMPLE 05-22-62 LOCATION LENGTH   SYMBOLIC ANALYZER

AF29   XXPRTSTWRK                                                         001269    00030
AE09              RCVS          XXPRTSTWRK-
AE05   XXPRTST    RCV           XXPRTSTWRK
AE18              SUB    14     XXPRTSTWRK-
AE25              RAD     2     XXPRTSTWRK-
AE26              RAD    15     XXPRTSTWRK-
AE27              CMP     1     XXPRTSTWRK-
AE30              TMTS    1     XXPRTSTWRK&
AE55              TMTS    1     XXPRTSTWRK&
AE58              TMTS    1     XXPRTSTWRK&
AE64              RAD     2     XXPRTSTWRK-
AE67              UNL     3     XXPRTSTWRK-
AE70              RAD     1     XXPRTSTWRK-
AF03              RAD     2     XXPRTSTWRK-
AF04              RAD    15     XXPRTSTWRK-
AF09              ST      1     XXPRTSTWRK-
AF11              RAD     1     XXPRTSTWRK-
AF14              ADCON 01      XXPRTSTWRK-

AE63   XXPRTST01                                                          001059    00005
AE10              SGN    01     XXPRTST01 -
AE11              ADM    01     XXPRTST01 -
AE40              TRE           XXPRTST01
AE54              TR            XXPRTST01 &
AE65              SGN     1     XXPRTST01 -

AE44   XXPRTST02                                                          000964    00005
AE49              TMTS    1     XXPRTST02 &
AE12              ADM    01     XXPRTST02 -
AE19              ULA    14     XXPRTST02 &
AE47              SGN     1     XXPRTST02 -

AF02   XXPRTST03                                                          001109    00005
AE13              ADM    01     XXPRTST03 -
AE63   XXPRTST01  SWN           XXPRTST03 -
AF05              SGN     1     XXPRTST03 -
AF10              TR            XXPRTST03

AE57   XXPRTST04                                                          001029    00005
AE14              ADM    01     XXPRTST04 -
AE42              TRE           XXPRTST04
AE48              SGN     1     XXPRTST04 -
AE66              SGN     1     XXPRTST04 -

AE68   XXPRTST05                                                          001084    00005
AE15              ADM    01     XXPRTST05 -
AE53              SGN     1     XXPRTST05 -

AE37   XXPRTST06                                                          000929    00005
AE52              TR            XXPRTST06


AE56              TR            XXPRTST06
AE60              TR            XXPRTST06 &
AE68   XXPRTST05  SWN           XXPRTST06
AE69              TR            XXPRTST06 &
AF06              TR            XXPRTST06 &
```

Reader's Comments

IBM 705/7080 DATA PROCESSING SYSTEM

Programming Systems Publications, Form C28-6224-1

FROM

Name ______________________________

Address ______________________________

Your comments regarding the completeness, clarity, and accuracy of this publication will help us improve future editions. Please check the appropriate items below, add your comments, and mail.

	YES	NO
Does this publication meet the needs of you and your staff?	___	___
Is this publication clearly written?	___	___
Is the material properly arranged?	___	___

If the answer to any of these questions is "NO," be sure to elaborate.

How can we improve this publication? Please answer below.

☐ Suggested Addition (Page , Timing Chart, Drawing, Procedure, etc.)

☐ Suggested Deletion (Page)

☐ Error (Page)

COMMENTS:

FOLD FOLD

FIRST CLASS
PERMIT NO. 116
KINGSTON, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
NEIGHBORHOOD ROAD
KINGSTON, N. Y. 12401

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPARTMENT 637

CUT ALONG LINE

OLD FOLD

Printed in U.S.A.

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601

C28-6224-1

Printed in U.S.A. C28-6224-1

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601