



## Systems Reference Library

### IBM 7080 Processor: Autocoder Language

This publication contains specifications for using Autocoder, the basic symbolic language of the 7080 Processor. The types of statements that constitute the Autocoder language include area definitions, switch definitions, one-for-one instructions, macro-instructions, address constants, and instructions to the Processor. All statement types, except macro-instructions, are described in detail. A general discussion of macro-instructions is included. However, the detailed specifications for using them are provided in the publication 7080 Processor: General Purpose Macro-Instructions, Form C28-6356.

The Introduction to this publication reviews some basic aspects of programming, such as symbolic programming systems and the IBM 7080 programming systems. Other features of the manual include descriptions of the following: The organization of the object program deck, the format of the object program card, and the standard and optional documentation produced during an Autocoder program assembly. An extensive sample assembly is also included to illustrate what the 7080 Processor produces. The assembly contains many examples of correct and incorrect language usage.

MINOR REVISION (September 1964)

This publication, Form C28-6263-2, is a minor revision of the previous edition, Form C28-6263-1, and incorporates the contents of the Technical Newsletter N28-1172. This revision does not obsolete either the previous edition or the Technical Newsletter.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

Address comments concerning the contents of this publication to:

IBM Corporation, Programming Systems Publications, Dept. 637, Neighborhood Road, Kingston, N. Y. 124

# CONTENTS

	Page		Page
INTRODUCTION . . . . .	5	Tag Operands . . . . .	38
Basic Aspects of Programming . . . . .	5	Literal Operands . . . . .	39
Symbolic Programming Systems . . . . .	6	Types of Lozenges . . . . .	39
The Symbolic Language . . . . .	6	Omitted Operands . . . . .	39
The Processor . . . . .	7	Importance of Properly Defined Data Fields . . . . .	39
The Basic 7080 Programming System . . . . .	7	Examples of Macro-Instructions and Their Use . . . . .	40
The 7080 Processor . . . . .	7		
Autocoder Language . . . . .	8	ADDRESS CONSTANTS . . . . .	42
Input/Output Control Systems for Use with		ADCON Address Constant . . . . .	43
Autocoder Programs . . . . .	10	ACON4 Address Constant . . . . .	43
Higher Languages of the 7080 Processor . . . . .	10	ACON5 Address Constant . . . . .	43
		ACON6 Address Constant . . . . .	44
		Address Constant Literal . . . . .	44
STANDARD FORMAT OF AUTOCODER			
STATEMENTS . . . . .	11	INSTRUCTIONS TO THE PROCESSOR . . . . .	46
Program Identification . . . . .	11	Instructions to the Processor-Standard Assembly	
Pglin . . . . .	11	Procedures . . . . .	46
Tag . . . . .	12	Location Assignment -- LASN . . . . .	46
Operation . . . . .	12	Special Assignment -- SASN . . . . .	48
Num . . . . .	12	Relative Assignment -- RASN . . . . .	48
Operand . . . . .	12	Assignment of Subroutines within Macro-	
Comments . . . . .	12	Instructions -- SUBRO . . . . .	49
Flag . . . . .	13	Assignment of Library Subroutines --	
		SUBOR . . . . .	49
AREA DEFINITIONS . . . . .	14	Assignment of Literals -- LITOR . . . . .	50
Definition of a Record -- RCD . . . . .	14	Transfer Card -- TCD . . . . .	50
Definition of a Constant Factor -- CON . . . . .	17	Instructions to the Processor-Object Program	
Definition of a Floating-Point		Content . . . . .	51
Number -- FPN . . . . .	18	Include Subroutine -- INCL . . . . .	51
Definition of a Report Format -- RPT . . . . .	19	Translation -- TRANS . . . . .	52
Definition of a Continuous Portion of		Source Program Language -- MODE . . . . .	53
Memory-NAME . . . . .	23	Coding Generated in 7080 Mode . . . . .	53
		Instructions to the Processor-Program Listing . . . . .	54
SWITCH DEFINITIONS . . . . .	26	Skip to New Page -- EJECT . . . . .	54
Data Switches . . . . .	26	Title for Routine or Comment -- TITLE . . . . .	54
Character Code -- CHRCD . . . . .	27	Instructions to the Processor-Multiple	
Bit Code -- BITCD . . . . .	27	Literal Tables . . . . .	55
Program Switches . . . . .	28	Literal Start -- LITST . . . . .	55
Switch Set to Transfer -- SWT . . . . .	29	Literal End -- LITND . . . . .	56
Switch Set to No Operation -- SWN . . . . .	29	Restrictions on Multiple Literals . . . . .	56
Console Switches . . . . .	29	Flag Characters and Their Meaning . . . . .	57
Alteration Switches -- ALTSW . . . . .	29		
ONE-FOR-ONE INSTRUCTIONS . . . . .	31	ASSEMBLY OUTPUT . . . . .	59
One-for-One Instruction Format . . . . .	31	Object Program Deck . . . . .	59
Basic Operands . . . . .	31	Standard Assembly Documentation . . . . .	59
Tag . . . . .	31	Program Listing . . . . .	59
Literal . . . . .	31	Optional Documentation . . . . .	60
Actual . . . . .	33	Operator's Notebook . . . . .	60
Location Counter . . . . .	34	Symbolic Analyzer . . . . .	60
Blank . . . . .	34	Details of the Program Listing . . . . .	60
Additions to Basic Operands . . . . .	34		
Character Adjustment . . . . .	34	APPENDIX . . . . .	62
Operand Modifier . . . . .	35		
Indirect Address . . . . .	35	SAMPLE ASSEMBLY . . . . .	64
Multiple Additions to a Basic Operand . . . . .	35		
GENERAL PURPOSE MACRO-INSTRUCTIONS . . . . .	37	GLOSSARY OF TERMS . . . . .	89
General Purpose Macro-Header Format . . . . .	38	INDEX . . . . .	90
Types of Macro-Header Operands . . . . .	38		



This manual contains detailed specifications for coding programs using Autocoder, the basic symbolic language of the 7080 Processor. All parts of the language except macro-instructions are fully described. A brief introduction to the general-purpose macro-instructions provided by IBM for 7080 users is provided in this publication; a full explanation may be found in the publication, 7080 Processor: General Purpose Macro-Instructions, Form C28-6356. Procedures for writing new macro-instructions for incorporation into the language are described in the publication, 7080 Processor: Preparation of Macro-Instructions, Form C28-6264.

Just as the Autocoder language described in this publication is the basic language of the 7080 Processor, so is Autocoder III the basic language of the predecessor system, the 7058 Processor. The over-all similarity of the two languages is such that this manual has been modeled after the manual describing Autocoder III. The major improvements that distinguish 7080 Autocoder from Autocoder III have been fully integrated into the following pages and may not be apparent, even to long-time users of Autocoder III. Despite this, no attempt has been made in the body of the manual to call attention to the differences; to do so might prove distracting, particularly to readers without a background in Autocoder III. However, significant differences between the two languages have been summarized in the Appendix for the convenience of experienced programmers who want a rapid survey of 7080 Autocoder in the light of their knowledge of Autocoder III. But it is expected that every programmer, before writing programs in 7080 Autocoder, will have become familiar with all sections of this manual.

The background discussion that follows assumes that the reader has had little programming experience. Readers already familiar with the IBM 7080 Data Processing System may wish to go directly to the body of the publication. Further information on the IBM 7080 may be found in the reference manual, IBM 7080 Data Processing System, Form A22-6560-2, and in 7080 Systems Summary, Form A22-6775. Other publications that may be of interest to 7080 users are abstracted in the publication, IBM 7080 Bibliography, Form A22-6774.

## BASIC ASPECTS OF PROGRAMMING

This explanation is written for the inexperienced programmer. The material is not detailed or comprehensive in scope. It is an outline of basic

program requirements, symbolic programming languages, and the program assembly process. These concepts are considered within the framework of the IBM 7080 Data Processing and Programming Systems.

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. The main function of a program is to process these records as specified, and these record-processing routines may be considered the body of the program. They are often called the main-line routines or the main-line coding. However, the program does not consist solely of these routines.

Any program must also 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 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, and locations for the storage area or areas that the records will occupy. Records are usually grouped in blocks; consequently an entire block enters storage at one time. 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 such a 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.

A program must also provide routines for detecting and handling error conditions resulting from input/output 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 assignment
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 is the second of two data processing applications. The first application is the processing or assembly 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 can be used in subsequent data processing applications until it is obsolete, or until it 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 that 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, which is a source-program statement that is eventually replaced by a sequence of machine instructions. 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 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 that 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 that 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 that will occupy it. If it is

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 that makes the reference. Usually, it is desirable to create symbolic addresses that 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 that 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 analyzes 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 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 each time. 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 combinations of 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 into the program. The instructions will not vary from program to program unless the subroutine itself contains macro-instructions. The programmer can 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 7080 PROGRAMMING SYSTEM

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

### The 7080 Processor

The 7080 Processor, hereafter called "the Processor," is a machine-language program that assembles programs written in Autocoder for the IBM 7080. The Processor operates on the 7080 when it is in 7080 mode. The Processor itself is so large that it must operate through a number of interrelated sections, or phases. Each phase is a program which performs one or more of the various assembly functions. The phase 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 statements from higher-language statements, 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. 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 various types of information about the program, including the following:

1. Programmed halts and halt loops
2. Titles of, and comments on, 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 Operator's 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 publication) means specifying the data to be operated on or specifying an instruction to be executed. For example, an Autocoder statement that calls for data movement to a work area references the data and the work area; a statement that 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 insert them in the library. However, the preparation of macro-instructions is a complicated procedure, requiring a thorough knowledge of Autocoder and the Processor.

### Autocoder Language

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

In defining 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 switches, program switches, and console switches. 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.

Referring again to the employee records used as an example in the section on area definitions, suppose 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 numerical characters. The manner in which each record is processed depends on the number of exemptions; the program therefore contains a number of processing routines. As each record is placed in the work area, the data switch becomes the character contained in the exemption field of the records. 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 that 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 the 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; and 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. These switches 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 that is run each week may include a routine that 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 that 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.

An illustration of macro-instruction scope is: 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 the excess positions 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 that lacks one, or replacing the reference made by an instruction. An instruction makes a reference by designating the symbolic address of a data field of another 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 that 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 address portion of the instruction 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 (i.e., 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 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 CONTROL SYSTEMS FOR USE WITH AUTOCODER PROGRAMS

Input/Output Control Systems (IOCS) have been developed for the IBM 7080. IOCS consists of a

group of routines that handle all input/output functions. IBM 7080 IOCS routines are made available to an Autocoder program when IOCS macro-instructions in the Processor library are used in the program.

Titles, form numbers, and abstracts of available publications dealing with 7080 IOCS may be found in the publication, IBM 7080 Bibliography, Form A22-6774.

#### HIGHER LANGUAGES OF THE 7080 PROCESSOR

As mentioned earlier, the 7080 Processor accepts program statements written in several higher languages. The languages are: Fortran; Report/File language; Decision language; Arithmetic language, and Table-Creating language. Various Processor phases translate each of these statements into one or more Autocoder statements.

FORTRAN is the name for FORMula TRANslation language. As the name implies, complex problems can be stated in the form of mathematical formulas, using FORTRAN. Both fixed point and floating point calculations are possible.

Report/File language is a set of statements that 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 can be used to request a logical decision to be made on the basis of a test of the various conditions supplied in the statement.

Arithmetic language can be used to request that a series of mathematical computations be performed on the elements supplied in the statement.

Table-Creating language can be used to describe tables suitable for data-searching, along with the associated table entries.

Titles, form numbers, and abstracts of publications dealing with the higher languages of the 7080 Processor may be found in, IBM 7080 Bibliography, Form A22-6774.

Note: Throughout this publication, the field headed "SEQUENCE (PGLIN)" will be referred to as the pglin field, the field headed "NAME (TAG)" will be referred to as the tag field.

**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 statement.

PGLIN (COLUMNS 1-5)

The sequence of the coding sheets and the statements on the coding sheets is designated by the five-position entry in these columns. Columns 1 and 2 designate a two-position page number that is used to determine the sequence of the coding sheets.

\* A standard card form, IBM electro N14106, is available for punching source statements from this coding form.

**Figure 1. IBM 7080 Processor Coding Form**

Any alphameric character may be used in the entry. Normally, however, special characters are not used. The IBM 7080 collating sequence, shown in Figure 2, is used to determine the order of the pages.

Columns 3-5 designate a three-position line number that is used to determine the sequence of the statements on the coding sheets. Any alphameric character may be used in these positions, although special characters are not normally used. Ordering should be done according to the 7080 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 that 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. In the case of three-lines inserted 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 that represents the actual location of a data field or an instruction. The field is filled in starting in column 6. When an Autocoder 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 characters may be alphabetic, numerical, and blanks. A tag may not contain special characters. If composed of numerical 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 numerical tags not be used. It is best to create tags that describe the data fields or the instructions to which they are assigned. Tags should not be assigned unless they are referenced by program statements; unnecessary tags slow the assembly process and produce needless

messages. To avoid confusion and possible improper macro generation, it is strongly recommended that no tag begin with either of the following three-letter prefixes: CSF, IBM.

#### OPERATION (COLUMNS 16-20)

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

#### NUM (COLUMNS 21-22)

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

#### OPERAND (COLUMNS 23-39)

The use of the operand field varies according to the type of Autocoder 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 40-73)

Additional information about an Autocoder 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 can begin before or after column 40. 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 accompany a macro-instruction, 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 the pgin and comments fields.

If the comments do not accompany a macro-instruction, they do not have to be separated from

Blank . □ & \$ \* - / , % # @ 0 A through I 0 J through R ≠ S through Z 0 through 9

Figure 2. IBM 7080 Collating Sequence

the operand by blank spaces, and comments continuation lines may contain entries in any columns except 16 (first position of the operation field) and 21 - 22 (numerical field). However, to make the comments easier to read, it is recommended that the continuation lines be restricted to entries in the pglin and comments fields.

#### FLAG (COLUMN 74)

Characters written in this column are used for communicating with the Processor. The types of characters that may be placed in this column (and an explanation of their meanings) are described in the section "Instructions to the Processor."

## 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 that 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 tag in the operand of the Auto-coder 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 17-20 of the operation field as an expansion of the numerical field. (The operation field is assumed to be blank if column 16 is blank.)

**NUMERICAL FIELD:** The size of the data field is entered here. A one-digit entry is placed in column 22; it need not be preceded by a zero. When the operation field contains the RCD code, the numerical 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 17-20 of the operation field may be used as an expansion of the numerical field. Under these conditions, in effect, the numerical field consists of six 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 numerical fields containing integers only.
2. A description of an integer and decimal format. This is used to define numerical 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.

### Alphameric Fields and Numerical Fields of Integers Only:

<u>Code</u>	<u>Contents of Field</u>
+	Signed numerical data consisting of integers.
N	Unsigned numerical data consisting of integers.
F	Signed numerical 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-point constant appears in storage.
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.

Left protection should be provided when the subsequent field contains signed numerical data. The low-order position of the field providing left protection must be occupied by one of the following: an alphabetic character, a signed numerical 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.

Numerical 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. (The first method is the preferred use.)

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28 30 32 33 35 37 38 39	
UN.SIGNED	R.C.D.	8N	
ALPHA FIELD		125A+	
SIGNED		13+	
FLOAT		10F	
		1200A	

Figure 3

1. Enumerating the number of integer and decimal positions. Signed numerical fields are represented as #+xx.yy, and unsigned numerical 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 numerical digit is preceded by a zero. The sum of xx and yy must equal the entry in the numerical field. The maximum size data field that 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 signed numerical fields are being defined, a plus sign is placed in the first position of the operand, and is followed by the layout. The operand defining an unsigned numerical field starts with the layout itself. A blank position is not used to indicate unsigned numerical data.

The total number of Xs must equal the entry in the numerical field. Although both the decimal point and the sign occupy positions in the layout, neither is included in the count for the numerical field entry. Neither the point nor the sign exists 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 numerical fields would be defined by each of these methods. Note that SIGNED3 is too large to be defined by a layout.

Indicating the Position of Record Marks and/or Group Marks: This information should be supplied if the record that contains such characters is referenced by a macro-instruction. The position or positions the characters occupy must be defined

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28 30 32 33 35 37 38 39	
SIGNED1	R.C.D.	8#+0.5.03	
SIGNED2	R.C.D.	8+XXXXX.XXX	
		3	
UN.SIGNED1	R.C.D.	12#+1.1.01	
UN.SIGNED2	R.C.D.	12XXXXXXXXXX.X	
		3	
SIGNED2	R.C.D.	13#+0.13	
SIGNED2	R.C.D.	13+XXXXXXXXXXXXX	
		3	
UN.SIGNED2	R.C.D.	2#+0.02	
UN.SIGNED2	R.C.D.	2.XX	
		3	
SIGNED3	R.C.D.	73#+4.7.26	

Figure 4

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 portion of the record that contains the characters. The operand 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28 30 32 33 35 37 38 39	
FIRSTWAY	R.C.D.	31A	
		4	#
		3	
SECONDWAY	R.C.D.	34A	
		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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
NEWWAY	RCD	31A	
		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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
FIRSTPART	RCD	30A+	
GROUPMARK		4###	
SECONDPART		56A+	

Figure 7

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

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
FIRSTWAY	RCD	21A	
		4#	
		3	
SECONDWAY	RCD	24A	
		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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
FIRSTWAY	RCD	21A	
		4##	
		3	
SECONDWAY	RCD	23A	
		2##	

Figure 9

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

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
FIRSTWAY	RCD	21A	
		4###	

Figure 10

If a record of less than 51 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39 40 42			
MARKS ONLY	RCD	20	
			15#

Figure 11

**COMMENTS FIELD:** Comments may be started here. If comments continuation lines are written, columns 16, 21, and 22 of the continuation lines must be blank. If the statement following the last continuation line is blank in column 16 (but is not blank in columns 21 and 22), the Processor assumes that the line 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. The mnemonic code RCD may be omitted in this second statement. Columns 17-20 of the operation field may then be used as an extension of the numerical field. No space will be reserved for an RCD of zero length.

#### Restrictions on an RCD Statement

The size of a data field may not exceed mode memory size minus one. If a single RCD field specifies a larger field size, the Processor will assume a length of one for location and address assignment. (The macro generator will use the actual size specified unless it is greater than 159999. In that case, a size of one will be assumed.)



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 that 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 that is blank in column 16 of the operation field is a CON, and treats it accordingly. This assumption makes it possible in subsequent statements to use columns 17-20 of the operation field as an expansion of the numerical field.

**NUMERICAL 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 numerical field is limited to two positions. However, when the operation field is blank and the statement has been preceded by another CON statement, columns 17-20 of the operation field may be used as an expansion of the numerical field. Under these conditions, in effect, the numerical field consists of six 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 numerical 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 numerical field entry specifies fewer positions than those contained in the operand.
2. Supply low-order zeros or blanks when the numerical 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 numerical field for TAG2 indicates that the constant contains nine low-order blanks.

**Defining a Numerical Constant:** A constant consisting of signed numerical data must contain a

Name (Tag)						Operation			Num		Operand										
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39
TAG1						CON			5		A B C D E										
TAG2									20		T H E D A T E I S										
TAG3									4		A 3 7 2										

Figure 12

plus sign or a 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 numerical field entry. A signed numerical constant that exceeds 99 integer or 99 decimal positions should not be referenced by a general-purpose macro-instruction.

Unsigned numerical data consisting of integers only is written starting in column 23 of the operand field. Unsigned numerical 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. If this is done, the data 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.

Name (Tag)						Operation			Num			Operand											
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39		
TAG1						CON			4			+			75			.25					
TAG2									3			8			45								
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.

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.

Name (Tag)					Operation		Num		Operand												
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39
						R,C,D			3	3	A										
CONSTANT						CON			2		#										

Figure 14

COMMENTS FIELD: Comments may be started here. If comments continuation lines are written, columns 16, 21, and 22 must be blank. If the statement following the last continuation line is blank in column 16 (but is not blank in columns 21 and 22), the Processor assumes that the line is another CON statement.

Name (Tag)					Operation			Num		Operand											
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39
						R,C,D			4	2	A										
C.O.N.S.T.A.N.T.						C.O.N.			3	#	#	#									

Figure 15

#### Restrictions on a CON Statement

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 sign or a minus sign followed by numerical characters is desired, a one-position CON statement should be used to define the sign; another CON should be used to define the numerical characters as an unsigned numerical constant.

The size of a CON statement may not exceed mode memory size minus one. If a single CON field specifies a larger field size, the Processor will assume a length of one for location and address assignment. (The macro generator will use the actual size specified unless it is greater than 159999. In that case, a size of one will be assumed.)

#### DEFINITION OF A FLOATING POINT NUMBER -- FPN

The function of an FPN statement is to define a data field for constant numerical data and to provide the data in floating-point form. Numerical 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 that is blank in column 16 of the operation field is an FPN statement and treats it accordingly.

NUMERICAL FIELD: This field is left blank. The Processor assumes ten positions.

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

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. A plus sign indicates that the decimal has been moved to the left; the minus sign indicates that the decimal has been moved to the right.

As indicated by DDDDDDD, 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. Comments continuation lines are not allowed. Any continuation line following an FPN is assumed to be another FPN.

#### Restrictions on an FPN Statement

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

Fixed Point Form

Name (Tag)						Operation		Num		Operand											
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39
1.																					

Figure 16

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 numerical 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 numerical 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-instructions that place the numerical 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 numerical 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 that is blank in column 16 of the operation field is an RPT statement and treats it accordingly.

**NUMERICAL 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 numerical data and any commas, decimal points, dollar signs, or any 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 numerical 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 that provide the macro-instructions with additional information about the desired print format.

Three sets of data will be used as examples throughout this section to explain the method of laying out the format. 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 Numerical Characters, Commas, Decimal Point:** Xs and Zs are used to indicate the position of each numerical character in the format. If commas and/or a decimal point are desired, the symbols for them are placed in the appropriate positions. The numerical 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 by an X. If the position is defined by a Z, the zero will be retained and printed. If the position 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
MIXED	RPT	8X,XXX	ZZ
DECIMAL		4,ZZZ	
INTEGER		5XXXXX	

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 that 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
MIXED	RPT	8X,XXZ	ZZ
INTEGER		5XZZZ	

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. It is requested by an indicator, which is placed immediately after the format layout. The indicator consists of a lozenge, an asterisk, and a lozenge ( $\square * \square$ ); it is not included in the count for the numerical column. 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
INTEGER	RPT	5XXXXXX	
MIXED		8X,XXX	ZZXX

Figure 19

The position of the decimal point can be indicated to macro-instructions that handle numerical 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 numerical field. This may be seen in Figure 20.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
MIXED	RPT	7X,XXX	DZZ
DECIMAL		3DZZZ	

Figure 20

**Indicating the Position and Treatment of Dollar Signs:** If the dollar sign is desired in the printed report, it is written to the left of the high-order position of the format layout and is included in the count for the numerical 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 ( $\square$ ), and are not included in the count for the numerical 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 ( $\square * \square$ ). 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 numerical character in each set of data. It is requested by an indicator consisting of a lozenge, a dollar sign, and a lozenge ( $\square \$ \square$ ) 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:

1. A floating dollar sign.
2. A fixed dollar sign with asterisk protection.
3. A fixed dollar sign without asterisk protection and with leading zeros converted to blanks.
4. A fixed dollar sign without asterisk protection and with up to three leading zeros retained.
5. No dollar sign but asterisk protection.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
MIXED1	RPT	9	X,XXX,ZZ,X
MIXED2		9	X,XXX,ZZ,X
MIXED3		9	X,XXX,ZZ
MIXED4		9	X,ZZZ,ZZ
MIXED5		8	X,XXX,ZZ,X

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

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

Depending on which set of characters is requested, one or two blank positions must be added to the low-order portion of the format layout. These blank positions must be included in the count for the numerical field entry, and 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 that 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 that 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. If a field sign is being requested, 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:

$\square X_1 \square X_2 \square X_3 \square X_4 \square$

where

- $X_1$  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 may be used for either. 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 numerical field entry.

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

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22 23	25 27 28 30 32 33 35 37 38 39 40	
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
032570	\$325.70-
000000	\$ .00*
457638	\$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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22 23	25 27 28 30 32 33 35 37 38 39 40 42 43	
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
003955	\$\$\$39.55CR
000000	\$****.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 numerical 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22 23	25 27 28 30 32 33 35 37 38 39	
DECIMAL	RPT	5DZZZ	#####

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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22 23	25 27 28 30 32 33 35 37 38 39	
INTEGER1	RPT	4ZZZZZ	

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
00278	00278-
00000	00000
34628	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 converted to blanks. This includes the dollar sign, comma, and decimal-point positions.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22 23	25 27 28 30 32 33 35 37 38 39 40	
INTEGER1	RPT	9\$X,X,XXX	\$\$\$###

Figure 26

**Blank-If-Zero Option:** If this is requested, any defined commas, the decimal point, and floating or fixed dollar signs will be converted to blanks along with the numeric positions when the field contains all zeros. 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.

Name (Tag)					Operation					Num		Operand																		
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39	40								
M I X E D 1									R P T	7	X	X	X	X	Z	Z	H	H	H	B	Z	H								
M I X E D 2									R P T	1	0	\$	X		X	X	X	Z	Z	H	X	H								

Figure 27

**COMMENTS FIELD:** Comments may be started here. If comments continuation lines are written, columns 16, 21, and 22 must be blank. If the statement following the last continuation line is blank in column 16 (but is not blank in columns 21 and 22), the Processor assumes that the line is another RPT statement.

#### Restrictions on an RPT Statement

The format layout of an RPT operand may not exceed five positions. One-position 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 numerical 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 should not exceed nine.

#### DEFINITION OF A CONTINUOUS PORTION OF MEMORY - NAME

A NAME 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 statements define fields that may be enclosed by a NAME statement:

1. Area definitions:  
RCD, CON, FPN, RPT, NAME
2. Switch definitions:  
CHRCO, BITCO
3. Address constants:  
ACON4, ACON5, ACON6, ACON

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 that 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. 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. If a subsequent entry to the NAME contains a blank in columns 16, 21, and 22, the entry is assumed to be another NAME statement.

**NUMERICAL FIELD:** This field is left blank if the 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
0 or 5	0 or 5
1 or 6	1 or 6
2 or 7	2 or 7
3 or 8	3 or 8
4 or 9	4 or 9
A	0
B	00
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 generator. (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 that 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28	30 32 33 35 37 38 39
STARTNAME	NAME	ENDNAME	
FIELD1	RCD	4N	
FIELD2		125A+	
FIELD3		5# + 0.3 0.2	
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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28	30 32 33 35 37 38 39
	NAME	2	
ANYTAG	RPT	7# 2.2.2 2.2	

Figure 29

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

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28	30 32 33 35 37 38 39
BEGIN	NAME	END	
FIELD1	FPN	+ 0.3 + 4 3.8	
END		+ 0.2 + 6 7.8 4.5	

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 (12345).

**COMMENTS FIELD:** Comments may be started here. Comments continuation lines are not allowed.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28	30 32 33 35 37 38 39
ALPHA	NAME	ENDALPHA	
	CON	1+	
ENDALPHA		5 1.2.3.4.5	

Figure 31

### 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 that terminates it. Internal NAMES may be terminated by the same field that terminates the outer NAME, or they may be terminated by fields that 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28	30 32 33 35 37 38 39
OUTERNAME	NAME	ENDOUTER	
FIELD1	RCD	5+	
FIELD2		150A+	
INNERNAME	NAME	ENDINNER	
FIELD3	RCD	12A+	
FIELD4		7# + 0.4 0.3	
ENDINNER		1#	
FIELD5	RPT	10\$X.X, XX.X 2.2 XX.X	
FIELD6	RCD	35A	
ENDOUTER	CON	5###	

Figure 32

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





## SWITCH DEFINITIONS

Switches are programming or hardware devices that are 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 a switch definition statement 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 that 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 that indicates whether a character code or a 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 that 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, a 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; it 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 the code is present, the switch is ON. If the switch contains anything other than the designated code, the switch is OFF.

A bit-code switch is set ON by setting the designated bits ON; it 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 the designated bit is ON, the switch is ON, otherwise, the switch is OFF.

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

Suppose, in a similar example, that the switch is a bit-code switch. Let the Surplus condition be represented by turning ON the 1-bit, and let the Deficit condition be represented by turning ON the 2-bit. In this case, 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, for the switch to be ON for both the conditions of Surplus and Deficit.

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
T <sub>1</sub>	CHRCD	n	X <sub>1</sub>
T <sub>2</sub>			C <sub>1</sub>
T <sub>3</sub>			C <sub>2</sub>
etc.			C <sub>3</sub> etc.

- n is blank when defining a one-position switch, or is 2 when defining a two-position switch.
- X<sub>1</sub> is the initial contents of the switch, or is blank.
- T<sub>1</sub>, T<sub>2</sub>, T<sub>3</sub>, ... are the tags of the codes. They specify the conditions the codes represent.
- C<sub>1</sub>, C<sub>2</sub>, C<sub>3</sub>, ... 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 numerical 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. Note that the switch is enclosed by a NAME statement. The NAME operand indicates that the statement tagged CANCELED terminates the NAME.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
RECORD AREA NAME			CANCELED
COMPANY	RCD	25A	
	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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
	CHRCD	218	
NEW YORK			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 that reference the switch.

Tag	Operation	Num	Operand	LOC	INSTR	SU	ADDRESS
	CHRCD			000343			
BLUE			A				
GREEN			B				
RED			C				
Instructions that reference the switch:							
	CMP	1	GREEN	002129	403U3	01	000343
	{						
	CMP	1	RED	002624	403U3	01	000343
	{						
	CMP	1	BLUE	002679	403U3	01	000343

Figure 36

## Restrictions on a CHRCD Switch

A code should be represented not as a signed numerical character but as the alphabetic character equivalent to the signed numerical 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 that 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
T <sub>1</sub> T <sub>2</sub> T <sub>3</sub> T <sub>4</sub>	BITCD	B <sub>1</sub> B <sub>2</sub> B <sub>3</sub> B <sub>4</sub>	X <sub>1</sub>

- X<sub>1</sub> is the initial setting of the switch, or is blank.
- T<sub>1</sub>... T<sub>4</sub> are the tags of the bits. They specify the conditions that the bits represent when they are ON.
- B<sub>1</sub>... B<sub>4</sub> 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-bit, 4-bit, and A-bit 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
RECORD AREA NAME			OTHER
EMPLOYEE	RCD	25A+	
	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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 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 that 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 that reference the switch.

Tag	Operation	Num	Operand	LOC	INSTR	SU	ADDRESS
EAST	BITCD	1		000237			
WEST		2					
NORTH		4					
Instructions that reference the switch:							
	RCVS		EAST	002319	U0237		000237
	RCVS		WEST	002464	U0237		000237
	RCVS		NORTH	002739	U0237		000237

Figure 39

### Restrictions on a BITCD Switch

A bit-code switch may not be used in a program for 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 that 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 that will be ON initially. The format of the SWT statement is as follows:

Tag	Operation	Num	Operand
T <sub>1</sub>	SWT		X <sub>1</sub>

T<sub>1</sub> is the tag of the switch.  
X<sub>1</sub> 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 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 <sub>1</sub>	SWN		X <sub>1</sub>

T<sub>1</sub> is the tag of the switch.  
X<sub>1</sub> 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 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 <sub>1</sub>	ALTSW	X <sub>1</sub>	

T<sub>1</sub> is the tag of the switch statement.  
X<sub>1</sub> 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22	23 25 27 28 30 32 33 35 37 38 39
WEEKLY RUN	ALTSW	A	
MONTHLY RUN	ALTSW	B	

Figure 42

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

<u>Prefix</u>	<u>Suffix</u>
operand modifier	character adjustment
indirect address	

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

The details of each 7080 operation are supplied in the reference manual, IBM 7080 Data Processing System, Form A22-6560.

## ONE-FOR-ONE INSTRUCTION FORMAT

Like other Autocoder 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 numerical 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 operands follows:

### Tag

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

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22	23 25 27 28	30 32 33 35 37 38 39
F.I.E.L.D.	R.C.D.	1,2,3,4,5,6	
	?		
I.N.S.T.R.	R.A.D.	F.I.E.L.D.	

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.

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 numerical constant is desired, the first character following the literal sign must be a plus sign or a minus sign. In storage, the low-order position of the constant will be signed. If the numerical data is a mixed or pure decimal, the decimal point will not appear in the constant. If an unsigned numerical constant is desired, the first character following the literal sign must be the first character of the numerical 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.

Name of Instruction	Mnemonic Code	Use In Programs For		
		Second'y Mode		7080
		7051I	7051II	
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 <sup>2</sup>			x
Control Write (Write 04)	CWR <sup>2</sup>			x
Divide	DIV	x	x	x
Dump Memory (Write 01)	DMP <sup>2</sup>	x	x	x
Enable Compare Backward	ECB			x
Enable Indirect Address	EIA			x
Enter Interrupt Mode	EIM			x
Enter 7080 Mode	EEM			x
Forward Space (Read 01)	FSP <sup>2</sup>	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 <sup>3</sup>			x
Load Storage 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 <sup>2</sup>	x	x	x
Read 02 (Read Memory Address)	RMA <sup>2</sup>		x	x
Read 03 (Sense Status Trigger)	SST <sup>2</sup>			x
Read 04 (Control Read)	CRD <sup>2</sup>			x
Read 05 (Read Memory Block)	RMB <sup>2</sup>			x
Read Memory Address (Read 02)	RMA <sup>2</sup>		x	x
Read Memory Block (Read 05)	RMB <sup>2</sup>			x
Read While Writing	RWW	x	x	x
Receive	RCV <sup>4</sup>	x	x	x
Receive Serial	RCVS <sup>4</sup>	x	x	x
Receive Ten Characters	RCVT <sup>4</sup>			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 <sup>2</sup>			x
Set Bit Alternate	SBA		x	x
Set Bit 1	SBN <sup>1</sup>		x	x
Set Bit Redundant	SBR		x	x
Set Bit 0	SBZ <sup>1</sup>		x	x
Set Control Condition (Write 03)	SCC <sup>2</sup>			x
Set Density High	SDH			x
Set Density Low	SDL			x
Set Left	SET	x	x	x
Set Record Counter (Write 02)	SRC <sup>2</sup>		x	x
Set Starting Point Counter	SPC			x
Shorten	SHR	x	x	x
Sign	SGN	x	x	x
Skip Tape	SKP		x	x

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

Name of Instruction	Mnemonic Code	Use In Programs For		
		Second'y Mode		7080
		7051I	7051II	
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 Transmilt	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	TNS			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 <sup>1</sup>		x	x
Transmilt	TMT	x	x	x
Transmilt 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 <sup>3</sup>			x
Unload Storage Bank	USB			x
Write 00	WR	x	x	x
Write 01 (Dump Memory)	DMP <sup>2</sup>	x	x	x
Write 02 (Set Record Counter)	SRC <sup>2</sup>		x	x
Write 03 (Set Control Condition)	SCC <sup>2</sup>			x
Write 04 (Control Write)	CWR <sup>2</sup>			x
Write 05 (Write Multiple Control)	WMC <sup>2</sup>			x
Write and Erase 00	WRE	x	x	x
Write and Erase 01	WRE 01	x	x	x
Write Multiple Control (Write 05)	WMC <sup>2</sup>			x
Write Tape Mark	WTM	x	x	x
<u>IBM 760 Operations</u>				
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
<u>IBM 777 Operations</u>				
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
See NOTES below.				

Figure 44. Mnemonic Codes for One-for-One Instructions.



## NOTES

- <sup>1</sup> Place a 1, 2, 4, 8, A, or B in column 22 to designate the bit (TZB can also have a C). If column 21 is not blank, the Processor assumes that ASU zoning, valid or invalid, has been designated.
- <sup>2</sup> Preferred mnemonics; RD 01 to 05 and WR 01 to 05 are also acceptable.
- <sup>3</sup> A blank or a 4 should be placed in column 22 if the Processor is to perform a 4/9 check. If a 1, 2, 3, or 5 is written, a 1/6, 2/7, 3/8, or 0/5 check, respectively, results.
- <sup>4</sup> The three different Autocoder mnemonics for the receive instruction (RCVS, RCV, and RCVT) indicate to the Processor the type of address to be assigned. If the mnemonic is RCVS, the location assigned is the high-order address of the field specified in the operand of the instruction. For an RCV, four 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 a 5. An RCVT is assigned the high-order address of the field plus nine. Since RCVT is used when a 9 ending is desired (i.e., with a TCT), the high-order address of the field should end in a 0.

If the generated address does not end in a 4 or a 9 (RCV) or 9 (RCVT), a 4/9 check or a 9 check message is prepared.

An example of assembled machine-language coding for the three forms of the receive instruction is shown below. The field tagged WORKAREA, has a high-order address of 3750. Note that the machine-language operation code (U) is the same for all three statements:

OP	Operand	Op	Address
RCVS	WORKAREA	U	3750
RCV	WORKAREA	U	3754
RCVT	WORKAREA	U	3759

The operands of all forms of the Receive instruction can be character adjusted. Thus, if the operands above were WORKAREA-3, the actual addresses would be three less than shown.

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 0370000000.

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.

If the literal is used with a TCT instruction, its length must be a multiple of ten with a record mark in the low-order position. The Processor will properly position the literal in a 9 location.

The Processor places all constants that it creates from literal operands in storage areas called literal tables. Literal constants may be placed either in the main literal table or in

Name (Tag)					Operation		Num		Operand												
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39
ONE						RAD					#,50,34,.27,#										
						{															
TWO						LOD					#,79,8#										
						{															
THREE						TMT					#,LOAD TAPE##										

Figure 45

multiple literal tables (see "Multiple Literal Tables.") A literal appears only once in a literal table, even when it has been used in several different statements.

The Processor classifies literals and makes literal-table assignments according to whether the literals are signed or unsigned:

1. Any literal containing a sign in the first position is automatically classified as signed. If the signed literal supplies numerical data, it appears in storage as previously described. If the literal contains a non-numerical character in the low-order position, the existing zoning in that character is replaced by the sign.
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. However, the flag character B can be used to allow literal symbols in any literal position (see "Flag Characters and Their Meanings").

## Actual

An actual operand is a set of numerical 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 an operand containing less than five numerical characters is used with one of the following operations: BLM, BLMS, CTL, HLT, LIP, LNG, RND, SEL, SET, SHR, SPC, SRC, TRANS. Note in Figure 46 that the SET and BLM instructions have been written two ways.

## Restrictions on an Actual Operand

An actual operand greater than the core-storage size specified to the Processor should not be used.

Name (Tag)	Operation	Num	Operand
ONE	ST	9995	
TWO	SET	000005	
THREE	SET	5	
FOUR	BLM	000020	
FIVE	BLM	20	

Figure 46

If such an operand is encountered during assembly, the Processor subtracts the maximum core-storage size from the actual and uses the difference as the operand. A message to this effect is provided at assembly time.

For example, if an 80,000 core-storage size has been specified, any actual operand in excess of 79999 will have 80000 subtracted from it; the remainder will be used as the operand. The list below indicates the largest actual operand that may be used with each available core-storage size:

Core-Storage Size	Maximum Actual Operand
20,000	19999
40,000	39999
80,000	79999
160,000	159999

### 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.

Name (Tag)	Operation	Num	Operand
	LOC	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 that has blanks in the first ten 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 an appropriate address.

Name (Tag)	Operation	Num	Operand
	BSP		
	ULLA	14	

Figure 48

### ADDITIONS TO BASIC OPERANDS

A description of the suffix and the prefixes that may be added to an Autocoder 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 numerical 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.

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.

Name (Tag)					Operation					Num					Operand																						
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39																
EMPLOYEE					RCD					250+																											
					{					5+																											
					RCD					EMPLOYEE+5																											

Figure 49

### Restrictions on Character Adjustment

The numerical portion of a character adjustment cannot exceed six positions, and its absolute value cannot be greater than 159999. If it is greater, 160000 will be subtracted until the absolute value is less than 160,000. If the numeric portion of the adjustment is less than six positions, the position immediately following must be non-numerical.

Name (Tag)					Operation					Num					Operand																						
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39																
					TRP					X+10																											
					APP					X+30#																											
					S.T.					FIELD																											

Figure 50

Further restrictions apply to operands that are a location counter, an actual, or a literal. These operands can use only the + or - operators. If any other operator is used, both the operator and the adjustment will be ignored.

Literal operands, in addition to being restricted to a + or a - operator, cannot have an adjustment value of more than 99. If the adjustment is more than 99, the Processor will use the two low-order digits for the adjustment value. Thus, an adjustment of -156 will be treated as if it were -56.

### Operand Modifier

An operand modifier is a two-character prefix that 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
T,	High-speed nine position

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.

Name (Tag)					Operation					Num		Operand												
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39			
FIELD						RCD				8N														
						{																		
						LOD				32	FIELD													

Figure 51

NOTE: If the modifier "S," has been used in Figure 51, the LOD instruction would reference the contents of 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.

Name (Tag)					Operation			Num		Operand														
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	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-mode portion of the program, it would have been followed by the generated instructions as shown in Figure 53.

### MULTIPLE ADDITIONS TO A BASIC OPERAND

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

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

Figure 53

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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22	23 25 27 28 30 32 33 35 37 38 39	
FIELD	RCD	IOA	
	LOD	L, FIELD+1	

Figure 54

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

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22	23 25 27 28 30 32 33 35 37 38 39	
ONE	RAD	RECORD 1	
	TR	I, X+10	
TWO	RAD	RECORD 2	
	TR	COMPUTE	

Figure 55

A macro-instruction is a source-program statement which represents multiple operations. When the program is assembled, each macro-instruction is replaced by a sequence 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 7080 Processor library are shown in Figure 56. The purpose of this chapter is to present them as a part of the Autocoder 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 publication 7080 Processor: General Purpose Macro-Instructions, Form C28-6356. Procedures for writing new macro-instructions for incorporation into the language are described in the publication, 7080 Processor: Preparation of Macro-Instructions, Form C28-6264. Input/output macro-instructions are a part of the 7080 Input/Output Control System, and are described in the 7080 IOCS publications. The titles, form numbers, and abstracts of references to all publications dealing with macro-instructions for the IBM 7080 may be found in IBM 7080 Bibliography, Form A22-6774.

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 the chapter "One-for-One Instructions." Programmers writing their own macro-instructions may also designate an entry in the numerical 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 IBM publication on the preparation of macro-instruction for the 7080 Processor.

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 Interrupt Program	(ENTIP)
Leave Interrupt Program	(LEVIP)
Leave 80 Mode	(LEV80)
Enter 80 Mode	(ENT80)
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)
Define CASU	(CASU)
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	
Dead-End Halt	(STOP)
Link to Subroutine	(LINK)
Transfer Indirect	(TRIN)
Type a Message	(TYPE)

Figure 56. 7080 Processor General-Purpose Macro-Instructions for Use in Autocoder Programs.

The remainder of this chapter is an introduction to the general-purpose macro-instructions in the

7080 Processor library. The discussion is based on the conventions and restrictions that apply to these macro-instructions.

#### GENERAL-PURPOSE MACRO-HEADER FORMAT

The portion of a macro-instruction that is written as a source-program statement is called a macro-header. Like other Autocoder statements, a macro-header is tagged if it is to be referenced. The mnemonic code is placed in the operation field. Entries in the numerical field are rarely permitted. Those entries 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 fifty 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 that 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 49 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 numerical 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 MACRO-HEADER 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 one-for-one instructions, macro-instructions, and address constants. Other

tags that may be used as operands are those of Class A subroutine items and generated descriptive tags. 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 of 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. That is, 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 to those previously defined for the field, or it may supply a different set of characteristics. A secondary definition must be used in a macro-header operand that references a data field indirectly, because the defined characteristics of the data field are not available to the Processor in such a situation.

The macro-header operand containing the definition is written in this order: the tag of the data field, a comma, the secondary definition. 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.

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.

A generated descriptive tag may not be used as a secondary definition.

Using a Literal: A macro-header operand containing a literal secondary definition would be one such as TAG, #+XXX.X# □. 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 numerical 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 in item 2 above would be written as TAG, #+03.01 □. This form is fully discussed earlier in this manual. This method can be used to define signed or unsigned fields only.

#### Literal Operands

A literal is actual data enclosed by pound signs (#) (see "One-for-One Instructions"). 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 numerical 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 stated in the explanation of FPN. A literal must not exceed 35 positions, exclusive of the pound signs.

#### 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 must never be omitted. If the operand it terminates is omitted, the fixed lozenge is placed back to back with the lozenge that 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 that 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 rapidly 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 omitted, 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 that is defined by either an area definition or a switch definition. In order to generate coding that is proper for the field, the Processor must know the characteristics of the data that 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 that 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 that is used to compare the contents of two fields. The fields may be alphameric or numerical. The one-for-one instructions which should be used to compare alphameric data differ from those which should be used to compare numerical 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 numerical data but as alphameric data.

EXAMPLES OF MACRO-INSTRUCTIONS AND THEIR USE

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

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 <sub>1</sub>	BLANK		X <sub>1</sub> X <sub>2</sub> X <sub>3</sub> ..... X <sub>50</sub>

T<sub>1</sub> is the tag of the macro-header, or is omitted.  
X<sub>1</sub> ... X<sub>50</sub> are the tags of the fields 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
ONE	NAME	0	
TWO	RCD	5	+
	RPT	8	XXXX. ZZ
	{		
TAG1	BLANK		ONE TWO
TAG1	RCV		ONE
	BLM		@00001
	RCVS		TWO
	BLMS		@00008

Figure 57  
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 <sub>1</sub>	IFON		X <sub>1</sub> X <sub>2</sub> X <sub>3</sub>

T<sub>1</sub> is the tag of the macro-header, or is omitted.  
X<sub>1</sub> is the tag of the switch to be tested.  
X<sub>2</sub> 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<sub>3</sub> 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, ON and OFF must be assumed to be the tags of instructions. If OFF and its associated lozenge had been omitted, the final instruction would not have been generated.

Tag	Operation	Num	Operand
NEW YORK	CHRC		A
CHICAGO	D		B
	{		
TAG2	IFON		NEW YORK ON OFF
TAG2	LOD	1	#A#
	CMP	1	NEW YORK
	TRE		ON
	TR		OFF

Figure 58  
Add -- ADDX

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



Tag	Operation	Num	Operand
T <sub>1</sub>	ADDX	1	X <sub>1</sub> X <sub>2</sub> X <sub>3</sub>

T<sub>1</sub> is the tag of the macro-header or is omitted.  
X<sub>1</sub> is the tag of one numerical source field; i.e., the field that is the source of one set of data to be added.  
X<sub>2</sub> is the tag of the other numerical source field.  
X<sub>3</sub> is the tag of the numerical or RPT result field; i.e., the field in which the result is to be placed.

Tag	Operation	Num	Operand
NINE	RCD	5	#+02.03
TEN		6	#+03.03
TAG3	ADDX		NINE X <sub>1</sub> X <sub>2</sub> X <sub>3</sub> TEN
TAG3	RAD		NINE
	SET		@00006
	ADD		#+75.000#
	ST		TEN

Figure 59

#### 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 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 <sub>1</sub>	INCRA		X <sub>1</sub> X <sub>2</sub>

T<sub>1</sub> is the tag of the macro-header, or is omitted.  
X<sub>1</sub> is the tag of an instruction that makes the field reference to be incremented.  
X<sub>2</sub> 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. (M00017#02 is a tag generated by the Processor).

Tag	Operation	Num	Operand
OTHER	RCD	8	A
FIELD		3	A
MACRO	BLANK		OTHER X <sub>1</sub> FIELD X <sub>2</sub>
MACRO	RCVS		OTHER
	BLMS		@00008
M00017#02	RCVS		FIELD
	BLMS		@00003
TAG4			
TAG4	INCRA		MACRO X <sub>1</sub> X <sub>2</sub> #+500#
	RAD	15	#+500#
	AAM	15	M00017#02

Figure 60

## ADDRESS CONSTANTS

An address constant is a numerical 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, the literal, or the 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 that is not created until the program is assembled.

Address constants are used to initialize instructions, a procedure that alters the reference made by an instruction or supplies a reference to an instruction that 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 that references FIRST and an instruction that 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, a literal, an actual, or a location counter. Operand modifiers may be used with a tag or a literal to request a generated constant:

<u>Modifier</u>	<u>Address Constant Generated From</u>
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

<u>Modifier</u>	<u>Address Constant Generated From</u>
High-speed nine Size	A left-hand address plus nine 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 160,000. The symbols are as follows:

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

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 ten 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 <sub>1</sub>	ADCON	nn	X <sub>1</sub>

T<sub>1</sub> is the tag of the address constant.  
 nn is ASU zoning or is blank.  
 X<sub>1</sub> 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
RECORDONE	RCD	35A+	
	}		
FIRST	ADCON		RECORDONE

Figure 61

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

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
INSTRUCTION	RCD	START	
	}		
TAG1	ADCON	15L	INSTRUCTION

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 <sub>1</sub>	ACON4	nn	X <sub>1</sub>

T<sub>1</sub> is the tag of the address constant.  
 nn is an ASU number or is blank.  
 X<sub>1</sub> 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
RECORDAREA	RCD	35A+	
	}		
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 <sub>1</sub>	ACON5	s	X <sub>1</sub>

T<sub>1</sub> is the tag of the address constant.  
s is + for a positive constant, or is - for a negative constant, or is blank for an unsigned constant.  
X<sub>1</sub> 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. Note that the address constant will be signed. The sign of the address constant is not related to the sign of the literal.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39		
TAG 1	ACON 5	+	5000

Figure 65

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

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39		
FIELD 2	RPT	2	2
TAG 2	ACON 5	s	FIELD 2 * 2

Figure 66

#### Restrictions on an ACON5 Statement

ASU zoning may not be specified in an ACON5 statement.

Any ACON5 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 constant is requested for such an address, 80,000 is subtracted from the address. 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 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 <sub>1</sub>	ACON6	s	X <sub>1</sub>

T<sub>1</sub> is the tag of the address constant.  
s is + for a positive constant, or is - for a negative constant, or is blank for an unsigned constant.  
X<sub>1</sub> is a tag, literal, actual, or location counter.

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

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39		
TAG 1	ACON 6		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 for 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 a literal
- 2. Operand modifier, plus ASU zoning plus a tag or a 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,
High speed nine	T@	T@nn,

NOTE: The modifier and ASU zoning may also be written in the form R@n, L@n, etc., when specifying ASUs 1 through 9.

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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28	30 32 33 35 37 38 39
FIELD	RCD	35A+	
ADCONLIT	LDD	15R@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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28	30 32 33 35 37 38 39
FIELD	RCD	25A+	
ADCONLIT	LDD	15R@05, FIELD	

Figure 69

Arithmetic instructions, such as ADD, SUB, etc., cause a six-position signed constant to be created; the constant is signed plus. In a secondary mode, a five-position constant, signed plus, is created. All instructions requiring a 4 or 9 address, such as LDA, AAM, TR, TMT, etc., cause a four-position unsigned constant to be created and properly positioned in a 4 or 9 location regardless of the mode. All other instructions cause a four-position unsigned constant, positioned in a 4 or 9 location, to be created for 705 II mode; a five-position unsigned constant to be created for 705 III mode; and a six-position, unsigned constant to be created for 7080 mode. In each case the maximum constant allowed is dependent on mode memory size.

Restrictions on an Address Constant Literal Operand

Character adjustment may be used for the purpose of modifying the constant itself. If character adjustment is written in an address constant literal operand, it will not 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.

## INSTRUCTIONS TO THE PROCESSOR

Instructions to the Processor concern the assembly process. They are executed by the Processor at assembly time. Consequently, these instructions do not appear in 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
  - Relative Assignment - RASN
  - Assignment of Macro-Instruction Subroutines - SUBRO
  - 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
4. Multiple Literal Tables
  - Literal Start - LITST
  - Literal End - LITND
5. Flags

### INSTRUCTIONS TO THE PROCESSOR - 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:

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 00500. The parts of the object program are assigned in the following sequence: the machine-language equivalent of the source program, the library subroutines, the main literal table. If no subroutines have been requested by either the source program or the Processor itself, the main literal table is placed after the source program. Standard "00" transfer control card: The Processor produces this as the terminal card of the object program deck. (The next chapter 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 parts of the object program
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 00500. 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. By 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 00500, 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.

**NUMERICAL 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 to 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 numerical 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.
4. A location counter, with or without adjustment. If there is no adjustment the assignment continues; i.e., it starts in the next available location.

To reset the counter to location 00500, 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 "Flag Characters and Their Meanings.")

**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, "Flag Characters and Their Meanings."

In Figure 70, storage assignments are shown to the right of the hand-coded Autocoder statements. Note 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 numerical field.

Tag	Operation	Num	Operand	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	000500
	{			{
	LASN			003025

Figure 70

**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 main 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 main 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 main literal table may be altered by LITOR and SUBOR. These are instructions to the Processor and are explained on subsequent pages. The assignment of multiple literal tables is controlled by LITST and LITND, as explained under "Multiple Literal Tables.")

#### Restrictions on a LASN Statement

A LASN statement may not be referred to by another Autocoder 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 <sub>1</sub>

X<sub>1</sub> is an actual operand, or is the tag of a statement appearing anywhere in the program before the SASN, or is a location counter.

The tag or location counter may be character-adjusted.

Note that the tag and numerical fields must be left blank. Comments may be placed in the comments field.

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

Tag	Operation	Num	Operand	ASSIGNMENTS
	{ LASN		@2000	{ 002000
	{ SASN		@3000	{ 002499 003000
	{ LASN			{ 004000 002500

Figure 71

#### Restrictions on a SASN Statement

A SASN statement may not be referred to by another Autocoder statement.

#### Relative Assignment -- RASN

This instruction allows a program or portion of a program to be assembled at one location and to cause all references to or within the program to be treated as if they were assembled at a different location. Various subroutines therefore, can be assembled relative to the same location, and at object time one of them can be moved for actual execution.

Locations will be assigned in the normal manner to the entries following a RASN, but references to them or any one of them will effectively be to their relative address.

A relative assignment will be terminated by any LASN, SASN, or TCD.

In Figure 72, the routine beginning with TAGA will be assembled starting at location 2000, but all references to the routine will be assembled as if the routine started at location 0300. The instruction used to move the routine should reference actual location 2000.

In Figure 73, the routine beginning with TAGA will be assembled starting at location 5005, but all references to the routine will be assembled as if the routine started at location 0300. The LASN is used to terminate the RASN. The instruction used to move the routine should reference REFTAG + 5.

There are certain limitations to be observed when using a RASN:

1. As with SASN, a RASN has no effect on the high assignment counters.



2. If location assignment is under control of a LASN or SASN at the time a RASN is encountered, it continues under control of the LASN or SASN.

3. At the time a RASN is encountered, the following (in effect) occurs: The location counter is incremented by one, and the high-order location of the operand of the RASN is obtained. The difference between these two must be a multiple of five, or inconsistent results will occur. Therefore, it is recommended that a RASN always be preceded by a LASN or a SASN; and that both have as operands actual addresses or tags that are similarly positioned with respect to the low-order location.

Tag	Operation	Num	Operand	LOC	INSTR	SU	ADDRESS
TAGA	TR		OUT	5004	18004		008004
	LASN		@2000	2000			
	RASN		@300	0300			
	CMP	CON 1		2004	40343		000343
	TRE		*+25	2009	L0334		000334
TAGB	SHR	1		2014	C0001		000001
	TRZ		TAGB	2019	N0329		000329
	TR		TAGA	2024	10304		000304
	HLT		9999	2029	J9999		009999
	LOD	01	CON 2	2034	803U4	01	000344
CON 1	TR		*+10	2039	10349		000349
	CON	04	XXXX	2043			
	CON	01	Y	2044			
	LASN			5005			
	LOD	01	CON 2	5009	803U4	01	000344

Figure 72

Tag	Operation	Num	Operand	LOC	INSTR	SU	ADDRESS
REFTAG	TR		OUT	5004	18004		008004
TAGA	RASN		TAGAT300	0300			
	CMP	CON 1		5009	40343		000343
	TRE		*+25	5014	L0334		000334
	SHR	1		5019	C0001		000001
	TRZ		TAGB	5024	N0329		000329
TAGB	TR		TAGA	5029	10304		000304
	HLT		9999	5034	J9999		009999
	LOD	01	CON 2	5039	803U4	01	000344
	TR		*+10	5044	10349		000349
	CON	04	XXXX	5048			
CON 1	CON	01	Y	5049			
	LASN			5050			
	LOD	01	CON 2	5054	803U4	01	000344

Figure 73

A RASN statement is written in the format shown below.

Tag	Operation	Num	Operand
	RASN		X <sub>1</sub>

X<sub>1</sub> is an actual operand, or  
is the tag of a statement appearing anywhere  
in the program before the RASN, or

is a location counter.

A tag or location counter may be character-adjusted.

The tag and numerical fields must be left blank.  
Comments may be placed in the comments field.

#### Restrictions on a RASN Statement

A RASN statement may not be referred to by another Autocoder statement.

#### Assignment of Subroutines Within Macro-Instructions - SUBRO

The function of a SUBRO statement is to cause the Processor to treat the coding that follows it as a subroutine and to locate it out of line. The Processor assigns storage locations to SUBRO routines after it has assigned locations to Class A subroutines. The storage location at which the Processor is to begin assigning addresses is designated in the operand of the SUBRO statement.

NOTE: A SUBRO statement must not be written in a source program. It is designed to be used with user-written macro-instructions. A complete explanation of the usage of a SUBRO is given in the publication on the preparation of macro-instructions.

#### 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 is 7080, and the secondary mode may be 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 <sub>1</sub>

X<sub>1</sub> is an actual operand, or is the tag of an Autocoder statement, or is a location counter.  
The tag or location counter may be character for consistency adjusted. The tagged statement must precede the SUBOR statement.

Comments may be placed in the comments field.

Figure 74 indicates that the programmer assumes the subroutines cannot possibly occupy more than 5,000 positions.

Name (Tag)					Operation			Num		Operand											
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39
						SUBOR					@160										
						LASN					@5160										
RECORD						NAME						END RECORD									
						{															
						}															

Figure 74

#### Restrictions on a SUBOR Statement

A SUBOR statement may not be referred to by another Autocoder statement.

#### Assignment of Literals -- LITOR

The function of a LITOR statement is to specify the starting location for the assignment of the main literal table. The LITOR assignment supersedes the standard main literal table placement, which is after the subroutine block or after the last instruction of the program if no subroutines are used. LITOR enables the user to position the main 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 also applies to LITOR.)

The LITOR statement is written as follows:

Tag	Operation	Num	Operand
	LITOR		X <sub>1</sub>

X<sub>1</sub> is an actual operand, or is the tag of an Autocoder statement, or is a location counter.

The tag or location counter may be character-adjusted. The tagged statement must precede the LITOR statement.

Comments may be placed in the comments field.

In Figure 75, the Processor is instructed to start the main 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 main literal table is actually placed in storage or that the contents of TAG are placed in storage after the main literal table is no longer needed.

Name (Tag)					Operation		Num		Operand												
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39
							}														
						L	I	T	O	R		T	A	G							
							}														

Figure 75

#### Restrictions on a LITOR Statement

A LITOR statement may not be referred to by another Autocoder statement.

A LITOR statement cannot be used to position multiple literal tables. The LITST and LITND statements must be used for this purpose.

#### 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. If a Z character is placed in column 74 of the TCD statement, the generated TCD "00" transfer control card will be produced at the end of the object program and will replace the standard "00" card (see the section "Flag Characters and Their Meanings").

The TCD statement must be followed by Autocoder statements that specify the contents of the card; i.e., by the instructions or the instructions and data the card will contain. The last of these Autocoder 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
	TCD		

Comments about the "00" card may be written in the comments field. A tag is not needed.

**THE EFFECT OF THE "00" CARD ON THE LOADING PROCESS:** As previously explained, as soon as a "00" card is loaded into storage, it causes the loading program to interrupt the loading procedure and to execute the instructions on the card. 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 76. 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
HOUSEKEEP	SEL		0500
ENDHOUSEK	TR		ZEROCARD
	TCD		
	TR		HOUSEKEEP
ZEROCARD	TR		000004
	LASN		HOUSEKEEP
CALCULATE	ADDX		ONE, TWO, THREE

Figure 76

## Restrictions on a TCD Statement

The machine-language version of the Autocoder statement specifying the "00" card content may not exceed 65 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 <sub>1</sub>

X<sub>1</sub> 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 77, is to provide linkage to a subroutine. The subroutine is ROOTS; the tag of its entry point is STEP 1.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15 16 18 20 21 22 23 25 27 28 30 32 33 35 37 38 39			
	LINK		STEP 1
	INCL		ROOTS

Figure 77

**TYPES OF LIBRARY SUBROUTINES:** Programmers may write subroutines in Autocoder 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.

1. Class A. These may contain any Autocoder statement.
2. Class B. These may contain any Autocoder statement, including NAME entries, except the following: a macro-instruction other than ENT80 or LEV80; an INCL that designates a Class A subroutine; a TRANS entry having the tag of another location as an operand.

An INCL statement may not be referenced by another Autocoder Statement.

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

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, all references will be equated to the location designated by the TRANS.

**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.

**NUMERICAL 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 as the operand of an object program requesting instruction, regardless of the memory orientation of the operation.
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 any character adjustment. The location of the instruction immediately following the TRANS with character adjustment applied will appear in an object-program instruction wherever the tag of the TRANS appears as a source program operand.
4. A tag of another location, including the location of another TRANS. The operand may have a character adjustment and/or an operand modifier

In Figure 78, 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.

Name (Tag)	Operation	Num	Operand
6    8    10 11    13    15	16    18    20 21 22	23    25    27 28    30    32 33    35    37 38 39	
	SEL		MASTER TAPE
	S		
MASTER TAPE TRANS		0200	

Assume that location 05009 is assigned to the first instruction generated from the ADDX macroinstruction in Figure 79. 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.

Name (Tag)	Operation	Num	Operand
6 8 10 11 13 15	16 18 20	21 22 23 25 27 28	30 32 33 35 37 38 39
	IR	NEXT	
	}		
NEXT	TRANS	*	
	ADDX	ONE TWO THREE	

If the RD instruction in Figure 80 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.

Name (Tag)	Operation	Num	Operand
6    8    10 11    13    15	16    18    20	21 22 23    25    27 28	30    32 33    35    37 38 39
	TR		ERROR
	}		
	BSP		
ERROR	TRANS	X-5	
	RD	AREA	

52

## Restrictions on a TRANS Statement

If a TRANS statement has a location counter, an actual operand, an operand modifier, or character adjustment, the statement that references the tag of the TRANS cannot have an operand modifier. In any of these cases, an operand modifier would have no significance.

## Source-Program Language -- MODE

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

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

The term "higher languages of the 7080 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 programs that contain Report/File 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 81 must precede each FORTRAN portion of an Autocoder program.

Name (Tag)						Operation	Num	Operand																			
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39						
						MODE					FORTRAN																

**Figure 81**

The operand FORTRAN indicates that the subsequent statements are in standard FORTRAN format.

**REPORT/FILE MODE STATEMENT:** The statement shown in Figure 82 must precede each Report/File portion of an Autocoder program.

Name (Tag)					Operation	Num	Operand																			
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39					
						MODE				REPORT																

Figure 82

**AUTOCODER MODE STATEMENT:** The statement shown in Figure 83 must precede each Autocoder portion of a program if that portion follows Report/

**File or FORTRAN statements.** The statement is used whether or not the Autocoder portion also contains Decision, Arithmetic, and Table statements.

Name (Tag)						Operation			Num		Operand										
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39
						MODE				AUTOCODER											

Figure 83

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

CODING GENERATED IN 7080 MODE

The terms "7080 mode" and "secondary mode" 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 Leave Eighty Mode (LEV80) and Enter Eighty Mode (ENT80), both of which are described in the macro-instruction manual. The 7080 mode is assumed until a LEV80 is encountered. Of course, if the entire program is in 7080 mode, the LEV80 and ENT80 are not necessary. Since these macro-instructions are Assembly Control macro-instructions, they should be considered along with other instructions to the Processor.

LEV80 and ENT80 affect the coding generated from the statements in the portion of the program that each of them precedes. The Processor generates 7080 instructions until it encounters a LEV80. It then generates 705 II or 705 III coding (depending on which is designated as the secondary mode for the assembly) until ENT80 is encountered.

The Processor then resumes generation in 7080 mode. 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 EIA 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 84. It may not be tagged, and it may contain only one line of comments.

Name (Tag)					Operation		Num		Operand														
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39		
						E	J	E	C	T													

Figure 84

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 the section, "Details of the Program Listing.")

### 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:

**OPERATION FIELD:** The mnemonic code TITLE is placed here (Figure 85). 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.

**NUMERICAL 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 numerical field be left blank at all times.

**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.

### 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 some 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. The TITLE page of the Operator's Notebook 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 86 and 87, 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 part of the program, 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 86, 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.

Name (Tag)					Operation		Num		Operand													Comments																							
6	8	10	11	13	15	16	18	20	21	22	23	25	27	28	30	32	33	35	37	38	39	40	42	43	45	47	48	50	52	53	55	57	58	60	62	63	65	67	69	71	73	74			
						T	I	T	L	E																																			

Figure 85







table. This can be effected by placing an L flag in column 74 (see "Flag Characters and Their Meanings").

## FLAG CHARACTERS AND THEIR MEANINGS

Flags are a means of communicating with the Processor. Specific single-character flags, explained below, have been defined for use in column 74 of all input to the Processor except FORTRAN and COBOL statements. Additional flags may be allocated in the future, and they will be made available as soon as they are completely defined. Should any character be encountered in column 74 when its use is unintentional, inconsistencies may occur in the assembled program.

### @ -- Force Program Card

This flag will cause the output produced from the entry containing the flag to begin on a new program card.

### A -- Reduce Location Assignment Phase Assembly Time

This flag is for use within Class B subroutines. It is placed in column 74 of statements which have tags that will be the operands of assignment statements (e.g., LASN, SASN, RASN).

All entries bearing this flag will be placed in a table that is used when assignment statements are encountered. This reduces the assembly time for Class B subroutines (which are processed in the location-assignment phase).

### B -- Scan Entry from Right to Left

This flag will cause the Processor to scan the operand of the entry containing it from right to left, rather than from left to right.

On encountering a left literal symbol in the operand of a one-for-one instruction that contains the B flag, the Processor will then scan from column 73 left to a literal symbol. Everything between the two literal symbols will be considered an unsigned literal. Valid modifiers and character adjustments will be honored.

The B flag with an operand of a macro-header will cause a scan from column 73 left to a lozenge. Everything from column 24 through the column two positions to the left of the lozenge will be treated as an unsigned literal of that length. (The characters in column 23 and the column to the left of the lozenge will be assumed to be literal symbols, and will be dropped.) The operand to be so treated, with this

flag, must be on a line (card) that does not contain any other operand.

### C -- Entire Card is a Comment

Columns 6 through 73 of an entry containing this flag will be considered a comment. Entries so flagged will also be printed, single spaced, on a separate page of the Operator's Notebook. Entries with this flag that are contained in the input to a librarian run will not be treated as components of macro-instructions, and will be removed. Their function in this case is solely for the purpose of listing on an IBM 407.

### D -- Delete All Messages Created for This Entry

An entry containing this flag will be processed normally but diagnostic messages (if any) will not be produced for it.

### F -- Processor Chain Indicator

This flag indicates the beginning and end of a macro-instruction chain. It is used when the chain contains macro suffix tags and/or generated descriptive tag operands. (Its use is explained in the macro-instruction manual.)

### G -- Treat Change Entry as Generated Entry

This flag is provided for use with change entries introduced in a high-speed assembly run. It will cause the entries containing it to be considered as generated entries during a subsequent reassembly. That is, during a subsequent reassembly the entries will be deleted, and during a subsequent high-speed assembly the entries will be retained.

### H -- Halt Loop

This flag, intended for use in entries that constitute the error-indication portions of a program, will cause entries containing it to be listed on a separate page of the Operator's Notebook. The H flag is valid only on one-for-one instructions.

### L -- Main Table Literal

This flag is intended for use with statements that have literal or address constant literal operands, and occur between a LITST and a LITND. When the Processor finds such a statement containing an L flag, it will treat the operand as a main-table literal rather than as one belonging in a multiple literal table. The L flag provides a convenient means of

preventing repeated generations of the same literal in a program that uses multiple literal tables.

#### M -- Operand is to be Modified

This character may be used to flag all entries having operands that are not blank, but are to be initialized and/or modified, and will cause these entries to be printed on the page of the Operator's Notebook containing entries with blank operands. The M flag is valid only on one-for-one instructions. When a generated instruction is referenced by another macro-instruction by means of a macro suffix tag, the macro generator automatically places an M flag on the referenced instruction, unless another flag is already present on it.

#### R -- Reset Location Counter

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: Suppose that when location counter 1 contains 59999, 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.

#### S -- Suppress Program Cards

An Autocoder entry containing this flag will indicate the beginning of program card suppression. This entry and all following entries will be processed normally, except that program cards will not be produced. A second entry containing this flag will indicate that program card suppression is to end after this entry is processed.

#### T -- Test-Assembly Entry

Entries containing this flag will be retained during an assembly when the run-type control card so indicates. Otherwise, all entries containing this flag will be deleted automatically. Statements may therefore be assembled for testing purposes, and easily removed.

#### Z -- Relocate "00" Transfer Control Card

This flag is only used with a TCD statement. It causes the TCD "00" transfer control card to be placed at the end of the program in place of the standard "00" card. If more than one TCD statement contains this flag, the last one encountered prevails.

#### l -- Weight Inner Macro-Instruction as One

This flag may be used with macro-headers when they are used as components of macro-instructions. It specifies that regardless of how frequently the macro-instruction containing it is used, the inner macro-instruction will be called by it very infrequently; therefore, the Processor is to consider that the inner macro-instruction is called one time as a component of the particular outer macro-instruction. Effective use of the flag will cause the Frequency Count Table to more accurately reflect the frequency with which each macro-instruction is used, so that the assignment of memory macro-instructions will be more efficient.

One card is punched for each line of the coding sheet, as explained in the section on statement format. 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 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 program (LD7080)
2. Main literal table
3. Machine-language equivalent of source program
4. Class A subroutines
5. Subroutines portions of macro-instructions
6. Class B subroutines
7. Standard "00" transfer control card

Note that the main 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:

**Program Identification:** Six positions. This is the source program identification (ident field on coding sheet).

**Serial number:** Three 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 (Pglin field on coding sheet).

**Initial address:** Four positions. This indicates the storage location at which the first character on the card is to be placed.

**Number of columns:** Two positions. This is the amount of data being supplied by the card. A maximum of 65 positions may be indicated; this is the space required by 13 instructions. The "00" card contains zeros in these positions.

**Instructions and/or constants:** One to sixty-five positions. This is the actual portion of the object program being supplied by the card. It is placed at the storage location specified as the initial address (see above).

## STANDARD 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 below in the section, "Details of the Listing."

## Program Listing

The program listing is provided only on tape. The contents of the listing are as follows:

**First Page:** This page is blank except for a heading line and a notation of the highest memory position used not resulting from a RASN or SASN.

**Main Literal Table:** The main literal table is divided into seven parts. (A signed literal is a literal in which the first position after the pound sign (#) is occupied by a plus or a minus sign.)

1. Signed literals, length not a multiple of 5 or 10
2. Signed literals, length a multiple of 5
3. Signed literals, length a multiple of 10
4. Unsigned literals, length a multiple of 10
5. Unsigned literals, length a multiple of 5
6. Unsigned literals, length not a multiple of 5 or 10.
7. Address constant literals, broken down in the

following order:

- a. unsigned, length of 6
- b. signed, length of 6
- c. signed, length of 5
- d. unsigned, length of 5
- e. all lengths of 4 ending in a 4 or 9 location

**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(s) that caused the generation.

Multiple literal tables are also included in the source program, if they are requested. They are divided into seven parts corresponding to those in the main literal table. However, within the groups of signed and unsigned literals, individual literals are not sorted according to size. Each multiple literal table will begin on a new page of the program listing.

**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.

**Subroutine Portions of Macro-Instructions:** The order of subroutines is the same as that of the macro-headers causing their generation.

Class B Subroutines: The subroutines are inserted alphabetically.

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 condition, 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. Diagnostic messages are described in the publication on 7080 Processor system operation.

Unreferenced Tags (NO REQS): On a separate page, hand-coded tags that are not referred to elsewhere in the program are listed.

## OPTIONAL DOCUMENTATION

### Operator's Notebook

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

TITLES	-- All TITLE statements
C FLAG	-- Comment statements with a C flag
H FLAG	-- Statements with an H flag; all halts
80 SP OP	-- All ENT80, LEV80, ENTIP, LEVIP, SPC, TIP, and LIP statements
80 SP 1	-- All statements in 7080 mode containing indirect address; i.e., the "I," prefix
ASSGNS	-- All LASN, SASN, RASN, and SUBRO statements
SWITCHES	-- All SWN and SWT statements
TRANS	-- All TRANS statements with descriptive operands; i.e., operands that are tags
M FLAG	-- All statements with an M flag; all statements with blank operands

### Symbolic Analyzer

This is an index of every hand-coded and generated tag in the program. The tags are listed in collating sequence. Each tag is followed by a list of every instruction, hand-coded or generated, that references the tag. Tags that are used incorrectly are flagged with an error indicator appearing as \*ERR\*.

Each program entry that defines a tag will be listed. All entries having operands that reference the tag will be listed, three per line, following the tag definition. Any operand modifier and/or

character adjustment in a referencing entry will be included, but comments, and ASU zoning in address constant literals will not. Entries that refer to undefined tags will be listed separately. When multiple literal tables occur in a program, the symbolic analyzer will contain a section on them preceding the index to descriptive tags.

## DETAILS OF THE PROGRAM LISTING

The heading of each page in the listing contains the program identification, revision number (if any), and the date (from the date control card), and page number.

The listing page contains 16 fields. The entries in the PGLIN through the FLAG fields comprise an Autocoder statement. The machine-language translation of the statement (i.e., an object-program instruction or constant) appears in the INSTR field. 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 or added, or one that follows a deleted statement. The INDEX number is not identical to the pglin number on the coding sheet.

S: Origin of entry (i.e., whether it is a source-program statement or a Processor-generated entry) and type of entry. Both items of information are conveyed by a single-character code, as follows:

<u>Code</u>	<u>Origin</u>	<u>Type of Statement</u>
A	Source Program	One-for-One
B	Source Program	Macro-Header
E	Source Program	Decision, Arithmetic, Table
F	Source Program	Report/File
G	Source Program	FORTRAN
I	Source Program	TITLE, C flag, and COBOL
J	Generated	One-for-One
K	Generated	Macro-Header
N	Generated	Decision, Arithmetic, Table
O	Generated	Report/File
P	Generated	FORTRAN
R	Generated	TITLE and C Flag
*	Generated	EIA and Related Instruction, and Multiple Literal Tables

NOTE: All subroutine entries are generated.

PGLIN: The entry in this field corresponds to the pglin entry 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 numerical field on the coding sheet.

**AT:** An entry in the AT (address type) 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 of this field varies just as it does when hand-coded. The field corresponds to the operand field on the coding sheet with the exception of the placement of a prefix to the basic operand. The prefix appears in the AT field explained in the preceding paragraph.

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

**F:** Flag code.

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

**INSTR:** The entry is a five-position field containing the actual operation code of the instruction followed by the actual address with ASU zoning.

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

**ADDR:** This field contains the actual address portion of an instruction as six positions.

**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 character on the object-program card. Subsequent lines with blanks in the SER field contain data that appear on the same card.

**REF:** An entry in this field is the INDEX number of the operand, and serves as a cross-reference. (Within a NAME, the number in this column is the cumulative length of the NAME.)

## APPENDIX

The more significant features that have been incorporated into Autocoder for the 7080 Processor are summarized below, by section headings. The reader can consult the appropriate sections of this manual for details on the changes.

Source programs that could be assembled by the 7058 Processor can also be assembled by the 7080 Processor. However, certain mnemonics which were accepted by the previous processor will not be accepted by the 7080 Processor. These invalid mnemonics are listed below:

1. DRCD, DCON, or DFPN
2. AACON, LACON, or RACON
3. AASN, OASN, or CASN
4. \*ASUnn
5. Actual operation codes

In addition, CTL, while it may be used and will be accepted, will cause a warning message to be produced; it will be assumed that the programmer has indicated the proper operand.

Certain differences between 7058 Autocoder and 7080 Autocoder result from expansion of the language and the incorporation of new features. Those differences are listed below.

1. A character in column 74 of a source statement, except one in FORTRAN or COBOL, will be considered a flag having specific significance to the 7080 Processor. The flag codes are described in the section on flags.
2. A character adjustment following an address constant literal request (e.g., L@TAG+5) will cause an increment to the assembled location of the address constant.
3. A literal may not be followed by a multiply or divide character adjustment, nor may the amount of the character adjustment be outside the range  $\pm 99$ ; i.e., be stated in more than two significant numbers. However, an increment or decrement can be written with leading zeros; e.g., +1 and +001 will cause the same increment, and -55 and -00055 will cause the same decrement.
4. No operand of a macro-header may exceed 10 positions unless it is surrounded by literal symbols. No literal used as a macro-header operand or in a macro-instruction component may exceed 35 positions including the sign and decimal point, but not including the literal symbols.
5. If the numeric portion of a character adjustment is less than six positions, the position immediately following the adjustment must be non-numerical.

Standard Format of Autocoder Statements: A new multipurpose coding form has been developed for use with the 7080 Processor. Column headings have been changed to accommodate certain new features of the Processor.

Area Definitions: Area-definition length may be specified by a six-digit number written in columns 17-22. Restrictions on comments continuation lines with area definitions have been altered to reflect the new meaning of the columns. RPT statements are restricted to nine commas in the layout format.

One-for-One Instructions: The list of acceptable mnemonics has been expanded and provision has been made for additional numerical codes to accompany various operation codes. The changes are detailed in Figure 44. Restrictions on character adjustment have been expanded, particularly with respect to literal operands. A new operand modifier (T, ) has been provided for both one-for-one instructions and address constants.

General Purpose Macro-Instructions: Up to 50 operands can be written in the macro-header. As many as 50 lines in the coding form can be used for the operands of one macro-instruction. Literal operands must not exceed 35 characters excluding the literal (#) signs.

Address Constants: An ACON6 can have a sign associated with it. Address constant literal requests of arithmetic operations will be six positions long with a signed plus. Formerly, such address constant literals were five positions. Character adjustment may be used for the purpose of modifying the constant itself.

Instructions to the Processor: The initial setting of the location counter is now 00500. Restrictions on LASN, SASN, SUBOR, and LITOR statements have been eased. The location counter, with or without adjustment, is now a valid operand for these statements. Two new assignment statements (RASN and SUBRO) have been added. Two statements (LITST and LITND) have been provided for creating multiple literal tables. A TRANS statement can have the tag of another location as its operand. A TCD statement can now occupy 65 positions. 7080 mode is assumed until a LEV80 is

encountered. To return to 7080 mode following a LEV80, the ENT80 macro-instruction is given. Additional instructions to the Processor in the form of Flag characters have been added to the Autocoder language. The use of Flags, particularly the F Flag, should be carefully considered.

Assembly Output: The listings that are provided have been expanded considerably. This entire section should be reviewed.

# SAMPLE ASSEMBLY

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	PATCHES	PG	001	F	LOC	INSTR	SU	ADDR	SER	REF
																			005449

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	002	F	LOC	INSTR	SU	ADDR	SER	REF
DA01			SIGNED LITERAL		1									005175				001	
DA02			SIGNED LITERAL		1		A							005176					
DA03			SIGNED LITERAL		2		16							005178					
DA04			SIGNED LITERAL		4		123D							005182					
DA05			SIGNED LITERAL		4		395G							005186					
DA06			SIGNED LITERAL		7		BALANCN							005193					
DA07			SIGNED LITERAL		7		987654C							005200					
DA08			SIGNED LITERAL		5		0000A							005209					
DA09			SIGNED LITERAL		5		0000D							005214					
DA10			SIGNED LITERAL		5		0021E							005219					
DA11			SIGNED LITERAL		10		0M5678000E							005229					
DA12			UNSIGNED LITERAL	50			AGE			CLOSING LIT SYMBOL OMITTED				005279				002	
DA13			UNSIGNED LITERAL	50			THIS LITERAL OVERFLOWS INTO THE NEXT CARD WHICH IS							005329				003	
DA14			UNSIGNED LITERAL	5			ABCDE							005334					
DA15			UNSIGNED LITERAL	5			APPLE							005339					
DA16			UNSIGNED LITERAL	1			F							005340					
DA17			UNSIGNED LITERAL	1			G							005341					
DA18			UNSIGNED LITERAL	1			J							005342					
DA19			UNSIGNED LITERAL	1			1							005343					
DA20			UNSIGNED LITERAL	2										005345					
DA21			UNSIGNED LITERAL	2			60							005347					
DA22			UNSIGNED LITERAL	3			300							005350					
DA23			UNSIGNED LITERAL	4			ABLE							005354					
DA24			UNSIGNED LITERAL	4			DUPE							005358					
DA25			UNSIGNED LITERAL	4			0010							005362					
DA26			UNSIGNED LITERAL	7			1234567							005369				004	
DA27			UNSIGNED LITERAL	8			-BALANCE							005377					
DA28			UNSIGNED LITERAL	9			LOCATIONA							005386					
DA29			UNSIGNED LITERAL	14			NOT AVAILABLE							005400					
EA01			NAMEA	RIGHT	6		001099							005406				AC51	
SA01			NAMEA	SIZE	6		00004E							005413				AC51	
*A01			NAMEA	SIZE	5		0004E							005419				AC51	
-A01			NAMEA	RIGHT	5		01099							005424				AC51	
/A01			123D	RIGHT	4		5182							005429				DA04	
/A02			*E000025	RIGHT	4		2C14							005434				005	
/A03			EXIT	RIGHT	4		1599							005439				AF55	
/A04			NAMEA	HI-SP	4		1074							005444				AC51	
/A05			NAMEA	RIGHT	4		1A03							005449				AC51	



INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	003	F	LOC	INSTR	SU	ADDR	SER	REF
AA 01	I	AA01					TITLE	7080 PROCESSOR - SAMPLE ASSEMBLY											
AA 02	I	AA02					TITLE	INTRODUCTION											
AA 03	I	AA03						THIS ASSEMBLY ILLUSTRATES CORRECT AND INCORRECT USAGES OF THE 7080											C
AA 04	I	AA04						PROCESSOR. SHORT CODING EXAMPLES ARE USED TO SHOW WHAT THE											C
AA 05	I	AA05						PROCESSOR PRODUCES, INCLUDING ERROR AND CAUTIONARY MESSAGES, FOR											C
AA 06	I	AA06						TYPICAL VALID AND INVALID STATEMENTS. COMMENT AND TITLE STATEMENTS											C
AA 07	I	AA07						AND THE COMMENTS FIELD OF ILLUSTRATIVE STATEMENTS, HAVE BEEN USED TO											C
AA 08	I	AA08						DESCRIBE THE USAGES. THIS ASSEMBLY IS FOR ILLUSTRATIVE PURPOSES ONLY											C
AA 09	I	AA09						AND DOES NOT REPRESENT AN EXECUTABLE PROGRAM. THE OBJECT MACHINE IS											C
AA 10	I	AA10						ASSUMED TO BE AN 80K 7080, ASUS 1-6 ARE ASSUMED SET TO LENGTHS OF											C
AA 11	I	AA11						1-6 RESPECTIVELY, AND THE OTHER ASUS AND ACC ARE AT SOME RANDOM											C
AA 12	I	AA12						LENGTH.											C
AA 13	I	AA13					TITLE	NORMAL ORIGIN											
AA 14	I	AA14						SINCE NO STARTING LOCATION IS SPECIFIED, THE ORIGIN OF THE											C
AA 15	I	AA15						PROGRAM IS ASSUMED TO BE AT LOCATION 0500.											C
AA 16	A	AA16					RCD	1											000500
AA 17	I	AA17					TITLE	AREA DEFINITIONS											
AA 18	I	AA18						DEFINITION OF A RECORD FIELD - RCD											
AA 19	A	AA19	RCDA				RCD	10	N										000510
AA 20	A	AA20								TEN DIGIT UNSIGNED NUMERIC FIELD									
AA 21	A	AA21						17	A	SEVENTEEN POSITION ALPHA-NUMERIC									000527
AA 22	A	AA22								FIELD WHOSE LOW ORDER POSITION MAY NOT PROVIDE									
AA 23	A	AA23								LEFT PROTECTION FOR ANY SIGNED NUMERIC FIELD IT									
AA 24	A	AA24								PRECEDES.									
AA 25	A	AA25						2	00	A6									000727
AA 26	A	AA26								TWO HUNDRED POSITION ALPHA-NUMERIC									
AA 27	A	AA27								FIELD WHOSE LOW ORDER POSITION WILL ALWAYS SUPPLY									
AA 28	A	AA28								LEFT PROTECTION. NOTE THAT THE LENGTH INDICATION									
AA 29	A	AA29								OVERFLOWS INTO THE OPERATION FIELD. THIS IS									
AA 30	A	AA30								PERMISSIBLE ON A CONTINUATION ENTRY AS LONG AS									
AA 31	A	AA31								COLUMN 16 IS BLANK.									
AA 32	A	AA32																	
AA 33	A	AA33						10	G	TEN DIGIT SIGNED INTEGER. DECIMAL									000737
AA 34	A	AA34								POINT IS ASSUMED TO RIGHT OF THE LOW ORDER DIGIT.									
AA 35	A	AA35								LEFT PROTECTION IS PROVIDED FOR THE FOLLOWING FIELD									
AA 36	A	AA36																	
AA 37	A	AA37	RCDS5X3					8	GXXXXX.XXX	TWO ALTERNATE DEFINITIONS OF AN									000745
AA 38	A	AA38	RCDS5X3A					8	#605.03	EIGHT DIGIT SIGNED NUMERIC FIELD									000753
AA 39	A	AA39								HAVING FIVE INTEGER AND THREE DECIMAL POSITIONS.									
AA 40	A	AA40																	
AA 41	A	AA41	RCDSOX3					3	G.XXX	TWO ALTERNATE DEFINITIONS OF A									000756
AA 42	A	AA42								THREE DIGIT SIGNED DECIMAL.									000759
AA 43	A	AA43																	
AA 44	A	AA44	RCDSN2X3A					5	XX.XXX	TWO ALTERNATE DEFINITIONS OF A									000764
AA 45	A	AA45						05	# 02.03	FIVE DIGIT UNSIGNED NUMERIC FIELD									000769
AA 46	A	AA46								WITH TWO INTEGER AND THREE DECIMAL POSITIONS.									
AA 47	A	AA47																	
AA 48	A	AA48						1	#	RECORD MARK INDICATION.									000770
AA 49	A	AA49																	
AA 50	A	AA50						10	F	TEN POSITION FLOATING POINT RCD.									000780
AA 51	I	AA51								INVALID USAGES									C
AA 52	A	AA52					RCD	0											000780
AA 53	A	AA53						1000	00	A									000781
AA 54	A	AA54								ALTHOUGH IT IS VALID TO SPECIFY A									
AA 55	A	AA55								SIX DIGIT LENGTH IN THIS FASHION, THE SIZE OF									
AA 56	A	AA56								OBJECT MEMORY IS SPECIFIED AS 80K FOR THIS PROGRAM.									
AA 57	A	AA57								THIS STATEMENT WOULD BE VALID IF MEMORY SIZE WAS									
AA 58	A	AA58								SPECIFIED AS 160K.									
AA 59	A	AA59					RCD	4		THIS WILL RESERVE FOUR PLACES BUT									000785
AA 60	A	AA60								WILL BE TREATED AS AN UNDEFINED RCD AREA BECAUSE									
AA 61	A	AA61								G, N, A, OR A6 ARE NOT INDICATED IN THE OPERAND									
AA 62	A	AA62																	
AA 63	A	AA63					FIELD	.		THE WORD FIELD, INTENDED AS A									000794
AA 64	A	AA64								COMMENT CONTINUATION, WAS TREATED AS A NOP BECAUSE									A0000
AA 65	A	AA65								IT WAS IN THE OPERATION FIELD AND WAS NOT A VALID									000000
AA 66	A	AA66								OPERATION.									006
AA 67	A	AA67																	
AA 68	A	AA68						2	N	THIS STATEMENT, INTENDED AS A RCD									000796
AA 69	A	AA69								CONTINUATION, WILL COMPILE AS A CON BECAUSE IT HAS									
AA 70	A	AA70								A BLANK OPERATION AND FOLLOWS A STATEMENT WHOSE									
AA 71	A	AA71								OPERATION IS NOT A DATA DEFINITION. IT WILL COMPILE									
AA 72	A	AA72								AS N FOLLOWED BY A BLANK.									

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	004	F	LOC	INSTR	SU	ADDR	SER	REF
AB 01	I	AB01	THE FOLLOWING THREE INVALID RCD ENTRIES PRODUCE INCONSISTENT																
AB 02	I	AB02	DATA DEFINITIONS.																
AB 03	A	AB03					RCD	3		#00.03									000799
AB 04	A	AB04								THIS OPERAND SHOULD HAVE BEEN									
AB 05	A	AB05								#00.03. OMITTING THE # SIGN CAUSED IT TO COMPILE									
AB 06	A	AB06								AS A THREE DIGIT SIGNED INTEGER WITH NO INTEGER AND									
AB 07	A	AB07								NO DECIMAL POSITIONS.									
AB 08	A	AB08																	000802
AB 09	A	AB09								#00.03									
AB 10	A	AB10								OMITTING THE ZEROS CAUSED IT TO COMPILE									
AB 11	A	AB11								AS A THREE POSITION SIGNED INTEGER WITH ONE INTEGER									
AB 12	A	AB12								AND NO DECIMAL POSITIONS.									
AB 13	A	AB13																	000806
AB 14	A	AB14								#02.02									
AB 15	A	AB15								OMITTING THE BLANK CAUSED IT TO COMPILE									
AB 16	A	AB16								AS A FOUR POSITION UNSIGNED FRACTION WITH 21									
										INTEGER PLACES AND 20 DECIMAL PLACES.									
AB 17	I	AB17					TITLE			DEFINITION OF A CONSTANT FIELD - CON									
AB 18	A	AB18					CCN	5		ABCOE									000811
AB 19	A	AB19	CONA					5		00003									000816
AB 20	A	AB20	CONN5X0					5		4JK9*									000821
AB 21	A	AB21	CONMIXED																
AB 22	A	AB22						6		-123499									000827
AB 23	A	AB23								SIX POSITION SIGNED INTEGER									
AB 24	A	AB24								CONSTANT. WILL APPEAR AS 12349R IN MEMORY.									
AB 25	A	AB25						6		£1234.99									000833
AB 26	A	AB26								SIX POSITION SIGNED CONSTANT WITH									
AB 27	A	AB27								FOUR INTEGER AND TWO DECIMAL POSITIONS. WILL									
AB 28	A	AB28								APPEAR AS 12349I IN MEMORY.									
AB 29	A	AB29						6		123.45									000839
AB 30	A	AB30								SIX POSITION CONSTANT WHICH WILL									
AB 31	A	AB31								APPEAR AS 123.45 IN MEMORY.									
AB 32	A	AB32						3		A									000842
AB 33	A	AB33								THREE POSITION CONSTANT OF WHICH									
AB 34	A	AB34								THE FINAL TWO POSITIONS ARE BLANKS.									
AB 35	A	AB35						2		□#									000844
AB 36	A	AB36								TWO POSITION CONSTANT CONSISTING									
										OF A GROUP MARK AND A RECORD MARK.									
AB 37	I	AB37								INVALID USAGES									C
AB 38	A	AB38						2		ABCDE									000846
AB 39	A	AB39	WORSTCASES				CCN			CON WITH OPERAND OF GREATER LENGTH									
AB 40	A	AB40								THAN NUMERIC FIELD STATES. WILL COMPILE AS AB WITH									
AB 41	A	AB41								NO MESSAGE.									
AB 42	A	AB42						3		£120									000849
AB 43	A	AB43								SIGNED CONSTANT WITH OPERAND									
AB 44	A	AB44								SHORTER THAN NUMERIC FIELD STATES. IT WAS PUNCHED									
AB 45	A	AB45								£12 BUT WILL COMPILE AS 120 WITH THE LAST DIGIT									
AB 46	A	AB46								SIGNED PLUS. HERE THE LISTING SHOWS THE ZERO.									
AB 47	A	AB47						0		123									000849
AB 48	A	AB48								THIS WILL NOT COMPILE BECAUSE THE									
AB 49	A	AB49								NUMERIC FIELD STATES A LENGTH OF ZERO POSITIONS.									
AB 50	A	AB50						62		THE NUMERIC FIELD STATES A LENGTH WHICH INCLUDES A									
AB 51	A	AB51								SECOND CARD. THE FIRST LINE WILL COMPILE, FOLLOWED									
AB 52	A	AB52								BY 12 BLANKS. THE REST IS TREATED AS A COMMENT.									
AB 53	A	AB53																	000911
AB 54	A	AB54						14		-59969096439550									000925
AB 55	A	AB55								THIS CON, INTENDED AS PART OF A									
AB 56	A	AB56								MESSAGE AND PUNCHED -ERROR ROUTINE, WAS STRIPPED OF									
AB 57	A	AB57								ZONING AND TREATED AS A SIGNED NUMERIC CON BECAUSE									
										THE LEADING DASH WAS INTERPRETED AS A MINUS SIGN.									
AB 58	I	AB58					TITLE			DEFINITION OF A FLOATING POINT CONSTANT - FPN									
AB 59	A	AB59					FPN			£03£123456									000935
										REPRESENTS £123.456									
AB 60	I	AB60								NOTE THAT THE NUMERIC FIELD IS BLANK AND THAT THE MANTISSA IS ONLY									C
AB 61	I	AB61								SIX DIGITS. A LENGTH OF TEN WILL BE ASSUMED AND TRAILING ZEROS ADDED									C
AB 62	I	AB62								TO MAKE AN EIGHT DIGIT MANTISSA. THE FPN APPEARS IN MEMORY AS									C
AB 63	I	AB63								OC12345600 WITH THE UNITS DIGIT SIGNED PLUS. THE LISTING DOES NOT									C
AB 64	I	AB64								SHOW THE ADDED ZEROS OR ASSUMED LENGTH.									C
AB 65	I	AB65																	C
AB 66	I	AB66								INVALID USAGES									C
AB 67	A	AB67								£04£9876543210									000945
AB 68	A	AB68								THIS OPERAND EXCEEDS THE MAXIMUM									
AB 69	A	AB69								LENGTH. THE MANTISSA IS TRUNCATED TO EIGHT DIGITS.									
										IT APPEARS IN MEMORY AS 0D9876543B.									
AB 70	I	AB70																	000955
AB 71	I	AB71																	000965
AB 72	I	AB72																	
AB 73	I	AB73																	
AB 74	I	AB74																	
AB 75	I	AB75																	
AB 76	I	AB76																	
AB 77	I	AB77																	
AB 78	A	AB78								FPN									000975
										££123456									
AB 79	I	AB79																	C
AB 80	I	AB80																	C
AB 81	A	AB81																	000985

INDEX	S	PGLIN	TAG	OP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	005	F	LOC	INSTR	SU	ADDR	SER	REF
AC 01	I	AC01					TITLE	DEFINITION OF A REPORT FORMAT - RPT											
AC 02	I	AC02					THESE ILLUSTRATIONS ALL SHOW EIGHT NUMERIC POSITIONS WITH VARIOUS							C					
AC 03	I	AC03					PUNCTUATION AND SIGN INDICATIONS.							C					
AC 04	I	AC04																	
AC 05	I	AC05					IN THIS SERIES NO COMMAS, DECIMAL POINTS, DOLLAR SIGNS, OR ASTERISKS							C					
AC 06	I	AC06					ARE SPECIFIED. ONE POSITION IS RESERVED FOR A BLANK OR MINUS SIGN.							C					
AC 07	I	AC07					IN THE FIRST FORMAT ALL EIGHT POSITIONS WILL PRINT, LEADING ZEROS							C					
AC 08	I	AC08					INCLUDED. IN THE SECOND FORMAT LEADING ZEROS IN ANY OF THE FIVE HIGH							C					
AC 09	I	AC09					ORDER POSITIONS ARE NOT PRINTED. IN THE THIRD FORMAT, NO LEADING							C					
AC 10	I	AC10					ZEROS WILL PRINT.							C					
AC 11	A	AC11					RPT 9 ZZZZZZZZ							000994					
AC 12	A	AC12					9 XXXXXZZZ							001003				010	
AC 13	A	AC13					9 XXXXXXXX							001012					
AC 14	I	AC14					IN THIS FORMAT VARIOUS EDIT PUNCTUATION IS ADDED. THE DOLLAR SIGN							C					
AC 15	I	AC15					WILL ALWAYS PRINT EIGHT POSITIONS TO THE LEFT OF THE DECIMAL POINT.							C					
AC 16	I	AC16					THE COMMA WILL PRINT IF THERE ARE ANY SIGNIFICANT FIGURES TO THE							C					
AC 17	I	AC17					LEFT OF IT. THE DECIMAL POINT AND THE POSITIONS TO THE RIGHT OF IT							C					
AC 18	I	AC18					WILL ALWAYS PRINT, EVEN FOR A ZERO AMOUNT. A TWO POSITION SIGN							C					
AC 19	I	AC19					INDICATOR IS SPECIFIED AS CR, **, CR DR FOR MINUS, ZERO, OR PLUS							C					
AC 20	I	AC20					AMOUNTS, RESPECTIVELY.							C					
AC 21	A	AC21					RPT 13 \$XXX,XXX.ZZ @@CR@@@DR@							001025					
AC 22	I	AC22					THESE TWO EXAMPLES ILLUSTRATE AMOUNT PROTECTION IN A RPT FORMAT. IN							C					
AC 23	I	AC23					THE FIRST, THE \$ SIGN IS FIXED BUT * WILL PRINT IN ALL SPACES							C					
AC 24	I	AC24					BETWEEN IT AND THE HI-ORDER DIGIT PRINTED. IN THE SECOND, THE \$ SIGN							C					
AC 25	I	AC25					WILL PRINT IMMEDIATELY TO THE LEFT OF THE HI-ORDER DIGIT PRINTED.							C					
AC 26	A	AC26					RPT 12 \$XXX,XXZ.ZZ @*@							001037					
AC 27	A	AC27					12 \$XXX,XXZ.ZZ @#@							001049					
AC 28	I	AC28					THE OPERAND BZ IN THIS EXAMPLE INDICATES THAT THE ENTIRE FIELD,							C					
AC 29	I	AC29					INCLUDING THE DECIMAL POINT AND POSITIONS TO THE RIGHT OF IT, IS TO							C					
AC 30	I	AC30					BE BLANKED IF THE RESULT IS ZERO.							C					
AC 31	A	AC31					RPT 07 XXXX.ZZ@@@BZ@@							001056					
AC 32	I	AC32					INVALID USAGES							C					
AC 33	A	AC33					RPT 9 ZZZXXXXX ZS AND XS REVERSED							001065				011	
AC 34	I	AC34					TITLE COLLECTIVE AREA DEFINITION - NAME												
AC 35	I	AC35					NORMAL USE												
AC 36	A	AC36	NAMEA			NAME	A NAMEAEND							001070			001099	AC50	
AC 37	A	AC37				RCD	2 N							001071				2	
AC 38	A	AC38					4 A							001075				6	
AC 39	A	AC39				CCN	3 XXX							001078				9	
AC 40	A	AC40					1							001079				10	
AC 41	A	AC41				RPT	8 XXXX.ZZ							001087				18	
AC 42	A	AC42				RCD	4 @							001091				22	
AC 43	A	AC43				CCN	3							001094				25	
AC 44	A	AC44				BITCD	2							001095				26	
AC 45	A	AC45	COND1				2												
AC 46	A	AC46	COND2				A												
AC 47	A	AC47				CHRC	D							001096				27	
AC 48	A	AC48	CONDP				P												
AC 49	A	AC49	CONDQ				Q												
AC 50	A	AC50	NAMEAEND			CCN	3 @#							001099				30	
AC 51	J	AC51	NAMEA											001099					
AC 52	A	AC52																	
AC 53	A	AC53																	
AC 54	A	AC54																	
AC 55	A	AC55																	
AC 56	I	AC56					THE FOLLOWING SERIES ILLUSTRATES THE USE OF CONCURRENT NAME							C					
AC 57	I	AC57					DEFINITIONS. NAMEC IS ENTIRELY WITHIN NAMEB. NAMED IS ONLY PARTLY							C					
AC 58	I	AC58					WITHIN NAMEB. BOTH USAGES ARE VALID.							C					
AC 59	A	AC59	NAMEB			NAME	6 NAMEBEND							001100			001129	AC67	
AC 60	A	AC60	RCD				6 @							001105				6	
AC 61	A	AC61	NAMEC			NAME	NAMECEND							001106			001113	AC63	
AC 62	A	AC62				RCD	2 N							001107				8	
AC 63	A	AC63	NAMECEND				6 A							001113				14	
AC 64	J	AC64	NAMEC											001113					
AC 65	A	AC65	NAMED			NAME	NAMEDEND							001114			001136	AC70	
AC 66	A	AC66				RCD	4 A							001117				18	
AC 67	A	AC67	NAMEBEND				12 A							001129				30	
AC 68	J	AC68	NAMEB											001129					
AC 69	A	AC69					3 N							001132					
AC 70	A	AC70	NAMEDEND				4 @							001136					
AC 71	J	AC71	NAMED											001136					

INDEX	S	PGLIN	TAG	OP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	006	F	LOC	INSTR	SU	ADDR	SER	REF
AD 01	I	AD01		TITLE						SPECIAL USES OF NAME STATEMENTS									
AD 02	A	AD02		NAME	0					ALTHOUGH THIS NAME STATEMENT HAS A				001140					
AD 03	A	AD03		CCN	6					BLANK TAG AND OPERAND, THE ZERO				001145				014	
AD 04	A	AD04								IN THE NUMERIC FIELD WILL CAUSE THE CON DEFINITION									
AD 05	A	AD05								WHICH FOLLOWS IT TO BEGIN IN THE NEXT 0/5 MEMORY									
AD 06	A	AD06								LOCATION RATHER THAN THE NEXT SEQUENTIAL LOCATION.									
AD 07	A	AD07		NAME	8					THE B IN THE NUMERIC FIELD OF				001200					
AD 08	A	AD08		RCD	01					THIS NAME STATEMENT CAUSES THE RCD				001200					
AD 09	A	AD09								WHICH FOLLOWS IT TO BEGIN IN THE NEXT 100 LOCATION.									
AD 10	I	AD10								INVALID USAGES				C					
AD 11	A	AD11	NAMEE	NAME			NAMEEEND			THIS NAME IS INVALID BECAUSE IT				001201			001210		AD15
AD 12	A	AD12		RCD	2		A			CONTAINS AN ITEM WHICH IS NOT AN				001202					2
AD 13	A	AD13		CND	2		RH			AREA DEFINITION, CON MISSPELLED.				001209		000000	015		
AD 14	J		NAMEE							THIS DEFINITION OCCUPIES				001209					
AD 15	A	AD14	NAMEEEND	RCD	1					CHARACTER POSITIONS				001210					
AD 16	I	AD15								THIS NAME ENTRY WILL NOT COMPILE CORRECTLY BECAUSE THE NUMERIC				C					
AD 17	I	AD16								FIELD OF THE INTERNAL NAME ENTRY SPECIFIES A STARTING LOCATION NOT				C					
AD 18	I	AD17								IMMEDIATELY FOLLOWING THE PORTION OF THE NAME ENTRY ALREADY DEFINED.				C					
AD 19	A	AD18	NAMEF	NAME	0		NAMEFEND							001215			001222		AD23
AD 20	A	AD19		RCD	2		A							001216					2
AD 21	A	AD20	NAMEG	NAME	4		NAMEFEND							001219			001222		AD23
AD 22	A	AD21		RCD	3									001221					7
AD 23	A	AD22	NAMEFEND		1		N			THIS DEFINITION OCCUPIES				001222					8
AD 24	J		NAMEF							8 CHARACTER POSITIONS				001222					
AD 25	J		NAMEG							4 CHARACTER POSITIONS				001222					
AD 26	A	AD23	NAMEF1	NAME			NAMEF1END			THIS IS INVALID FOR A SIMILAR				001223			001234		AD28
AD 27	A	AD24		RCD	4		A			REASON, THE ACON BREAKS THE				001226					4
AD 28	A	AD25	NAMEF1END	ADCON			CONTINUE			CONTINUITY OF ASSIGNMENT.				001234	A1674		001674	016	AG33
AD 29	J		NAMEF1							THIS DEFINITION OCCUPIES				001234					
AD 30	A	AD26	NAMEH	NAME			NOTEND			THIS WILL NOT COMPILE CORRECTLY				001235					
AD 31	A	AD27		RCD	4		A			BECAUSE THE OPERAND OF THE NAME				001238					4
AD 32	A	AD28		CCN	2					DOES NOT SPECIFY THE TAG OF THE				001240					6
AD 33	A	AD29	NOWEND		3		XXX			ENDING SEGMENT.				001243					9
AD 34	A	AD30		NCP			*			FORCE TERMINATION OF NAMEH				001249	A1249		001249	018	
AD 35	J		NAMEH							THIS DEFINITION OCCUPIES				001249					
AD 36	A	AD31	NAMEI	NAME			NAMEJ			NAMEI IS INVALID BECAUSE IT ENDS				001250			001260		AD42
AD 37	A	AD32		RCD	2		AC			AT THE SAME TAG AT WHICH NAMEJ				001251					2
AD 38	A	AD33			3		E			BEGINS.				001254					5
AD 39	A	AD34	NAMEJ	NAME			NAMEJEND							001255			001260		AD41
AD 40	A	AD35		RCD	05		E							001259					10
AD 41	A	AD36	NAMEJEND		1					THIS DEFINITION OCCUPIES				001260					11
AD 42	J		NAMEJ							6 CHARACTER POSITIONS				001260					
AD 43	A	AD37		NCP			*			FORCE TERMINATION OF NAMEI				001269	A1269		001269	019	
AD 44	J		NAMEI							THIS DEFINITION OCCUPIES				001269					
AD 45	I	AD38		TITLE						SWITCH DEFINITIONS									
AD 46	I	AD39								DATA SWITCHES									
AD 47	I	AD40								CHARACTER CODE - CHRCD									
AD 48	A	AD41	AGE	CHRCO	2	40				A TWO DIGIT CODE WHOSE INITIAL				001271					
AD 49	A	AD42	TWENTY			20				VALUE IS 40, AS SPECIFIED BY									
AD 50	A	AD43	FORTY			40				THE NUMERIC AND OPERAND FIELDS									
AD 51	A	AD44	SIXTY			60				OF THE CHRCD STATEMENT.									
AD 52	A	AD45		RCD	1	N								001272					
AD 53	A	AD46	SEX	CHRCO						A ONE POSITION CODE WILL BE SET UP				001273					
AD 54	A	AD47	MALE			M				WITHOUT INITIALIZATION SINCE A									
AD 55	A	AD48	FEMALE			F				CHRCO WHICH FOLLOWS A RCD WITHOUT									
AD 56	A	AD49								ANY INTERVENING STATEMENTS CAN NOT SPECIFY AN									
AD 57	A	AD50								INITIAL VALUE. IT IS CONSIDERED PART OF THE RCD.									
AD 58	I	AD51		TITLE						BIT CODE - BITCD									
AD 59	A	AD52	PAYTYPE	BITCD		G				A ONE POSITION BIT CODE FIELD WILL				001274					020
AD 60	A	AD53	HOURLY		1					BE DEFINED. THE TITLE ENTRY CAUSES									
AD 61	A	AD54	WEEKLY		2					THE INITIAL VALUE TO BE VALID.									
AD 62	A	AD55	BIWEEKLY		4					ALTHOUGH ALL SIX OF THE SPECIFIED									
AD 63	A	AD56	MONTHLY		8					CODES WILL BE SET UP AND CAN BE									
AD 64	A	AD57	COMMISSION		A					TESTED, THE USE OF THE B OR 8 BIT									
AD 65	A	AD58	FLAT FEE		B					IS QUESTIONABLE SINCE IT MAY									
AD 66	A	AD59								RESULT IN CREATING INVALID CHARACTERS IN MEMORY.									
AD 67	I	AD60								INVALID USAGES				C					
AD 68	A	AD61	SPLIT TAG	RCD	1	A								001275					
AD 69	A	AD62		BITCD		G				THIS BITCD DEFINITION WILL GENERATE				001276					
AD 70	A	AD63	BAD1		1					AND CAN BE REFERENCED BUT WILL NOT									
AD 71	A	AD64	BAD2		2					BE INITIALIZED TO THE VALUE SHOWN.									

INDEX	S	PGLIN	TAG	CP	NU	AT	CP	ERAND	80SMPL-001	08-28-63	COMMENTS	PG	007	F	LOC	INSTR	SU	ADLR	SER	REF
AE 01	I	AE01		TITLE					PROGRAM SWITCHES											
AE 02	A	AE02	SWA	SWT				*E15	PROGRAM SWITCH, INITIALLY ON.						001284	11299		001299	021	
AE 03	A	AE03	SWB	SWN				*E10	PROGRAM SWITCH, INITIALLY OFF.						001289	11299		001299		
AE 04	I	AE04							INVALID USAGES											
AE 05	A	AE05	SWC	NCP				*-10	WHILE THIS GENERATES A SWITCH IT						001294	11284		001284		
AE 06	A	AE06							CANNOT BE REFERENCED BY THE BRANCH CONTROL MACROS											
AE 07	A	AE07							AND WILL NOT APPEAR IN THE SWITCH LISTING IN THE											
AE 08	A	AE08							OPERATORS NOTEBOOK. IF IT IS REFERENCED BY THE											
AE 09	A	AE09							BRANCH CONTROL MACROS IT WILL BE TREATED AS AN A-J											
AE 10	A	AE10							SWITCH AS SHOWN BELOW.											
AE 11	B	AE11		SETCF				SWC												
AE 12	J			RCVS				SWC							001299	11290		001290		AE05
AE 13	J			TMTS	01			#J#							001304	95302	C1	005342		A18
AE 14	I	AE12		TITLE					CONSOLE SWITCHES											
AE 15	A	AE13	ALTSW911	ALTSW		A			THE SYMBOLIC VALUE IN THE TAG WILL											
AE 16	A	AE14	ALTSW912			B			BE ASSIGNED TO THE HARDWARE SWITCH											
AE 17	A	AE15	ALTSW913			C			REPRESENTED BY THE CODE IN THE											
AE 18	A	AE16	ALTSW914			D			NUMERIC FIELD. NOTE THAT											
AE 19	A	AE17	ALTSW915			E			CONTINUATIONS ARE VALID.											
AE 20	A	AE18	ALTSW916			F														
AE 21	I	AE19		TITLE					BRANCH CONTROL MACRO-INSTRUCTIONS											
AE 22	B	AE20	TESTSW	SETCN				SWA#SWB#SIXTY#WEEKLY#COMMISSION#												
AE 23	J		TESTSW	LCD	01			#1#							001309	85303	01	005343		A19
AE 24	J			UNL	01			SWA	-000004						001314	712Y0	C1	001280		AE02
AE 25	J			UNL	01			SWB	-000004						001319	712Y5	C1	001285		AE03
AE 26	J			RCVS				SIXTY							001324	11270		001270		AD51
AE 27	J			TMTS	2			#60#							001329	953M6	02	005346		A21
AE 28	J			SBZ	2			WEEKLY							001334	112P4	02	001274		AD61
AE 29	J			SBZ	A			COMMISSION							001339	115X4	05	001274		AD64
AE 30	B	AE21		IFCN				SWB#TESTSW#EXIT#												
AE 31	J			RCVS				SWB							001344	11285		001285		AE03
AE 32	J			TZB	B			TESTSW							001349	11T-9	06	001309	022	AE23
AE 33	J			TR				EXIT							001354	11599		001599		AF55
AE 34	B	AE22		IFCN				FEMALE#TESTSW#EXIT#												
AE 35	J			LCD	01			#F#							001359	85300	01	005340		A16
AE 36	J			CMP	01			FEMALE							001364	412X3	C1	001273		AD55
AE 37	J			TRE				TESTSW							001369	11309		001309		AE23
AE 38	J			TR				EXIT							001374	11599		001599		AF55
AE 39	B	AE23		SETOF				SWB#HOURLY#WEEKLY#MONTHLY#												
AE 40	J			RCVS				SWB							001379	11285		001285		AE03
AE 41	J			TMTS	1			#E1#							001384	951X6	01	005176		AD02
AE 42	J			SBN	1			HOURLY							001389	11KX4	09	001274		AD60
AE 43	J			SBN	2			WEEKLY							001394	11KP4	10	001274		AD61
AE 44	J			SBN	8			MONTHLY							001399	11E74	12	001274		AD63
AE 45	B	AE24		IFCN				ALTSW912#TESTSW#EXIT#												
AE 46	J			TAB				TESTSW							001404	113-9	02	001309		AE23
AE 47	J			TR				EXIT							001409	11599		001599		AF55
AE 48	I	AE25							INVALID USAGES											
AE 49	I	AE26							THE FOLLOWING MACRO ATTEMPTS TO SET ON TWO UNDEFINED SWITCHES WHICH											
AE 50	I	AE27							ARE THE TAGS OF CHRCD AND BITCD HEADERS. THEY ARE TREATED AS A-J											
AE 51	I	AE28							TYPE SWITCHES.											
AE 52	B	AE29		SETCN				SEX#PAYTYPE#												
AE 53	J			LCD	01			#E1#							001414	851X6	C1	005176	023	AD02
AE 54	J			UNL	01			SEX							001419	712X3	C1	001273		AD53
AE 55	J			UNL	01			PAYTYPE							001424	712X4	C1	001274		AD59
AE 56	I	AE30							THE NEXT MACRO ATTEMPTS TO SET ON AN ALTSW.											
AE 57	B	AE31		SETCN				ALTSW916#												
AE 58	I	AE32							THE FOLLOWING MACRO ATTEMPTS TO INITIALIZE A BITCD USING MOVE MACRO.											
AE 59	B	AE33		MCVE				#G#BAD1#	A BITCD IS NOT VALID AS A MOVE OPERAND.											
AE 60	J			HLT				229000							001429	JRC00		029000		
AE 61	J			ADCCN				#G#							001434	A5341		005341		A17
AE 62	J			ADCCN				BAD1							001439	A1276		001276		AD70

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	008	F	LOC	INSTR	SU	ADDR	SER	REF
AF 01	I	AF01					TITLE	ONE-FOR-ONE INSTRUCTIONS											
AF 02	I	AF02						BASIC OPERANDS											
AF 03	I	AF03						TAG OPERANDS											
AF 04	A	AF04					NCP	SPLIT TAG		SINGLE BLANKS ARE VALID IN TAGS.				001444	A1275		001275		AD68
AF 05	I	AF05						THE MEANING OF A TAG OPERAND DEPENDS ON THE INSTRUCTION AS WELL AS											
AF 06	I	AF06						THE DATA DEFINITION FOR THE TAG.											
AF 07	A	AF07					SET	RCDSOX3		SET ACC TO SIZE OF RCDSOX3.				001449	B0003		000003		AA41
AF 08	A	AF08					LOD	RCDSOX3		LOD ACC WITH VALUE OF RCDSOX3.				001454	B0756		000756		AA41
AF 09	I	AF09						INVALID USAGES											
AF 10	A	AF10					SND	04		Worst cases				001459	/0#00	04	000000		
AF 11	A	AF11					TR			RD/WR				001464	10000		000000		
AF 12	A	AF12	GAP				NCP	*		HERE, GAP HAS A LEADING BLANK.				001469	A1469		001469		
AF 13	A	AF13	GAP				NCP	*		HERE, GAP HAS NO LEADING BLANK.				001474	A1474		001474		
AF 14	A	AF14	RD/WR				SGN	L, GAP		LEADING BLANK ON GAP IS IGNORED.				001479	T1470		001470	024	AF13
AF 15	I	AF15						LITERAL OPERANDS											
AF 16	A	AF16					RAD	#000215#		A FIVE DIGIT SIGNED LITERAL				001484	H5219		005219		DA10
AF 17	A	AF17					LOD	#APPLE#		A FIVE PLACE UNSIGNED LITERAL				001489	B5339		005339		DA15
AF 18	A	AF18					WR	#NOT AVAILABLE#		A FOURTEEN PLACE LITERAL MESSAGE				001494	B5387		005387		DA29
AF 19	A	AF19					CMF	#		TWO BLANKS				001499	A5345		005345		DA20
AF 20	A	AF20					LOD	10		FPN LITERAL &.00005678				001504	B5KK9	10	005229		DA11
AF 21	I	AF21						INVALID USAGES											
AF 22	A	AF22					ADD	E14#		OPENING LIT SYMBOL OMITTED				001509	G0000		000000		
AF 23	A	AF23					LOD	#LOCATIONA#		LITERAL INDICATED WITH TAG OPERAND				001514	B5386		005386		DA28
AF 24	A	AF24								INTENDED.									
AF 25	A	AF25					TRE	#DUPE#		TRANSFER TO A LITERAL				001519	L5358		005358		DA24
AF 26	A	AF26					ADD	#0010#		ADD REQUIRES A SIGNED OPERAND				001524	G5362		005362		DA25
AF 27	A	AF27					LOD	#AGE		CLOSING LIT SYMBOL OMITTED				001529	B5279		005279		DA12
AF 28	A	AF28					WR			#THIS LITERAL OVERFLOWS INTO THE NEXT CARD WHICH IS				001534	B5280		005280		DA13
AF 29	A	AF29								INVALID# NOTE THAT ONLY THE FIRST LINE COMPILES.									
AF 30	A	AF30																	
AF 31	A	AF31					LOD	#-BALANCE#		BECAUSE OF THE DASH THIS LIT WILL				001539	B5193		005193		DA06
AF 32	A	AF32								COMPILE AS BALANCN.									
AF 33	I	AF33						ACTUAL OPERANDS											
AF 34	A	AF34					SET	000005		TWO ALTERNATE WAYS OF WRITING AN				001544	B0005		000005	025	
AF 35	A	AF35					SET	5		INSTRUCTION TO SET ACC TO FIVE.				001549	B0005		000005		
AF 36	I	AF36						INVALID USAGES											
AF 37	A	AF37					ST	995		ST REQUIRES THE @ SIGN FOR ACTUALS				001554	F0000		000000		
AF 38	A	AF38																	
AF 39	A	AF39					WR	@82500		82500 IS OUTSIDE THE MEMORY SIZE				001559	R2500		002500		
AF 40	A	AF40								SPECIFIED FOR THE OBJECT PROGRAM.									
AF 41	A	AF41																	
AF 42	A	AF42					TR	@0001234		ACTUAL EXCEEDS SIX DIGITS				001564	10123		000123		
AF 43	A	AF43					LOD	@APPLES		AN ACTUAL IS INDICATED WHEN A				001569	B0000		000000		
AF 44	A	AF44								LITERAL IS INTENDED.									
AF 45	I	AF45						LOCATION COUNTER OPERANDS											
AF 46	A	AF46					LOD	04	*	THE LOCATION OF THE LOD IS PLACED				001574	B1V74	04	001574		
AF 47	A	AF47								IN ASU 04.									
AF 48	I	AF48						FURTHER EXAMPLES WILL BE SHOWN UNDER CHARACTER ADJUSTMENT.											
AF 49	I	AF49						BLANK OPERANDS											
AF 50	A	AF50					LOCATIONA	BSP		NO ADDRESS IS REQUIRED FOR THESE				001579	B0004		000004		
AF 51	A	AF51						EIM		INSTRUCTIONS. IT IS EITHER IGNORED				001584	,0#-0	06	000000		
AF 52	A	AF52						CNO		OR IS INSERTED BY THE PROCESSOR.				001589	,0-EO	11	000000		
AF 53	A	AF53																	
AF 54	A	AF54					ULA	06	EXIT	HERE THE ADDRESS OF THE TR WILL BE				001594	*1VR9	06	001599		AF55
AF 55	A	AF55					TR			INITIALIZED BY UNLOADING ASU 06.				001599	10000		000000		
AF 56	I	AF56						A SPECIAL CASE OF A LASN WITH BLANK OPERAND WILL BE SHOWN LATER.											

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	009	F	LOC	INSTR	SU	ADDR	SER	REF
AG 01	I	AG01		TITLE						ADDITIONS TO BASIC OPERANDS									
AG 02	I	AG02								CHARACTER ADJUSTMENT									
AG 03	A	AG03		LDD	3		RCDSOX3E3			THESE TWO STATEMENTS REFER TO THE	001604			807E9	03	000759		AA41	
AG 04	A	AG04		TMTS	3		RCDN2X3A-4			SAME DATA FIELD. THE ADJUSTMENTS	001609			907E6	03	000756	026	AA44	
AG 05	A	AG05								ARE BASED ON THE POSITION WITHIN THE TAGGED FIELD									
AG 06	A	AG06								WHICH EACH INSTRUCTION TYPE NORMALLY ADDRESSES.									
AG 07	A	AG07																	
AG 08	A	AG08		LDD	1		CONA-2			THESE STATEMENTS USE ADJUSTMENTS	001614			808#9	01	000809		AB18	
AG 09	A	AG09		CMP	1		#1234567#-2			TO ADDRESS A POSITION WITHIN A	001619			453W7	01	005367		BA26	
AG 10	A	AG10								DEFINED FIELD OR WITHIN A LITERAL.									
AG 11	A	AG11																	
AG 12	A	AG12		TR			CONTINUEE20			TO 4TH INSTR FOLLOWING CONTINUE.	001624			11694		001694		AG33	
AG 13	A	AG13		RCVS			*E6			RCVS AT OP CODE OF 2ND INSTRUCTION	001629			U1635		001635			
AG 14	A	AG14								FOLLOWING.									
AG 15	A	AG15																	
AG 16	A	AG16		SET	6		NAMEA/5			DIVIDE OPERATOR & ADJUSTMENT USED	001634			B0#-6	06	000006		AC51	
AG 17	A	AG17		SND	6		NAMEA			TO GET SET TO MOVE NAMEA BY SND.	001639			/1#P4	06	001074		AC51	
AG 18	I	AG18								CHARACTER ADJUSTMENT TO ADDRESS CONSTANT LITERALS IS A SPECIAL									
AG 19	I	AG19								CASE AND WILL BE ILLUSTRATED LATER.									
AG 20	I	AG20								INVALID USAGES									
AG 21	A	AG21		LDD			*E80005			EXCEEDS SPECIFIED MEMORY SIZE.	001644			81649		001649			
AG 22	A	AG22																	
AG 23	A	AG23		LDA			*E3			GIVES INVALID ADDRESS FOR LDA.	001649			#1652		001652			
AG 24	A	AG24																	
AG 25	A	AG25		TMTS	1		NAMEA-0000030			AN ADJUSTMENT CAN NOT HAVE MORE	001654			91CW7	01	001067		AC51	
AG 26	A	AG26								THAN SIX DIGITS. THIS WILL BE TRUNCATED TO 000003.									
AG 27	A	AG27																	
AG 28	A	AG28		TRE			*E10			THE GENERATED EIA UPSETS THIS ADJUSTMENT	001659			L1669		001669			
AG 29	A	AG29								CALCULATION. TRE CONTINUE WOULD BE CORRECT.									
AG 30	A	AG30		TR		I,EXIT				TO EXIT LINKAGE ON UNEQUAL									
AG 31	*			EIA		EXIT				TO EXIT LINKAGE ON UNEQUAL	001664			,1AR9	10	001599		AF55	
AG 32	*			TR		EXIT				TO EXIT LINKAGE ON UNEQUAL	001669			11599		001599		AF55	
AG 33	A	AG31	CONTINUE	NOP		*					001674			A1674		001674	027		
AG 34	A	AG32																	
AG 35	A	AG33		LDD			#300#E100			ADJUSTMENTS TO A LITERAL MAY NOT BE	001679			85350		005350		BA22	
AG 36	A	AG34								MORE THAN TWO DIGITS. THIS WILL BE TRUNCATED TO 00.									
AG 37	A	AG35																	
AG 38	A	AG36		SET	6		#E1234#*5			ONLY & AND - ARE VALID ADJUSTMENT	001684			B0#-4	06	000004		BA04	
AG 39	A	AG37								OPERATORS FOR ADJUSTMENTS TO LITERALS.									
AG 40	I	AG38		TITLE						OPERAND MODIFIERS									
AG 41	I	AG39								THIS SERIES SHOWS THE USE OF MODIFIERS TO CHANGE THE NORMAL ADDRESS									
AG 42	I	AG40								ORIENTATION OF AN INSTRUCTION. NAMEA IS 30 POSITIONS FROM 1070-1099.									
AG 43	A	AG41		CMP	1		NAMEA			CMP NORMALLY REFERENCES THE RIGHT	001689			41C29	01	001099		AC51	
AG 44	A	AG42		CMP	1		L,NAMEA			HAND CHARACTER.	001694			41CX0	01	001070		AC51	
AG 45	A	AG43		CMP	1		R,NAMEA			REDUNDANT MODIFIER	001699			41C29	01	001099		AC51	
AG 46	A	AG44																	
AG 47	A	AG45		TMTS	1		NAMEA			TMTS NORMALLY REFERENCES THE LEFT	001704			91CX0	01	001070		AC51	
AG 48	A	AG46		TMTS	1		R,NAMEA			HAND CHARACTER.	001709			91C29	01	001099		AC51	
AG 49	A	AG47		TMTS	1		L,NAMEA			REDUNDANT MODIFIER	001714			91CX0	01	001070		AC51	
AG 50	A	AG48																	
AG 51	A	AG49		RCV			NAMEA			RCV NORMALLY REFERENCES THE LEFT	001719			U1C74		001074		AC51	
AG 52	A	AG50		RCVS			H,NAMEA			HAND E4 CHARACTER, RCVS THE LEFT.	001724			U1C74		001074		AC51	
AG 53	A	AG51		RCV			H,NAMEA			REDUNDANT MODIFIER	001729			U1074		001074		AC51	
AG 54	A	AG52																	
AG 55	A	AG53		RCVT			NAMEA			RCVT NORMALLY REFERENCES THE LEFT	001734			U1C79		001079		AC51	
AG 56	A	AG54		RCVS			T,NAMEA			HAND E9 CHARACTER, RCVS THE LEFT,	001739			U1079		001079	028	AC51	
AG 57	A	AG55		RCV			T,NAMEA			AND RCV THE LEFT E4.	001744			U1C79		001079		AC51	
AG 58	A	AG56		RCVT			T,NAMEA			REDUNDANT MODIFIER	001749			U1C79		001079		AC51	
AG 59	A	AG57																	
AG 60	A	AG58		SET			NAMEA			SET NORMALLY REFERENCES SIZE.	001754			B0C30		000030		AC51	
AG 61	A	AG59		SET			S,NAMEA			REDUNDANT MODIFIER	001759			B0C30		000030		AC51	
AG 62	A	AG60		NOP			S,NAMEA			CREATE CONSTANT OF LENGTH OF NAMEA	001764			A0C30		000030		AC51	
AG 63	I	AG61								OPERAND MODIFIERS MAY BE COMBINED WITH CHARACTER ADJUSTMENT.									
AG 64	A	AG62		LDD	1		L,NAMEA#E3			LOAD ASU 1 WITH L.	001769			853T8	01	005338		BA15	
AG 65	I	AG63								INVALID USAGES									
AG 66	A	AG64		TMT			T,NAMEA			THE FIRST FIVE POSITIONS OF NAMEA	001774			91C79		001079		AC51	
AG 67	A	AG65								WILL NOT BE MOVED.									
AG 68	A	AG66																	
AG 69	A	AG67		LDD	05		L,NAMEA#			THE LOAD WILL EXTEND INTO THE NEXT	001779			85TT0	05	005330		BA14	
AG 70	A	AG68								LITERAL WHICH IS NOT USUALLY PREDICTABLE.									
AG 71	A	AG69																	
AG 72	A	AG70		RAD			S,NAMEA			THIS IS THE SAME AS RAD @30 WHEN	001784			H0030		000030		AC51	
AG 73	A	AG71								WHAT WAS INTENDED WAS RAD #E30#.									
AG 74	A	AG72																	
AG 75	A	AG73		TMT			R,NAMEA			THIS MOVES ONLY THE LAST FIVE	001789			91C99		001099		AC51	
AG 76	A	AG74								POSITIONS OF NAMEA.									
AG 77	A	AG75																	
AG 78	A	AG76		TCT			H,NAMEA			H, INCONSISTENT WITH RCVT AND TCT.	001794			,1-74	08	001074		AC51	

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	010	F	LOC	INSTR	SU	ADDR	SER	REF
AH 01	I	AH01					TITLE			INDIRECT ADDRESSING									
AH 02	A	AH02					LEV80												
AH 03	A	AH03	INDEX1				ADCCN			NAMEA				001799	A1099		001099		AC51
AH 04	A	AH04								AN ADDRESS CONSTANT									
AH 05	A	AH05					L0D	6	I,INDEX1	LOAD ASU 6 AT THE ADDRESS NOW IN INDEX1.				001804	81XRZ	06	001799	029	AH03
AH 06	A	AH06																	
AH 07	A	AH07					ENT80			SAME INSTRUCTION IN 80 MODE									
AH 08	A	AH08					L0D	6	I,INDEX1	OPERAND AND COMMENTS REPEAT.									
AH09 *							EIA		INDEX1	OPERAND AND COMMENTS REPEAT.				001809	1PR9	10	001799		AH03
AH10 *							L0D	6	INDEX1	OPERAND AND COMMENTS REPEAT.				001814	81XRZ	06	001799		AH03
AH 11	I	AH09								INVALID USAGES									
AH 12	A	AH10					LEV80												
AH 13	A	AH11	INDEX3				CON	06	001234					001820					
AH 14	A	AH12					RAD		I,INDEX3	INDEX3 IS NOT IN A 4/9 LOCATION.				001829	H182#		001820	030	AH13
AH 15	A	AH13																	
AH 16	A	AH14																	
AH 17	A	AH15					LCD		I,2110234	THE LOCATION OF THE REFERENCED ADDRESS IS ABOVE OBJECT MEMORY LIMITS.				001834	8623U		030234		
AH 18	A	AH16																	
AH 19	A	AH17					ENT80												
AH 20	I	AH18								IN 80 MODE THE TAG OF AN I, WILL BE PUT ON THE GENERATED EIA. ANY ADDRESS MODIFICATION MUST TAKE THIS INTO ACCOUNT.									
AH 21	I	AH19																	
AH 22	A	AH20					LCA	6	EXIT	THIS LDA-ULA IS INTENDED TO MODIFY THE ADDRESS PORTION OF THE LOD.				001839	#1VR9	06	001599		AF55
AH 23	A	AH21					ULA	6	TAGZ					001844	#1YM9	06	001849		AH25
AH 24	A	AH22	TAGZ				L0D		I,INDEX1										
AH25 *			TAGZ				EIA		INDEX1					001849	1PR9	10	001799		AH03
AH26 *							L0D		INDEX1					001854	81799		001799		AH03
AH 27	I	AH23					TITLE			SPECIAL MNEMONICS									
AH 28	I	AH24								ADDRESS CHECK ON LFC-UFC									
AH 29	A	AH25					LFC		LASNTAGA	ON A LFC OR UFC WITH BLANK NUM THE ADDRESS IS 4/9 CHECKED.				001859	51K9	02	005129		AK09
AH 30	A	AH26					LFC		LASNTAGA2					001864	51L1	02	005131		AK09
AH 31	A	AH27					LFC	1	LASNTAGA2	WITH A NUM OF 1 IT IS 1/6 CHECKED				001869	51L1	02	005131		AK09
AH 32	A	AH28					LFC	1	LASNTAGA	ETC. SEE MESSAGES.				001874	51K9	02	005129		AK09
AH 33	I	AH29					TITLE			NUM ON SET BIT INSTRUCTIONS									
AH 34	A	AH30					SBZ	4	220000	NORMAL SBZ 4 WITH 4 IN COL 22.				001879	2-0&0	03	020000		
AH 35	A	AH31					SBZ	4	220000	SB 4 WITH 4 IN COL 22.				001884	2-0&0	03	020000		
AH 36	A	AH32					SBZ	4	220000	SB 4 WITH 4 IN COL 21.				001889	2-0&0	03	020000		
AH 37	A	AH33					SBZ	04	220000	SB 04 REFERENCES 8 BIT				001894	2-#00	04	020000		031
AH 38	A	AH34					SBA		220000	SB 7 BECOMES SBA				001899	2-#E0	07	020000		
AH 39	A	AH35					SBN	4	220000	SBN 04 REFERENCES 4 BIT				001904	2-#E0	11	020000		
AH 40	A	AH36					SBR	10	220000	SBR, NUM IS IGNORED				001909	2-#00	08	020000		
AH 41	A	AH37					SBA	10	220000	SBA, NUM IS IGNORED				001914	2-#E0	07	020000		
AH 42	I	AH38								INVALID USAGES									
AH 43	A	AH39					SBZ	3	220000	SB 3				001919	2-#00		020000		
AH 44	A	AH40					SBZ	5	220000	SB 5				001924	2-#00		020000		
AH 45	A	AH41					SBZ	6	220000	SB 6				001929	2-#00		020000		
AH 46	A	AH42					SBN	9	220000	SB 9				001934	2-#00		020000		
AH 47	I	AH43					TITLE			FLAG CODES									
AH 48	I	AH44								THE FLAG CODES C, R, AND Z, ARE SHOWN ELSEWHERE. CODES I, A, F, T, AND G ARE NOT SHOWN SINCE THEIR EFFECT IS NOT APPARENT HERE.									
AH 49	I	AH45																	
AH 50	I	AH46																	
AH 51	A	AH47					RAD		RCDA	GIVES CAUTIONARY MESSAGE				001939	H0510		000510		AA19
AH 52	A	AH48					RAD		RCDA	FLAG D SUPPRESSES THE MESSAGE				D	001944	H0510		000510	AA19
AH 53	A	AH49																	
AH 54	A	AH50					KCD	1	A					001945					
AH 55	A	AH51					NCP		*	THIS IS THE ONLY INSTR. ON A CARD				001954	A1954		001954	032	
AH 56	A	AH52					NCP		*	FLAG @ FORCES THIS ONTO THE NEXT				@	001959	A1959		001959	033
AH 57	A	AH53																	
AH 58	A	AH54					NOP		EXIT	THIS SPIN LOOP IS EQUIVALENT TO A M				001964	A1599		001599		AF55
AH 59	A	AH55					TR		*-5	HLT. AN INTERRUPT CAN CHANGE THE				H	001969	11964		001964	
AH 60	I	AH56																	
AH 61	I	AH57								NOP TO TR. FLAG M PUTS THE NOP ON THE M FLAG PAGE									
AH 62	I	AH58								OF THE NOTEBOOK, FLAG H PUTS THE TR ON THE H FLAG									
AH 63	I	AH59								PAGE OF THE NOTEBOOK.									
AH 64	A	AH60					LCD		#-BALANCE#	R-L SCAN TREATS - AS DASH NOT NEG.B				001974	85377		005377		PA27



INDEX	S	PGLIN	TAG	OP	NU	AT	OPERAND	805MPL-001	08-28-63	COMMENTS	PG	011	F	LOC	INSTR	SU	ADDR	SER	REF
AI 01	I	AI01					TITLE	MACRO INSTRUCTIONS											
AI 02	I	AI02					THE INSTRUCTIONS GENERATED BY A MACRO DEPEND ON THE DATA CHARACTERISTICS OF THE FIELDS REFERENCED BY THE OPERANDS. THE FIRST CASE, BELOW, ADDS TWO SIMILAR FIELDS AND PLACES THE RESULT IN ONE.												
AI 03	I	AI03											C						
AI 04	I	AI04											C						
AI 05	B	AI05		ADDX			RCDS5X3#RCDS0X3#RCDS5X3#	SIMPLE ADD											
AI06	J			RAD			RCDS0X3							001979	H0756		000756		AA41
AI07	J			ACD			RCDS5X3							001984	G0745		000745		AA37
AI08	J			ST			RCDS5X3							001989	F0745		000745		AA37
AI 09	B	AI06		ADDX			RCDS5X3A#&98765.43#RCDS6X0#	WITH RND AND LNG											
AI10	J			RAD			RCDS5X3A							001994	H0753		000753		AA38
AI11	J			SHR			0000001							001999	C0001		000001		
AI12	J			SET			0000008							002004	B0008		000008		
AI13	J			ADD			#&98765.43#							002009	G5200		005200		AA07
AI14	J			RND			0000002							002014	E0002		000002		
AI15	J			ST			RCDS6X0							002019	F1105		001105		AC60
AI 16	B	AI07		ADDX			RCDS5X3#RCDS5X3A#RCDS5X3#EXIT#TRUNCATE#	OVFLO PROT											
AI17	J			RAD			RCDS5X3A							002024	H0753		000753	034	AA38
AI18	J			SET			0000009							002029	B0009		000009		
AI19	J			ADD			RCDS5X3							002034	G0745		000745		AA37
AI20	J			CMP			XACA	0000008						002039	44030		004030		AQ12
AI21	J			TRH			EXIT							002044	K1599		001599		AF55
AI22	J			SET			0000008							002049	B0008		000008		
AI23	J			ST			RCDS5X3							002054	F0745		000745		AA37
AI 24	B	AI08		ADDX			RCDS5X3#RCDS0X3#XAC1,#&06.02#	SECONDARY FIELD DEF											
AI25	J			RAD			RCDS0X3							002059	H0756		000756		AA41
AI26	J			SET			0000009							002064	B0009		000009		
AI27	J			ADD			RCDS5X3							002069	G0745		000745		AA37
AI28	J			RND			0000001							002074	E0001		000001		
AI29	J			ST			XAC1							002079	F4C21		004021		AQ11
AI 30	B	AI09		MCVE			NAMEB#NAMEA#	ALPHA TO ALPHA											
AI31	J			RCV			NAMEA							002084	U1C74		001074		AC51
AI32	J			SET			0000006							002089	B0006		000006	035	
AI33	J			SND			NAMEB							002094	/1104		001104		AC68
AI 34	B	AI10	MOVE1	MCVE			NAMEA#NAMEB#	ALPHA TO ALPHA HS											
AI35	J		MOVE1	RCV			NAMEB							002099	U1104		001104		AC68
AI36	J		M00019#01	TMT			NAMEA						M	002104	91C74		001074		AC51
AI 37	B	AI11		INCRA			MOVE1#1#&10#	ADDRESS MODIFICATION											
AI38	J			RAD	15		#&10#							002109	H5A68	15	005178		AA03
AI39	J			AAM	15		M00019#01							002114	A2AE4	15	002104		AI36
AI 40	B	AI12		MCVE			CONN5X0#RCDS6X0#	5 DIG UNSIGNED TO 6 DIG SIGNED											
AI41	J			SET			0000005							002119	B0005		000005		
AI42	J			LCD			CONN5X0							002124	B0816		000816		AB19
AI43	J			SET			0000006							002129	B0006		000006		
AI44	J			ST			RCDS6X0							002134	F1105		001105		AC60
AI 45	I	AI13					TITLE	PROGRAM CARD SUPPRESSION WITH S FLAGS											
AI 46	B	AI14	TAGA	MCVE			CONA#CONN5X0#						S						
AI47	J		TAGA	TRANS			#							002139					
AI48	J		T00022#02	RCVS			CONN5X0						M	002139	U0812		000812		AB19
AI49	J		T00022#01	TMTS	05		CONA						M	002144	90Y#7	05	000807		AB18
AI 50	B	AI15	TAGB	MCVE			CONN5X0#CONMIXED#						S						
AI51	J		TAGB	RCVS			CONMIXED							002149	U0817		000817	036	AB20
AI52	J		T00023#01	TMTS	05		CONN5X0						M	002154	90Y/2	05	000812		AB19
AI 53	I	AI16					TITLE	MACROS WITH F FLAGS											
AI 54	B	AI17		MOVEA			TAGA#1#TAGB#1#						F						
AI55	J			LCA	15		T00022#01							002159	#2AD4	15	002144		AI49
AI56	J			ULA	15		T00023#01							002164	#2AE4	15	002154		AI52
AI 57	B	AI18		MOVEA			XACC#TAGC#												
AI58	J			ADD	15		#&00004#							002169	G5BA4	15	005214		AA09
AI59	J			ULA	15		TAGC							002174	#2AH9	15	002189		AI63
AI 60	B	AI19		ACDA			XACC#5#TAGA#2#						F						
AI61	J			ADD	15		#&00001#							002179	G5BE9	15	005209		AA08
AI62	J			ULA	15		T00022#02							002184	#2AC9	15	002139		AI48
AI 63	A	AI20	TAGC	TR										002189	10C00		000000		

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	012	F	LOC	INSTR	SU	ADDR	SER	REF
AJ 01	I	AJ01					TITLE	ADDRESS CONSTANTS											
AJ 02	I	AJ02						ADCON											
AJ 03	A	AJ03	DUMMYTAG	RCD	8	A								002197					
AJ 04	A	AJ04																	
AJ 05	A	AJ05		ADCON			DUMMYTAG	TAG OPERAND						002204	A2197		002197	037	AJ03
AJ 06	A	AJ06		ADCON	13		DUMMYTAG	ADCON WITH ASU ZONING.						002209	A2A27	13	002197		AJ03
AJ 07	A	AJ07		ADCON			DUMMYTAG&4	TAG OPERAND WITH CHARACTER ADJ.						002214	A2201		002201		AJ03
AJ 08	A	AJ08		ADCON		L	DUMMYTAG	WITH OPERAND MODIFIER						002219	A2190		002190		AJ03
AJ 09	A	AJ09		ADCON			#ABLE#	LITERAL OPERAND						002224	A5354		005354		DA23
AJ 10	A	AJ10		ADCON			043155	ACTUAL OPERAND						002229	A315N		043155		
AJ 11	A	AJ11		ADCON			*-55	ADJUSTED LOCATION COUNTER OPERAND						002234	A2179		002179		
AJ 12	I	AJ12						INVALID USAGES											
AJ 13	A	AJ13		ADCON			LASNTAG*35	VALUE OF OPERAND EXCEEDS 160K.						002239	AZ515		019515		AK09
AJ 14	A	AJ14		ADCON	8		043424	THE 8 INTERFERES WITH THE ADDRESS						002244	A342M		043424		
AJ 15	I	AJ15					TITLE	ACON4, ACON5, AND ACON6											
AJ 16	A	AJ16		ACON4			DUMMYTAG	TAG OPERAND						002248	2197		002197		AJ03
AJ 17	A	AJ17		ACON5			DUMMYTAG	TAG OPERAND						002253			002197		AJ03
AJ 18	A	AJ18		ACON6			DUMMYTAG	TAG OPERAND						002259			002197		AJ03
AJ 19	A	AJ19		ACON4	12		DUMMYTAG	ACON4 WITH ASU ZONING						002263	2A97	12	002197		AJ03
AJ 20	A	AJ20		ACON5	8		DUMMYTAG	ACON5 SIGNED PLUS						002268			002197	038	AJ03
AJ 21	A	AJ21		ACON6	-		DUMMYTAG	ACON6 SIGNED MINUS						002274			002197		AJ03
AJ 22	A	AJ22		ACON4			DUMMYTAG&8	WITH CHARACTER ADJUSTMENT						002278	2205		002205		AJ03
AJ 23	A	AJ23		ACON5	L		DUMMYTAG	WITH OPERAND MODIFIER						002283			002190		AJ03
AJ 24	A	AJ24		ACON6	L		DUMMYTAG&2	WITH MODIFIER AND ADJUSTMENT						002289			002192		AJ03
AJ 25	A	AJ25		ACON4	S		DUMMYTAG&1	WITH SIZE MODIFIER AND ADJUSTMENT						002293	0009		000009		AJ03
AJ 26	A	AJ26		ACON5	-		#63957#	SIGNED ACON OF LITERAL. NOTE THAT						002298			005186		DA05
AJ 27	A	AJ27						THE SIGN OF THE ACON IS OPPOSIT TO THE LIT SIGN.											
AJ 28	A	AJ28																	
AJ 29	A	AJ29		ACON6			*E10	ACON OF ADJUSTED LOCATION CTR						002304			002314		
AJ 30	A	AJ30		ACON5			012345	UNSIGNED WITH ACTUAL OPERAND						002309			012345		
AJ 31	I	AJ31						INVALID USAGES											
AJ 32	A	AJ32		ACON4			S,DUMMYTAG-10	ADJUSTMENT IS LARGER THAN S,						002313	0002		000002		AJ03
AJ 33	A	AJ33		ACON4	8		040100	8 SIGN INTERFERES WITH ADDRESS						002317	010-		040100		
AJ 34	A	AJ34		ACON5	15		DUMMYTAG	AN ACON5 CANNOT HAVE ASU ZONING						002322			002197		AJ03
AJ 35	A	AJ35		ACON5			084324	OPERAND TOO LARGE FOR ACON5						002327			004324		
AJ 36	I	AJ36					TITLE	ADDRESS CONSTANT LITERAL											
AJ 37	A	AJ37		LCD	04		R&NAMEA	NON-ARITH, NON-4/9 OPERATION IN 80						002334	85L06	04	005406	039	EA01
AJ 38	A	AJ38						MODE GIVES 6 DIGITS UNSIGNED.											
AJ 39	A	AJ39		ADD			S&NAMEA&10	80 MODE ARITH GIVES 6 DIG SIGNED						002339	G5413		005413		SA01
AJ 40	I	AJ40						ON THE STATEMENT ABOVE NOTE THE WAY THE ADJUSTMENT IS APPLIED. THE											
AJ 41	I	AJ41						VALUE OF S,NAMEA IS 30. THE ADJUSTMENT IS ADDED TO THIS VALUE											
AJ 42	A	AJ42		LEV80				REPEAT SERIES IN 705111 MODE.											
AJ 43	A	AJ43		LOD	04		R&NAMEA	GIVES 5 DIGITS UNSIGNED						002344	85U24	04	005424		-A01
AJ 44	A	AJ44		ADD			S&NAMEA&10	GIVES 5 DIGITS SIGNED						002349	G5419		005419		MA01
AJ 45	A	AJ45																	
AJ 46	A	AJ46		EIA				4/9 OPERATIONS IN ANY MODE GIVE 4						002354	0--0	10	000000		
AJ 47	A	AJ47		ULA	06		R&EXIT	DIGIT UNSIGNED MACHINE ADDRESSES.						002359	*5UL9	06	005439		/A03
AJ 48	A	AJ48		LDA	05		R&12,NAMEA&4	ASU ZONING CAN BE SPECIFIED						002364	*5LU9	05	005449		/A05
AJ 49	A	AJ49		LDA	05		R&15,*E25	AN ADCON LIT OF A LOCATION COUNTER						002369	*5UT4	05	005434		/A02
AJ 50	A	AJ50		LDA	06		R&E1234#	ADDRESS OR OF A LITERAL IS VALID.						002374	*5UK9	06	005429		/A01
AJ 51	I	AJ51						INVALID USAGES											
AJ 52	A	AJ52		TMT			H&NAMEA	THE TMT IS FROM THE ADCON LITERAL						002379	95445		005445		/A04
AJ 53	A	AJ53						RATHER THAN FROM H,NAMEA. ALSO A 4/9 INSTR WITH											
AJ 54	A	AJ54						H, OR T, ORIENTATION GIVES INCONSISTENT ADDRESSING											
AJ 55	A	AJ55						WHEN USED WITH AN ADCON LIT OPERAND.											

INDEX	S	PG	LINE	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	013	F	LOC	INSTR	SU	ADDR	SER	REF
AK 01	I		AK01					TITLE			INSTRUCTIONS TO THE PROCESSOR									
AK 02	I		AK02								ASSIGNMENT STATEMENTS									
AK 03	I		AK03								LASN									
AK 04	I		AK04								THE FOLLOWING EXAMPLES SHOW THE INDEPENDENCE OF THE LASN COUNTERS OF									
AK 05	I		AK05								EACH OTHER AND THEIR RELATION TO THEIR HIGH ASSIGNMENT COUNTERS AND									
AK 06	I		AK06								TO THE LOCATION COUNTER.									
AK 07	A		AK07					RCD	1	A	TO SHOW THE CURRENT VALUE OF THE ASSIGNMENT CTR.					002380				
AK 08	A		AK08					LASN			@5123					005123				
AK 09	A		AK09	LASNTAGA				NCP			*					005129	A5129		005129	040
AK 10	A		AK10					LASN	1		LASNTAGA@20					005145				
AK 11	A		AK11					NCP			*					005149	A5149		005149	041
AK 12	A		AK12					LASN			*					005150				
AK 13	A		AK13					NCP			*					005154	A5154		005154	
AK 14	A		AK14					LASN	1		LASNTAGA@10					005135				
AK 15	A		AK15					NCP			*					005139	A5139		005139	042
AK 16	A		AK16					LASN	1		*					005150				
AK 17	A		AK17					NCP			*					005154	A5154		005154	043
AK 18	A		AK18					LASN	1		LASNTAGA					005125				
AK 19	A		AK19					NCP			*					005129	A5129		005129	044
AK 20	A		AK20					LASN			@5100					005100				
AK 21	A		AK21					NCP			*					005104	A5104		005104	045
AK 22	A		AK22					LASN	1		*					005130				
AK 23	A		AK23					NCP			*					005134	A5134		005134	046
AK 24	A		AK24					LASN			*					005155				
AK 25	A		AK25					NCP			*					005159	A5159		005159	047
AK 26	I		AK26					TITLE			SASN									
AK 27	A		AK27					SASN			LASNTAGA@100					005225				
AK 28	A		AK28					NCP			*					005229	A5229		005229	048
AK 29	A		AK29					LASN			*					005160				
AK 30	A		AK30					NCP			*					005164	A5164		005164	049
AK 31	A		AK31					SASN			@5000					005000				
AK 32	A		AK32					NCP			*					005004	A5004		005004	050
AK 33	A		AK33					LASN			*					005165				
AK 34	A		AK34					NCP			*					005169	A5169		005169	051
AK 35	A		AK35					SASN			@8000					008000				
AK 36	A		AK36					NCP			*					008004	A8004		008004	052
AK 37	I		AK37								INVALID USAGES									
AK 38	A		AK38					LASN			LASNTAGB					005170				
AK 39	A		AK39	LASNTAGB				NCP			*					005174	A5174		005174	053
AK 40	A		AK40					SASN								005175				
AK 41	I		AK41								RASN									
AK 42	A		AK42	OUTSIDE				ADCON			NAMEA					005179	A1099		001099	AC51
AK 43	A		AK43					LASN			@5000					005000				
AK 44	A		AK44					RASN			@15000					015000				
AK 45	A		AK45	RASNA				LCA	06		OUTSIDE					005004	#5/P9	06	005179	054
AK 46	A		AK46					ULA	06		RASNB					005009	*V#J4	06	015014	AK47
AK 47	A		AK47	RASNB				LDD	05							005014	80#0	05	000000	
AK 48	A		AK48					UNL	05		*625					005019	7V#U4	05	015044	
AK 49	A		AK49					LASN			@3000					003000				
AK 50	A		AK50					LDD			RASNA					003004	8V004		015004	055
AK 51	A		AK51																	AK45
											END RASN RANGE									
											REF TO TAG IN RASN RANGE IS									
											AFFECTED.									

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	014	F	LOC	INSTR	SU	ADDR	SER	REF
AL 01	I	AL01		TITLE				SUBOR AND LITOR											
AL 02	A	AL02		SUBOR			OUTSIDE	THESE STATEMENTS ILLUSTRATE WAYS OF						005175					AK42
AL 03	A	AL03		SUBOR			@28704	STATING A STARTING LOCATION FOR						028704					
AL 04	A	AL04		SUBOR			*@1000	SUBROUTINES AND LITERALS. NOTE						004005					
AL 05	A	AL05		LITOR			@35000	THAT THE LAST ASSIGNMENT IS THE ONE						035000					
AL 06	A	AL06		LITOR			OUTSIDE	WHICH IS EFFECTIVE.						005175					AK42
AL 07	I	AL07		TITLE				GENERATE 00 CARD - TCD											
AL 08	A	AL08		TCD				TCD TO BE GENERATED IN MIDDLE OF						000095					
AL 09	A	AL09		SEL			@100	THE PROGRAM. IT READS A CONTROL						000099	20100		000100	056	
AL 10	A	AL10		RD			@1000	CARD AND THEN CONTINUES LOADING.						000104	Y1C00		001000		
AL 11	A	AL11		TR			@0004							000109	10004		000004		
AL 12	A	AL12		CON		5								000114					
AL 13	A	AL13				17	READ CONTROL CARD	COMMENT TO GO ON TCD CARD						000131					
AL 14	A	AL14		LASN				TERMINATE TCD						005175					
AL 15	A	AL15		TCD				TERMINAL TCD TO REPLACE STANDARD						Z 000095					
AL 16	A	AL16		SET		1	1							000099	B0C#1	01	000001		
AL 17	A	AL17		SET		2	2							000104	B0C-2	02	000002		
AL 18	A	AL18		SET		3	3							000109	B0C#3	03	000003		
AL 19	A	AL19		SET		4	4							000114	B0#04	04	000004		
AL 20	A	AL20		SET		5	5							000119	B0##5	05	000005		
AL 21	A	AL21		SET		6	6							000124	B0#-6	06	000006		
AL 22	A	AL22		TR			CONTINUE							000129	11674		001674		AG33
AL 23	A	AL23		LASN				TERMINATE TCD						005175					
AL 24	I	AL24		TITLE				SUBROUTINE CALLS-INCL											
AL 25	A	AL25		INCL			9HEAD	EACH OF THESE STATEMENTS CALLS											
AL 26	A	AL26		INCL			8HEAD	FOR A SUBROUTINE FROM THE LIBRARY											
AL 27	A	AL27		INCL			9HEAD	NOTE THAT EACH SUBROUTINE ONLY											
AL 28	A	AL28		INCL			8HEAD	APPEARS ONCE IN THE PROGRAM, NO											
AL 29	A	AL29		INCL			9HEAD	MATTER HOW OFTEN IT IS CALLED.											
AL 30	I	AL30						INVALID USAGES						C					
AL 31	A	AL31		INCL			NOTIN	SUBROUTINE NOT IN LIBRARY											
AL 32	I	AL32		TITLE				DEFINE A TAG - TRANS											
AL 33	A	AL33	TRANSA	TRANS			500	THE TRANS DEFINES A TAG, WITH AN						000500					
AL 34	A	AL34						ACTUAL, IN THIS CASE.											
AL 35	A	AL35		SEL			TRANSA	REFERENCES TO THE TAG WILL GET						005179	20500		000500	057	AL33
AL 36	A	AL36		LGD			TRANSA	THIS DEFINITION AS THE TAG VALUE.						005184	80500		000500		AL33
AL 37	A	AL37		SET			TRANSA							005189	80500		000500		AL33
AL 38	A	AL38																	
AL 39	A	AL39	TRANSC	TRANS			*@10	A TRANS TO A LOCATION COUNTER						005204					
AL 40	A	AL40		NOP			TRANSC	ADDRESS IS VALID.						005194	A5204		005204		AL39
AL 41	A	AL41																	
AL 42	A	AL42		TR			TRANSB	THIS SERIES ILLUSTRATES A USEFUL						005199	15209		005209		AL44
AL 43	A	AL43		HLT			*	TECHNIQUE FOR WRITING MACRO						005204	J5204		005204		
AL 44	A	AL44	TRANSB	TRANS			*	COMPONENTS, OF USING A TAGGED						005209					
AL 45	A	AL45		NOP			*	TRANS * TO REFERENCE THE NEXT						005209	A5209		005209		
AL 46	A	AL46						IN LINE INSTRUCTION.											
AL 47	A	AL47																	
AL 48	A	AL48	TRANSD	TRANS			NAMEA	TRANS TO TAG OPERAND IS VALID.						001099					AC51
AL 49	A	AL49																	
AL 50	A	AL50	TRANSE	TRANS			L,NAMEA&6	MODIFICATION AND ADJUSTMENT						001076					AC51
AL 51	A	AL51																	
AL 52	A	AL52		RCVS		05	TRANSD	THESE FOUR INSTRUCTIONS SHOW THAT						005214	U1#X0	05	001070		AC51
AL 53	A	AL53		SET			S,TRANSD	BOTH LENGTH AND LOCATION ARE						005219	80030		000030		AC51
AL 54	A	AL54		LGD			TRANSD	OBTAINED WITH A TRANS TO A TAG.						005224	81099		001099		AC51
AL 55	A	AL55		LGD			1 L,TRANSD&4	UNMODIFIED, UNADJUSTED TAG.						005229	810X4	01	001074		AC51
AL 56	I	AL56						INVALID USAGES						C					
AL 57	I	AL57		TAGS DEFINED BY TRANS *, TRANS @, OR A TRANS TO A MODIFIED OR										C					
AL 58	I	AL58		ADJUSTED TAG, SHOW A FIELD LENGTH OF ZERO. MODIFICATION OF SUCH TAGS										C					
AL 59	A	AL59		TMT			TRANSA	IS MEANINGLESS. USE OF SUCH TAGS						005234	90500		000500		AL33
AL 60	A	AL60		TCT			TRANSA	WITH H, T, OR L, ORIENTED						005239	0N00	08	000500		AL33
AL 61	A	AL61		RD			TRANSC	INSTRUCTIONS MAY GIVE INCONSISTENT						005244	Y5204		005204	058	AL39
AL 62	A	AL62		LDA			1 R,TRANSE	ADDRESSING.						005249	#10X6	01	001076		AC51

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	015	F	LOC	INSTR	SU	ADDR	SER	REF
AM 01	I	AL64					TITLE	MULTIPLE LITERALS - LITST, LITND											
AM 02	A	AL65					LITST												
AM 03	A	AL66					RAC	#61#						005254	H5296		005296		AN03
AM 04	A	AL67					LCD	05 #ABCDE#						005259	85TT4	05	005334		AN10
AM 05	A	AL68					LDA	06 R@#612345#						005264	#5T04	06	005364		AN18
AM 06	A	AL69					LCD	L@#WJR111#						005269	85354		005354		AN15
AM 07	A	AL70					LDA	15 S@OUTSIDE						005274	#5CF9	15	005369		AN20
AM 08	A	AL71					RAD	#624#						005279	H5298		005298		AN04
AM 09	A	AL72					RAD	H@#62468013579#						005284	H5349		005349		AN14
AM 10	A	AL73					LCD	T@#ABCDEFGHIJ#						005289	85359		005359		AN16
AM 11	A	AL74					LDA	R@01,#ABC#						005294	#5374		005374		AN22
AM 12	A	AL75					LITND												

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	016	F	LOC	INSTR	SU	ADDR	SER	REF
AN01	*							MULTIPLE LITERAL TABLE NUMBER 0001											
AN02	*							01						005295					
AN03	*		SIGNED LITERAL		01	\	A							005296					
AN04	*		SIGNED LITERAL		02		2D							005298					
AN05	*							01						005299					
AN06	*		SIGNED LITERAL		05		1234E							005304				059	
AN07	*							05						005309					
AN08	*		SIGNED LITERAL		10		246801357I							005319					
AN09	*		UNSIGNED LITERAL		10		ABCDEFGH1J							005329					
AN10	*		UNSIGNED LITERAL		05		ABCDE							005334					
AN11	*		UNSIGNED LITERAL		06		WJR111							005340					
AN12	*		UNSIGNED LITERAL		03		ABC							005343					
AN13	*							01						005344					
AN14	*		2468013	HI-SP	05		0531D							005349				AN08	
AN15	*		WJR111	LEFT	05		05335							005354				AN11	
AN16	*		ABCDEFG	HISP9	05		05329							005359				AN09	
AN17	*							01						005360					
AN18	*		1234E	RIGHT	04		5304							005364				AN06	
AN19	*							01						005365					
AN20	*		OUTSIDE	SIZE	04		0005							005369				060	AK42
AN21	*							01						005370					
AN22	*		ABC	RIGHT	04		53U3							005374				AN12	

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	017	F	LOC	INSTR	SU	ADDR	SER	REF
AO 01	I	AM01					TITLE	ASSEMBLY DOCUMENTATION											
AO 02	I	AM02	THE COMMENTARY ILLUSTRATES THE USE OF TITLE AND COMMENT STATEMENTS											C					
AO 03	I	AM03	TO ENHANCE PROGRAM DOCUMENTATION. NOTE THAT TITLE STATEMENTS WHICH											C					
AO 04	I	AM04	EXTEND BEYOND THE LIMITS OF COL 23 TO COL 73 WILL BE DIVIDED INTO											C					
AO 05	I	AM05	FIELDS AS IN THE EXAMPLE BELOW WHICH WAS ONE WORD, ENTITLED.											C					
AO 06	I	AM06					EN TITLE D												
AO 07	I	AM07	THE COMMENT STATEMENT, A NEW FEATURE OF THE 7080 PROCESSOR, IS											C					
AO 08	I	AM08	DESIGNATED BY A CODE OF C IN THE FLAG FIELD, COL 74. IT MAY EXTEND											C					
AO 09	I	AM09	FROM COL 6 TO COL 73 AND IS NOT OVERPRINTED. AN EXTRA SPACE IS GIVEN											C					
AO 10	I	AM10	BEFORE A COMMENT STATEMENT UNLESS IT FOLLOWS ANOTHER COMMENT ENTRY.											C					
AO 11	I	AM11					TITLE	OVERFLOW CONTROL											
AO 12	I	AM12	PAGE-TO-PAGE OVERFLOW IS NORMALLY UNDER THE CONTROL OF A LINE COUNT											C					
AO 13	I	AM13	WHICH INCLUDES BLANK LINES. IT IS COMPARED TO A MAXIMUM LINE COUNT											C					
AO 14	I	AM14	SPECIFIED IN THE COMMUNICATION WORD AND WHEN THIS MAXIMUM IS REACHED											C					
AO 15	I	AM15	AN OVERFLOW OCCURS.											C					

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	COMMENTS	PG	018	F	LOC	INSTR	SU	ADDR	SER	REF
AP 01	I	AN01					TITLE			EJECT ENTRY									
AP 02	I	AN02	THE STATEMENT IMMEDIATELY PRECEDING THE TITLE EJECT ENTRY HAD THE												C				
AP 03	I	AN03	WORD EJECT IN THE OPERATION FIELD. THIS PRODUCED AN IMMEDIATE PAGE												C				
AP 04	I	AN04	BREAK REGARDLESS OF THE LINE COUNT.												C				

[illegible]

INDEX	S	PGLIN	TAG	OP	NU AT OPERAND	80SMPL-001	08-28-63	COMMENTS	PG 020	F	LOC	INSTR SU	ADDR	SER	REF
AR01	K							THE FOLLOWING ENTRIES ARE CLASS B SUBROUTINES							
AR02	R	AA01	BHEAD	TITLE				THIS TITLE BLOCK APPEARS IN LIEU OF A CLASS B							
AR03	R	AA02						SUBROUTINE.							

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	MESSAGES	PG 001	F	LOC	INSTR	SU	ADDR	REF
AA53		AA53		1000	00		AREA OVER MODE MEM										
AA63		AA63		FIELD	.		JUST NUM										
AA63		AA63		FIELD	.		OP NOT FD										
AA63		AA63		FIELD	.		IMPR NUM IGNRD										
AA68		AA68			2		ASSUME CON										
AB08		AB08			3		STRIPPED DATA										
AB08		AB08			3		NUM NCT EQ TO DESIG										
AB13		AB13			4		STRIPPED DATA										
AB13		AB13			4		STRIPPED DATA										
AB13		AB13			4		NUM NCT EQ TO DESIG										
AB42		AB42			3		STRIPPED CON										
AB54		AB54			14		STRIPPED CON										
AC33		AC33		RPT	9		IMPR RPT										
AC42		AC42		RCD	4		CHK LEFT PROTECTION										22
AD03		AD03		CCN	6		CHK LEFT PROTECTION										
AD13		AD13		CNC	2	RH				NO TAG RH							
AD14			NAMEE				INVAL NAME BEG AD11										
AD35			NAMEH				INVAL NAME BEG AD26										
AD44			NAMEI				INVAL NAME BEG AD31										
AE11				SETCF			OPERND 01 SWITCH TYPE UNKNOWN ASSUME A-J TYPE										
AE52				SETCN			OPERND 01 SWITCH TYPE UNKNOWN ASSUME A-J TYPE										
AE52				SETCN			OPERND 02 SWITCH TYPE UNKNOWN ASSUME A-J TYPE										
AE57				SETCN			OPERND 01 SETOF ALTSW			NO GENERATION							
AE59				MCVE			IMPROPER DATA DEFINITION NO GEN.										
AE59				MCVE			IMPROPERLY WRITTEN										
AF10		AF10		SND	04		WORST CASES			NO TAG WORST CASE							
AF11		AF11		TR			RD/WR			NO TAG RD							
AF11		AF11		TR			ADJ 0										
AF12		AF12	GAP	NCP			JUST TAG										
AF13		AF13	GAP	NCP			*			DUPL TAG AF12							
AF18		AF18		WR			#NOT AVAILABLE #			0/5 CHECK							DA29
AF22		AF22		ACD			E14#			NO TAG E14#							
AF25		AF25		TRE			#DUPE#			4/9 CHECK							DA24
AF25		AF25		TRE			#DUPE#			LIT OPND IMPROPER							DA24
AF26		AF26		ACD			#0010#			SGN CHK L							DA25
AF27		AF27		LCD			NO RT LIT										DA12
AF28		AF28		WR			NO RT LIT										DA13
AF31		AF31		LCD			IMPR SGND LIT										DA06
AF37		AF37		ST			995			NO TAG 995							
AF39		AF39		WR			282500			ADDR OVR 079999							
AF42		AF42		TR			00001234			4/9 CHECK							
AF43		AF43		LCD			IMPR CPND										
AG21		AG21		LCD			*E80005			ADDR OVR 079999							
AG23		AG23		LCA			*E3			4/9 CHECK							
AG35		AG33		LCD			IMPR ADJ TRUNC										DA22
AG35		AG33		LCD			ADJ 0										DA22
AG38		AG36		SET	6		INVAL ADJ OP										DA04
AG72		AG70		RAD			S,NAMEA			SGN CHK							AC51
AG78		AG76		TCT			H,NAMEA			9 CHECK							AC51
AH14		AH12		RAD			I,INDEX3			4/9 CHECK							AH13
AH17		AH15		LCD			I,0110234			ADDR OVR 079999							
AH30		AH26		LFC			LASNTAGA62			4/9 CHECK							AK09
AH32		AH28		LFC	1		LASNTAGA			1/6 CHECK							AK09
AH36		AH32		SBZ	4		JUST NUM										
AH37		AH33		SBZ	04		POSS IMPR NUM										
AH43		AH39		SBZ	3		JUST NUM										
AH43		AH39		SBZ	3		IMPR BIT										
AH44		AH40		SBZ	5		IMPR BIT										
AH45		AH41		SBZ	6		IMPR BIT										
AH46		AH42		SBN	9		IMPR BIT										
AH51		AH47		RAD			RCD A			SGN CHK							AA19
AI46				MCVE			MOVING ALPHA TO NUMERIC										
AJ13		AJ13		ACCCN			LASNTAGA*35			ADDR OVR 079999							AK09
AJ14		AJ14		ACCCN	8		043424			SIGNED ADDR OVER 40K							
AJ32		AJ32		ACON4			S,DUMMYTAG-10			LOWER WRAP ARND							AJ03
AJ33		AJ33		ACON4	8		040100			SIGNED ADDR OVER 40K							
AJ34		AJ34		ACON5	15		DUMMYTAG			ZONE ON ACON5-6							AJ03
AJ34		AJ34		ACON5	15		IMPR NUM IGNRD										AJ03
AJ35		AJ35		ACON5			084324			ADDR OVR 079999							
AJ52		AJ52		TMT			H0NAMEA			4/9 CHECK							AA04
AK38		AK38		LASN			ASSIGN OPND NOT DEFINED										AK39
AK40		AK40		SASN			INVL CPND										
AL31		AL31		INCL			NOT IN CLASS B NAME TABLE										
AL59		AL59		TMT			TRANSA			4/9 CHECK							AL33
AL60		AL60		TCT			TRANSA			9 CHECK							AL33
AL61		AL61		RD			TRANSC			0/5 CHECK							AL39
AL62		AL62		LDA	1		R,TRANSE			4/9 CHECK							AC51

INDEX	S	PGLIN	TAG	OP	NU	AT	OPERAND	80SMPL-001	08-28-63	NO	REQS	PG	002	F	LOC	INSTR	SU	ADDR	REF
AB38		AB38	WORSTCASES	CCN	2		ABCDE												
AC45		AC45	COND1		2														
AC46		AC46	COND2		A														
AC48		AC48	CONDP				P												
AC49		AC49	CONDD				Q												
AD33		AD29	NOWEND		3		XXX												9
AD48		AD41	AGE	CHRC	2		40												
AD49		AD42	TWENTY				20												
AD50		AD43	FORTY				40												
AD54		AD47	MALE				M												
AD62		AD55	BIWEEKLY		4														
AD65		AD58	FLAT FEE		8														
AD71		AD64	BAD2		2														
AF14		AF14	RD/WR	SGN			L, GAP												
AF50		AF50	LOCATIONA	BSP															AF13

INDEX	S	PGLIN	TAG	OP	NU	AT	OPERAND	80SMPL-001	08-28-63	TITLES	PG	003	F	LOC	INSTR	SU	ADDR	REF
AA 01	I	AA01		TITLE			7080 PROCESSOR - SAMPLE ASSEMBLY											
AA 02	I	AA02		TITLE			INTRODUCTION											
AA 13	I	AA13		TITLE			NORMAL ORIGIN											
AA 17	I	AA17		TITLE			AREA DEFINITIONS											
AA 18	I	AA18		TITLE			DEFINITION OF A RECORD FIELD - RCD											
AB 17	I	AB17		TITLE			DEFINITION OF A CONSTANT FIELD - CON											
AB 58	I	AB58		TITLE			DEFINITION OF A FLOATING POINT CONSTANT - FPN											
AC 01	I	AC01		TITLE			DEFINITION OF A REPORT FORMAT - RPT											
AC 04	I	AC04		TITLE			COLLECTIVE AREA DEFINITION - NAME											
AC 34	I	AC34		TITLE			NORMAL USE											
AC 35	I	AC35		TITLE			SPECIAL USES OF NAME STATEMENTS											
AD 01	I	AD01		TITLE			SWITCH DEFINITIONS											
AD 45	I	AD38		TITLE			DATA SWITCHES											
AD 46	I	AD39		TITLE			CHARACTER CODE - CHRC											
AD 47	I	AD40		TITLE			BIT CODE - BITCD											
AD 58	I	AD51		TITLE			PROGRAM SWITCHES											
AE 01	I	AE01		TITLE			CONSOLE SWITCHES											
AE 14	I	AE12		TITLE			BRANCH CONTROL MACRO-INSTRUCTIONS											
AE 21	I	AE19		TITLE			CNE-FOR-ONE INSTRUCTIONS											
AF 01	I	AF01		TITLE			BASIC OPERANDS											
AF 02	I	AF02		TITLE			TAG OPERANDS											
AF 03	I	AF03		TITLE			LITERAL OPERANDS											
AF 15	I	AF15		TITLE			ACTUAL OPERANDS											
AF 33	I	AF33		TITLE			LOCATION COUNTER OPERANDS											
AF 45	I	AF45		TITLE			BLANK OPERANDS											
AF 49	I	AF49		TITLE			ADDITIONS TO BASIC OPERANDS											
AG 01	I	AG01		TITLE			CHARACTER ADJUSTMENT											
AG 02	I	AG02		TITLE			OPERAND MODIFIERS											
AG 40	I	AG38		TITLE			INDIRECT ADDRESSING											
AH 01	I	AH01		TITLE			SPECIAL MNEMONICS											
AH 27	I	AH23		TITLE			ADDRESS CHECK ON LFC-UFC											
AH 28	I	AH24		TITLE			NUM ON SET BIT INSTRUCTIONS											
AH 33	I	AH29		TITLE			FLAG CODES											
AH 47	I	AH43		TITLE			MACRO INSTRUCTIONS											
AI 01	I	AI01		TITLE			PROGRAM CARD SUPPRESSION WITH S FLAGS											
AI 45	I	AI13		TITLE			MACROS WITH F FLAGS											
AI 53	I	AI16		TITLE			ADDRESS CONSTANTS											
AJ 01	I	AJ01		TITLE			ADCON											
AJ 02	I	AJ02		TITLE			ACON4, ACON5, AND ACON6											
AJ 15	I	AJ15		TITLE			ADDRESS CONSTANT LITERAL											
AJ 36	I	AJ36		TITLE			INSTRUCTIONS TO THE PROCESSOR											
AK 01	I	AK01		TITLE			ASSIGNMENT STATEMENTS											
AK 02	I	AK02		TITLE			LASN											
AK 03	I	AK03		TITLE			SASN											
AK 26	I	AK26		TITLE			RASN											
AK 41	I	AK41		TITLE			SUBOR AND LITOR											
AL 01	I	AL01		TITLE			GENERATE OO CARD - TCD											
AL 07	I	AL07		TITLE			SUBROUTINE CALLS-INCL											
AL 24	I	AL24		TITLE			DEFINE A TAG - TRANS											
AL 32	I	AL32		TITLE			MULTIPLE LITERALS - LITST, LITND											
AM 01	I	AL64		TITLE			MULTIPLE LITERAL TABLE NUMBER 0001											
AN01	*			TITLE			ASSEMBLY DOCUMENTATION											
AD 01	I	AM01		TITLE			OVERFLOW CONTROL											
AD 06	I	AM06		TITLE			EJECT ENTRY											
AD 11	I	AM11		TITLE			THE CLASS A SUBROUTINE WHICH FOLLOWS IS CALLED BY											
AP 01	I	AN01		TITLE			THE PROCESSOR. IT CONSISTS OF MACRO INSTRUCTIONS											
AQ02	R	AA01	9HEAD	TITLE			WHICH ARE ONLY GENERATED IF THEY ARE NEEDED.											
AQ03	R	AA02		TITLE			THIS TITLE BLOCK APPEARS IN LIEU OF A CLASS B											
AQ04	R	AA03		TITLE			SUBROUTINE.											
AR02	R	AA01	BHEAD	TITLE														
AR03	R	AA02		TITLE														



[illegible]

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	C FLAG	PG 005	F	LOC	INSTR	SU	ADDR	REF
AH 42	I	AH38								INVALID USAGES							
AH 48	I	AH44								THE FLAG CODES C, R, AND Z, ARE SHOWN ELSEWHERE. CODES I, A, F,							
AH 49	I	AH45								T, AND G ARE NOT SHOWN SINCE THEIR EFFECT IS NOT APPARENT HERE.							
AH 50	I	AH46															
AH 60	I	AH56								NOP TO TR. FLAG M PUTS THE NOP ON THE M FLAG PAGE							
AH 61	I	AH57								OF THE NOTEBOOK, FLAG H PUTS THE TR ON THE H FLAG							
AH 62	I	AH58								PAGE OF THE NOTEBOOK.							
AH 63	I	AH59															
AI 02	I	AI02								THE INSTRUCTIONS GENERATED BY A MACRO DEPEND ON THE DATA							
AI 03	I	AI03								CHARACTERISTICS OF THE FIELDS REFERENCED BY THE OPERANDS. THE FIRST							
AI 04	I	AI04								CASE, BELOW, ADDS TWO SIMILAR FIELDS AND PLACES THE RESULT IN ONE.							
AJ 12	I	AJ12								INVALID USAGES							
AJ 31	I	AJ31								INVALID USAGES							
AJ 40	I	AJ40								ON THE STATEMENT ABOVE NOTE THE WAY THE ADJUSTMENT IS APPLIED. THE							
AJ 41	I	AJ41								VALUE OF S,NAMEA IS 30. THE ADJUSTMENT IS ADDED TO THIS VALUE							
AJ 51	I	AJ51								INVALID USAGES							
AK 04	I	AK04								THE FOLLOWING EXAMPLES SHOW THE INDEPENDENCE OF THE LASN COUNTERS OF							
AK 05	I	AK05								EACH OTHER AND THEIR RELATION TO THEIR HIGH ASSIGNMENT COUNTERS AND							
AK 06	I	AK06								TO THE LOCATION COUNTER.							
AK 37	I	AK37								INVALID USAGES							
AL 30	I	AL30								INVALID USAGES							
AL 56	I	AL56								INVALID USAGES							
AL 57	I	AL57								TAGS DEFINED BY TRANS *, TRANS @, OR A TRANS TO A MODIFIED OR							
AL 58	I	AL58								ADJUSTED TAG, SHOW A FIELD LENGTH OF ZERO. MODIFICATION OF SUCH TAGS							
AO 02	I	AO02								THE COMMENTARY ILLUSTRATES THE USE OF TITLE AND COMMENT STATEMENTS							
AO 03	I	AO03								TO ENHANCE PROGRAM DOCUMENTATION. NOTE THAT TITLE STATEMENTS WHICH							
AO 04	I	AO04								EXTEND BEYOND THE LIMITS OF COL 23 TO COL 73 WILL BE DIVIDED INTO							
AO 05	I	AO05								FIELDS AS IN THE EXAMPLE BELOW WHICH WAS ONE WORD, ENTITLED.							
AO 07	I	AO07								THE COMMENT STATEMENT, A NEW FEATURE OF THE 7080 PROCESSOR, IS							
AO 08	I	AO08								DESIGNATED BY A CODE OF C IN THE FLAG FIELD, COL 74. IT MAY EXTEND							
AO 09	I	AO09								FROM COL 6 TO COL 73 AND IS NOT OVERPRINTED. AN EXTRA SPACE IS GIVEN							
AO 10	I	AO10								BEFORE A COMMENT STATEMENT UNLESS IT FOLLOWS ANOTHER COMMENT ENTRY.							
AO 12	I	AO12								PAGE-TO-PAGE OVERFLOW IS NORMALLY UNDER THE CONTROL OF A LINE COUNT							
AO 13	I	AO13								WHICH INCLUDES BLANK LINES. IT IS COMPARED TO A MAXIMUM LINE COUNT							
AO 14	I	AO14								SPECIFIED IN THE COMMUNICATION WORD AND WHEN THIS MAXIMUM IS REACHED							
AO 15	I	AO15								AN OVERFLOW OCCURS.							
AP 02	I	AP02								THE STATEMENT IMMEDIATELY PRECEDING THE TITLE EJECT ENTRY HAD THE							
AP 03	I	AP03								WORD EJECT IN THE OPERATION FIELD. THIS PRODUCED AN IMMEDIATE PAGE							
AP 04	I	AP04								BREAK REGARDLESS OF THE LINE COUNT.							

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	H FLAG	PG 006	F	LOC	INSTR	SU	ADDR	REF
AE60	J			HLT			@29000						001429	J		029000	
AH 59	A	AH55		TR			*-5			HLT. AN INTERRUPT CAN CHANGE THE			001969	I		001964	
AL 43	A	AL43		HLT			*			TECHNIQUE FOR WRITING MACRO			005204	J		005204	

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	80 SP OP	PG 007	F	LOC	INSTR	SU	ADDR	REF
AH 02	A	AH02		LEV80													
AH 07	A	AH07		ENT80						SAME INSTRUCTION IN 80 MODE							
AH 12	A	AH10		LEV80													
AH 19	A	AH17		ENT80													
AJ 42	A	AJ42		LEV80						REPEAT SERIES IN 705111 MODE.							

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	80 SP I	PG 008	F	LOC	INSTR	SU	ADDR	REF
AG 30	A	AG30		TR			I,EXIT			TO EXIT LINKAGE ON UNEQUAL							
AH 08	A	AH08		L0D			6 I,INDEX1			OPERAND AND COMMENTS REPEAT.							
AH 24	A	AH22	TAGZ	L0D			I,INDEX1										

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	ASSGNS	PG	009	F	LOC	INSTR	SU	ADDR	REF
AK 08	A	AK08		LASN			@5123		SET BLANK CTR TO 5123					005123			005129	
AK 10	A	AK10		LASN	1		LASNTAGA@20		SET CTR 1 TO 5145					005145			005149	AK09
AK 12	A	AK12		LASN			*		SET BLANK CTR TO LOCATION CTR					005150			005154	
AK 14	A	AK14		LASN	1		LASNTAGA@10		SET CTR 1 TO LOWER VALUE					005135			005139	AK09
AK 16	A	AK16		LASN	1				SET CTR 1 TO PREVIOUS HI ASSIGNMENT					005150			005154	
AK 18	A	AK18		LASN	1		LASNTAGA		RESET CTR 1 HI ASSIGNMENT & CTR 1	R				005125			005129	AK09
AK 20	A	AK20		LASN			@5100		SET BLANK CTR TO LOWER VALUE					005100			005104	
AK 22	A	AK22		LASN	1				SET CTR 1 TO NEW HI ASSIGNMENT					005130			005134	
AK 24	A	AK24		LASN					SET BLNK CTR TO BLNK CTR HI ASSIGNMENT					005155			005159	
AK 27	A	AK27		SASN			LASNTAGA@100		SET TO HIGHER THAN LASN BLANK CTR					005225			005229	AK09
AK 29	A	AK29		LASN					RETURN TO BLANK CTR HI ASSIGNMENT					005160			005164	
AK 31	A	AK31		SASN			@5000		SET BELOW LASN BLANK CTR					005000			005004	
AK 33	A	AK33		LASN					RETURN TO BLANK CTR HI ASSIGNMENT					005165			005169	
AK 35	A	AK35		SASN			@8000							008000			008004	
AK 38	A	AK38		LASN			LASNTAGB		A LASN TO A TAG NOT YET DEFINED IS					005170			005174	AK39
AK 40	A	AK40		SASN					SASN BLANK IS IGNORED.					005175			005179	
AK 43	A	AK43		LASN			@5000		ASSEMBLE ROUTINE AT 5000					005000			005000	
AK 44	A	AK44		RASN			@15000		AS IF IT WAS AT 15000					015000			005019	
AK 49	A	AK49		LASN			@3000		END RASN RANGE					003000			000131	
AL 14	A	AL14		LASN					TERMINATE TCO					005175			000129	
AL 23	A	AL23		LASN					TERMINATE TCO					005175			004030	

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	SWITCHES	PG	010	F	LOC	INSTR	SU	ADDR	REF
AE 02	A	AE02	SWA	SWT			*@15		PROGRAM SWITCH, INITIALLY ON.					001284	1		001299	
AE 03	A	AE03	SWB	SWN			*@10		PROGRAM SWITCH, INITIALLY OFF.					001289	A		001299	

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	TRANS	PG	011	F	LOC	INSTR	SU	ADDR	REF
AL 48	A	AL48	TRANSD	TRANS			NAMEA		TRANS TO TAG OPERAND IS VALID.					001099				AC51
AL 50	A	AL50	TRANSE	TRANS			L,NAMEA@6		MODIFICATION AND ADJUSTMENT					001076				AC51

INDEX	S	PGLIN	TAG	CP	NU	AT	OPERAND	80SMPL-001	08-28-63	M FLAG	PG	012	F	LOC	INSTR	SU	ADDR	REF
AA 63	A	AA63		FIELD					THE WORD FIELD, INTENDED AS A					000794	A		000000	
AF 55	A	AF55	EXIT	TR					INITIALIZED BY UNLOADING ASU 06.					001599	1		000000	
AH 58	A	AH54		NCP			EXIT		THIS SPIN LOOP IS EQUIVALENT TO A	M				001964	A		001599	AF55
AI36	J		M00019#01	TMT			NAMEA							002104	9		001074	AC51
AI48	J		T00022#02	RCVS			CONN5X0							002139	L		000812	AB19
AI49	J		T00022#01	TMTS	05		CONA							002144	9	05	000807	AB18
AI52	J		T00023#01	TMTS	05		CONN5X0							002154	9	05	000812	AB19
AI 63	A	AI20	TAGC	TR										002189	1		000000	
AK 47	A	AK47	RASNB	LGD	05				BLANK OPERAND NOT AFFECTED					005014	8	05	000000	

## DEFINITIONS

## REQUESTS

80SMPL-001 08-28-63

PG 013 SYMBOLIC ANALYZER

## SIGNED LITERALS

A	01	AE41	AE53
1E	02	AI38	
123D	04	AJ50	AG38
395G	04	AJ26	
BALANCN	07	AF31	
987654C	07	AI13	
0000A	05	AI61	
0000D	05	AI58	
0021E	05	AF16	
0M56780	10	AF20	

## UNSIGNED LITERALS

AGE	50	AF27	
THIS LI	50	AF28	
ABCDE	05	AG69	
APPLE	05	AG64	AF17
F	01	AE35	
G	01	AE61	
J	01	AE13	
1	01	AE23	
	02	AF19	
60	02	AE27	
300	03	AG35	
ABLE	04	AJ09	
DUPE	04	AF25	
0010	04	AF26	
1234567	07	AG09	
-BALANC	08	AH64	
LOCATIO	09	AF23	
NOT AVA	14	AF18	

## ACTUALS

*002394	AJ49	LDA	05						
0000000	AF43	LOD							
0000001	AI11	SHR	AI28	RND	AL16	SET	1		
0000002	AI14	RND	AI17	SET	2				
0000003	AL18	SET	3						
0000004	AL11	TR	AL19	SET	4				
0000005	AF34	SET	AF35	SET	AI41	SET	AL20	SET	5
0000006	AI32	SET	AI43	SET	AL21	SET	6		
0000008	AI12	SET	AI22	SET					
0000009	AI18	SET	AI26	SET					
0000100	AL09	SEL							

DEFINITIONS	REQUESTS	80SMPL-001 08-28-63				PG 014	SYMBOLIC ANALYZER			
@000123	AF42	TR								
@000500	AL33	TRANSA	TRANS							
@001000	AL10	RD								
@003000	AK49	LASN								
@005000	AK31	SASN	AK43	LASN						
@005100	AK20	LASN								
@005123	AK08	LASN								
@008000	AK35	SASN								
@012345	AJ30	ACCN5								
@015000	AK44	RASN								
@020000	AH34	SBZ	4	AH35	SBZ	4	AH36	SBZ	4	AH37
	AH38	SBA		AH39	SBN	4	AH40	SBR	10	AH41
	AH43	SBZ	3	AH44	SBZ	5	AH45	SBZ	6	AH46
@028704	AL03	SUBOR								
@029000	AE60	HLT								
@035000	AL05	LITOR								
@040100	AJ33	ACON4	E							
@043155	AJ10	ADCON								
@043424	AJ14	ADCON	E							
@082500	AF39	WR								
@084324	AJ35	ACCN5								
@110234	AH17	LOD								

DEFINITIONS	REQUESTS	80SMPL-001 08-28-63		PG 015	SYMBOLIC ANALYZER
MULTIPLE LITERALS					
AN03 A	AM03	RAD			
AN10 ABCDE	AM04	LOD	05		
AN06 1234E	AN18 LITERAL			R,	
AN18 1234E	AM05	LDA	06		
AN11 WJR111	AN15 LITERAL			L,	
AN15 WJR111	AM06	LOD			
AN20 OUTSIDE	AM07	LDA	15		
AN04 2D	AM08	RAD			
AN08 2468013	AN14 LITERAL			H,	
AN14 2468013	AM09	RAD			
AN09 ABCDEFG	AN16 LITERAL			T,	
AN16 ABCDEFG	AM10	LOD			
AN12 ABC	AN22 LITERAL			R,	
AN22 ABC	AM11	LDA			

DEFINITIONS	REQUESTS	80SMPL-001 08-28-63				PG 016	SYMBOLIC ANALYZER			
DESCRIPTIVES										
*ERR*E14#	AF22	ADD								
AD48 AGE										
AD70 BAD1	AE62	ADCON								
AD71 BAD2										
AD62 BIWEEKLY										
AD64 COMMISSION	AE29	SBZ	A							
AB18 CONA	AG08	LOD	1	-000002	AI49	T00022#01	TMTS	05		
AC48 CONDP										
AC49 CONDQ										
AC45 CONDI										
AC46 COND2										
AB20 CONMIXED	AI51 TAGB	RCVS								
AB19 CONN5X0	AI42	LOD		AI48	T00022#02	RCVS		AI52	T00023#01	TMTS 05
AG33 CONTINUE	AG12	TR	E000020	AD28	NAMEFIEND	ADCON		AL22		TR
AJ03 DUMMYTAG	AJ24	ACON6	L,E000002	AJ08		ADCON	L,	AJ23	ACON5	L,
	AJ32	ACON4	S,-000010	AJ25		ACON4	S,E000001	AJ07	ADCON	E000004
	AJ22	ACON4	E000008	AJ05		ADCON		AJ06	ADCON	13
	AJ16	ACON4		AJ17		ACON5		AJ18	ACON6	
	AJ19	ACON4	12	AJ20		ACON5	E	AJ21	ACON6	-
	AJ34	ACON5	15							
AF55 EXIT	AJ47	ULA	06	RAE33		TR		AE38		TR
	AE47	TR		AF54		ULA	06	AG31		EIA
	AG32	TR		AH22		LDA	6	AH58		NOP
	AI21	TRH								
AD55 FEMALE	AE36	CMP	01							
AD65 FLAT FEE										
AD50 FORTY										
AF12 GAP										
AF13 GAP	AF14 RD/WR	SGN	L,							
AD60 HOURLY	AE42	SBN	1							
AH03 INDEX1	AH05	LOD	6	I,AH09		EIA		AH10		LOD 6
	AH25 TAGZ	EIA		AH26		LOD				
AH13 INDEX3	AH14	RAD	I,							
AK09 LASNTAG	AJ13	ADCON		*000035	AH30	LFC	E000002	AH31	LFC	1 E000002
	AK14	LASN	1	E000010	AK10	LASN	1	E000020	AK27	SASN E000100
	AH29	LFC		AH32		LFC	1		AK18	LASN 1
AK39 LASNTAGB	AK38	LASN								
AF50 LOCATIONA										
AD54 MALE										
AD63 MONTHLY	AE44	SBN	8							
AI35 MOVE1										
AI36 M00019#01	AI39	AAM	15							
AC51 NAMEA	AJ39	ADD	S0E000010	AJ37	LOD	04	RAJ44	ADD	S0E000010	
	AJ43	LOD	04	RAJ52	TMT	H0	AJ48	LDA	05	RAE000004
	AG45	CMP	1	RAJ48	TMTS	1	AG75	TMT		R,
	AL50	TRANS	L,E000006	AG44	CMP	1	AG49	TMTS	1	L,
	AG52	RCVS	H,	AG53	RCV	H,	AG78	TCT	H,	
	AG56	RCVS	T,	AG57	RCV	T,	AG58	RCVT	T,	
	AG66	TMT	T,	AG61	SET	S,	AG62	NOP	S,	
	AG72	RAD	S,	AG16	SET	6	/000005	AG25	TMTS	1 -000003
	AG17	SND	6	AG43	CMP	1		AG47	TMTS	1
	AG51	RCV		AG55	RCVT			AG60	SET	
	AH03	ADCON		AI31	RCV			AI36	M00019#01	TMT
	AK42	ADCON		AL48	TRANS					

DEFINITIONS	REQUESTS	80SMPL-001 08-28-63				PG 017	SYMBOLIC ANALYZER	
AC50 NAMEAEND	AC36 NAMEA	NAME	A					
AC68 NAMEB	AI33	SND		AI35 MOVE1	RCV			
AC67 NAMEBEND	AC59 NAMEB	NAME						
AC64 NAMEC								
AC63 NAMECEND	AC61 NAMEC	NAME						
AC71 NAMED								
AC70 NAMEDEND	AC65 NAMED	NAME						
AD14 NAMEE								
AD15 NAMEEEND	AD11 NAMEE	NAME						
AD24 NAMEF								
AD23 NAMEFEND	AD19 NAMEF	NAME	0	AD21 NAMEG	NAME	4		
AD29 NAMEF1								
AD28 NAMEF1END	AD26 NAMEF1	NAME						
AD25 NAMEG								
AD35 NAMEH								
AD44 NAMEI								
AD42 NAMEJ	AD36 NAMEI	NAME						
AD41 NAMEJEND	AD39 NAMEJ	NAME						
*ERR*NOTENC	AD30 NAMEH	NAME						
AD33 NOWEND								
AK42 OUTSIDE	AN20 LITERAL AL06	LITOR	S,	AK45 RASNA	LDA	06	AL02	SUBOR
AD59 PAYTYPE	AE55	UNL	01					
AK45 RASNA	AK50	LOD						
AK47 RASNB	AK46	ULA	06					
AA19 RCDA	AH51	RAD		AH52	RAD			
AA44 RCDN2X3A	AG04	TMTS	3	-000004				
AA41 RCD50X3	AG03 AI06	LOD RAD	3	8000003	AF07 AI25	SET RAD	AF08	LOD
AA37 RCD55X3	AI07 AI23	ADD ST		AI08 AI27	ST ADD		AI19	ADD
AA38 RCD55X3A	AI10	RAD		AI17	RAD			
AC60 RCD56X0	AI15	ST		AI44	ST			
*ERR*RD	AF11	TR						
AF14 RD/WR								
*ERR*RH	AD13	CNO	2					
AD53 SEX	AE54	UNL	01					
AD51 SIXTY	AE26	RCVS						
AD68 SPLIT TAG	AF04	NOP						
AE02 SWA	AE24	UNL	01	-000004				
AE03 SWB	AE25	UNL	01	-000004	AE31	RCVS	AE40	RCVS
AE05 SWC	AE12	RCVS						
AI47 TAGA								
AI51 TAGB								
AI63 TAGC	AI59	ULA	15					
AH25 TAGZ	AH23	ULA	6					

DEFINITIONS	REQUESTS	80SMPL-001 08-28-63		PG 018	SYMBOLIC ANALYZER		
AE23 TESTSW	AE32	TZB	B	AE37	TRE	AE46	TAB
AL33 TRANSA	AL35 AL59	SEL TMT		AL36 AL60	LOD TCT	AL37	SET
AL44 TRANSB	AL42	TR					
AL39 TRANSC	AL40	NOP		AL61	RD		
AL48 TRANSD	AL55 AL54	LOD LOD	1 L, &0000004	AL53	SET	S,	AL52 RCVS 05
AL50 TRANSE	AL62	LDA	1 R,				
AD49 TWENTY							
AI49 T00022#01	AI55	LDA	15				
AI48 T00022#02	AI62	ULA	15				
AI52 T00023#01	AI56	ULA	15				
AD61 WEEKLY	AE28	SBZ	2	AE43	SBN	2	
*ERR*WORST CASE	AF10	SND	04				
AB38 WORSTCASES							
AQ10 XAC							
AQ12 XACA	AI20	CMP	&0000008				
AQ11 XAC1	AI29	ST					
*ERR*995	AF37	ST					



## GLOSSARY OF TERMS

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, numerical, 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 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.

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

Processor library: The portion of the 7080 Processor System tape that contains the elements of each macro-instruction and subroutine.

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

Record: A set of adjacent data fields.

Secondary mode: Any mode other than 7080 mode.

Special characters: The following group of characters: . □ ₣ & \$ \* - / , % # @ + †

- ACON4 Statement 43
- ACON5 Statement 43
- ACON6 Statement 44
- Actual Operand, Defined 33
- Actual Language - See Machine Language
- ADCON Statement 42
- Address, Defined 62
- Address Constant, Defined 9, 42
- Address Constant Literal 44
- Alphabetic Character, Defined 62
- Alphameric Character, Defined 62
- ALTSW Statement 29
- Area-Definition Statement 8, 14
- Arithmetic Operator 91, 34, 42
- Assembly Documentation 59
- Assembly Input 59
- Assembly Output 59
- Asterisk Protection, Defined 20
- Autocoder MODE Statement 53
- Autocoder Operands, Defined 31
  - additions to, multiple additions to 34
- Autocoder Statements, How to Write 11
  
- Basic Programming System for 7080 7
- Bit-Code Switch, Defined 26
  - see also BITCD
- BITCD Statement 27
- Blank Character, Defined 62
- Blank Counter 47
- Blank Operand, Defined 34
- Blank-if-Zero Option 22
  
- Character Adjustment 34, 42
- Character Code Switch, Defined 26
- CHRC D Statement 27
- Class A and B Subroutines 51
- Coding Sheet, How To Use 11
- Collating Sequence, 7080 11
- Comments in Autocoder Statements 12
- Comments Continuation Lines, Rules for Writing
  - in CON 15
  - in RPT 23
  - in switch-definition statements 26
- Comments Flag 57
- CON Statement 17
- Conditional Lozenges 39
- Console Switch, Defined 8, 29
  - see also ALTSW
- Constant 17, 31
  
- Data Field, Defined 62
- Data Switch, Defined 8, 26
  - see also BITCD, CHRC D
  
- EJECT Statement 54
- ENT80 53
- Exponent, Defined 19
  
- Field-Sign Indicators 21
- Fixed Dollar Sign 20
- Flag Characters 13, 57
- Floating Dollar Sign 20
- Floating-Point Number, Defined 18
  - by a literal 32
  - calculations with 18
  - FPN 18
  - RCD 14
- Format Layout, Defined 62
  - RPT 19
  
- FORTTRAN MODE Statement 53
  
- General-Purpose Macro-Instructions 37
- Generated, Defined 62
- Generated Coding, 7080 Mode 53
- Group Marks 15, 17
  
- Hand-Coded, Defined 62
- Higher Languages of 7080 Processor 10, 53
  
- INCL Statement 51
- Indirect Address 35, 53
- Initialization, Defined 62
  - by address constant 42
- Insertions on Coding Sheet 12
- Insignificant Zeros, Defined 19
- Instructions to the Processor 10, 46
- Integer Positions, Defined 62
- Interior Fields of NAME 23
- Internal NAME 24
  
- LASN Statement 48
- Leading Zeros, Defined 19
- Left Protection, Defined 14
- LEV80 53
- Library Subroutine - See Subroutine
- Literal - See Literal Operand
- Literal Constant - See Literal Operand
- Literal Operand, Defined 31, 39
- Literal Sign 31
- Literal Tables 31, 48, 50, 55, 59
  - see also Main Literal Table, Multiple Literal Tables
- LITND Statement 56
- LITOR Statement 50
- LITST Statement 56
- Location, Defined 62
- Location Assignment, by Processor 48
  - see also LASN, RASN, SASN
- Location Counter, Used by Processor 46
  - see also LASN
- Location Counter Operand, Defined
- Lozenges 39
  
- Machine Language, Defined 62
- Macro-Header, Defined 38
- Macro-Instruction, Defined 9
  - general purpose, list of 37
- Macro Suffix Tag
- Main Literal Table 31, 48, 50, 59
- Mantissa, Defined 19
- Mnemonic Codes, 7080 Operations 32
- Mode, Coding for 7080 53
- MODE Statements 53
- Modification, Defined 62
- Multiple Literal Tables 31, 50, 55, 59
  
- NAME Statement 24
- Non-Printing Decimal Point 20
- Numerical Characters, Defined 62
- Numerical Constant 17, 18, 38
  
- Object Program, Defined 6
- Object-Program Card 59
- Object-Program Contents 7, 51, 59
- Object-Program Deck 59
- ON/OFF Statue
  - of a bit 27
  - of a bit code switch 27
  - of a character code switch 27
  - of a program switch 28

One-for-One Instruction, Defined	9, 31	Secondary Field Definition, Use of	38
mnemonic codes for	32	Significant Zeros, Defined	19
additions to basic operand	21, 34	Source Program, Defined	6
Operand Modifier	35, 42	Special Characters, Defined	62
Operation Codes, 7080	32	SUBOR Statement	49
Operator's Notebook	8, 60	SUBRO Statement	49
Overlapping, Defined	46	Subroutine	
		assignment of	49, 51
Processor, 7080	7	Class A and B	52
Processor Library, Defined	62	inclusion in program	51
Program Listing, Contents and Details of	59	Switch Definitions	8, 26
Program Switch, Defined	9, 28	SWN Statement	29
see also SWN, SWT		SWT Statement	29
Pure Decimal, Defined	62	Symbolic Analyzer	8, 60
RASN Statement	49	Tag, Rules for Writing	12
RCD Statement	14	Tag Operand	31, 38
Record, Defined	62	TCD Statement	50
Record Mark	15, 17	TITLE Statement	54
Referencing, Defined	8	Trailing Zeros, Defined	19
Report/File Mode Statement	53	TRANS Statement	52
Reset Character	58	Transfer Card	46, 50
RPT Statement	19	see also TCD	
SASN Statement	48	"00" Transfer Card	46, 50
Secondary Mode, Definition	62		



**International Business Machines Corporation**  
**Data Processing Division**  
**112 East Post Road, White Plains, New York**