File No. S360(Mod.20)-21
Form GC24-9002-5

DPS
TPS

IBM Systems Reference Library

IBM System/360 Model 20 Disk and Tape Programming Systems Assembler Language

This publication provides the information enabling the programmer to write programs in the IBM System/360 Model 20 DPS/TPS Assembler language and the macro language.

The Model 20 Assembler language allows the use of mnemonic operation codes and symbolic representations of storage addresses and other values. A program is written in symbolic language. This program is processed by the DPS/TPS Assembler program, which reads the symbolic statements and produces a program in machine language.

By means of the macro language, the programmer can reduce considerably the amount of repetitive coding required for routines used frequently within a given program or in many different programs. The programmer must code the routine only once and include it in the macro library. He writes a macro instruction at the point in the source program where the routine is required. During assembly, the Assembler reads the macro instruction, extracts the routine from the library, and inserts it in the object program. The programmer can cause the Assembler to tailor the routine to fit the particular problem program by specifying the appropriate symbolic operands in the macro instruction.

The reader of this publication should be familiar with basic programming concepts and with the operating principles of his system as described in the appropriate SRL publications. For a list of pertinent publications see IBM System/360 Model 20, Bibliography, Form GA26-3565.

Sixth Edition (April, 1970)

This is a major revision of, and obsoletes, GC24-9002-4 and Technical Newsletters GN33-9059 and GN33-9076.

Minor changes have been made throughout the text. Many sections have been rearranged to improve readability. Therefore, the table of contents should be studied carefully. A section on Planned Overlay Structure has been added. Changes to the text and small changes to the illustrations are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition applies to release 9 of IBM System/360 Model 20 DPS, to release 13 of IBM System/360 Model 20 TPS, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 Model 20 SRL Newsletter, Form GN20-0361, for the editions that are applicable and current.

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

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

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

Contents

Assembler Language —— Introduction

Computer programs may be expressed either in machine language, i.e., language directly interpreted by the computer, or in a symbolic language which is more meaningful to you, the programmer. The symbolic language, however, must be translated into machine language before the computer can execute the program. This is the function of translator programs such as the Assembler.

Of the various symbolic programming languages, Assembler languages are closest to machine language in form and content.

The Assembler language discussed in this manual is a symbolic programming language for the IBM System/360 Model 20. It enables you to use all IBM System/360 Model 20 machine functions as if you were coding in System/360 Model 20 machine language.

The Assembler program translates or processes (assembles) Assembler-language programs into machine language for execution by the computer. A program written in the Assembler language and used as input to the Assembler program is called the source program; the machine-language program produced as output from the Assembler program is called the object program. The translation or processing procedure performed by the Assembler program to produce the object program is called assembling or assembly.

The entire process is illustrated in Figure 1. The Assembler program is supplied by IBM.

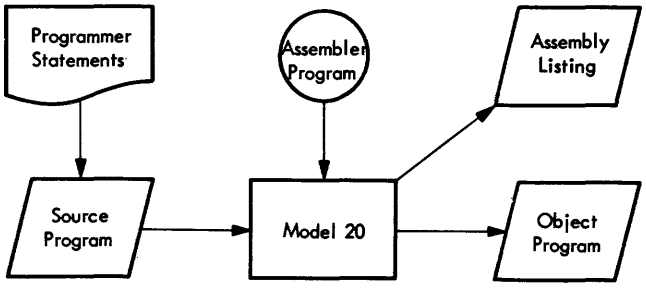


Figure 1. Schematic Representation of the Assembly Process

There are two outputs from the assembler run. The first is an object program consisting of actual machine instructions corresponding to the source program statements written by you. The object program is punched either into cards or it is written on magnetic tape or on disk.

The second output is a program listing or assembly listing. This document shows the original source program statements side by side with the object program instructions created from them. Many programmers work from the assembly program listing as soon as it is available, hardly ever referring to their coding sheets again. An example is shown in Figure 2. This figure is explained below.

(Proceeding from right to left):

a. The items listed under A should be exactly the same as the handwritten entries on the coding sheet. This provides a good check on the accuracy of the keypunching.

b. The items under B are a representation, in hexadecimal notation, of the corresponding instructions and constants.

c. C shows the addresses (in hexadecimal notation) of the instructions, constants, and areas of storage specified by you. For more details see Appendix G.

TYPES OF ASSEMBLER-LANGUAGE STATEMENTS

An assembler-language program may consist of up to four types of statements:

- machine instruction statements (hereafter called machine instructions)
- Assembler-instruction statements (hereafter called Assembler instructions)
- macro-instruction statements (hereafter called macro instructions)
- comments statements (hereafter called comments).

```
LOCATN  OBJECT CODE   ADD1 ADD2 STMT    SOURCE STATEMENT
  C          B                                               A

 0100                           0001          START 256
 0100 0DB0                      0002 BEGIN    BASR  11,0
 0102                           0003          USING *,11
 0102 4880 B01E       0120      0004          LH    8,DATA           LOAD REGISTER 8
 0106 4A80 B022       0124      0005          AH    8,TEN            ADD 10
                                0006 * THE FOLLOWING OPERATION WILL MULTIPLY BY 2.
 010A 1A88                      0007          AR    8,8
 010C 4B80 B020       0122      0008          SH    8,DATA+2      NOTE RELATIVE ADDRESSING
 0110 4080 B024       0126      0009          STH   8,RESULT
 0114 4890 B026       0128      0010          LH    9,BIN1
 0118 4A90 B028       012A      0011          AH    9,BIN2
                                0012 * THE NEXT MACRO INSTRUCTION
                                0013 *  WILL CALL THE END OF JOB MACRO.
                                0014          EOJ
 011C 47F0 00C2       00C2      0015+         BC    15,194(0,0)
                                0016 *
 0120 0019                      0017 DATA     DC    H'25'
 0122 000F                      0018          DC    H'15'
 0124 000A                      0019 TEN      DC    H'10'
 0126                           0020 RESULT   DS    H
 0128 000C                      0021 BIN1     DC    H'12'
 012A 004E                      0022 BIN2     DC    H'78'
 0100                           0023          END   BEGIN
```

Figure 2. Assembly Listing Produced by the Assembly of the Program

Predefined mnemonic codes are provided in the Assembler language for all machine instructions, Assembler instructions, and IBM-supplied macro instructions. Additional extended mnemonics are provided for the various forms of the Branch-on-Condition machine instruction.

The Assembler language provides for the symbolic representation of any addresses, machine components (such as registers), and actual values needed in source statements. Also provided is a variety of forms of data representations: decimal, binary, hexadecimal, or character representation. You can select the representation best suited to express a given data item.

Machine instructions: Machine instructions are one-to-one representations of System/360 Model 20 machine instructions. The Assembler produces an equivalent machine instruction in the object program for each machine instruction in the source program.

Assembler instructions: Assembler instructions specify auxiliary functions to be performed by the Assembler program in addition to its function of translating. These auxiliary functions assist you in

- checking and documenting programs,
- controlling storage-address assignment,
- program sectioning and linking,
- data storage field definition, and
- controlling the Assembler program itself.

With a few exceptions, Assembler instructions do not result in the generation of any machine-language code by the Assembler program.

Macro instructions: Macro instructions cause the Assembler to retrieve a coded symbolic routine, called macro definition, from the macro library, modify the routine according to the information in the macro instruction, and insert the modified routine into the source program for translation into machine language. IBM supplies macro definitions (mainly for input/output operations) as part of the macro library.

You may also define your own macro definitions and refer to them through macro instructions which you define yourself. These definitions and statements are defined according to the macro language and are processed by the Assembler in the same manner as the IBM supplied macro definitions. The macro language is described also in this publication.

Comments: Comments allow you to state, for your own reference or for any other reader of your program, what you intended to be

done in the particular instruction. Your comments should be as precise as possible.

CHARACTER SET

Assembler-language statements may be written using the following alphabetic, numeric, and special characters:

Alphabetic characters: 29 characters are classified as alphabetic characters. These include the characters @, #, and $ as well as the characters A through Z. The three additional characters are included so that the category can accommodate certain non-English languages. (The printer graphic may vary according to the national character set.)

Numeric characters: digits 0 through 9

Special characters: + - , = . * ( ) '/ & blank

These letters, digits, and special characters are only 51 of the 256 EBCDIC (Extended Binary-Coded Decimal Interchange Code) characters. Each of the 256 characters (including the 51 characters above) has a unique card punch code.

Most of the terms used in Assembler-language statements are expressed by the letters, digits, and special characters shown above. However, such Assembler-language features as character self-defining terms and character constants permit the use of any of the 256 card codes. Appendix J shows the 256 EBCDIC character codes.

MAJOR ASSEMBLER LANGUAGE FEATURES

Program Listings: A listing of the source-program statements and the resulting object-program statements is produced by the Assembler for each source program it assembles. You can partly control the form and contents of the listing (see Figure 2).

Error Indications: As a source program is assembled, it is analyzed for actual or potential errors in the use of the Assembler language. Detected errors are indicated in the program listing.

Relocatability: The object programs produced by the Assembler may be in a format enabling relocation from the originally assigned storage area to any other suitable area through the Linkage Editor Program.

Sectioning and Linking: The Assembler language and program provide facilities for partitioning an Assembler-language program into one or more parts called control sections. Because control sections do not have to be loaded contiguously in main storage, a sectioned program may be loaded and executed even though a continuous block of storage large enough to accommodate the entire program is not available.

The linking facilities of the Assembler language and program allow symbols to be defined in one assembly and referred to in another, thus effecting a link between separately assembled programs. This permits you to reference data and/or transfer control between programs.

OPERATING ENVIRONMENT

The Assembler program is either tape- or disk-resident. The TPS Assembler program operates under control of the TPS Basic Monitor program and the DPS Assembler program under the control of the DPS Monitor program. Appendix K contains the minimum and maximum system configuration.

For the TPS Assembler program, the Assembler control card and the associated source-program input must be read on a card reading device. The object program may be punched into cards or written onto tape.

For the DPS Assembler program, the Assembler control card and the associated source-program input may be read on a card reading device or, in card-image format, from a magnetic tape. The object program is placed in the Relocatable Area on the system disk pack and, in addition, may be punched into cards or written onto tape.

The absolute or relocatable object program will then be processed as described in the Model 20 SRL publications describing the DPS and TPS Control and Service Programs (Form numbers GC24-9006 and GC24-9000, respectively).

Assembler Language Coding Conventions

This section discusses the general coding conventions associated with use of the Assembler language.

DESCRIPTION OF CONVENTIONS

Coding Form

A source program is a sequence of source statements punched into cards. The statements may be written on the standard IBM coding form, X28-6509 (Figure 3). One line of coding on the form is punched into one card. The vertical columns on the form correspond to card columns.

Space is provided at the top of the form for program identification. You can also give instructions to the keypunch operator; any character code that does not have a corresponding printer graphic can be assigned any special graphic to identify the code to the keypunch operator, who can then punch the corresponding card punch code wherever he encounters the special graphic. (See under Character Set for the representation of the valid character codes that can be used in a source program.) Neither the program information (Program, Programmer, Date etc.) nor the instructions to the keypunch operator are punched into a card; they are for your own use.

The body of the form is composed of two fields: the statement field, columns 1-71, and the identification-sequence field, columns 73-80. The identification-sequence field is not part of a statement.

Statement Boundaries

Source statements are normally contained in columns 1 - 71 (statement field) of the statement lines. However, macro instructions (and only those) may be continued in columns 16 - 71 of as many continuation lines as required. Therefore, columns 1, 71, and 16 are referred to as the "begin", "end", and "continue" column, respectively.

If a macro instruction line extends beyond column 71 it is to be continued on the next line. This is indicated by a continuation character in column 72. The continuation character may be any non-blank character and is not considered part of the statement coding. The columns of the continuation line preceding the continue column, columns 1-15, must be blank.

The above statement boundaries may be altered by means of the ICTL (Input Format Control) statement discussed later in this publication.

Statement Format

Statements may consist of one to four entries in the statement field. These entries are, from left to right: name, operation, operands, and comments. The entries must be written in the order stated and separated from each other by one or more blanks.

The coding form is ruled to provide an eight-character name field, a five-character operation field, and a 56-character operand and/or comments field.

If you wish, you may disregard boundaries and write the name, operation, operand, and comment entries in other positions, subject to the following rules:

1. The entries must not extend beyond statement boundaries (either the conventional boundaries, or the ones you have designated by means of the ICTL statement).
2. The entries must be in proper sequence, as stated above.
3. The entries must be separated from each other by one or more blanks.
4. If used, a name entry must be written starting in the begin column.

A description of the name, operation, operands, and comments entries follows:

Name: The name (also called label) is a symbol you create yourself to identify a statement or to represent an address or an arbitrary value. Whether a name entry is required, optional, or not permitted depends on the particular statement.

The symbol must consist of eight characters or less; it must be entered with the first character appearing in the begin column. If the begin column is blank, the Assembler program assumes no name has been entered. No blanks must appear within the symbol.

Operation: The operation is a mnemonic code specifying the machine operation or Assembler function desired. An operation entry is mandatory and must start at least one position to the right of the begin column. Valid mnemonic operation codes for machine and Assembler operations are contained in Appendixes A and C of this publication.

Valid operation codes of your self-defined macro instructions must be alphameric and must not be longer than five characters. The leftmost character must be alphabetic. Special characters and/or embedded blanks are not permitted.

Operands: Operands identify and describe data to be acted upon by the instruction; they indicate such things as registers, storage locations, masks, storage-area lengths, or types of data.

Depending on the needs of the instruction, one or more operands may be written. Operands are required for all machine instructions.

Operands must be separated from each other by commas. No blanks are permitted between operands and the separating commas.

Symbols appearing in the operand field of a statement must be defined. A symbol is considered to be defined when it appears either in the name field of a statement or in the operand field of an EXTRN statement.

The operands must not contain embedded blanks. However, if character representation is used to specify a constant, a literal, or immediate data in an operand, the character string may contain blanks.

Comments: Comments are descriptive items of information about the program that are to be inserted in the program listing. All valid characters including blanks (see Character Set) may be used in writing a comment. The entry must not extend beyond the end column (column 71), and at least one blank must separate it from the operand.

An entire line may be used for a comment by placing an asterisk in the begin column. Extensive comments entries may be written by using a series of lines with an asterisk in the begin column of each line.

In statements where either an optional operand is omitted or an operand is not permitted but a comments entry is desired, the absence of the operand must be indicated by a comma preceded and followed by one or more blanks, as follows:

Name	Operation	Operand
	END	, comment

Statement Example: The following example illustrates the use of name, operation, operand, and comments entries. An Add instruction has been named by the symbol ADD; the operation entry (AR) is the mnemonic for a register-to-register add operation, the two operands, eight and nine, designate the two general registers. The comments entry will remind you that you are adding "new sum" to "old" with this instruction.

Name	Operation	Operand
ADD	AR	8,9 NEW SUM TO OLD

Figure 3 shows an example entered on the standard coding form. Since, in this example, the keyboard is assumed not to have a graphic for the character code >, the character code & has been chosen as a substitute. This is indicated to the keypunch operator on the coding sheet.

Identification-Sequence Field

The identification-sequence field of the coding form (columns 73 -- 80) is used to enter program identification and/or statement-sequence characters. The entry is optional. If the field, or a portion of it, is used for program identification, the identification is punched by the user in the statement cards, and reproduced by the Assembler in the printed listing of the source program.

To aid in keeping source statements in order, you may code an ascending sequence of characters in this field or a portion of it. These characters are punched into their respective cards. During assembly, you may request the Assembler to verify this sequence by the use of the ISEQ (Input Sequence Checking) statement. This instruction is discussed later in this publication.

SUMMARY OF CODING CONVENTIONS

The "begin", "end", and "continue" columns are 1, 71, and 16 respectively unless the statement boundaries are altered by means of an ICTL instruction.

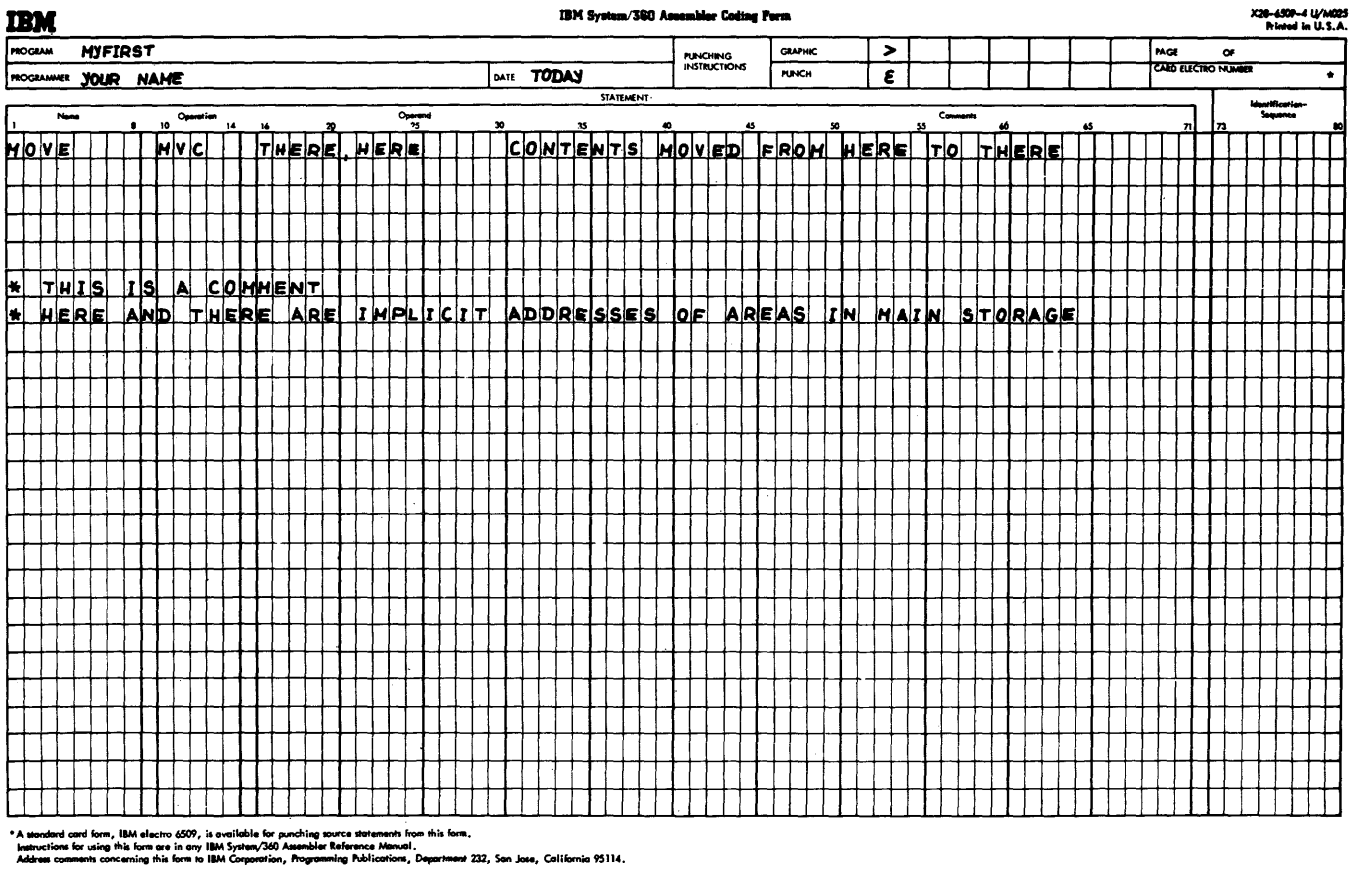
All entries must be contained within the designated begin and end column boundaries. The entries in a statement must always be separated by at least one blank and must be in the following order: name, operation, operand(s), comment.

Depending on the particular statement, a name entry is either required, or optional, or not permitted. Every statement, with the exception of comments statement, requires an operation entry. Operand entries are required for all machine instructions and most Assembler instructions. Comment entries are optional.

The name and operation entries must not contain blanks. Operand entries must not have a blank preceding or following the commas that separate them.

A name entry must always start in the begin column.

Column 72 must be blank, except for macro instructions, for which a continuation punch may be placed in column 72.



IBM

IBM System/360 Assembler Coding Form

X28-6509-4 U/M025
Printed in U.S.A.

PROGRAM MYFIRST
PROGRAMMER YOUR NAME
DATE TODAY
PUNCHING INSTRUCTIONS
GRAPHIC >
PUNCH £
PAGE OF
CARD ELECTRO NUMBER *

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
MOVE	MVC	THERE,HERE	CONTENTS MOVED FROM HERE TO THERE	
* THIS IS A COMMENT				
* HERE AND THERE ARE IMPLICIT ADDRESSES OF AREAS IN MAIN STORAGE				

*A standard card form, IBM electro 6509, is available for punching source statements from this form.
Instructions for using this form are in any IBM System/360 Assembler Reference Manual.
Address comments concerning this form to IBM Corporation, Programming Publications, Department 232, San Jose, California 95114.

Figure 3. Coding Form

Terms and Expressions

An operand is composed of one or more expressions, which, in turn, are composed of a term or an arithmetic combination of terms.

Terms and expressions are used in operands to define storage locations, general registers, immediate data, or constant values.

Terms

All terms represent a value. This value may be assigned by the Assembler program (symbols, symbol length attribute, Location Counter reference) or may be inherent in the term itself (self-defining terms).

Terms are classed as absolute or relocatable. They are absolute or relocatable according to the effect of program relocation upon them.

Program relocation is defined as:

- either reassembling the program with a different starting address
- or relocating the program - by means of the Linkage Editor Program - to storage locations other than those originally assigned by the Assembler program.

A term is absolute if its value does not change upon relocation. A term is relocatable if its value changes by n when the program is relocated n bytes away from the location where it is first assembled.

The section below discusses each type of term and the rules for its use.

SELF-DEFINING TERMS

A self-defining term is one whose value is inherent in the term. It is not assigned a value by the Assembler program. For example, the decimal self-defining term 15 represents a value of fifteen.

There are four types of self-defining terms: decimal, hexadecimal, binary, and character. Accordingly, we speak of decimal, hexadecimal, binary, or character representation of the machine-language binary value (or bit configuration) a term represents.

Self-defining terms are classed as absolute terms since the value they represent does not change upon program relocation.

Using Self-Defining Terms: Self-defining terms are the means of specifying machine-language binary values or bit configurations without equating the value to a symbol.

Self-defining terms may be used to specify such program elements as immediate data, masks, registers, and addresses. The type of term selected (decimal, hexadecimal, binary, or character) depends on what is being specified.

Self-defining terms are not to be confused with data constants or literals. When a self-defining term is used in a machine instruction, its value is assembled into the instruction. When a data constant or literal is specified in the operand of an instruction, its address is assembled into the instruction.

Limitations on the value of the term depend on its use. For example, a decimal term that designates a general register should have a value between 8 and 15 inclusively; one that represents a displacement should not exceed 4095.

Decimal Self-Defining Term: A decimal self-defining term is an unsigned decimal number written as a sequence of decimal digits. High-order zeros may be used.

A decimal term must not consist of more than five digits, or exceed 32,767 (2^{15}-1). A decimal term is assembled as its binary equivalent.

Some examples of decimal self-defining terms are: 8, 147, 4092, 00021.

Hexadecimal Self-defining Term: A hexadecimal self-defining term is an unsigned hexadecimal number written as a sequence of hexadecimal digits. The digits must be enclosed in apostrophes and preceded by the letter X; for example, X'C49'.

Each hexadecimal digit is assembled as its four-bit binary equivalent. Thus, a hexadecimal term used to represent an eight-bit mask would consist of two hexadecimal digits. The maximum value of a hexadecimal term is X'7FFF'.

The hexadecimal digits and their bit patterns are as follows:

Hex Dig.	Pattern	Hex Dig.	Pattern
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

A table for converting hexadecimal to decimal representations is provided in Appendix L.

A hexadecimal self-defining term that is not specified as a complete byte is assembled as one byte. The specified bits are assembled right-justified, and the portion of the byte not specified is padded with binary zeros. For example, X'F' would be assembled as 00001111.

Binary Self-Defining Term: A binary self-defining term is written as an unsigned sequence of ones and zeros enclosed in apostrophes and preceded by the letter B. For example, B'10001101'. This term would appear in storage as shown within the apostrophes and occupy one byte. A binary term may have up to eight bits represented.

Binary representation is used primarily in designating bit patterns of masks or in logical operations.

The following example illustrates a binary term used as a mask in a TM (Test-Under-Mask instruction. The contents of GAMMA are to be tested, bit by bit, against the pattern of bits represented by the binary term.

Name	Operation	Operand
ALPHA	TM	GAMMA,B'10101101'

A binary self-defining term that is not specified as a complete byte is assembled as one byte. The specified bits are assembled right-justified, and the portion of the byte not specified is padded with binary zeros. For example, B'101011' would be assembled as 00101011.

Character Self-Defining Term: A character self-defining term consists of one character enclosed by apostrophes and preceded by the letter C. All letters, decimal digits, and special characters may be used in a character term. In addition, any of the 256 punch combinations (shown in Appendix J) may be used in a character self-defining term. Examples of character self-defining terms are as follows:

```
C'/'                C' ' (blank)
C'A'                C'1'
```

Because of the use of apostrophes in the Assembler language and ampersands in the macro language as syntactic characters, the following rule must be observed when using these characters in a character term:

For each apostrophe or ampersand desired in a character term, two apostrophes or ampersands must be written. For example, the character value ' would be written as C'''' and the value & as C'&&'.

The character is assembled as its eight-bit code equivalent (see Appendix J). The two apostrophes or ampersands that must be used to represent an apostrophe or an ampersand are assembled as one apostrophe or ampersand.

ASSEMBLER PROGRAM DEFINED TERMS

Terms whose value depends on the Assembler program are classified as Assembler program defined terms although you actually create them yourself. The classification is made to distinguish these terms from the self-defining terms.

Symbols

A symbol is a character or combination of characters used to identify a statement or to represent addresses or arbitrary values.

Symbols are used as names and in operands to provide you with an efficient way to name and to refer to a program statement. A symbol, which you create for use as a name entry or in an operand, must conform to the following rules:

1. The symbol must not consist of more than eight characters, the first of which must be alphabetic. The other characters may be letters, digits, or a combination of the two. Since symbols used by IOCS begin with I, symbols in problem programs should not begin with the letter I. Also, the symbol or the first portion of a symbol (up to seven characters) in problem programs should not be the same as the file name in a DTF header entry. (For further details, refer to the SRL publications describing the pertinent Input/Output Control System.)

2. No special characters are permitted in a symbol.

3. No blanks are permitted in a symbol.

The following are examples of valid symbols:

READER	LOOP2	$13
A23456	N	@PRICE
X4F2	S4	#LB1

The following symbols are invalid, for the reasons noted:

256B	first character is not alphabetic
RECORDAREA2	more than eight characters
BCD*34	contains a special character, namely *
IN AREA	contains a blank

Defining Symbols: A symbol is defined when it appears as the name of a source statement or as the operand of an EXTRN statement. The Assembler program assigns a value to each symbol appearing as a name entry in a source statement. The value assigned to symbols naming storage areas, machine instructions, constants, and control sections represents the address of the leftmost byte of the storage field containing the named item. Since the addresses of these items change upon program relocation, the symbols naming them are relocatable terms.

A symbol used as a name entry in the EQU (Equate Symbol) Assembler instruction is assigned the value stated as the operand of the instruction. Since the operand may represent either a relocatable or an absolute value, the symbol is considered a relocatable or absolute term depending upon the value to which it is equated.

The value of a relocatable symbol may vary between 0 and 2^{15}-1 (=32767). The value of absolute symbols may vary between -2^{15} (=-32768) and 2^{15}-1 (=32767).

Symbol definition also involves the assignment of a length attribute to the symbol. (The Assembler program maintains an internal table, the symbol table, in which the values and attributes of symbols are kept. When the Assembler program encounters a symbol in an operand, it refers to the table for the values associated with the symbol.) The length attribute of a symbol is the size, in bytes, of the storage field whose address is represented by the symbol. For example, a symbol naming an instruction that occupies four bytes of storage has a length attribute of four.

Normally, symbols are defined in the same program in which they are used as operands. However, you can define a symbol in one program and use it in another program that was assembled separately from the first (see under Symbolic Linkages).

Previously Defined Symbols: A symbol is called "previously defined" if it has appeared as a name in an instruction or as the operand in an EXTRN statement prior to being used as an operand in a different instruction. Symbols used in the operands of the Assembler instructions ORG and EQU must have been previously defined.

General Restrictions On Symbols: A symbol may be defined only once in an assembly. That is, each symbol used as the name of a statement or as the operand of an EXTRN instruction must be unique to that assembly.

Symbol Length Attribute Reference (absolute)

The length attribute may be used as a term. Reference to the attribute is made by coding L' followed by the symbol, e.g., L'BETA. The L'.... term allows coding where lengths are unknown.

The following example illustrates the use of L'symbol in moving a character constant into either the high-order or low-order end of a storage field.

Name	Operation	Operand
A1	DS	CL8
B2	DC	CL2'AB'
HIORD	MVC	A1(L'B2),B2
LOORD	MVC	A1+L'A1-L'B2(L'B2),B2

A1 names a storage field eight bytes in length and is assigned a length attribute of eight. B2 names a character constant two bytes in length and is assigned a length attribute of two. The statement named HIORD moves the contents of B2 into the leftmost two bytes of A1. The term L'B2 in parentheses provides the length specification required by the instruction. When the instruction is assembled, the length is placed into the proper field of the machine instruction.

LOORD moves the contents of B2 into the right-most two bytes of A1. A1+L'A1-L'B2 results in the addition of the length of A1 to the beginning address of A1, and the subtraction of the length of B2 from this value. The result is the address of the seventh byte in field A1. The constant represented by B2 is moved into A1 starting at this address. L'B2 in parentheses provides length specification as in HIORD.

Location Counter Reference

You may refer to the current value of the location counter at any place in a program, by using an asterisk in an operand. The asterisk represents the current value of the location counter.

Using an asterisk in a machine instruction or DC-instruction is the same as placing a symbol in the name field of the particular instruction and then using that symbol rather than the asterisk in the operand.

A reference to the location counter must not be made in an address constant specified in literal form.

The Location Counter: In each control section a location counter is used to assign storage addresses to program instructions occupying storage. As each machine or DC-instruction or data area is assembled, the location counter is first adjusted to the proper boundary for the item, if adjustment is necessary. After the instruction is assembled the location counter is incremented by the length of the assembled item. Thus, it always points to the next available location. If an instruction is named by a symbol, the value attribute of the symbol is the value of the location counter after boundary adjustment, but before addition of the length.

The location counter setting can be controlled by using the START and ORG Assembler instructions, which are described under Program Sectioning and Linking. The counter affected by either of these Assembler instructions is the counter for the control section in which they appear. The maximum value for the location counter is $2^{15}-1$ (=32767).

Expressions

Expressions are operand entries consisting of either a single term or an arithmetic combination of terms.

Up to three terms can be combined with the following arithmetic operators:

- `+` addition, e.g., ALPHA+2
- `-` subtraction, e.g., ALPHA-BETA
- `*` multiplication, e.g., 5*L'DATA

Note: The character * (asterisk) has two meanings when used in an operand:

1. Reference to the location counter (in this case it is not an operator).
2. Arithmetic operator (multiplication).

Two of the terms within a 3-term expression can be grouped within parentheses to indicate the order in which they are to be evaluated. When terms in parentheses are encountered in combination with another term, the combination of terms inside the parentheses is first reduced to a single value. This value then is used in reducing the rest of the expression to another single value.

The rules for combining terms are discussed under Absolute and Relocatable Expressions. In addition to these, the following rules apply to the coding of expressions:

1. An expression must not start with an arithmetic operator (+,-,*).
2. An expression must not contain two terms or two operators in succession.
3. An expression must not consist of more than 3 terms.
4. An expression must not have more than one pair of parentheses.
5. A multi-term expression must not contain a literal.

The following are examples of valid expressions:

```
AREA1+X'2D'   (EXIT-ENTRY)*8    29
*+32          =H'1234'          L'FIELD
N-25          L'BETA*10         C'A'
FIELD         B'101'            LAMBDA+GAMMA
FIELD+332
```

In the example *+32, the asterisk is not used as an operator.

EVALUATION OF EXPRESSIONS

A single term expression, e.g., 29, BETA, *, or L'SYMBOL, takes on the value of the term involved. A multi-term expression (e.g., BETA+10, ENTRY-EXIT, 10+A*B) is reduced to a single value, as follows:

1. Each term is given its value.
2. Expressions within parentheses are evaluated first.
3. Arithmetic operations are performed left to right. Multiplication is done before addition and subtraction, e.g., A+B*C is evaluated as A+(B*C), not (A+B)*C. The computed result is the value of the expression.

Final values of expressions representing storage addresses may vary between 0 and $2^{15}-1$. However, intermediate results may

vary between -2^{15} (=-32768) and $2^{15}-1$ (=32767).

ABSOLUTE AND RELOCATABLE EXPRESSIONS

An expression is called absolute if its value is not affected by program relocation. An expression is called relocatable if its value changes upon program relocation. The two types of expressions, absolute and relocatable, take on these characteristics from the term or terms they contain.

Two terms of an expression are said to be paired if both are relocatable, defined in the same control section, and have opposite signs. Any other term of an expression is called unpaired.

Absolute Expressions

An absolute expression may be an absolute term or any arithmetic combination of absolute terms. An absolute term may be an absolute symbol, any of the self-defining terms, or the length attribute reference. Addition, subtraction, and multiplication are permitted between absolute terms.

An absolute expression may contain two relocatable terms (RT) -- alone or in combination with an absolute term (AT) -- under the following conditions:

1. The relocatable terms must be paired. The paired terms do not have to be contiguous, e.g., RT+AT-RT.

2. No relocatable term must enter into a multiply operation. Thus, RT-RT*10 is invalid. However, (RT-RT)*10 is valid.

The pairing of relocatable terms cancels the effect of relocation. Therefore, the value represented by the paired terms remains constant, regardless of program relocation. For example, in the absolute expression $A-R_1+R_2$, A is an absolute term, and R_2 and R_1 are relocatable terms from the same control section. If A = 50, R_1 = 25, and R_2 = 10, the value of the expression would be 35. If R_2 and R_1 are relocated by a factor of 100 their values would then be 125 and 110. However, the expression would still be evaluated as 35 (50-125+110=35).

Absolute expressions are reduced to a single absolute value. Absolute expressions may only be negative in address constants (see DC instruction).

The following examples illustrate absolute expressions. A is an absolute term; R_2 and R_1 are relocatable terms from the same control section.

$A-R_1+R_2$
A
A*A
R_2-R_1+A
$*-R_1$ (a reference to the location counter is paired with another relocatable term from the same control section).

Relocatable Expressions

A relocatable expression is one whose value would change by n if the program in which it appears is relocated n bytes away from its originally assigned area of storage. All relocatable expressions have a positive value.

A relocatable expression may be a relocatable term. A relocatable expression may also contain several relocatable terms -- alone or in combination with absolute terms -- under the following conditions:

1. There must be an odd number, 1 or 3, of relocatable terms.

2. If a relocatable expression contains three relocatable terms, two of them must be paired.

3. The unpaired term must be positive.

4. Relocatable terms must not enter into multiply operations.

A relocatable expression is reduced to a single relocatable value. This value is the value of the unpaired relocatable term, adjusted by the values represented by the absolute terms and/or paired relocatable terms associated with it.

For example, in the expression $R_3-R_2+R_3$, R_3 and R_2 are relocatable terms from the same control section. If, initially, R_3 equals 10 and R_2 equals 5, the value of the expression is 15. However, upon relocation this value will change. If a relocation factor of 100 is applied, the value of the expression is 115. Note that the value of the paired terms R_3-R_2 remains constant at 5 regardless of relocation. Thus, the new value of the expression, 115, is the result of the value of the unpaired term (R_3) adjusted by the values of R_3-R_2.

The following examples illustrate relocatable expressions. A is an absolute term, R_3 and R_2 are relocatable terms from the same control section. R_1 is a relocatable term from a different control section.

R_1-32*A R_3-R_2+*	=H'1234' (literal)
$R_3-R_2+R_1$	A*A+R_3
* (reference to location counter)	$R_3-R_2+R_3$
	R_1

This section deals with the coding of the machine instructions featured in the Assembler language. Machine instruction statements are used to tell the Assembler to generate the object (machine language) coding for Model 20 instructions. Format and function of each machine instruction are described and the use of each instruction is illustrated by an example.

Object Format of Machine Instructions

The instruction format indicates the length of the instruction and the type of operation to be performed. The length of the instruction can be one, two, or three halfwords. The types of instruction formats are shown in Figure 4.

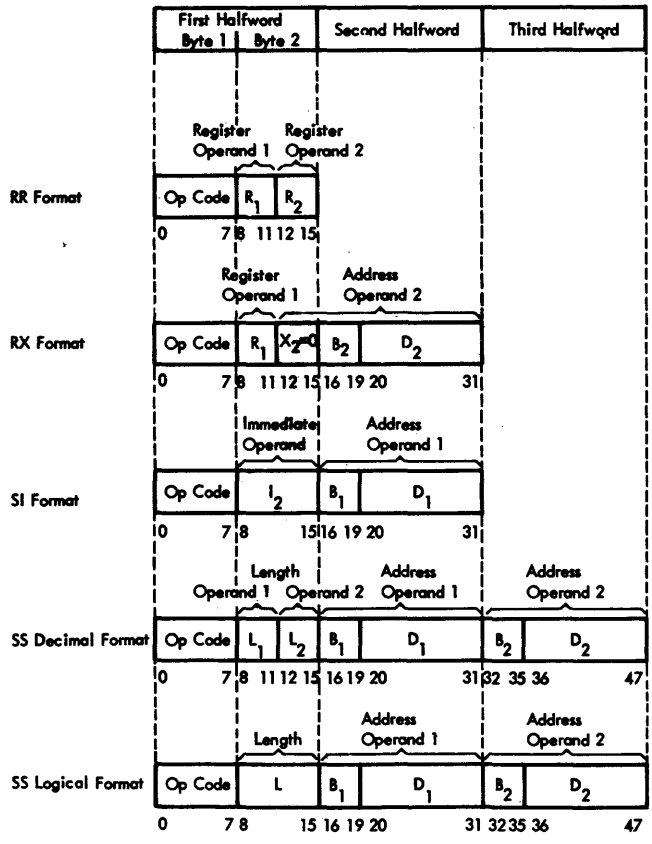


Figure 4. Object Format of Machine Instructions

RR Format: Denotes a register-to-register operation.

RX Format: Denotes a register-to-storage or a storage-to-register operation. In this format, bits 12 through 15 must be zero.

SI Format: Denotes a storage-immediate operation. In this format the I2 field of the instruction is the second operand.

SS Format: Denotes a storage-to-storage operation.

In each format, the first byte of the first halfword contains the operation code, commonly referred to as the op-code.

The second byte of the first halfword may be used to contain data, specify operand lengths, or specify registers to be used by the operation. Each instruction consists of an op-code and two operands.

Machine-Instruction Alignment

All machine instructions are automatically aligned by the Assembler on halfword boundary. If any instruction that causes information to be assembled requires alignment, the byte skipped is filled with hexadecimal zeros.

Machine-Instruction Mnemonic Codes

The mnemonic operation codes (shown in Appendix A) are designed to be easily-remembered codes that indicate the functions of the instructions. The normal format of the code is shown below; the items in brackets are not necessarily present in all codes:

Verb [Modifier] [Data Type] [Machine Format]

The verb, which is usually one or two characters, specifies the function. For example, A represents Add and MV represents Move. The function may be further defined by a modifier. For example, the modifier L indicates a logical function and the C a character as data type, as in CLC for Compare Logical Character.

The letters R and I are added to the codes to indicate, respectively, RR and SI machine instruction formats. Thus, AR indicates Add in the RR format. Functions

involving character and decimal data types imply the SS format.

EXTENDED MNEMONIC CODES

For your convenience, the Assembler provides extended mnemonic codes, which allow conditional branches to be specified mnemonically as well as through the use of the BC machine-instruction. These extended mnemonic codes specify both the machine branch instruction and the condition on which the branch is to occur. The codes are not part of the machine instruction, but are translated by the Assembler into the corresponding operation and condition combinations.

The extended mnemonic codes and their operand formats are shown in Appendix A together with their machine instruction equivalents. Unless otherwise noted, all extended mnemonics shown are for instructions in the RX format. The only difference between the operand fields of the extended mnemonics and those of their machine-instruction equivalents is the absence of the R1 field and the comma that separates it from the rest of the operand field.

The extended mnemonic list, like the machine-instruction list, shows explicit address formats only. Each address can also be specified as an implied address. Examples illustrating instructions using extended mnemonic codes are given below.

Name	Operation	Operand
	B	40(0,8)
	BNL	GO
	BO	8
	BR	REG9

The first instruction specifies an unconditional branch to an explicit address. The address is the sum of the contents of base register 8 and the displacement 40. The second instruction specifies a branch on not low to the address implied by GO. The next to last instruction is a branch on one to the address contained in register 8. The last instruction is an unconditional branch to the address contained in the register equated to REG9 elsewhere in the program.

Machine-Instruction Operands

The operands of a machine instruction are referred to as first and second operands. They have, in the following examples, a subscript (1 or 2) to the code letter for the field to indicate a particular operand (e.g., R_1, R_2, L_1, D_2 etc.).

There are three types of operands:

1. Operands that are main-storage addresses.

2. Immediate data operands that are one byte constants.

3. Operands that are the general registers.

The address specified in an instruction always refers to the leftmost byte of the field addressed. There is no relation between the address specified in the operand and that of the instruction.

The length of an addressed data field may be fixed or variable. In the latter case, the length is indicated in the length field (L) of the operand. The L-field indicates the number of bytes used. The maximum length of a field is 256 bytes.

Immediate data is used only as the second operand in logical operations in the SI-Format. The length is one byte and, being part of an instruction, immediate data has no address.

Data in registers have a fixed length of one halfword.

OPERAND FIELDS AND SUBFIELDS

Some symbolic operands are written as a single field. Other operands are written as a field followed by one or two subfields. For example, addresses consist of the contents of a base register and a displacement. An operand that specifies a base register and displacement is written as a displacement field followed by a base register subfield, as follows: 40(8). Since the Model 20 does not have index registers, the base register subfield must be preceded by a zero and a comma in the RX format, e.g., 40(0,8). In the SS format, a length subfield and a base register subfield are written as follows: 40(21,8).

A comma must be written to separate operands. Parentheses must be written to enclose a subfield or subfields, and a comma must be written to separate two subfields within parentheses. When parentheses are used to enclose one subfield, and the subfield is omitted, the parentheses must be omitted.

In the case of two subfields separated by a comma and enclosed by parentheses, the following rules apply:

1. If both subfields are omitted, the separating comma and the parentheses must also be omitted. For example:

```
LH   12,48(0,15)
LH   12,FIELD      (implicit address)
```

2. If the first subfield in the sequence is omitted, the comma that separates it from the second subfield is written. The parentheses must also be written. For example:

```
MVC  32(16,15),FIELD2
MVC  BETA(,15),FIELD2 (implicit
                       length)
```

3. If in the RX format a base register is specified, the first subfield (index register) must be specified as a zero because this subfield is not used. This zero must not be omitted. For example: LH 12,48(0,15)

4. If the second subfield in the sequence is omitted, the comma that separates it from the first subfield must be omitted. The parentheses must be written. For example:

```
MVC  32(16,15),FIELD2
MVC  FIELD1(16),FIELD2      (implicit
                             address)
```

Fields and subfields in a symbolic operand may be represented by absolute or relocatable expressions, depending on what the field requires. Refer to Appendix B for a detailed description of field requirements.

Blanks must not appear in an operand unless provided by a character self-defining term or a character literal. Thus, blanks are not permitted between fields and the comma separators, between parentheses and fields, etc.

In the following, when we speak of a data field or storage field, we mean the field in main storage defined by the fields and subfields of the first or second operand of a machine instruction.

Explicit and Implicit Adressing

Byte locations in storage are expressed in binary form and are numbered consecutively from hexadecimal 0000 to the upper limit of the available storage. The first 144 bytes (bytes 0000-0143) are reserved for internal CPU control and thus not available to the program. The location of any field or group of bytes is specified by the address of the leftmost byte.

Appendix B shows two types of addressing formats for RX, SI, and SS instructions. In each case, the first type shows the method of specifying an address explicitly as a base register and a displacement. The second type indicates how to specify an implied address as a relocatable expression.

EXPLICIT ADDRESSING

If you use explicit addressing in an operand you must specify a base register and a displacement. For example, explicit addressing is used in the first operand of the following Move-Immediate instruction:

MVI $D_1(B_1)$,X'F0'

where D_1 is the displacement and B_1 is the base register. B_1 may be an absolute expression with a value between 0 and 15 inclusive. D_1 may be an absolute expression with a value between 0 and 4095 inclusive. The address specified in an operand occupies one halfword of the object code.

At object time, the Model 20 differentiates between a base register specification of $0 \leq B_1 \leq 7$ and $8 \leq B_1 \leq 15$.

Case $0 \leq B_1 \leq 7$ (Direct Addressing)

The content of the halfword containing the address is taken as the effective address by the CPU. For example, the source statement

MVI 4095(3),X'F0'

will be assembled as follows

92F03FFF (object code).

The CPU takes the second halfword (3FFF) of the object code directly as the effective address (16383) of the field addressed by the first operand. Therefore, one speaks of direct addressing.

Case $8 \leq B_1 \leq 15$ (Indirect Addressing)

Here, the first four bits of the halfword containing the address specify one of the general registers 8 through 15. The other 12 bits contain the displacement. The CPU adds the content of the general register to the displacement to form the effective address. For example, the source statement

MVI 1095(9),X'F0'

will be assembled as follows

92F09447 (object code).

The CPU adds the content of register 9 (assumed to have been loaded previously with a value of 14288 or 37D0 hexadecimal) and 1095 (hexadecimal 447) to get the effective address 16383 (hexadecimal 3FFF). This is referred to as effective or indirect addressing.

IMPLICIT ADDRESSING

If you use implicit addressing you must specify an expression to represent an address. The expression may either be absolute or relocatable.

Absolute Expression

The value of the expression must not exceed 4095 (hexadecimal FFF). The Assembler regards this absolute expression as displacement and automatically assumes base register 0. For example, the source statement

```
BC    15,EOJ
```

where the absolute expression EOJ has the value 194 (hexadecimal 0C2), will be assembled as follows

```
47F000C2    (object code).
  B D
```

Again, at object time, we have direct addressing as described above.

Relocatable Expression

In this case, the Assembler uses the value of the relocatable expression to calculate base register and displacement. To this end, you must tell the Assembler which register to use as base register by issuing USING and DROP instructions.

You can find an explanation on how to use the USING and DROP instructions in the section Base Register Instruction Statements. You will find that the implicit addressing feature of the Assembler language is a great help to you. It relieves you of the necessity to separate each storage address into a displacement value and a base address value, thus eliminating a likely source of error and reducing the time required to check out your program.

For example, assume that FIELD is a relocatable symbol, which has been assigned a value of 7400. Assume also that the Assembler has been notified (by a USING instruction) that general register 8 currently contains a relocatable value of 4096 and is available as a base register. The following example shows a machine instruction as it would be written in Assembler language and as it would be assembled. Note that the value of D_2 is the difference between 7400 and 4096 and that X_2 is assembled as zero, since double indexing is not possible on Model 20. The assembled instruction is presented in hexadecimal notation:

```
Source statement:  STH  14,FIELD
```

Assembled instruction:

Op.Code	R_1	X_2	B_2	D_2
40	E	0	8	CE8

Here again, direct and indirect addressing is possible depending on whether you specify one of the pseudo registers 0 through 7 or one of the general registers 8 through 15. Direct and indirect addressing is explained under Explicit Addressing.

A special application of implicit addressing is relative addressing:

Relative addressing is the technique of addressing instructions and data areas by designating their location in relation to the location counter or to some symbolic location. This type of addressing is always in bytes, never in bits, halfwords, or instructions. Thus, the expression *+4 specifies an address that is four bytes greater than the current value of the location counter.

In the sequence of instructions shown in the following example, the location of the SR machine instruction can be expressed in two ways, ALPHA+2 or BETA-4, because all of the mnemonics in the example are for instructions with a length of two bytes.

Name	Operation	Operand
ALPHA	AR	13,14
	SR	14,15
	BCR	1,14
BETA	AR	12,13
	B	ALPHA+2

EXPLICIT AND IMPLICIT LENGTHS

The length in SS instructions can be explicit or implied. To imply a length, simply omit a length field from the operand. The omission indicates that the length field is either of the following:

1. The length attribute of the expression specifying the displacement, if an

explicit base and displacement have been written (explicit addressing).

2. The length attribute of the expression specifying the effective address, if the base and displacement have been implied (implicit addressing).

In either case, the length attribute for an expression is the length attribute of the leftmost term in the expression.

A self-defining term has the length attribute 1. Both a symbol referring to a machine instruction and a Location Counter reference have the length of the instruction in which they appear. The length attribute of a literal is determined the same way as that of a constant in a DC instruction.

An explicit length is written in the operand as an absolute expression. The explicit length overrides any implied length.

Whether the length is explicit or implied, it is always an effective length. The value inserted into the length field of the assembled instruction (object code) is one less than the effective length. If the specified length is a zero value, a zero is inserted into the length field.

In the following example:

```
|Name    |Operation |Operand             |
|--------|----------|--------------------|
|        |MVC       |SYMBOL,A            |
|        |.         |                    |
|        |.         |                    |
|SYMBOL  |DS        |CL3                 |
```

three bytes are moved since the operand SYMBOL has an implicit length of 3 as defined by the DS instruction. As shown below, the value inserted into the length field of the object code is two.

```
|D2|02|B1|D1|B2|D2|
```

Note the length specification of two.

Using an explicit length, e.g:

```
MVC SYMBOL(5),A
```

would have the following effect:

```
|D2|04|B1|D1|B2|D2|
```

Note the length specification of four.

You may combine explicit and implicit addressing with explicit and implicit lengths. Examples are given below.

Examples

The following examples are grouped according to machine-instruction format. They illustrate the various symbolic operand formats. All symbols used in the examples are assumed to be defined either within the same assembly or by means of an EXTRN statement within another assembly. All symbols specifying register numbers, masks, and lengths are assumed to be equated, by an EQU instruction, elsewhere to absolute values.

Implicit addressing, control section addressing, and the function of the USING Assembler instruction are not considered here. For discussion of these considerations and for examples of coding sequences that illustrate them, refer to Program Sectioning and Explicit and Implicit Linking and Addressing.

RR Format

Both operands must be absolute expressions.

```
|Name|Operation|Operand                  |
|----|---------|-------------------------|
|A1  |AR       |11,12                    |
|A2  |SR       |REG11,REG12              |
|B1  |BASR     |REG10,0                  |
|B2  |BASR     |LINKREG,LINKREG          |
|C1  |BCR      |2,LINKREG                |
|C2  |BCR      |HIGH,LINKREG             |
```

RX Format

The first operand must be an absolute expression. Explicit or implicit addressing or a literal may be used in the second operand. A length cannot be specified.

```
|Name|Operation|Operand                  |
|----|---------|-------------------------|
|A1  |LH       |CALCREG,38(0,10)         |
|A2  |AH       |CALCREG,DISPL1(0,REG10)  |
|B1  |CH       |CALCREG,MAXIMUM          |
|B2  |BC       |LOW,*+8                  |
|B3  |SH       |CALCREG,BIN1000          |
|B4  |STH      |CALCREG,RESULT           |
|B5  |BAS      |REG8,GOON                |
|C1  |AH       |REG14,=H'1000'           |
```

Instructions A1 and A2 use explicit addressing; the first subfield within the parentheses must not be omitted and must be zero because double indexing is not possible in the Model 20. Instructions B1, B2,

B3, B4, and B5 use implicit addressing. C1 contains a literal.

SI Format

Explicit and implicit addressing may be used in the first operand. The second operand - if any - must be an absolute expression with a value between 0 and 255 (hexadecimal 00 and FF) inclusively. A length cannot be specified.

Name	Operation	Operand
A1	CLI	40(9),X'40'
A2	MVI	DISPL(REG9),BLANK
A3	HPR	STOP01D0,0
B1	NI	SWBYTE,X'FF'-BIT0-BIT7
B2	OI	SWBYTE,BIT0+BIT7
B3	TM	BYTE,MASK
B4	SPSW	NEWPSW

Instructions A1, A2, and A3 use explicit addressing, instructions B1, B2, B3, and B4 use implicit addressing.

SS Decimal Format

A combination of explicit and implicit addressing with explicit and implicit length is possible in both operands. Thus, for both operands you have the following four possibilities:

- explicit addressing with explicit length
- explicit addressing with implicit length
- implicit addressing with explicit length
- implicit addressing with implicit length

Literals may be used in the second operand only.

Name	Oper-ation	Operand
A1	MP	20(10,8),10(6,13)
A2	DP	10(LEN10,R8),DISPL0(SIX,12)
A3	AP	D4(9,REG11),0(,10)
A4	SP	FIVE(LFOUR,RBASE),ZERO(,RG11)
B1	CP	RESULT(2),PFOUR(1)
B2	MVO	FIELD2(LEN3),FIELD1
B3	PACK	PFIELD,ZFIELD(L'PFIELD+1)
B4	UNPK	ZFIELD,PFIELD
C1	YAP	RESFIELD,=P'0'

All A-instructions use explicit addressing and all B-instructions use implicit addressing. Explicit length is shown in the second operand of instructions A1, A2, B1, and B3; implicit length is shown in the second operand of instructions A3, A4, B2, and B4. Instruction C1 contains a literal.

SS Logical Format

Explicit and implicit addressing may be combined with explicit and implicit length in the first operand just as in the SS decimal format. In the second operand, explicit or implicit addressing or a literal may be used.

Name	Operation	Operand
A1	MVN	2(20,9),22(9)
A2	MVZ	DISPL+19(,R9),DISPL+18(R9)
B1	TR	FIELD(10),TRATABLE
B2	CLC	FIELD+1(L'FIELD-1),FIELD
B3	ED	PATTFLD,RESFLD
C1	MVC	PRINTAR,=C'RESULT'

Instructions A1 and A2 show explicit addressing, instructions B1, B2, and B3 show implicit addressing. Explicit length is shown in instructions A1, B1, and B2 and implicit length in instructions A2, B3, and C1. Instruction C1 uses a literal.

Types and Functions of Machine Operations

There are five types of operations:

1. Binary arithmetic operations.
2. Decimal arithmetic operations.
3. Logical operations.
4. Branch operations.
5. I/O operations.

These operations differ not only in their internal logic but also in the format of data, use of registers, and format of instructions. The first four operations are discussed in the subsequent sections.

Some operations set a condition code in bits two and three of the Program Status Word (PSW). This condition code indicates the relationship (less than/greater than, zero, negative, positive etc.) between the two operands as a result of the last operation effecting the condition code setting. For details about the PSW see the SRL publication IBM System/360 Model 20 Functional Characteristics, Form GA26-5847.

BINARY ARITHMETIC OPERATIONS

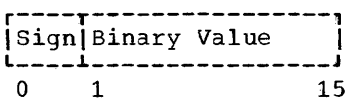
Binary arithmetic is used for operands like addresses, indexes, counters, and binary data. The length of each operand is one

halfword including the sign. Negative numbers are given in the two's-complement form. The first operand must be in one of the general registers. The other operand may be either in a register or in main storage. For detailed information refer to the SRL publication IBM System/360 Model 20 Functional Characteristics, Form GA26-5847.

Data Format

Binary numbers have a fixed length of one halfword (16 bits). The first (leftmost) bit contains the sign, the other 15 bits the binary value. Binary numbers may be stored in one of the general registers or in main storage. In main storage, the address of the left byte must be even.

Binary halfword

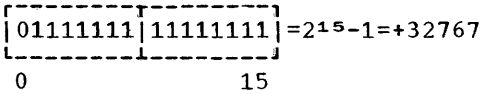


Representation of Binary Numbers

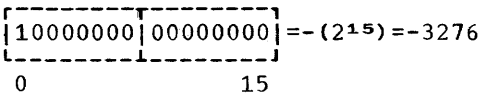
Binary numbers are represented as signed integers. Positive numbers are represented in true form with a 0-bit as sign. Negative numbers are in the twos-complement form with a 1-bit as sign. The twos-complement form is found by reversing each bit (0 to 1 and 1 to 0) and adding a 1 to the rightmost bit.

A zero is always positive by definition. The absolute value of the lowest possible negative number is higher by 1 than the highest possible positive number.

Highest possible positive number:



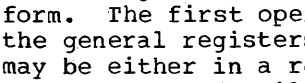
Lowest possible negative number:



MACHINE FORMATS OF INSTRUCTIONS FOR BINARY OPERATIONS

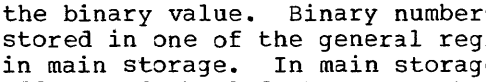
Instructions for binary operations use the RR- or RX-Format.

RR-Format



R_1 indicates a general register containing the first binary number and R_2 a general register containing the second binary number. R_1 and R_2 may refer to the same register. The result of an instruction in the RR-Format replaces the first operand.

RX-Format



R_1 indicates a general register containing the first binary number. The address of the second binary number is formed by adding the contents of the register named in the B_2-field to the displacement given in the D_2-field.

Condition Code After Binary Operations

Condition code	00	01	10	11
AR-Add Register	zero	<zero	>zero	-
SR-Subtract Reg.	zero	<zero	>zero	-
CH-Comp.Halfword*	equal	low	high	-
AH-Add Halfword	zero	<zero	>zero	-
SH-Subtract Halfw.	zero	<zero	>zero	-

*first operand compared to second.

All other binary operations leave the condition code unchanged.

BINARY ARITHMETIC ERROR CONDITIONS

Error conditions that may occur during the execution of binary operations are:

1. Operation code invalid
2. Addressing error
   a. An instruction address or an operand address refers to the protected first 144 bytes of main storage (addresses 0 to 143).
   b. An instruction address or an operand address is outside available main storage.
   c. The last (highest) main-storage position contains any part of an instruction that is to be executed.
   d. The R_1 or R_2 fields of a binary instruction contain binary values 0 through 7.

3. Specification error

   a. The low-order bit of an instruction address is one, i.e., no halfword boundary.
   b. The half-word second operand is not located on a halfword boundary.
   c. Bits 12 through 15 of an RX format instruction are not all zero.

4. Binary overflow check

5. CPU parity error

INSTRUCTIONS FOR BINARY ARITHMETIC

Name	Op-code	Format
Add Register (AR)	1A	RR
Subtract Register (SR)	1B	RR
Store Halfword (STH)	40	RX
Load Halfword (LH)	48	RX
Compare Halfword (CH)	49	RX
Add Halfword (AH)	4A	RX
Subtract Halfword (SH)	4B	RX

AR -- ADD REGISTER

Name	Operation	Operand
blank or symbol	AR	R_1,R_2

Function: The content of the first operand field is added to the content of the second operand field. The result is stored in the register specified by the first operand. The second operand remains unchanged.

The sign is determined by the rules of algebra. A zero result is always positive. If the result is higher than 2^{15}-1 (=32767) or lower than -2^{15} (=-32768), a binary overflow check occurs.

Condition Code:

00 Result = zero
01 Result < zero
10 Result > zero

Example: Assume register 8 contains hexadecimal 0123 and register 9 contains hexadecimal 0532.

Source statement:

```
AR    8,9
```

From this source statement the Assembler creates the following object code:

Op-code	R_1	R_2
1A	8	9

After execution, register 8 contains hexadecimal 0655. The condition code is 10.

SR -- SUBTRACT REGISTER

Name	Operation	Operand
blank or symbol	SR	R_1,R_2

Function: The content of the second operand field is subtracted from the content of the first operand field. The result will be in the register specified by R_1. Both operands and the result consist of 15 numeric bits plus the sign. The second operand remains unchanged.

The subtraction is performed by adding the twos-complement of the second operand to the first operand. All 16 bits of both operands are added. If the result is higher than 2^{15}-1 (=32767) or lower than -2^{15} (=-32768), a binary overflow check occurs.

A register may be cleared to zero by subtraction from itself.

There is no two's-complement for the highest negative number. This number remains unchanged when a complementation is performed. Nonetheless, the subtraction is still executed correctly.

Condition Code:

00 Result = zero
01 Result < zero
10 Result > zero

Example: Assume register 8 contains hexadecimal 047F and register 13 contains hexadecimal 00D7.

Source statement:

```
SR    8,13
```

From this source statement the Assembler generates the following object code:

Op-code	R_1	R_2
1B	8	D

After execution register 8 contains hexadecimal 03A8. The condition code is 10.

STH -- STORE HALFWORD

Name	Operation	Operand
blank or symbol	STH	R_1,D_2(0,B_2)

Function: The content of the register specified by R_1 is stored in the halfword at the main-storage location addressed by B_2 and D_2. The first operand remains unchanged.

Condition Code: No change.

Example: Assume register 9 contains hexadecimal 68AF, register 11 contains hexadecimal 001E, and the displacement in the second operand is hexadecimal 29E (decimal 670).

Source statement:

```
STH   9,670(0,11)
```

From this source statement the Assembler generates the following object code:

Op-code	R_1	X_2=0	B_2	D_2
40	9	0	B	29E

After execution the field starting at storage location hexadecimal 2BC (decimal 700) contains 68AF.

LH -- LOAD HALFWORD

Name	Operation	Operand
blank or symbol	LH	R_1,D_2(0,B_2)

Function: The halfword at the main storage location addressed by B_2 and D_2 is placed into the register specified by R_1. The second operand remains unchanged.

Condition Code: No change.

Example: Assume register 9 contains hexadecimal AAAA, register 12 contains hexadecimal 0032, the displacement in the second operand is 1F4 (decimal 500), and the field starting at storage location hexadecimal 226 (decimal 550) contains 80AF.

Source statement:

```
LH    9,500(0,12)
```

From this source statement the Assembler generates the following object code:

Op-code	R_1	X_2=0	B_2	D_2
48	9	0	C	1F4

After execution register 9 contains hexadecimal 80AF.

CH -- COMPARE HALFWORD

Name	Operation	Operand
blank or symbol	CH	R_1,D_2(0,B_2)

Function: The content of the register specified by R_1 is compared with the halfword at the main storage location addressed by B_2 and D_2. The comparison is algebraic, i.e. the signs must be taken into consideration. Both operands remain unchanged. A condition code is set.

Condition Code:

00 First operand = second operand
01 First operand < second operand
10 First operand > second operand

Example: Assume register 9 contains hexadecimal 0001, the displacement in the second operand is hexadecimal 690 (decimal 1680), register 13 contains hexadecimal 0025, and the halfword at storage location hexadecimal 6B5 is AF99.

Source statement:

```
CH      9,1680(0,13)
```

From this source statement the Assembler generates the following object code:

Op-code	R_1	X_2=0	B_2	D_2
49	9	0	D	690

After comparison the resulting condition code setting will be: 10.

AH -- ADD HALFWORD

Name	Operation	Operand
blank or symbol	AH	$R_1,D_2(0,B_2)$

Function: The halfword in main storage, addressed by B_2 and D_2, is added to the content of the register specified by R_1. The sign is determined by the rules of algebra. A zero result is positive by definition.

If the result is higher than $2^{15}-1$ (=32767) or lower than -2^{15} (=-32768), a binary overflow check will occur.

Condition Code:

00 Result = zero
01 Result < zero
10 Result > zero

Example: Assume register 9 contains hexadecimal 047F, register 11 contains hexadecimal 0028, the displacement in the second operand is 1EA (decimal 490), and the field at storage location hexadecimal 212 (530) contains hexadecimal 1F29.

Source statement:

```
AH   9,490(0,11)
```

From this source statement the Assembler generates the following object code:

Op-code	R_1	X_2=0	B_2	D_2
4A	9	0	B	1EA

After execution register 9 contains hexadecimal 23A8 and the condition code is 10.

SH -- SUBTRACT HALFWORD

Name	Operation	Operand
blank or symbol	SH	$R_1,D_2(0,B_2)$

Function: This instruction is identical to the Add Halfword instruction with the following exception: The two's complement of the second operand, addressed by B_2 and D_2, is added in place of the true value.

Condition Code:

00 Result = zero
01 Result < zero
10 Result > zero

Example: Assume register 9 contains hexadecimal 047F, register 11 contains hexadecimal 0050, the displacement in the second operand is hexadecimal 320 (decimal 800), and the field starting at storage location hexadecimal 370 (decimal 880) contains hexadecimal 00D7.

Source statement:

```
SH   9,800(0,11)
```

From this source statement the Assembler generates the following object code:

Op-code	R_1	X_2=0	B_2	D_2
4B	9	0	B	320

After execution register 9 contains hexadecimal 03A8 and the condition code is 10.

DECIMAL ARITHMETIC OPERATIONS

Decimal arithmetic can be performed only with data in packed format. Packed format means that there are two digits in one byte except for the low order byte. It contains one digit and the sign.

Data is transferred to and from the external I/O devices in zoned format. Thus, the data has to be packed and unpacked before and after processing respectively. In zoned format, each byte contains a zone in the left halfbyte and a digit in the right halfbyte except the last one which contains the sign and a digit.

The address in an instruction always specifies the left-most byte of the data field. The length field in an assembled instruction indicates how many bytes are part of the data field in addition to the addressed (left) byte.

Data Format

Decimal operations are performed in main storage. The data fields may have a length from 1-16 bytes. A field may start at any address including an odd one. In zoned format there may be a maximum of 16 digits, in packed format a maximum of 31 digits plus the sign in a field. The two operands may be of different length. Multiplicand and divisor are restricted to a maximum of 15 digits plus the sign.

The values in the operand fields are assumed to be right aligned, with leading zeros where required. The operands are processed as integers from right to left. If a result extends beyond the field indicated by the address and the length field, the extending (high order) part is ignored and the condition code is set to 11.

Representation of Numbers

Decimal numbers consist of binary coded digits and a sign. The decimal digits 0-9 are represented in the four bit code by the bit combinations 0000-1001.

The combinations 1010-1111 are reserved for representations of a sign (+,-). 1011 and 1101 represent a minus, the other four combinations a plus. The representations 1100, 1101, 1010, and 1011 are created during calculations in main storage.

Negative numbers are represented in true form.

The two decimal formats are:

Packed decimal number (e.g. five digits)

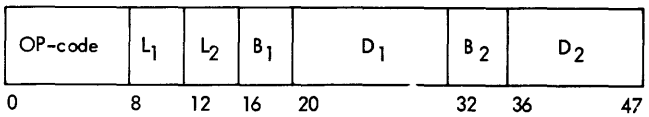
Byte	Byte	Byte
Digit | Digit	Digit | Digit	Digit | Sign

Zoned decimal number (e.g. three digits)

Byte	Byte	Byte
Zone | Digit	Zone | Digit	Sign | Digit

Machine Formats of Instructions for Decimal Arithmetic

Decimal operations have the SS format:



The fields B_1 and D_1 give the main-storage address of the left byte of the first data field; L_1 gives the number of bytes in addition to the leftmost byte. L_1 may vary between zero and 25 inclusively.

In the source instruction statement you specify the effective length of the data field. The Assembler inserts a value one less than the effective length into the length field of the assembled instruction.

The instruction fields B_2, D_2, and L_2 give the respective information for the second data field.

The result of a decimal operation replaces the content of the first data field. It cannot occupy more storage area than indicated in the L_1 field. The second data field remains unchanged. Exception: overlapping fields.

The general registers are not affected by decimal operations.

CONDITION CODE AFTER DECIMAL OPERATIONS

The decimal operations listed in the table below set a condition code.

	00	01	10	11
ZAP	zero	< zero	> zero	-
CP*	equal	low	high	-
AP	zero	< zero	> zero	overflow
SP	zero	< zero	> zero	overflow

*First operand compared to second.

All other decimal operations leave the condition code unchanged.

DECIMAL ARITHMETIC ERROR CONDITIONS

The following error conditions may occur during the execution of decimal arithmetic operations:

1. Operation code invalid.
2. Addressing error
   a. An instruction address or an operand address refers to the protected first 144 bytes of main storage.
   b. An instruction address or an operand address is outside available storage.
   c. An instruction occupies the last two (highest) main-storage positions.
3. Specification error
   a. The low-order bit of an instruction address is one, i.e., no halfword boundary.

b. For Zero and Add, Compare Decimal, Add Decimal, and Subtract Decimal instructions the length code L_2 is greater than the length code L_1.
c. For Multiply Decimal and Divide Decimal instructions, the length code L_2 is greater than 7 or greater than or equal to the length code L_1.

4. Data error

   a. A sign or digit code of an operand in the Zero and Add, Compare Decimal, Add Decimal, Subtract Decimal, Multiply Decimal, or Divide Decimal instruction is incorrect.
   b. The operand fields in these instructions overlap incorrectly.
   c. The first operand in a Multiply Decimal instruction has insufficient high-order zeros.

5. Decimal divide check
   The resultant quotient in a Divide Decimal instruction exceeds the specified data field instruction (including division by zero) or the dividend has no leading zero.

6. CPU parity error.

INSTRUCTIONS FOR DECIMAL ARITHMETIC

Name	Op-code	Format
Move with Offset (MVO)	F1	SS
Pack (PACK)	F2	SS
Unpack (UNPK)	F3	SS
Zero and Add Packed (ZAP)	F8	SS
Compare Decimal Packed (CP)	F9	SS
Add Decimal Packed (AP)	FA	SS
Subtract Decimal Packed (SP)	FB	SS
Multiply Decimal Packed (MP)	FC	SS
Divide Decimal Packed (DP)	FD	SS

MVO -- MOVE WITH OFFSET

Name	Operation	Operand
blank or symbol	MVO	$D_1(L_1,B_1),D_2(L_2,B_2)$

Function: The contents of the second data field are moved to the location specified by the first operand. The move is executed with an offset of half a byte (one digit) to the left. The right halfbyte of the first data field remains unchanged. There is no check for validity. The fields need not have equal lengths. Leading zeros are inserted if the first field is longer than the second. If the second field is longer than the first, the high-order digits of the second field are ignored.

The move proceeds from right to left one byte at a time. The second field may overlap the first excluding the rightmost byte of the first field.

Condition Code: No change.

Example: Assume register 12 contains hexadecimal 0250, register 15 contains hexadecimal 040F, the displacement given in both operands is zero, storage location hexadecimal 040F-0412 contains hexadecimal 123456, and storage location hexadecimal 0250-0253 contains hexadecimal 7788990C.

Source statement:

```
MVO    0(4,12),0(3,15)
```

From this source statement the Assembler produces the following object code:

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
F1	3	2	C	000	F	000

After execution the field at location hexadecimal 0250-0253 contains hexadecimal 0123456C.

PACK -- PACK

Name	Operation	Operand
blank or symbol	PACK	$D_1(L_1,B_1),D_2(L_2,B_2)$

Function: The unpacked content of the second data field is packed and placed into the first data field. The second data field must contain an unpacked decimal number. It may have a maximum size of 16 bytes. There is no check for validity of digits and sign.

The lengths of the fields need not be equal. Leading zeros are inserted if the first field is too long for the result. The high-order digits of the second field are ignored if the first field is too short for the result. The fields are processed from right to left one byte at a time.

Condition Code: No change.

Example: Assume register 11 contains hexadecimal 044A, register 9 contains hexadecimal 02C0, the displacement in the first operand is hexadecimal 244, in the second operand it is hexadecimal 180, and that storage location hexadecimal 0440-0444 contains hexadecimal F1F2F3F4C5.

Source statement:

```
 PACK   580(4,11),384(5,9)
```

From this source statement the Assembler produces the following object code:

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
F2	3	4	B	244	9	180

After execution the field at storage location hexadecimal 068E contains 0012345C.

UNPK -- UNPACK

Name	Operation	Operand
blank or symbol	UNPK	$D_1(L_1,B_1),D_2(L_2,B_2)$

Function: The packed contents of the second data field are changed to zoned format and stored in the first data field. The second data field must contain a packed decimal number. Sign and digits are not checked for validity.

After processing, the zoned decimal number in the first data contains the sign (high-order four bits) and one digit in the rightmost byte. Each of the other bytes contains a zone and a digit.

The fields are processed from right to left. If the first operand field is too long it is filled with leading zeros. If the first operand field is too short to contain all the digits of the second operand, the leading digits are ignored. The operands may overlap but you must exercise caution.

Condition Code: No change.

Example: Assume register 10 contains hexadecimal 0FA0, the displacement in the first operand is hexadecimal FB4, that in the second operand is hexadecimal 65, and location hexadecimal 1004-1007 contains hexadecimal 0123456D.

Source statement:

```
UNPK    4020(5,10),100(4,10)
```

From this source statement the Assembler produces the following object code:

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
F3	4	3	A	FB4	A	65

After execution location hexadecimal 1F54-1F58 contains F2F3F4F5D6.

ZAP -- ZERO AND ADD PACKED

Name	Operation	Operand
blank or symbol	ZAP	$D_1(L_1,B_1),D_2(L_2,B_2)$

Function: The first data field is zeroed out and the contents of the second data field are placed into the first data field. This operation is equivalent to an addition into a zero-field. The second field must be in packed format.

A zero result is positive by definition. The second field may be shorter than the first field. If the second field is longer, then a machine stop occurs and the instruction is not executed.

Processing proceeds from right to left. All digits and the sign of the second field are checked for validity. High order zeros are supplied if needed. The fields may overlap if the rightmost byte of the first operand is coincident with, or to the right of, the rightmost byte of the second operand.

Condition Code:

00 Result = zero
01 Result < zero
10 Result > zero

Example: Assume register 10 contains hexadecimal 01F4, the displacement in the first operand is hexadecimal 294, that in the second operand is hexadecimal 37A, and storage location hexadecimal 056E-0570 contains 01234D.

Source statement:

```
ZAP     660(4,10),890(3,10)
```

From this source statement the Assembler produces the following object code:

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
F8	3	2	A	294	A	37A

After execution location 0487-048A contains 00012340.

CP -- COMPARE DECIMAL PACKED

Name	Operation	Operand
blank or symbol	CP	$D_1(L_1,B_1),D_2(L_2,B_2)$

Function: The contents of the first data field are compared to the contents of the second data field and the result is indicated by a new condition code.

The comparison proceeds from right to left and is algebraic, i.e. the sign and all digits are compared one byte at a time. (Negative values are smaller than positive values).

A negative zero is equal to a positive zero. The sign and all digits are checked for validity. A halt occurs if the second field is longer than the first field and the instruction is not executed. If the second field is shorter it is extended with leading zeros.

The contents of both fields do not change. An overflow cannot occur. The two fields may overlap if the rightmost bytes coincide. Therefore, it is possible to compare a number to itself.

Note the difference between "Compare Decimal Packed" and "Compare Logical Characters" (CLC).

Condition Code:

00 First operand = second operand
01 First operand < second operand
10 First operand > second operand

Example: Assume register 12 contains hexadecimal 0040, register 11 contains hexadecimal 02F0, the displacement in the first operand is hexadecimal 640, that in the second operand is hexadecimal 3E8, location hexadecimal 0680-0682 contains 01000C, and location 06D8-06D9 contains 999C.

Source statement:

```
CP      1600(3,12),1000(2,11)
```

From this source statement the Assembler produces the following object code:

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
F9	2	1	C	640	B	3E8

After comparison the condition code is 10.

AP -- ADD DECIMAL PACKED

Name	Operation	Operand
blank or symbol	AP	$D_1(L_1,B_1),D_2(L_2,B_2)$

Function: The contents of the second data field are added to the contents of the first data field. The result replaces the content of the first field.

The sign is determined by the rules of algebra. A zero result is positive by definition. Exception: It is possible that a remaining zero result after an overflow has a negative sign. A condition code is set.

If the second field is longer than the first a program error halt occurs and the instruction is not executed. If the second field is shorter than the first it is expanded with leading zeros and addition will take place normally. Signs and digits are checked for validity. Addition proceeds from right to left. The result is in packed format.

The two fields may overlap if the rightmost bytes coincide. Thus, it is possible to double a number.

Condition Code:

00 Result = zero
01 Result < zero
10 Result > zero
11 Overflow

Example: Assume register 8 contains hexadecimal 0014, storage location 0329 (hexadecimal) contains 00222D, storage location 500 (hexadecimal) contains 01000C, the displacement in the first operand is 315 (hexadecimal), and that in the second operand is 4EC (hexadecimal).

Source statement:

```
AP      789(3,8),1260(3,8)
```

From this source statement the Assembler produces the following object code:

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
FA	2	2	8	315	8	4EC

After execution storage location 0329-032B (hexadecimal) contains 00778C.

SP -- SUBTRACT DECIMAL PACKED

Name	Operation	Operand
blank or symbol	SP	$D_1(L_1,B_1)D_2(L_2,B_2)$

Function: The contents of the second field are subtracted from the contents of the first data field. The result is placed into the first field. The sign is determined by the rules of algebra. A zero result is positive by definition. Exception: A zero result remaining in case of an overflow may have a minus sign.

If the second field is longer than the first a program error halt occurs and the instruction is not executed. If the second field is shorter, it is expanded with leading zeros and subtraction will take place normally.

All digits and the signs are checked for validity. The operation proceeds from right to left by reversing the sign of the second number and then adding the second number to the first. The result is in packed format.

The fields may overlap if the rightmost bytes coincide. Thus it is possible to clear a field to zero.

Condition Code:

00 Result = zero
01 Result < zero
10 Result > zero
11 Overflow

Example: Assume register 9 contains (hexadecimal) 00C8, register 8 contains (hexadecimal) 012C, storage location 898 (hexadecimal) contains 012C, storage location 0CE4 (hexadecimal) contains 008C, the displacement in the first field is 7D0 (hexadecimal), and that in the second field is BB8 (hexadecimal).

Source statement:

```
SP      2000(2,9),3000(2,8)
```

From this source statement the Assembler produces the following object code:

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
FB	1	1	9	7D0	8	BB8

After execution storage location 0898 (hexadecimal) contains 00A0. The condition code is 10.

MP -- MULTIPLY DECIMAL PACKED

Name	Operation	Operand
blank or symbol	MP	$D_1(L_1,B_1),D_2(L_2,B_2)$

Function: The multiplicand in the first data field is multiplied by the multiplier in the second data field. The product is placed into the first field. The second field may have a maximum of 15 digits (L_2=7) plus the sign and must be shorter than the first operand. If $L_2 > 7$ or $L_2 \geq L_1$ a program error halt occurs and the instruction is not executed.

The length of the product is equal to the sum of the lengths of multiplier and multiplicand (L of product = L_1+L_2). Therefore, the multiplicand must be expanded with leading zeros by the number of bytes of the multiplier. Otherwise a halt occurs. An overflow is not possible. The product may have a maximum length of 30 digits plus the sign. It contains at least one leading zero.

The factors and the result are considered to be signed integers. The sign is determined by the rules of algebra. The fields may overlap if their rightmost bytes coincide. Thus, it is possible to square a number.

Note: You can save computing time by using the larger of the two factors as the second operand.

Condition Code: No change.

Example:

1. Multiplicand x multiplier = product
   MAND x MOR = PROD

2. Length MAND + length MOR = length PROD

3. The MAND must be right-aligned and have leading zeros before the multiplication is executed.

```
   | Name | Operation | Operand              |
   |------|-----------|----------------------|
   |      |     .     |                      |
   |      |     .     |                      |
   |      |     .     |                      |
1  |      |   ZAP     | PROD,MAND            |
2  |      |   MP      | PROD,MOR             |
   |      |     .     |                      |
   |      |     .     |                      |
   |      |     .     |                      |
   | MOR  |   DS      | CL3                  |
   | MAND |   DS      | CL2                  |
   | PROD |   DS      | CL5                  |
   |      |     .     |                      |
   |      |     .     |                      |
   |      |     .     |                      |
```

Assume the Assembler has allocated storage location (hexadecimal) 1C92 to statement MOR. Then, MAND has location 1C95 and PROD has location 1C97. Further assume that the storage locations implicitly addressed by MOR and MAND contain 37219D and 425C respectively and register 12 contains (hexadecimal) 1194. (The Assembler automatically calculates the displacement shown in the object coding by subtracting the contents of register 12 from the location counter value of the symbolic address).

Source statement:

```
 ZAP         PROD,MAND
```

Assembler produced object code:

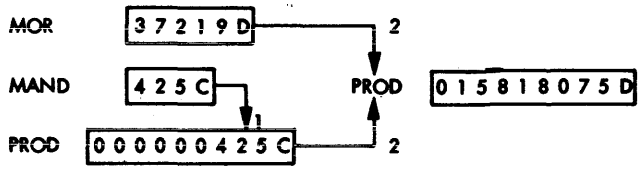
Op-code	L_1	L_2	B_1	D_1	B_2	D_2
F8	4	1	C	B03	C	B01

and

```
 MP          PROD,MOR
```

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
FC	4	2	C	B03	C	AFE

The results of the two instructions is shown in Figure 5.



Note: Maximum length of product is 16 bytes. Maximum length of MOR is 8 bytes.

Figure 5. Decimal Multiplication

DP -- DIVIDE DECIMAL PACKED

Name	Operation	Operand
blank or symbol	DP	$D_1(L_1,B_1),D_2(L_2,B_2)$

Function: The dividend in the first data field is divided by the divisor in the second data field. The quotient and the remainder are placed into the first data field.

The quotient occupies the left part of the first field, i.e. the address of the quotient is the same as the address of the dividend. The remainder occupies the right part of the first field and has a length equal to that of the divisor.

The quotient and the remainder together occupy the entire dividend field (first operand). This means the dividend field must be large enough to accomodate a divisor of maximum length and a quotient of maximum length. In the extreme case the dividend field has to be expanded with zeros to the left by the number of bytes of the divisor.

The length of the quotient field (in bytes) is L_1-L_2. The divisor field may have a maximum of 15 digits plus the sign and must be smaller than the dividend field.

If $L_2 > 7$ or $L_2 \geq L_1$ a halt occurs and the operation is not executed. The dividend must have at least one leading zero.

Dividend, divisor, quotient, and remainder are signed integers. The sign is determined according to the rules of algebra from the signs of dividend and divisor. The sign of the remainder is always ident-

ical to the sign of the dividend. This also holds true if the quotient or the remainder are zero.

If the quotient contains more than 29 digits plus the sign, or if the dividend has no leading zero, then a halt occurs and the operation is not executed. The divisor and the dividend remain unchanged and there is no overflow. The two operands may overlap if their rightmost bytes coincide.

Condition code: No change.

Example:

1. Dividend : Divisor = Quotient
   DEND : DOR = QUOT

2. Length of processing field = length QUOT + length DOR

   maximum length of processing field (PROFE) = length DEND + length DOR (packed bytes).

3. The dividend must be right-aligned with at least one leading zero before the division is performed.

```
|Name |Operation|Operand                  |
|-----|---------|-------------------------|
|     |    .    |                         |
|     |    .    |                         |
|     |    .    |                         |
|     | ZAP     |PROFE,DEND               |
|     | DP      |PROFE,DOR                |
|     |    .    |                         |
|     |    .    |                         |
|     |    .    |                         |
|DEND | DS      |CL4                      |
|DOR  | DS      |CL2                      |
|PROFE| DS      |CL5                      |
|     |    .    |                         |
|     |    .    |                         |
|     |    .    |                         |
```

Assume the Assembler has allocated storage locations as follows: DEND hexadecimal A09, PROFE hexadecimal F40, and DOR hexadecimal CAC. Register 9 contains hexadecimal 0400. The Assembler automatically calculates the displacements for the two operands by subtracting the contents of register 9 from the respective storage address values.

The source and object codings for the ZAP and DP are shown below.

Source statement:

```
 ZAP       PROFE,DEND
```

Assembler produced object code:

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
F8	4	3	9	758	9	609

and

Source statement:

```
 DP        PROFE,DOR
```

Assembler produced object code:

Op-code	L_1	L_2	B_1	D_1	B_2	D_2
FD	4	1	9	758	9	8AC

The results of the two instructions are shown in Figure 6.

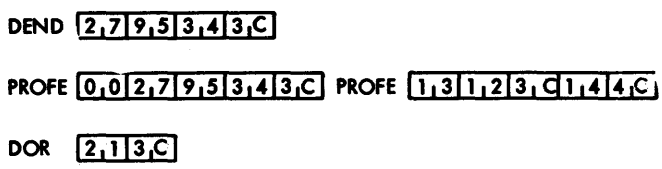


Figure 6. Decimal Division

LOGICAL OPERATIONS

There are special instructions for the non-arithmetic processing of data. The data fields are processed one byte at a time. In some cases the left four bits and the right four bits of a byte are treated separately.

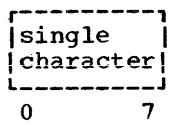
Processing of data fields in main storage proceeds from left to right. A field may start at any address excluding the reserved areas.

In logical operations the data fields are considered to contain alphameric data. An exception is the Edit-instruction which requires packed decimal numbers in the second data field.

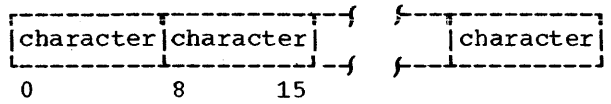
Data Format

The data are either in main storage or in the instruction itself. They may be a single character or an entire field. If two fields are used, they must be of equal length. Exception: the Edit-instruction. The two formats for logical data are:

Fixed Length (one byte; storage-immediate operations)



Variable Length (1 to 256 bytes; storage to storage operations)



In storage-to-storage (SS) operations, the fields may start at any address with exception of the first 144 bytes, which are reserved. The maximum length of a field is 256 bytes. Immediate data is limited to a length of one byte.

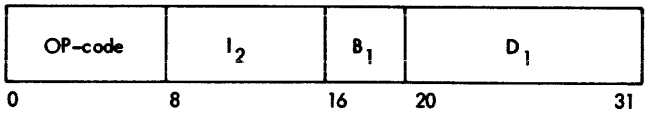
Only the EDIT operation handles data of packed format. The other instructions handle all bit combinations.

Storage-to-storage instructions may address overlapping fields. The result of overlapping depends on the particular operation. Overlapping does not influence the operation if the contents of the field remain unchanged (e.g. in a comparison). If one or both change, however, execution of the operation may be influenced by the overlapping and by the manner in which the data are rounded off and stored.

MACHINE FORMATS OF INSTRUCTIONS FOR LOGICAL OPERATIONS

Logical instructions are either in the SI-or the SS-format.

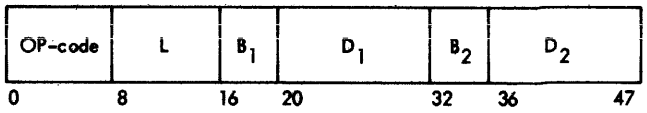
SI-Format



The first data field has a fixed length of one byte. The second operand also has a length of one byte but it is contained directly in the instruction.

The general registers are not affected by an SI-instruction.

SS-Format



The address of the each data field is the sum of the contents the respective B- and D-fields. The first and second operand fields must have the same length.

CONDITION CODE AFTER LOGICAL OPERATIONS

The results of the logical operations determine the condition code. Move-operations do not set a code. In case of the Edit-instruction the condition code indicates the status of the field to be transferred into the mask.

In the case of the Compare Logical Immediate the first data field is compared to the immediate data. In case of the Compare Logical Character the first data field is compared to the second data field.

Table of condition codes:

	00	01	10	11
Test under Mask	zero	mixed	--	one
And	zero	not zero	--	--
Compare Logical	equal	low	high	--
Or	zero	not zero	--	--
Edit	zero	< zero	> zero	--

All other logical operations leave the condition code unchanged.

Error Conditions

Error conditions which may occur during the execution of non-arithmetic operations are:

1. Operation code invalid
2. Addressing error
   a. An instruction address or an operand address refers to the protected first 144 bytes of main storage (addresses 0 to 143).
   b. An instruction address or an operand address is outside available storage.
   c. The last (highest) main-storage position contains any part of an instruction that is to be executed.

3. Specification error
   The low-order bit of an instruction address is one, i.e., no halfword boundary.

4. Data error
   An invalid digit code is contained within the second operand field of an Edit operation.

5. CPU parity error.

INSTRUCTIONS FOR LOGICAL OPERATIONS

Name	Op-Code	Format
Move Immediate (MVI)	92	SI
Move Characters (MVC)	D2	SS
Move Numerics (MVN)	D1	SS
Move Zones (MVZ)	D3	SS
Compare Logical Immediate (CLI)	95	SI
Compare Logical Character (CLC)	D5	SS
Edit (ED)	DE	SS
And Immediate (NI)	94	SI
Or Immediate (OI)	96	SI
Test under Mask (TM)	91	SI
Halt & Proceed (HPR)	99	SI
Translate (TR)	DC	SS

MVI -- MOVE IMMEDIATE

Name	Operation	Operand
blank or symbol	MVI	$D_1(B_1),I_2$

Function: The byte from I_2 is placed directly into the storage location addressed by B_1 and D_1.

Condition Code: No change.

Example: Assume register 10 contains (hexadecimal) 082E, storage location A22 (hexadecimal) contains A, the displacement in the first operand is 1F4, and the immediate data is the $.

Source statement:

```
MVI    500(10),C'$'
```

From this source statement the Assembler produces the following object code:

Op-code	I_2	B_1	D_1
92	5B	A	1F4

After execution storage location A22 contains hexadecimal 5B, a $ sign.

MVC -- MOVE CHARACTERS

Name	Operation	Operand
blank or symbol	MVC	$D_1(L,B_1),D_2(B_2)$

Function: The contents of the second data field are placed into the first data field. Processing is performed from left to right one byte at a time.

The two fields may overlap. If the first field is to the left of the second field, then transfer will proceed correctly. If the first field is exactly one byte to the right of the second field, then this byte will be propagated throughout the first field.

Condition Code: No change.

Example: Assume register 11 contains (hexadecimal) 0258, register 15 contains (hexadecimal) 04B0, storage location 3E8 (hexadecimal) contains optional data, storage location 07D0 (hexadecimal) contains C9C2D4, the displacement in the first field is 190 (hexadecimal), and that in the second field is 320 (hexadecimal).

Source statement:

```
MVC    400(3,11),800(15)
```

From this source statement the Assembler produces the following object code:

Op-code	L	B_1	D_1	B_2	D_2
D2	2	B	190	F	320

After execution storage location 03E8 contains C9C2D4.

MVZ -- MOVE ZONES

Name	Operation	Operand
blank or symbol	MVZ	$D_1(L,B_1),D_2(B_2)$

Function: The high-order four bits (the zones) of each byte in the second data field are placed into the high-order four bits of the first data field. The low order four bits (the numerics) of each byte remain unchanged. Movement is from left to right one byte at a time. The digits are not checked for validity. The fields may overlap.

Condition Code: No change.

Example: Assume register 10 contains (hexadecimal) 0890, storage location 08F4-08F7 (hexadecimal) contains F4F3F2C1, the displacement in the first operand is 67 (hexadecimal), and that in the second operand is 66 (hexadecimal).

Source statement:

```
 MVZ    103(1,10),102(10)
```

From this source statement the Assembler produces the following object code:

Op-code	L	B_1	D_1	B_2	D_2
D3	0	A	064	A	066

After execution storage location 08F4-08F7 contains F4F3F2F1.

MVN -- MOVE NUMERICS

Name	Operation	Operand
blank or symbol	MVN	$D_1(L,B_1),D_2(B_2)$

Function: The low order four bits (the numerics) of each byte in the second data field are placed, from left to right, into the corresponding low order four bits of the first field. The high order four bits (the zones) of each byte in the second field remain unchanged. The digits are not checked for validity. The fields may overlap.

Condition Code: No change.

Example: Assume register 15 contains (hexadecimal) 7DA, storage location 08A4-08A7 (hexadecimal) contains F4F3F2C1, storage location 096A (hexadecimal) contains F9F8F7D6, the displacement in the first field is C8 (hexadecimal), and that in the second field is 190 (hexadecimal).

Source statement:

```
 MVN    200(4,15),400(15)
```

From this source statement the Assembler produces the following object code:

Op-code	L	B_1	D_1	B_2	D_2
D1	3	F	0C8	F	190

After execution storage location 08A4-08A7 contains F9F8F7C6.

CLI -- COMPARE LOGICAL IMMEDIATE

Name	Operation	Operand
blank or symbol	CLI	$D_1(B_1),I_2$

Function: The eight-bit symbol of the immediate-data (the second operand) is compared to the eight bits of the first data field. The result sets the condition code. The two bytes are treated as eight-bit unsigned binary values. This results in the following order of comparison:

Special characters, lower case letters, upper case letters, digits. (System/360 collating sequence).

All 256 bit combinations are valid.

Condition Code:

00: first operand = second operand
01: first operand < second operand
10: first operand > second operand

Example: Assume register 15 contains (hexadecimal) 01F4, storage location 05DC (hexadecimal) contains E9, the displacement in the first operand is 03E8 (hexadecimal), and the immediate data is the letter A.

Source statement:

```
 CLI    1000(15),C'A'
```

From this source statement the Assembler produces the following object code:

Op-code	I_2	B_1	D_1
95	C1	F	3E8

After execution the condition code setting is 10.

CLC -- COMPARE LOGICAL CHARACTERS

Name	Operation	Operand
blank or symbol	CLC	$D_1(L,B_1),D_2(B_2)$

Function: The contents of the first data field are compared with those of the second data field. The fields may have a maximum length of 256 bytes. Comparison proceeds from left to right. The comparison is terminated as soon as inequality is encountered.

All bytes are treated alike as eight bit unsigned binary values. The order of comparison is the System/360 collating sequence: Special characters, lower case letters, upper case letters, digits. All 256 bit combinations are valid.

Condition Code:

00: first operand = second operand
01: first operand < second operand
10: first operand > second operand

Example: Assume register 11 contains (hexadecimal) 0320 storage location AF0-AF3 (hexadecimal) contains D1D6C8D5, storage location 0708-070B (hexadecimal) contains D1D6C5E8, the displacement in the first operand is 7D0 (hexadecimal), and that in the second operand is 3E8 (hexadecimal).

Source statement:

```
CLC    2000(4,11),1000(11)
```

From this source statement the Assembler produces the following object code:

Op-code	L	B_1	D_1	B_2	D_2
D5	3	E	7D0	E	3E8

After having compared the third character the condition code setting will be 10.

ED -- EDIT

Name	Operation	Operand
blank or symbol	ED	$D_1(L,B_1),D_2(B_2)$

Function: The format of the source field (the second data field) is changed from packed to zoned and is edited under control of the pattern (the first data field). The edited result replaces the pattern. The two fields must not overlap.

Editing includes sign and punctuation control and the suppressing and protecting of leading zeros. It also facilitates programmed blanking of all-zero fields. Several numbers may be edited in one operation, and numeric information may be combined with alphabetic information.

The length field applies to the pattern. It may have a maximum of 256 bytes. The pattern has unpacked format and may contain any character. The source field has packed format and must contain valid decimal digit-and sign-codes. Its left half-byte must always contain one of the digits 0-9. The right half-byte may be a digit or a sign.

Both fields are processed left to right one character at a time. Overlapping pattern-and source-fields give unpredictable results.

A so-called S-trigger controls the Edit-operation. Depending on various conditions during the operation the trigger is set either to ON or OFF. This setting determines whether a source digit or a fill character is inserted into the result field.

As mentioned before, the pattern may contain any unpacked character. However, three bit-combinations have special significance:

0010 0000 (hexadecimal 20) = digit-select character
0010 0010 (hexadecimal 22) = field-separator character
0010 0001 (hexadecimal 21) = significance-start character.

The digit-select character indicates a position in the result field into which the corresponding digit of the source field or a fill character is to be inserted.

The field-separator character is used if several source fields are to be inserted into one pattern. By setting the S-trigger to OFF it causes every source field to be treated separately. The field-separator character is always replaced by the fill character.

The significance-start character sets the S-trigger to ON. Now every character in the pattern is replaced by the respective digit of the source field or the fill character.

The S-trigger is set to OFF (0):

1. At the beginning of an Edit-operation.
2. By the field-separator character in the pattern.
3. By a positive sign (1010, 1100, 1110, 1111).

The S-trigger is set to ON (1):

1. By a valid digit (1-9) of the source field.
2. By the significance-start character in the pattern.
3. By a negative sign (1011, 1101).

During the processing of the left half-byte the sign of the right half-byte is checked and set accordingly. If a sign coincides with a valid digit or with a significance-start character in one position of the result field, the sign takes precedence and the S-trigger is set to OFF (0).

The new S-trigger setting always takes effect with the subsequent position.

The fill character, which under certain conditions, is placed into the result field, is always the first (left) character of a pattern; it is retained in the pattern (exception: the digit-select character and the significance-start character).

The S-trigger in OFF position causes:

1. The digit-select character (hexadecimal 20) and/or the significance-start character (hexadecimal 21) to be replaced by a valid digit (1-9) from the source field.
2. The fill character to be stored in place of a zero in the source field.
3. The fill character to be stored in place of any character in the pattern (exception: the digit select and the significance start characters).

The S-trigger in ON position causes:

1. The digit-select and/or the significance-start character to be replaced by any digit (0-9) from the source field.
2. A character in the pattern to remain unchanged (exception: the digit-select, field-separator, and significance-start characters).

All digits in the result field receive the zone 1111 in the binary-coded-decimal mode and the zone 0101 in the USASCII mode. The type of zone used depends on bit six, the mode bit, in the PSW.

Condition Code:
The condition code is set to:

1. 00 if the source field contains only zeros. The setting of the S-trigger has no effect.
2. 01 if the source field is not zero and the S-trigger is set to CN (1). (Negative result).
3. 10 if the source field is not zero and the S-trigger is set to CFF (0). (Positive result).

If several fields are edited with one pattern, then the condition code refers to the field being processed. If the pattern has a field-separator in the last place, then the condition code is set to zero.

The following symbols are used in the example below:

Symbol	Meaning
b (hexadecimal 40)	blank character
((hexadecimal 21)	significance-start character
) (hexadecimal 22)	field-separator character
d (hexadecimal 20)	digit-select character

If the number to be edited is a negative number, then the CR (hexadecimal C3D9) is commonly used in the last two bytes of the pattern. Since the minus sign does not reset the S-trigger, the CR will be left unchanged in the pattern. (CR stands for credit and indicates payments due).

Example: (The numbers are given in decimal notation with the hexadecimal equivalent in parentheses.)

Assume that register 12 contains 1000 (03E8),
D_1 is 0 (00),
D_2 is 200 (C8),
storage location 1000-1012 (3E8-3F4) contains bdd,dd(.ddbCR (unpacked),
storage location 1200-1203 (4B0-4B3) contained 0257426C (packed).

Source statement:

```
ED      0(13,12),200(12)
```

From this source statement the Assembler produces the following object code:

Op-code	L	B_1	D_1	B_2	D_2
DE	C	C	000	C	0C8

Processing proceeds left to right one character at a time as shown in Figure 7.

Pattern	Digit	S-trigger	Rule	Location 1000-1012
b		0	leave(1)	bdd, dd(.ddbCR
d	0	0	fill	bbd, dd(.ddbCR
d	2	1	digit	bb2,dd(.ddbCR(2)
,		1	leave	same
d	5	1	digit	bb2, 5d(.ddbCR
d	7	1	digit	bb2, 57(.ddbCR
(	4	1	digit	bb2,574.ddbCR
.		1	leave	same
d	2	1	digit	bb2, 574.2dbCR
d	6C	0	digit	bb2, 574.26bCR(3)
b		0	fill	same
C		0	fill	bb2, 574.26bbR
R		0	fill	bb2, 574.26bbb

Notes:
1. This character is saved as the fill character.
2. First non-zero digit sets S-trigger to one.
3. The plus sign in this byte sets the S-trigger to zero.

Figure 7. Processing of an Edit-Instruction

After execution location 1000-1012 (3E8-3F4) contains bb2,574.26bbb, the condition code is set to 10.

If the contents of location 1200-1203 are 00 00 02 6D, the following results are obtained:

(before) Loc 1000-1012 (3E8-3F4)
bdd,dd(.ddbCR
(after) Loc 1000-1012 (3E8-3F4)
bbbbbb.26bCR

Condition code is set to 01 (result less than zero).

In this case the significance-start character in the pattern causes the decimal point to be left unchanged. The minus sign does not reset the S-trigger so that the CR symbol is also preserved.

NI -- AND IMMEDIATE

Name	Operation	Operand
blank or symbol	NI	$D_1(B_1),I_2$

Function: The immediate data in the I_2 field and the contents of the storage location addressed in the first field are connected by the logical AND. The result (logical product) is placed into the first field.

The connective AND is applied bit by bit. If there is a 1-bit in both fields, then the 1-bit in the first operand remains unchanged. Otherwise the 1-bit in the first field will be changed to a 0-bit.

Condition Code: If all eight bits in the result field are zero, the condition code is set to 00. Otherwise it is set to 01.

Example: (The numbers are given in decimal notation with the hexadecimal equivalent in parentheses).

Assume that
register 8 contains 4096(1000),
D_1 is 1000(3E8),
I_2 is 2720(AA), in binary notation:
1010 1010,
location 5096(1060) contains 240(F0), in binary notation: 1111 0000.

Source statement:

```
NI     1000(8),X'AA'
```

From this source statement the Assembler produces the following object code:

Op-code	I_2	B_1	D_1
94	AA	8	3E8

After execution storage location 5096(1060) contains 160(A0) or in binary notation 1010 0000.

Condition code setting is 01.

OI -- OR IMMEDIATE

Name	Operation	Operand
blank or symbol	OI	$D_1(B_1),I_2$

Function: The immediate data in the I_2 field and the contents of the storage location addressed in the first field are connected by the inclusive OR. The result (logical sum) is placed into the first field.

The inclusive OR is applied bit by bit. A 0-bit in both fields will set the bit in the result field (first operand) to zero. Otherwise the resulting bit will always be one.

Condition Code: If all bits are zero, then the condition code is 00. Otherwise the condition code is set to 01.

Example: (The numbers are given in decimal notation with the hexadecimal equivalent in parentheses).

Assume that
register 8 contains 4096(1000),
D_1 is 1000(3E8),
I_2 is 2720(AA), in binary notation:
1010 1010,
storage location 5096(1060) contains
240(F0), in binary notation: 1111 1010.

Source statement:

```
OI      1000(8),X'AA'
```

From this source statement the Assembler produces the following object code:

Op-code	I_2	B_1	D_1
96	AA	8	3E8

After execution storage location 5096(1060) contains 250(FA) or in binary notation: 1111 1010.

Condition code is 01.

TM -- TEST UNDER MASK

Name	Operation	Operand
blank or symbol	TM	$D_1(B_1),I_2$

Function: The bit combination of the mask in the I_2 field is compared with the contents of the storage location addressed in the first data field. The result of the comparison sets the condition code.

The eight bits of the mask correspond bit by bit to the eight bits defined by the first data field. A comparison with a bit in the first data field is performed only if the corresponding bit in the mask contains a "1". If the bit in the mask is "0", the corresponding bit in the first data field field will not be tested.

Condition Code:

00: all bits tested were zero (also, if all bits in the mask were zero, i.e., no test).
01: some (not all) of the bits tested were one.
11: all bits tested were one.

Example: (The numbers are given in decimal notation with the hexadecimal equivalent in parentheses).

Assume that
register 8 contains 2000(07D0),
D_1 is 650(28A),
I_2 is 217(D9) or in binary notation:
1101 1001,
storage location 2650(A5A) contains 204(CC) or in binary notation: 1100 1100.

Source statement:

```
TM      650(8),X'D9'
```

From this source statement the Assembler produces the following object code:

Op-code	I_2	B_1	D_1
91	D9	8	28A

Condition code is 01.

HPR -- HALT AND PROCEED

Name	Operation	Operand
blank or symbol	HPR	$D_1(B_1),0$

Function: This instruction is used to halt the CPU. All input/output operations are continued to completion.

Execution of the program may be resumed with the next sequential instruction by pressing the Start key on the CPU.

This instruction uses the SI-Format in which the I_2 field is ignored. The effective address derived from the B_1-D_1 fields may be used to identify the Halt and Proceed instruction.

Condition Code: No change.

Example: (The numbers are given in decimal notation with the hexadecimal equivalent in parentheses).

Assume that
register 10 contains 450(01C2),
D_1 is 140(08C),
The halt number 590(24E) is shown on the E-S-T-R registers on the console as 024E.

Source statement:

```
 HPR    140(10),0
```

From this source statement the Assembler produces the following object code:

Op-code	I_2	B_1	D_1
99	00	A	08C

TR -- TRANSLATE

Name	Operation	Operand
blank or symbol	TR	$D_1(L,B_1),D_2(B_2)$

Function: This operation allows you to replace the values of one operand field by the corresponding values of a table.

Every byte in the first data field is used to look up a value in a table. The binary value of a byte is added to the starting address (given by the B_2/D_2 field) of the table. The sum is the address of the table-value wanted. This table-value replaces the byte in the first field used to locate the table-value.

Processing proceeds from left to right until the end of the first operand is reached. The maximum length may be 256 bytes. The table must contain as many bytes as indicated by the highest binary value used for searching.

Condition Code: No change.

Example: (The numbers are given in decimal notation with the hexadecimal equivalent in parentheses).

Assume that
register 10 contains 0(0000),
register 12 contains 0(0000),
D_1 is 1000 (3E8),
D_2 is 2000(7D0),
storage location 1000-1012(3E8-3F4) contains the EBCDIC characters 542156037835 and location 2000-2009(7D0-7D9) contains the EBCDIC characters 6MB0Ib3-2 (where b=blank).

Source statement:

```
 TR     1000(12,10),2000(12)
```

From this source statement the Assembler produces the following object code:

Op-code	L	B_1	D_1	B_2	D_2
DC	0B	A	3E8	C	7D0

After execution storage location 1000-1012 (3E8-3F4) contains the EBCDIC characters bIBMb360-20b (where b=blank).

BRANCH OPERATIONS

Normally the CPU processes instructions in the order of their location in main storage. Branch operations allow a departure from this sequence. They enable the machine to make logical decisions on the basis of certain conditions. For example:

- The program continues in its normal sequence.
- The program branches to a subroutine.
- Part of the program is repeated (loop).

The branch address may be obtained from one of the general registers or it may be specified in an instruction. The branch address is independent of the updated instruction address.

Branching is determined either by the condition code in the Program Status Word (PSW) or by the contents of the general registers used in the operations.

During a branch operation the rightmost half of the PSW, the updated instruction address, may be stored before it is replaced by the branch address. The stored information may be used to link the new instruction sequence with the preceding sequence.

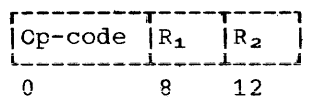
The condition code and the branch instruction are used to make logical decisions within a program. The branch operation itself does not change the condition code.

For your convenience, the Assembler program provides the facility of extended mnemonics for branch operations. Appendix A contains a list of all extended mnemonics.

MACHINE FORMATS OF INSTRUCTIONS FOR BRANCH OPERATIONS

Branching instructions can be in the RR or the RX format.

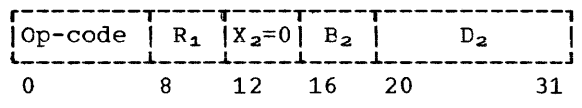
RR Format



The R_1 field may specify a general register into which the address of the next sequential instruction is to be stored as link information, or may contain a mask which is employed to identify the bit values of the condition code. In the latter case it is referred to as the M_1 field.

The R_2 field specifies the general register that contains the branch address.

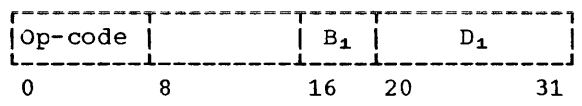
RX Format



The R_1 field may specify a general register into which the updated instruction address is to be stored as link information, or may contain a mask (then called M_1 field) that is employed to identify the bit values of the condition code.

The effective address derived from the B_2-D_2 fields is the branch address.

SI Format



The SI format is used by only one branching instruction, Set PSW. The effective address derived from the D_1-B_1 fields specifies the location of a word in main storage which is to replace the PSW (program status word). Bits 8-15 of the Set PSW instruction are ignored.

ERROR CONDITIONS

Error conditions which may occur during a branch operation are:

1. Operation code invalid.
2. Addressing error:
   a. An instruction address or a branch address refers to the protected first 144 bytes of main storage.
   b. An instruction address or a branch address is outside available storage.
   c. The R_1 field of a Branch and Store instruction contains binary values zero through seven, or the R_2 field of an RR format branch instruction contains binary values one through seven.
   d. An instruction part is located in the last (highest) two main storage positions.
3. Specification error:
   a. The low-order bit of an instruction address is one, i.e., no halfword boundary.
   b. Bits 12 through 15 of an RX format instruction are not all zero.
4. CPU parity error.

INSTRUCTIONS FOR BRANCH OPERATIONS

The branch instructions, their operation codes, formats, and mnemonics are shown in the following table:

Name	Op-Code	Format
Branch on Condition (BCR)	07	RR
Branch on Condition (BC)	47	RX
Branch & Store (BASR)	0D	RR
Branch & Store (BAS)	4D	RX
Set PSW(SPSW)	81	SI

BCR -- BRANCH ON CONDITION REGISTER

Name	Operation	Operand
blank or symbol	BCR	M_1,R_2

Function: The condition code is tested against the four bits in the mask M_1. If the condition is met, a branch occurs to the address in main storage specified by R_2. Otherwise, the next sequential instruction is executed.

There is a corresponding bit in the mask for each of the four possible condition code settings as shown below:

Bit in M_1	1	2	3	4
Condition Code	00	01	10	11

The condition for a branch is met if the mask bit corresponding to the current condition code setting is a 1-bit.

It is possible to connect several conditions by specifying a 1-bit in the corresponding mask-bit positions. An unconditional branch occurs if all four bits in the mask are 1-bits. The branch instruction is ignored if all four bits in the mask are 0-bits or if R_2 is zero.

Condition code: No change.

Example: Assume register 9 contains decimal 555 (hexadecimal 22B), the condition code in the PSW is 01, and the mask is given as hexadecimal 6.

Source statement:

```
BCR     X'6',9
```

Assembler produced object code:

Op-code	M_1	R_2
07	0110	9

A branch to the main storage location 022B will take place.

BC -- BRANCH ON CONDITION

Name	Operation	Operand
blank or symbol	BC	M_1,D_2(0,B_2)

Function: The condition code is tested against the mask M_1 (four bits). If the condition is met, a branch occurs to the storage address specified by B_2/D_2. Otherwise the next sequential instruction is executed.

For each of the four condition code settings there is a corresponding bit of the mask as shown below:

Bit in M_1	1	2	3	4
Condition Code	00	01	10	11

The condition for a branch is met if the mask bit corresponding to the current condition code setting is a 1-bit.

It is possible to connect several conditions by defining several bits in the mask accordingly. An unconditional branch occurs if all four bits in the mask are one. The branch instruction is ignored if all four bits in the mask are zero.

Condition Code: No change

Example: Assume that
D_2 is 875 decimal (36B hexadecimal),
Register 11 contains 0000,
Condition code in the PSW: 00.

Source statement:

```
BC      X'8',875(0,11)
```

Assembler produced object code:

Op-code	M_1	0	B_2	D_2
47	8	0	B	36B

A branch to main storage location 036B (hexadecimal) takes place (branch on equal).

BASR -- BRANCH AND STORE/REGISTER

Name	Operation	Operand
blank or symbol	BASR	R_1,R_2

Function: A branch is taken to the address specified by the contents of the register in the R_2 field. Next, the rightmost 16 bits of the PSW, the address of the next sequential instruction, are stored as link information in the general register specified in the R_1 field. If R_2 contains all zeros, then only the address of the next sequential instruction is loaded into the register specified by the R_1 field and no branching takes places.

Condition Code: No change.

Example:
The contents of register 10 are arbitrary. Assume that register 12 contains hexadecimal 0361 (decimal 865),
PSW 16-31 contains hexadecimal 026D (decimal 621).

Source statement:

```
 BASR   10,12
```

Assembler produced object code

Op-code	R_1	R_2
0D	A	C

After execution register 10 contains 026D and a branch is taken to storage location 0361 (hexadecimal).

BAS -- BRANCH AND STORE

Name	Operation	Operand
blank or symbol	BAS	R_1,D_2(0,B_2)

Function: The rightmost 16 bits of the PSW, the address of the next sequential instruction, are stored as link information in the general register specified by R_1. Next, the address specified by B_2/D_2 is stored as an instruction address in the PSW. This amounts to a branch to the address specified by B_2/D_2.

Condition Code: No change.

Example:
The contents of register 10 are arbitrary. Assume that register 11 contains hexadecimal 044B,
PSW bits 16-31 represent hexadecimal 036B,
D_2 is hexadecimal 12C (decimal 300).

Source statement:

```
 BAS    10,300(0,11)
```

Assembler produced object code:

Op-code	R_1	X=0	B_2	D_2
4D	A	0	B	12C

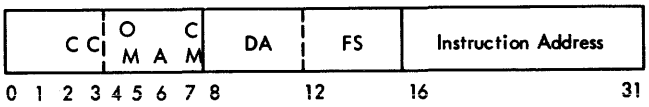
After execution register 10 contains hexadecimal 036B and a branch to storage location hexadecimal 0577 is taken.

SPSW -- SET PSW

Name	Operation	Operand
blank or symbol	SPSW	$D_1(B_1)$

Function: The only operand $D_1(B_1)$ specifies the address of a word in main storage which is to replace the PSW.

PSW Format



0-1	Not Used
2-3	Condition Code
4	Not Used
5	Overlap Mode (Submodel 5 only)
6	USASCII Mode Bit
7	Channel Mask
8-11	Device Address
12-15	Function Specification
16-31	Instruction Address

Programming Notes

1. The instruction address portion of the word which is transferred from main storage to the PSW by the Set PSW instruction should:
   a. Not refer to the protected first 144 bytes of main storage,
   b. Have the least significant bit zero, and
   c. Be within the limits of available storage.

   If these conditions are not satisfied, an addressing or specification error halt will occur.

2. The condition code is set by the Set PSW instruction to the value contained in the word transferred from main storage to the PSW.

3. Main-storage boundaries are not required of the first operand address in the Set PSW instruction.

4. The condition code, USASCII mode bit, channel mask, and overlap mode bit in the PSW are zero when the CPU is in the reset state. The instruction address portion of the PSW is not changed when the CPU is reset.

Example: Assume D_1 is 875 (hexadecimal 036B), and register 11 contains 555 (hexadecimal 022B). Bits 16 through 31 of the PSW contain 0444 (hexadecimal).

Source statement:

```
SPSW   875(11)
```

The address of the next sequential instruction as given by bits 16 through 31 of the PSW will now be 1430 (hexadecimal 0596).

INPUT/OUTPUT OPERATIONS

The Assembler program supports the following Input/Output operations:

- Control I/O (CIO)
- Test I/O and Branch (TIOB)
- Transfer I/O (XIO)

You can find a detailed description of these instructions in the SRL publication IBM System/360 Model 20 Functional Characteristics, Form GA26-5847.

It is recommended, however, that you use the IBM-supplied IOCS macro definitions for your input/output operations.

Literals

A literal is one way to introduce data into a program. It represents data itself rather than a reference to data.

Literals provide a means of entering constants (such as numbers for calculation, addresses, messages, etc.), into a program by specifying the constant in the operand of the machine instruction in which it is used. The Assembler program assembles the value specified by the literal, stores this value in a "literal pool", and places the address of the storage field containing the value in the operand field of the assembled source statement.

A literal is an alternative to using the DC Assembler instruction as a means to enter data into the program, and then using the name of the DC instruction in the operand. Literals can be used in machine instructions only. There, you may use a literal wherever a storage address is permitted as an operand.

Only one literal is allowed in a machine instruction. A literal must not be specified in the first operand of a machine instruction. It cannot be changed in storage, i.e., it must not be used as the receiving field of a machine instruction that modifies storage, e.g., STH.

A literal cannot be combined with other terms.

Literal Format: The method of describing and specifying a constant as a literal is nearly identical to the method of specifying it in the operand of a DC Assembler instruction. The major difference is that the literal must begin with an equal sign (=), which indicates to the Assembler that a literal follows. Refer to the discussion of the DC Assembler instruction operand format under Assembler Instructions for the means of specifying a literal. An address constant can be expressed as a literal. Some examples of literals are:

```
=Y(BETA)      -- address constant.
=H'1234'      -- a fixed-point number with
                 a length of two bytes.
=C'ABC'       -- a character constant.
=CL7'PAGE'    -- a character constant with
                 explicit length.
=X'1A4C'      -- a hexadecimal constant.
=B'10011110'-- a binary constant.
=P'+324'      -- a decimal constant (packed).
=Z'-541'      -- a decimal constant (zoned).
```

The instruction coded below shows one use of a literal.

Name	Operation	Operand
GAMMA	LH	10,=H'274'

The statement GAMMA is a load instruction that uses a literal as the second operand. When assembled, the second operand of the instruction will be the address at which the binary value represented by H'274' is stored.

A literal is not to be confused with the immediate data in an SI instruction. Immediate data is assembled into the instruction.

LITERAL POOL

The literals processed by the Assembler are collected and placed in a special area called the literal pool, and the location of the literal, rather than the literal itself, is assembled in the statement employing a literal.

You may control the position of the literal pool by using a LTORG instruction. Unless otherwise specified (through a LTORG instruction), the literal pool is placed at the end of the first or only control section. If this control section ends with an XFR card, the literal pool is inserted before the XFR card.

You may also specify that multiple literal pools be created by using several LTORG instructions. However, the sequence in which literals are ordered within the pool is controlled by the Assembler. Further information on positioning the literal pool(s) is given under LTORG -- Begin Literal Pool.

Assembler Instructions

Assembler instructions are requests to the Assembler to perform certain operations during the assembly. Assembler instructions, in contrast to machine instructions, do not cause machine instructions to be included in the assembled program. Some, such as DS, generate no instructions but do cause storage areas to be set aside for data. Others, such as SPACE, are effective only at assembly time; they generate nothing in the assembled program and have no effect on the location counter.

Some of the uses of assembler instructions are:

- To generate data constants for the object program.
- To reserve storage locations within the object program for use as input/output areas or as work areas.
- To control the assembly process; such as setting the location counter to some value.
- To control the listing by e.g., telling the assembler to eject to a new page.
- To tell the assembler when you intend to use a label that is defined in another program.

The following is a list of all the Assembler instructions.

Symbol-Definition Instruction

```
EQU - Equate Symbol
```

Data-Definition Instructions

```
DC  - Define Constant
DS  - Define Storage
DCCW- Define Channel Command Word
```

Program Sectioning and Linking Instructions

```
START - Start Assembly
CSECT - Identify Control Section
DSECT - Identify Dummy Section
ENTRY - Identify Entry-Point Symbol
EXTRN - Identify External Symbol
```

Base-Register Instructions

```
USING - Use Base Address Register
DROP  - Drop Base Address Register
```

Listing-Control Instructions

```
TITLE - Identify Assembly Output
EJECT - Start New Page
SPACE - Space Listing
PRINT - Print Optional Data
```

Program-Control Instructions

```
ORG   - Set Location Counter
LTORG - Begin Literal Pool
END   - End Assembly
REPRO - Reproduce Following Card
XFR   - Generate a Transfer Card
```

Symbol Definition Instruction

EQU -- EQUATE SYMBOL

The EQU instruction is used to define a symbol by assigning to it the attributes of an expression in the operand field. The format of the EQU instruction is as follows:

Name	Operation	Operand
A symbol	EQU	An expression

The expression in the operand field may be absolute or relocatable. Any symbols appearing in the expression must have been previously defined.

The symbol in the name field is given the same attributes as the expression in the operand field. The length attribute of the symbol is that of the leftmost (or only) term of the expression. The value attribute of the symbol is the value of the expression.

The EQU instruction is the means of equating symbols to register numbers, immediate data, and other arbitrary values. The following examples illustrate how this might be done:

Name	Operation	Operand
REG2	EQU	12 GENERAL REGISTER
TEST	EQU	X'3F' IMMEDIATE DATA

To reduce programming time, you may equate symbols to frequently used expressions and then use the symbols as operands in place of the expressions. Thus, in the statement:

Name	Operation	Operand
FIELD	EQU	ALPHA-BETA+GAMMA

FIELD is defined as ALPHA-BETA+GAMMA and may be used in place of the expression. ALPHA, BETA and GAMMA must all have been previously defined.

Data Definition Instructions

There are four data definition instructions: Literals, Define Constant (DC), Define Storage (DS), and Define Channel Command Word (DCCW).

These instructions are used to enter data constants into storage, to define and reserve areas of storage, and to specify the contents of channel command words. The instructions may be named so that other instructions can refer to the fields generated from them. The discussion of the DC instruction is far more extensive than that of the DS instruction, because the DS instruction is written nearly in the same format as the DC instruction. For this reason, the DC instruction is presented first and discussed in more detail than the DS instruction.

Boundary alignment varies according to the type of constant being specified. Only H- and Y-type constants are aligned to a half-word boundary unless a length modifier is specified.

Bytes that must be skipped to align the field at the proper boundary are not considered to be part of the constant.

A byte skipped in aligning statements that do not cause information to be assembled is not zeroed. Thus, a byte skipped to align a statement such as DC H'123' is zeroed, whereas a byte skipped to align a statement such as DS 2H is not zeroed.

All operand specifications are applicable to writing literals, the only differences being that

(1) the literal is preceded by an = sign

(2) a location-counter reference is not permitted in an address-constant literal.

DC -- DEFINE CONSTANT

The DC instruction is used to define constant data in storage. A variety of constants may be specified: binary, fixed-point, decimal, hexadecimal, character, and storage addresses. Appendix D summarizes, in chart form, the information concerning constants that is presented in this section. Data constants are generally called constants unless they represent storage addresses, in which case they are called address constants.

The format of the DC instruction is as follows:

Name	Operation	Operand
A symbol or blank	DC	One operand in the format described below

Format(s) of operand:

dtm'c' or dtm(c)

d = duplication factor (optional)
t = type (required)
m = length modifier (optional)
c = constant (required)

The symbol in the name field of the DC instruction statement is the name of the constant.

The value attribute of the symbol naming the DC instruction is the address of the leftmost byte (after alignment) of the constant. The length attribute depends on (1) the type of constant being defined and (2) the presence of a length specification. Implied lengths are assumed for the various constant types in the absence of a length modifier. The implied length is assigned before application of the duplication factor.

Examples of literals appear throughout the discussion of the DC instruction.

Duplication Factor: The duplication factor may be omitted. If specified, the constant is generated the number of times indicated by the factor. The duplication factor must be an unsigned decimal value. It is applied after the constant is fully assembled, i.e., after it has been developed into its proper format.

A duplication factor of zero is not permitted.

Type: The type defines the type of constant being specified. From the type specification, the Assembler determines how it

is to interpret the constant and translate it into the appropriate machine format. The type is specified by a letter code as shown in Appendix D. Further information about these constants is provided under Constant.

Length Modifier: A length modifier explicitly describes the length of a constant in bytes (in contrast to an implied length) and becomes the length attribute of the symbol in the name field.

The length modifier is written as Ln, where n is an unsigned decimal value. The value of n represents the number of bytes of storage that are assembled for the constant. The maximum value permitted for length modifier supplied for the various types of constants is summarized in Appendix D. This table also indicates the implied length for each type of constant; the implied length is used unless a length modifier is present.

A length modifier may be specified for any type of constant. You would use a length modifier when you want the assembler to pad the constant (extend it with either blanks or zeros).
For example, the instruction DC CL3'A' defines a constant having a length of three bytes, the leftmost byte containing the character and the other two bytes containing blanks.

Note: No boundary alignment will be performed when a length modifier is specified.

Constant: A data constant (all types except Y) is enclosed in apostrophes. An address constant (type Y) is enclosed in parentheses. Thus, the format for specifying the constant is one of the following:

- 'constant'
- (constant)

The total storage requirement for a data definition is the product of the length times the duplication factor (if present) plus any byte skipped for boundary alignment of the first constant.

The subsequent text describes each of the constant types and provides examples.

C -- Character Constant: Any of the valid 256 punch combinations may be designated in a character constant.

Special consideration must be given to representing apostrophes and ampersands as characters. Each apostrophe or ampersand desired as a character in the constant must be represented by two apostrophes or ampersands. Only one apostrophe or ampersand appears in storage.

The maximum length of a character constant is 32 bytes. No boundary alignment is performed. Each character is translated into one byte. Two apostrophes or two ampersands count as one character.

If no length modifier is given, the size in bytes of the character constant is equal to the number of characters in the constant. If a length modifier is provided, the result varies as follows:

1. If the number of characters in the constant exceeds the specified length, as many of the rightmost bytes as necessary are dropped.

2. If the number of characters is less than the specified length, the excess rightmost bytes are filled with blanks.

In the following example, the implied length attribute of FIELD is 12:

```
┌──────┬──────────┬────────────────────┐
│Name  │Operation │Operand             │
├──────┼──────────┼────────────────────┤
│FIELD │DC        │C'TOTAL IS 110'     │
└──────┴──────────┴────────────────────┘
```

However, in this next example, the explicit length attribute is 15, and three blanks appear in storage to the right of the zero:

```
┌──────┬──────────┬────────────────────┐
│Name  │Operation │Operand             │
├──────┼──────────┼────────────────────┤
│FIELD │DC        │CL15'TOTAL IS 110'  │
└──────┴──────────┴────────────────────┘
```

In the next example, the implied length attribute of FIELD is 12, although 13 characters appear in the operand. The two ampersands count as only one byte.

```
┌──────┬──────────┬────────────────────┐
│Name  │Operation │Operand             │
├──────┼──────────┼────────────────────┤
│FIELD │DC        │C'TOTAL IS &&10'    │
└──────┴──────────┴────────────────────┘
```

Note that in the next example, a length of four has been specified, but there are five characters in the constant.

```
┌──────┬──────────┬────────────────────┐
│Name  │Operation │Operand             │
├──────┼──────────┼────────────────────┤
│FIELD │DC        │3CL4'ABCDE'         │
└──────┴──────────┴────────────────────┘
```

The generated constant would be:

ABCDABCDABCD

The same constant could be specified as a literal as follows:

```
|Name   |Operation  |Operand               |
|-------|-----------|----------------------|
|       |MVC        |AREA(12),=3CL4'ABCDE' |
```

On the other hand, if the length had been specified as six instead of four, the generated constant would have been:

ABCDE ABCDE ABCDE

X -- Hexadecimal Constant: A hexadecimal constant consists of one or more of the hexadecimal digits 0-9 and A-F. The maximum length of a hexadecimal constant is 32 bytes (64 hexadecimal digits). No halfword boundary alignment is performed.

Constants that contain an even number of hexadecimal digits are translated as one byte per pair of digits. If an odd number of digits is specified, the leftmost byte has the leftmost four bits filled with a hexadecimal zero, while the rightmost four bits contain the odd (first) digit.

If no length modifier is specified, the implied length of the constant is half the number of hexadecimal digits in the constant (a hexadecimal zero is added to the high-order byte if there is an odd number of digits). If a length modifier is specified, the constant is handled as follows:

1. If the number of bytes the constant could occupy exceeds the specified length, the extending leftmost bytes are dropped.

2. If the number of bytes the constant could occupy is less than the specified length, the necessary bytes are added to the left and filled with hexadecimal zeros.

A four-digit hexadecimal constant provides a convenient way to set the bit pattern of a binary halfword. The constant in the following example would set the bits of the first byte of a halfword to ones:

```
|Name   |Operation  |Operand               |
|-------|-----------|----------------------|
|       |DS         |0H                    |
|TEST   |DC         |X'FF00'               |
```

The DS instruction sets the location counter to a halfword boundary.

In the following example, the digit A would be dropped, because five hexadecimal digits are specified for a length of two bytes:

```
|Name       |Operation  |Operand             |
|-----------|-----------|--------------------|
|ALPHACON   |DC         |3XL2'A6F4E'         |
```

The resulting constant would be 6F4E, which would occupy the specified two bytes. It would then be duplicated three times, as requested by the duplication factor. If it had merely been specified as X'A6F4E', the resulting constant would have had a hexadecimal zero in the leftmost position:

0A6F4E

B -- Binary Constant: A binary constant is written using ones and zeros enclosed in apostrophes. Duplication and length may be specified. The maximum length of a binary constant is eight bytes.

The implied length of a binary constant is the number of bytes occupied by the constant, which includes any necessary padding. Padding or truncation takes place on the left. The padding bit used is a zero.

The following example shows the coding used to designate a binary constant. BCON would have an implied length attribute of one.

```
|Name     |Operation  |Operand               |
|---------|-----------|----------------------|
|BCON     |DC         |B'11011101'           |
|BTRUNC   |DC         |BL1'100100011'        |
|BPAD     |DC         |BL1'101'              |
```

BTRUNC would be assembled with the leftmost bit dropped as follows:

00100011

BPAD would be assembled with five padding zeros, as follows:

00000101

H -- Fixed-Point Constant: A fixed-point constant is defined as an integer and written as a signed or unsigned decimal value. A positive sign is assumed if an unsigned number is specified.

The decimal value is converted to its binary equivalent and assembled as a halfword. It is aligned on halfword boundary if a length is not specified. An implied length of two bytes is assumed. A length of one or two bytes may be specified by a length modifier, in which case no boundary alignment occurs.

Highest positive and negative values for a fixed-point constant are:

Length	Max	Min
2	2^{15}-1(=32767)	-2^{15}(=-32768)
1	2^{7}-1(=127)	-2^{7}(=-128)

The binary number occupies the rightmost portion of the field in which it is placed. The unoccupied portion (i.e., the leftmost bits) is filled with the sign. A 1-bit for positive and a 0-bit for negative numbers.

If the value of the number exceeds the length, the necessary leftmost bits are dropped after conversion. A negative number is carried in twos complement form.

A halfword is generated from the statement shown below. The value attribute of CONWRD is the address of the left byte of the halfword, and the length attribute is two, which is the implied length for a halfword fixed-point constant.

```
|Name    |Operation |Operand            |
|--------|----------|-------------------|
|CONWRD  |DC        |H'658'             |
```

The next example uses a halfword constant as a literal and loads ones into bits 8 through 15 of register 15.

```
|Name  |Operation |Operand              |
|------|----------|---------------------|
|      |LH        |15,=H'255'           |
```

P and Z -- Decimal Constants: A decimal constant is written as a signed or unsigned decimal value. If the sign is omitted, a plus sign is assumed. The maximum length of a decimal constant is 16 bytes. No halfword boundary alignment is performed.

If zoned decimal format (Z) is specified, each decimal digit is translated into one byte. Except for the rightmost byte, the translation is done according to the character set shown in Appendix J. The rightmost byte contains the sign in its left half-byte and the rightmost digit of the decimal constant in its right half-byte.

In packed decimal format (P), the rightmost byte contains the rightmost decimal digit in its left half-byte and the sign in its right half-byte. The other decimal digits are "packed" two at a time into one byte.

If you specify an even number of decimal digits, one digit will be left unpaired, because the rightmost digit is paired with the sign. Therefore, in the leftmost byte, the leftmost four bits will be set to zeros and the rightmost four bits will contain the first digit. The bit configuration for the digits is identical to the configurations for the hexadecimal digits 0-9 as stated under Hexadecimal Self-Defining Term.

For both packed and zoned decimal numbers, a plus sign is translated into the hexadecimal digit C, and a minus sign into the digit D.

If no length modifier is given, the implied length for either constant is the number of bytes the constant occupies (taking into account the format, sign, and possible addition of zero bits for packed decimals). If a length modifier is given, the constant is handled as follows:

1. If the constant requires fewer bytes than the length specifies, the necessary number of bytes is added to the left. For zoned decimal format, the character zero is placed in each added byte. For packed decimals, all eight bits of each added byte are set to zero.

2. If the constant requires more bytes than the length specifies, the necessary number of leftmost digits or pairs of digits is dropped, depending on the specified format (zoned or packed).

For example, the instruction DC P'12' is translated into hexadecimal 012C, and the instruction DC Z'-543' into hexadecimal F5F4D3.

The following example illustrates the use of a packed decimal literal.

```
|Name  |Operation |Operand              |
|------|----------|---------------------|
|      |AP        |OUTAREA,=PL2'+25'    |
```

Y -- Address Constant: Address constants are normally used for initializing base registers to facilitate the addressing of storage. Furthermore, they provide the means of communicating between control sections of a multi-section program. The latter is explained in the section Base Register Instructions.

An address constant, unlike other types of constants, is enclosed in parentheses and specified as an absolute, relocatable, or complex relocatable expression. (Complex relocatable expressions are discussed below.)

The value of the expression may range between -2^{15}(=-32768) and 2^{15}-1(=32767). The implied length of an address constant is two bytes, and the value is placed in the rightmost portion. Alignment is to a

halfword boundary, unless a length is specified. A length modifier may be used, in which case no alignment will occur. The length that may be specified depends on the type of expression used for the constant; a length of 1-2 bytes may be used for an absolute expression, while a length of two bytes must be used for a relocatable or complex relocatable expression.

If an address constant contains a location-counter reference, the location counter value used is the storage address of the first byte the constant will occupy.

If you specify a duplication for an address constant containing a location-counter reference, the value of the location counter used in each duplication is incremented by the length of the constant.

In the following example, the field generated from the statement named ACONST contains a constant that occupies two bytes. Note that there is a location-counter reference. The value of the location counter will be the address of the first byte allocated to the constant. The second statement below shows an address constant used as a literal. Since a location-counter reference is not permitted within a literal, the instruction must be named and the name used in the literal if a location-counter reference is desired. The instruction ADCON will generate the address of the constant named FIELDA.

Name	Operation	Operand
ACONST	DC	Y(*+4096)
A	LH	14,=Y(A)
FIELDA	DC	H'101'
ADCON	DC	Y(FIELDA)

Complex Relocatable Expressions: These expressions contain two or three unpaired relocatable terms or a negative relocatable term in addition to any absolute or paired relocatable terms that may be present. A complex relocatable expression may only be used to specify an address constant. Unlike relocatable expressions, complex relocatable expressions may represent a negative value. A complex relocatable expression may consist of external symbols and designate an address in an independent assembly that is to be linked and loaded with the assembly containing the address constant.

For example, if SECTION1 and SECTION2 name two consecutive sections, the instruction

```
DC  Y(SECTION2-SECTION1)
```

is a complex relocatable expression constant describing the length of SECTION1.

Relocation Dictionary (RLD): If an address constant is specified by a relocatable or a complex relocatable expression, the Assembler automatically places certain information into the relocation dictionary. This information tells the Linkage Editor that this address constant must be updated when the program is relocated and how this updating is to be performed.

DS -- DEFINE STORAGE

The DS instruction is used to reserve areas of storage and to assign names to those areas. The use of this instruction is the preferred way of symbolically defining storage for work areas, input/output areas, etc.

Name	Operation	Operand
A symbol or blank	DS	One operand written in the format described below

The format of the DS operand is similar to that of the DC operand. It consists of a duplication factor, a type code, and a length modifier. The rules for DC instructions are also applicable for DS instructions with the following exceptions:

1. A duplication factor of zero is permitted. (It does not advance the location counter).
2. Only constants of types C and H are permitted in the DS instruction. A duplication factor is permitted for both types.
3. The length modifier may only be specified for the C-type constant. (Range 0-256).
4. The specification of data is not permitted in a DS operand.

If you have a symbol in the name field of a DS instruction, its value attribute is the location of the leftmost byte of the reserved area. The length attribute of the symbol is the length (implicit or explicit) of the type of data specified. Any positioning required for aligning the storage area to the proper type of boundary is done before the address value is determined. Skipped bytes are not zeroed.

A fixed-point field (H) has an implied length of two bytes. The leftmost byte is aligned to a halfword boundary. Use this code if you desire to reserve two bytes of storage aligned to a halfword boundary. A

duplication factor would have to be used to reserve a larger area, because the maximum length specification for this type is two bytes.

Character (C) fields have an implied length of one byte. If you use this code, you would have to specify a length modifier, unless you want to reserve just one byte. Although no alignment occurs, the use of a C-type field permits greater latitude in length specifications, the maximum for this type being 256 bytes.

The size of a storage area that can be reserved by using the DS instruction is limited only by the maximum value of the location counter. Since the maximum length specification is 256, an area larger than 256 must be specified with a duplication factor. For example, the statement

```
DS 2CL200
```

can be used to reserve 400 positions of main storage.

To define four 10-byte fields and one 100-byte field, the respective DS instructions might be as follows:

```
|Name      |Operation |Operand          |
|----------|----------|-----------------|
|FIELD     |DS        |4CL10            |
|AREA      |DS        |CL100            |
```

Although FIELD might have been specified as one 40-byte field, the preceding definition has the advantage of providing FIELD with a length attribute of 10. This would be important when using FIELD as a machine-instruction operand governed by a length consideration.

Additional examples of DS statements are shown below:

```
|Name |Operation|Operand                    |
|-----|---------|---------------------------|
|ONE  |DS       |CL80 (one 80-byte field,   |
|     |         | length attribute of 80)   |
|TWO  |DS       |80C (80 one-byte fields,   |
|     |         | length attribute of one)  |
|THREE|DS       |4H (four halfwords,        |
|     |         | length attribute of       |
|     |         | two)                      |
```

Note: A DS instruction causes the storage area to be reserved, but not to be set to zeros. You cannot assume that the area contains zeros or data saved from a previous program or program phase.

Special Uses of the Duplication Factor

Forcing Alignment: The location counter can be forced to a halfword boundary by using the H-type field with a duplication factor of zero. This method may be used to obtain boundary alignment that otherwise would not be provided. For example, the following statements would set the location counter to the next half-word boundary and then reserve storage for a 128-byte field (whose leftmost byte would be on a half-word boundary).

```
|Name      |Operation |Operand          |
|----------|----------|-----------------|
|          |DS        |0H               |
|AREA      |DS        |CL128            |
```

Defining Fields of an Area: A DS instruction with a duplication factor of zero may be used to assign a name and a length to an area of storage without actually reserving the area.

A DS statement for C-type fields with a duplication factor of zero does not advance the location counter. Additional DS and/or DC instructions may then be used to reserve the area and assign names to fields and constants within this area.

For example, assume that 80-character records are to be read into an area for processing and that each record has the following format:

Positions 5-10	Payroll number
Positions 11-30	Employee name
Positions 31-36	Date
Positions 47-54	Gross wages
Positions 55-62	Withholding tax

The following example illustrates how you might use DS instructions to assign a name to the record area, then define the fields of the area and allocate storage for them.

```
|Name      |Operation |Operand          |
|----------|----------|-----------------|
|RDAREA    |DS        |0CL80            |
|          |DS        |CL4              |
|PAYNO     |DS        |CL6              |
|NAME      |DS        |CL20             |
|DATE      |DS        |0CL6             |
|DAY       |DS        |CL2              |
|MONTH     |DS        |CL2              |
|YEAR      |DS        |CL2              |
|          |DS        |CL10             |
|GROSS     |DS        |CL8              |
|FEDTAX    |DS        |CL8              |
|          |DS        |CL18             |
```

The first instruction names the entire area by defining the symbol RDAREA; the instruction gives RDAREA a length attribute of 80 bytes, but does not reserve any storage. Similarly, the fifth statement names a 6-byte area by defining the symbol DATE; the three subsequent statements actually define the fields of DATE and allocate storage for them. The second, ninth, and last statements are used for spacing purposes and, therefore, are not named.

DCCW -- DEFINE CHANNEL COMMAND WORD

The DCCW instruction provides a convenient way to define and generate a 6-byte channel command word aligned at a half-word boundary. The format of the DCCW instruction is:

Name	Operation	Operand
A symbol or blank	DCCW	Four operands, separated by commas, specifying the contents of the channel command word

The internal machine format of a channel command word is described in the SRL publication IBM System/360 Model 20, Functional Characteristics, Form GA26-5847.

All four operands must appear. They are written, from left to right, as follows:

First operand: An absolute expression specifying the command code. The value of this expression is right-aligned in byte one.

Second operand: An absolute expression. The value of this expression is right-aligned in byte two.

Third operand: An absolute or relocatable expression specifying a storage address. The value of this expression is right-aligned in bytes 3-4.

Fourth operand: An absolute expression. The value of this expression is right-aligned in bytes 5 and 6.

For further details see the pertinent hardware SRL publication.

The following is an example of a DCCW instruction for a magnetic tape read:

Name	Operation	Operand
	DCCW	X'02',X'80',READAREA,80

If READAREA represents, for example, the value 1204, the assembled CCW is 028012040080.

If there is a symbol in the name field of the DCCW instruction, it is assigned the address value of the leftmost byte of the channel command word. The length attribute of the symbol is six.

Program Sectioning and Linking Instructions

It is often convenient, or necessary, to write a large program in sections. The sections may be assembled separately and then combined into one object program. The Assembler provides facilities for creating multi-section programs and symbolically linking separately assembled program sections.

Program sectioning and linking is closely related to the specification of base registers for each control section. Sectioning and linking examples are given under CSECT -- Identify Control Section and Addressing An External Control Section.

CONTROL SECTIONS

A control section is the smallest logical unit of a program. All elements of a control section are in a constant relationship to each other. Therefore, the control section is the smallest separately relocatable unit of a program. If a program is sectioned, it must be written so that control passes properly from one control section to another, regardless of the position of the control section in main storage.

A program is divided into control sections if it is to be assembled in several parts. (Program parts assembled at one time are often called an assembly.) In a multi-section program, each control section must be complete. An unsectioned program is considered a single control section.

Since you have described storage symbolically you know what eventually will be entered into storage, regardless of whether you write an unsectioned program, a multi-section program, or part of a multi-section program but you will, most likely, not know where in storage a section appears. There is no constant relatioship between individual control sections. Thus, knowing the location of one control section does not make another control section addressable by relative addressing.

The output of the Assembler consists of the assembled control sections, an External Symbol Dictionary and a Relocation Dictionary.

The External Symbol Dictionary contains information the Linkage Editor program needs to complete cross-referencing between control sections as it combines them into one object program. The Linkage Editor program can take control sections from various assemblies and combine them properly with the help of the corresponding External Symbol Dictionaries. Successful combination of separately assembled control sections depends on the techniques used to provide symbolic linkages between the control sections. This is described in the sections below describing the CSECT, ENTRY, and EXTRN instructions.

The Relocation Dictionary contains information about certain address constants (see DC-instruction) which must be updated by the Linkage Editor Program when a control section is relocated.

The External Symbol Dictionary is contained in the ESD-cards in front of the object deck. The Relocation Dictionary is contained in the RLD-cards mingled with the TXT-cards.

The Linkage Editor program assigns locations to control sections in such a way that the sections are placed in storage consecutively, in the same order as they occur in the program. Each control section subsequent to the first begins at the next available half-word boundary.

A control section is normally identified by the CSECT instruction. However, if it is desired to specify a tentative starting address, the START instruction may be used to identify the first control section of an assembly.
The first control section of an assembly has the following special properties.

1. Its tentative starting location may be specified as an absolute value.

2. It normally contains the literals requested in the program, although their positioning can be altered. For further explanation on positioning of literals see the discussion of the LTORG instruction.

Limitations
The combined number of control sections and dummy sections (see Dummy Control Sections) for an assembly must not exceed eight. The combined number of control sections and dummy sections plus the number of unique symbols in EXTRN statements for an assembly must not exceed 31. A maximum number of 20 ENTRY instructions can be processed in a single assembly.

START -- START ASSEMBLY

The START instruction may be used to give a name and starting address to the first (or only) control section of an assembly. The START instruction may be preceded only by AWORK, AOPTN (in this order), ICTL, ISEQ, REPRO, EJECT, SPACE, PRINT, TITLE instructions, and comments statements. There must be only one START instruction in an assembly.

The format of the START instruction is as follows:

Name	Operation	Operand
A symbol or blank	START	A self-defining term or blank

If a symbol names the START instruction, the symbol is established as the name of the control section. If not, the control section is considered to be unnamed. All subsequent statements are assembled as part of that control section. This continues until a CSECT instruction identifying the beginning of the next control section or a DSECT instruction is encountered.

A CSECT instruction named by the same symbol that names a START instruction is invalid. An unnamed CSECT instruction that occurs in a program initiated by an unnamed START instruction is also invalid.

The symbol in the name field is a valid relocatable symbol whose value represents the address of the first byte of the control section. It has a length attribute of one.

The Assembler uses the self-defining value specified by the operand as the starting location of the first control section. This value must be divisible by two. For example, either of the following statements could be used to assign the name PROG2 to the first control section and to indicate an initial assembly location of 2040:

Name	Operation	Operand
PROG2	START	2040
PROG2	START	X'7F8'

If the operand in a START instruction is blank, the Assembler checks if NORLD is specified as the operand of an AOPTN instruction, provided such an instruction is given. If NORLD is not specified, the

Assembler assumes that the program shall be relocatable and sets the starting address to zero. If it is specified, the Assembler regards the program as not relocatable and sets the starting address to the address of the first available halfword behind the Monitor.

If you omit the START instruction, the Assembler assumes one with blank name and operand fields.

CSECT -- IDENTIFY CONTROL SECTION

The CSECT instruction identifies the beginning of a control section. The format of the CSECT instruction is as follows:

Name	Operation	Operand
A symbol or blank	CSECT	Blank; or a comment preceded by a comma.

The symbol that names the CSECT instruction is the name of the control section; a blank indicates an unnamed section. All statements following the CSECT instruction are assembled as part of that control section until a statement identifying the beginning of the next control section (i.e., another CSECT or a DSECT instruction) is encountered.

The symbol in the name field is a valid relocatable symbol whose value represents the address of the first byte of the control section. It has a length attribute of one. Only one CSECT statement with the same name is permitted within a program.

If you wish to use a symbol defined in one control section as an operand in another of the same assembly, you must write a USING instruction telling the Assembler which register to use as the base register for that control section. The unpaired term in the operand v in the USING instruction (see USING -- Use Base Address Register) must be defined in that same control section.

An additional USING instruction is needed because a CSECT instruction causes the Assembler to disregard all previous USING instructions of the same assembly. Figure 8 illustrates these rules.

```
|Name   |Operation|Operand                          |
|-------|---------|---------------------------------|
|*************BEGIN OF PROGRAM************|
|SECT1  |START    |0                                |
|BEG1   |BASR     |10,0                             |
|       |USING    |*,10                             |
|       |.        |                                 |
|       |.        |                                 |
|       |USING    |SECT2,11                         |
|A1     |LH       |11,=Y(SECT2)                     |
|       |MVC      |FIELD1,FIELD2                    |
|       |.        |                                 |
|       |.        |                                 |
|       |USING    |SECT3,12                         |
|B1     |LH       |12,=Y(SECT3)                     |
|       |CLC      |FIELD1,FIELD3                    |
|       |.        |                                 |
|       |.        |                                 |
|FIELD1 |DS       |H                                |
|**********SECOND CONTROL SECTION*********|
|SECT2  |CSECT    |                                 |
|BEG2   |BASR     |11,0                             |
|       |USING    |*,11                             |
|       |.        |                                 |
|       |.        |                                 |
|       |USING    |FIELD1,10                        |
|A2     |LH       |10,=Y(FIELD1)                    |
|       |MVC      |FIELD2,FIELD1                    |
|       |.        |                                 |
|       |.        |                                 |
|       |USING    |FIELD3,12                        |
|B2     |LH       |12,=Y(FIELD3)                    |
|       |CLC      |FIELD2,FIELD3                    |
|       |.        |                                 |
|       |.        |                                 |
|FIELD2 |DS       |H                                |
|**********THIRD CONTROL SECTION**********|
|SECT3  |CSECT    |                                 |
|BEG3   |BASR     |12,0                             |
|       |USING    |*,12                             |
|       |.        |                                 |
|       |.        |                                 |
|       |USING    |BEG1+B1-A1,10                    |
|A3     |LH       |10,=Y(BEG1+B1-A1)                |
|       |MVC      |FIELD3,FIELD1                    |
|       |.        |                                 |
|       |.        |                                 |
|       |USING    |BEG2+B2-A2,11                    |
|B3     |LH       |11,=Y(BEG2+B2-A2)                |
|       |CLC      |FIELD3,FIELD2                    |
|       |.        |                                 |
|       |.        |                                 |
|FIELD3 |DS       |H                                |
|       |.        |                                 |
|       |.        |                                 |
|       |END      |, END OF PROGRAM                 |
```

Figure 8. Example of a Multi-Section Program

The MVC instruction in the control section named SECT1 uses FIELD2 as an operand and the CLC instruction uses FIELD3 as an operand. Both FIELD2 and FIELD3 are not defined in control section SECT1. Therefore a USING statement must be issued prior

to using each symbol as an operand. USING SECT2,11 tells the Assembler that a symbol defined in SECT2 will be used and that its base register is 11. Likewise, USING SECT3,12 tells that a symbol defined in SECT3 will be used and that its base register is register 12.

In the control section named SECT2 the instruction USING FIELD1,10 tells the Assembler to use register 10 as base register to address control section SECT1 since FIELD1 is defined in that control section. The assumed base address is the address of the instruction named FIELD1.

In the control section named SECT3, to use a different method, the instruction USING BEG2+B2-A2,11 tells the Assembler to use register 11 as base register to address control section SECT2 because the unpaired term BEG2 is defined in that control section. The assumed base address is the value of the expression BEG2+B2-A2.

The statements named A1,B1,A2,B2, and A3,B3 load the base register specified in the respective USING statements immediately preceding each statement with the address of the first operand in each USING statement.

Unnamed Control Section

If neither a named CSECT instruction nor a named START instruction appears at the beginning of the program, the Assembler determines that it is to assemble an unnamed control section as the first (or only) control section. Only one unnamed control section is permitted in a program. If you write a small program that is unsectioned, you need not use a CSECT instruction.

DUMMY CONTROL SECTIONS

A dummy control section is not part of the object program; it only serves to describe the layout of an area of storage without actually reserving storage. (It is assumed that the storage is reserved by another assembly).

DSECT -- IDENTIFY DUMMY SECTION

The DSECT instruction identifies the beginning of a dummy section. More than one dummy section may be defined per assembly, but each must be named. The format of the DSECT instruction is as follows:

Name	Operation	Operand
A symbol	DSECT	Blank; or a comment preceded by a comma.

The symbol in the name field must be a valid relocatable symbol whose value represents the first byte of the dummy section. It has a length attribute of one.

Symbols that appear in the name field of a DSECT instruction or in the name field of an instruction in a dummy section may be used in USING instructions. Therefore, they may be used in program elements (e.g., machine instructions and data definitions) that specify storage addresses. An example illustrating the use of a dummy section appears under Addressing Dummy Sections.

A symbol that names a statement in a dummy section may be used in an address constant (see DC instruction) only if it is paired with another symbol (with the opposite sign) from the same dummy section.

Dummy-Section Location Assignment

A location counter is used to determine the relative locations of named program elements in a dummy section. The location counter is always set to zero at the beginning of the dummy section, and the location values assigned to symbols that name statements in the dummy section are relative to the initial statement in the section.

Addressing Dummy Sections

Suppose you wish to describe the format of an area whose storage location will not be determined until the program is executed. You describe the format of the area in a dummy control section and use symbols defined in the dummy section as the operands of machine instructions. To reference the storage area, you must:

1. Provide a USING instruction specifying both a general register that the Assembler can assign to the machine instructions as a base register and an address value from the dummy section that the Assembler may assume the register contains.
2. Ensure that the same register is loaded with the actual address of the storage area.

Because the location counter is set to zero at the beginning of the dummy control section, the values assigned to symbols defined in a dummy control section are relative to the initial statement of that section. Thus, all machine instructions

referring to names defined in the dummy section will, at execution time, refer to storage locations relative to the address loaded into the register.

An example is shown in the following coding. Assume that two independent assemblies (assembly 1 and assembly 2) have been loaded and are to be executed as a single overall program. Assembly 1 is an input routine that places a record into a specified area of storage, places the address of the input area containing the record into general register 13, and branches to assembly 2. Assembly 2 processes the record. The coding shown in the example is from assembly 2.

```
|Name     |Operation |Operand              |
|---------|----------|---------------------|
|ASMBLY2  |START     |0                    |
|BEGIN    |BASR      |12,0                 |
|         |USING     |*,12                 |
|         |  .       |                     |
|         |  .       |                     |
|         |USING     |AREA,13              |
|         |CLI       |CODE,C'A'            |
|         |BE        |ATYPE                |
|         |  .       |                     |
|         |  .       |                     |
|ATYPE    |MVC       |WORKA,PUTA           |
|         |MVC       |WORKB,PUTB           |
|         |  .       |                     |
|         |  .       |                     |
|WORKA    |DS        |CL20                 |
|WORKB    |DS        |CL18                 |
|         |  .       |                     |
|         |  .       |                     |
|AREA     |DSECT     |                     |
|CODE     |DS        |CL1                  |
|PUTA     |DS        |CL20                 |
|PUTB     |DS        |CL18                 |
|         |  .       |                     |
|         |END       |                     |
```

The input area is described in assembly 2 by the dummy control section named AREA. Fields of the input area that are to be processed are named in the dummy control section as shown. The Assembler instruction USING AREA,13 designates general register 13 as the base register to be used in addressing the DSECT control section and indicates that general register 13 is assumed to contain the address of AREA.

Assembly 1, during execution, loads the actual beginning address of the input area into general register 13. Because the symbols used in the DSECT section are defined relative to the initial statement in the section, the address values they represent will, at the time of program execution, be the actual storage locations of the input area.

SYMBOLIC LINKAGES

Symbols may be defined in one assembly and referred to in another, thus allowing symbolic linkages between independently assembled sections. Linkages are only possible if the Assembler is able to provide information about the externally defined symbols to the Linkage Editor, which resolves these symbols into addresses. The Assembler places the necessary information into the External Symbol Dictionary if the particular symbols are specified in the ENTRY and EXTRN instructions. Symbolic linkages are described as linkages between independent assemblies; more specifically, they are linkages between independently assembled control sections.

In the program where the linkage symbol is defined (i.e., used as a name), it must also be identified to the Assembler by means of the ENTRY Assembler instruction, except when the symbol appears in the name field of a START or CSECT instruction. It is identified as a symbol that names an entry point, which means that another program will use that symbol in order to effect a branch operation or a data reference. The Assembler places this information in the External Symbol Dictionary.

Similarly, the program that uses a symbol defined in some other program must identify it by the EXTRN Assembler instruction. It is identified as an externally defined symbol (i.e., defined in another program) that is used to effect linkage to the point of definition. The Assembler places this information into the External Symbol Dictionary.

ENTRY -- IDENTIFY ENTRY-POINT SYMBOL

The ENTRY instruction identifies a linkage symbol that is defined in one assembly but may be used in another assembly. An ENTRY instruction must not appear in an unnamed control section or in a dummy section. The format of the ENTRY instruction is as follows:

```
|Name  | Operation | Operand                |
|------|-----------|------------------------|
|Blank | ENTRY     | A relocatable          |
|      |           | symbol that also       |
|      |           | appears as a state-    |
|      |           | ment name              |
```

The symbol in the ENTRY operand field may be used in the operand field of instructions in other assemblies. The symbol in the operand field must not be defined in an unnamed control section or in a dummy control section. The following example identifies the statements named SINE and COSINE as entry points to the assembly.

```
|Name  | Operation | Operand              |
|------|-----------|----------------------|
|      | ENTRY     | SINE                 |
|      | ENTRY     | COSINE               |
```

Note: The name of a control section need not be identified by an ENTRY instruction when another assembly uses it as an entry point. The Assembler automatically places information on control section names in the External Symbol Dictionary.

Limitation: A maximum of 20 ENTRY statements can be processed in a single assembly.

EXTRN -- IDENTIFY EXTERNAL SYMBOL

The EXTRN instruction identifies a linkage symbol that is used in this assembly but defined in some other assembly. Each linkage symbol must be identified, even symbols that name external control sections. The format of the EXTRN instruction is as follows:

```
|Name  | Operation | Operand              |
|------|-----------|----------------------|
|Blank | EXTRN     | A relocatable        |
|      |           | symbol               |
```

The symbol in the operand field must not appear as the name of a statement in this assembly.

Limitation: The combined number of control sections, dummy sections, and symbols specified in EXTRN instructions must not exceed 31 for one assembly.

The following example identifies three external symbols that have been used as operands in this assembly but are defined in some other assembly.

```
|Name  | Operation | Operand              |
|------|-----------|----------------------|
|      | EXTRN     | RATETBL              |
|      | EXTRN     | PAYCALC              |
|      | EXTRN     | WITHCALC             |
```

External symbols should be used only in address constants. But if you do wish to use an external symbol in a machine instruction, you must write an USING statement before using the symbol as an operand as illustrated in the following example:

```
|Name  |Operation|Operand                |
|------|---------|-----------------------|
|      |EXTRN    |FIELD                  |
|      |         |.                      |
|      |         |.                      |
|      |         |.                      |
|      |LH       |8,YFIELD               |
|      |USING    |FIELD,8                |
|      |CH       |9,FIELD                |
|      |         |.                      |
|      |         |.                      |
|      |         |.                      |
|YFIELD|DC       |Y(FIELD)               |
|      |.        |                       |
|      |.        |                       |
```

An example that employs the EXTRN instruction appears under Addressing An External Control Section.

ADDRESSING AN EXTERNAL CONTROL SECTION

To link a program to a control section in a different assembly, proceed as follows:

1. Identify the external symbol with the EXTRN instruction, and create an address constant from the symbol.
2. Load the constant into a general register, and use the register for base addressing or branch to the section via the register.

Figure 9 shows the coding that might be used to incorporate a subroutine named SUBROUT (which is an external control section), to branch to this subroutine, and to branch back to the main program.

```
| NAME      | OPERATION | OPERAND AND COMMENTS                                          |
|-----------|-----------|---------------------------------------------------------------|
|MAINPROG   | START     | 0              MAIN PROGRAM                                   |
|BEGIN      | BASR      | 12,0           STORE INSTRUCTION COUNTER INTO R12              |
|           | USING     | *,12           USE R12 FOR ADDRESSING THE MAIN PROGRAM        |
|           | .         |                                                               |
|           | .         |                                                               |
|           | .         |                                                               |
|           | EXTRN     | SUBROUT        DEFINE SUBROUT AS NAME OF EXTERNAL SECTION     |
|           | .         |                                                               |
|           | .         |                                                               |
|           | .         |                                                               |
|           | LH        | 10,SUBADDR     LOAD ADDRESS OF EXTERNAL SECTION INTO R10       |
|CALLSUBR   | BASR      | 11,10          BRANCH TO SUBROUT                              |
|           | .         |                                                               |
|           | .         |                                                               |
|           | .         |                                                               |
|SUBADDR    | DC        | Y(SUBROUT)     ADDRESS OF EXTERNAL SECTION                    |
|           | END       | BEGIN          BRANCH TO BEGIN                                |
```

```
| NAME      | OPERATION | OPERAND AND COMMENTS                                          |
|-----------|-----------|---------------------------------------------------------------|
|SUBROUT    | CSECT     | ,              CONTROL SECTION EXTERNAL TO MAIN PROGRAM       |
|           | USING     | *,10                                                          |
|           | .         |                                                               |
|           | .         |                                                               |
|           | .         |                                                               |
|           | BR        | 11             RETURN TO INSTRUCTION FOLLOWING CALLSUBR       |
|           | END       |                                                               |
```

Figure 9. Addressing an External Control Section

Base Register Instruction Statements

USING -- USE BASE ADDRESS REGISTER

By means of the USING instruction you tell the assembler

- which pseudo registers (0 through 7) or which general registers (8 through 15) are available as base registers for implicit addressing;
- for which control section such a base register is available;
- what value the register(s) will contain at object time.

A USING instruction does not load the registers specified. It is your responsibility to ensure that the specified base address values are placed into the registers (see the BASR instruction). An example follows the description of the DROP instruction.

A USING instruction has effect only within the control section where it is contained and, within that control section, it applies only to instructions following it in the program. With the beginning of a new control section (see CSECT and DSECT instructions) all previously available base registers are dropped automatically. You must use at least one USING instruction for each control section you want to address.

The format of the USING instruction is:

Name	Operation	Operand
Blank	USING	v,r_1,r_2,r_3,r_4

Operands v and r_1 are mandatory. Operands r_2, r_3, and r_4 are optional.

Operand v must be a relocatable expression. It specifies a value that the Assembler can use as a base address. The unpaired relocatable term of this expression refers to that control section for which base register(s) are to be made available by this instruction. No literals are permitted.

The operands r_1, r_2, r_3, r_4 must be absolute expressions, whose value must be between 0 and 15. Operand r_1 specifies the register that can be assumed to contain the base address represented by the operand v. Operands r_2, r_3, and r specify registers that can be assumed to contain v+4096, v+8192, v+12288, respectively. For example, the statement:

```
|Name   |Operation  |Operand              |
|-------|-----------|---------------------|
|       |USING      |*,8,9                |
```

tells the Assembler to assume that at object time the current value of the location counter (indicated by the *) will be in general register 8, and that the current value of the Location Counter, incremented by 4096, will be in general register 9.

The registers r_1, r_2, r_3, r_4 address that control section where the unpaired term of the expression v is defined. For an example see the section Program Sectcning and Linking Instructions. Thus, if you want to address two different control sections you must use two USING instructions.

If you change the value in a base register currently used and wish the Assembler to compute displacements from this value, you must tell the Assembler the new value by means of another USING statement. In the following sequence the Assembler first assumes that the value of ALPHA is in register 9. The second statement then causes the Assembler to assume that ALPHA+1000 is the value in register 9.

```
|Name  |Operation  |Operand              |
|------|-----------|---------------------|
|      |USING      |ALPHA,9              |
|      |.          |                     |
|      |.          |                     |
|      |USING      |ALPHA+1000,9         |
```

If you wish to use more than four registers as base registers to address one control section you must use two or more USING instructions.

Whenever a storage location is specified by a relocatable expression in an operand of a machine instruction, the Assembler checks for an available base register to separate the storage address into a base address value and a displacement value. To this end the assembler determines:

- which control section the relocatable expression refers to (i.e. in which control section the unpaired relocatable term of the expression is defined);
- if a base register is available for that control section (i.e. if you issued a USING instruction).

If a base register is available, the assembler determines, in order to get a positive displacement, whether or not the base address value to be assumed for this register (see USING instruction) is lower than or equal to the storage address to be separated. The difference between the base address and the storage address must not exceed 4095 (hexadecimal FFF), because exactly three halfbytes are reserved in an instruction to hold the displacement specification.

If more than one base register satisfies the above condition, the assembler will always choose the one giving the smallest displacement. If more than one register gives the same displacement, the numerically highest register will be chosen.

USING instructions may specify the pseudo registers 0 through 7 as base registers. This is referred to as direct addressing. In this case, the object program cannot be relocated by the Linkage Editor Program because the Linkage Editor Program does not update a direct address in the operand of a machine instruction.

The Assembler assumes fixed contents for pseudo base registers as shown in the following list:

Register	Contents
0	0
1	4,096
2	8,192
3	12,288
4	16,384
5	20,480
6	24,576
7	28,672

The Assembler always uses these values. However, a check is performed to determine whether the expression v matches the contents of the pseudo base register referred to, and a warning is issued if they do not match. Unlike the general registers 8 through 15, the pseudo registers need not be loaded in a program.

You may make the object program relocatable (referred to as indirect addressing) at some future time by making the following changes in the source program and reassembling it:

1. Replacing pseudo registers in the USING statement by general registers.
2. Loading the new specified base registers with a relocatable value.

The pseudo registers must not be used as registers for working with data.

DROP -- DROP BASE REGISTER

The DROP instruction specifies a previously available register that must no longer be used as a base register. The format of the DROP instruction is as follows:

Name	Operation	Operand
Blank	DROP	Up to four absolute expressions of the form r_1,r_2,r_3,r_4

Operand r_1 is mandatory, operands r_2, r_3, and r_4 are optional. r_1, r_2, r_3, and r_4 are absolute expressions indicating registers previously named in a USING statement and are now unavailable for implicit addressing. The following statement, for example, prevents the Assembler from using registers 9 and 11:

Name	Operation	Operand
	DROP	9,11

If more than four registers are to be made unavailable for base addressing, two or more DROP instructions must be issued.

It is not necessary to use a DROP statement before the base address in a register is changed by a USING statement; nor are DROP statements needed at the end of a source program.

A register made unavailable by a DROP instruction can be made available again by a subsequent USING instruction.

PROGRAMMING EXAMPLE

In the following sequence, the BASR instruction loads register 12 with the address of the first storage location immediately following. In this case, it is the instruction named FIRST. The USING instruction indicates to the Assembler that register 12 contains the address of this instruction. When you employ this method, the USING instruction must immediately follow the BASR instruction. No other USING or load instructions are required if the location named LAST is within 4095 bytes of FIRST.

Name	Operation	Operand
BEGIN	BASR	12,0
	USING	*,12
FIRST	.	
	.	
	.	
LAST	.	
	END	BEGIN

In the following example, the BASR and LH instructions load registers 12-15. The USING instruction indicates to the Assembler that these registers are available as base registers for addressing a maximum of 16,384 consecutive bytes of storage, beginning with the location named HERE. The number of addressable bytes may be increased or decreased by altering the number of registers designated by the USING and LH instructions and the number of address constants specified in the DC instructions.

Name	Operation	Operand
BEGIN	BASR	12,0
	USING	HERE,12,13,14,15
HERE	LH	13,BASEADDR
	LH	14,BASEADDR+2
	LH	15,BASEADDR+4
	B	FIRST
BASEADDR	DC	Y(HERE+4096)
	DC	Y(HERE+8192)
	DC	Y(HERE+12288)
FIRST	.	
	.	
	.	
LAST	.	
	END	BEGIN

Restrictions on Register Usage

Registers 8, 9, 10, 14, and 15 have special uses and are available to you only under certain conditions. Register 9 is used by the DPS IOCS. Registers 8, 14, and 15 are used by the FETCH routine. Register 10 is used by the LOAD routine. Neither the FETCH routine nor the IOCS nor the LOAD routine save the contents of these registers prior to using them. If you use these registers you must save their contents (and restore them later) or be finished with them before the FETCH routine or IOCS make use of the registers.

If you use IOCS-routines and specify a DTFEN overlay you must issue a new USING 10 after each OPEN and CLOSE instruction, because a DROP 10 instruction is given within the OPEN/CLOSE routine. For further details see the SRL publications

IBM System/360 Model 20, Tape Programming System, Input/Output Control System, Form GC24-9003,

IBM System/360 Model 20, Disk Programming System, Input/Output Control System, Form GC24-9007.

Registers 11-13 are available to you without any restriction. You will, as a matter of fact, decrease the possibility of errors if you try to use only these three registers. However, if there is a shortage of registers all general registers 8 through 15 are available to you under the restrictions stated above.

Listing—Control Instruction Statements

The listing-control instructions are used to identify an assembly listing and assembly output cards, to provide blank lines or skip pages in an assembly listing, and to designate how much detail is to be included in an assembly listing. In no case are instructions or constants generated in the object program.

TITLE -- IDENTIFY ASSEMBLY OUTPUT

The TITLE instruction enables you to identify the assembly listing and assembly output cards. The format of the TITLE instruction is as follows:

Name	Operation	Operand
Name or blank	TITLE	A sequence of characters, enclosed in apostrophes

If the first TITLE statement in a program appears before the START statement, it may contain an entry in the name field. This entry may contain one to four alphabetic or numeric characters in any combination. Any additional characters are ignored. The contents of the name field are punched into columns 73-76 of all the output cards for the program, except in cards produced by means of a REPRO Assembler instruction. Subsequent TITLE statements must not contain a name entry.

The operand field of a TITLE statement may contain up to 62 characters enclosed in apostrophes. The contents of the operand field are printed at the top of each page of the assembly listing that follows it, until another TITLE statement is encountered. The TITLE statement itself does not appear in the source listing unless it is found to be incorrect. Each TITLE statement causes the listing to be advanced to a new page (before the heading is printed).

For example, if the following statement is the first TITLE statement to appear in a program, and it appears before the START statement:

Name	Operation	Operand
PGM1	TITLE	'FIRST HEADING'

then PGM1 is punched into all of the output cards (columns 73-76), except those produced by a REPRO statement, and the heading FIRST HEADING appears at the top of each page that follows it.

If the following statement:

Name	Operation	Operand
	TITLE	'A NEW HEADING'

occurs later in the program, PGM1 is still punched into the output cards, but each following page begins with the heading: A NEW HEADING.

EJECT -- START NEW PAGE

The EJECT instruction affects only the assembly listing and provides a convenient way to separate program routines in the listing. This instruction causes the remainder of the present page to be skipped and the listing to continue at the top of the next page, below the heading line. If the line preceding the EJECT statement appears at the bottom of a page, the EJECT has no effect.

If two or more EJECT instructions are issued in succession, a complete page is skipped for each EJECT instruction after the first and the listing continues on the page that is in printing position after the last EJECT instruction is executed. Each page that is skipped is printed with a heading line, however. The format of the EJECT instruction is:

Name	Operation	Operand
Blank	EJECT	Blank; or a comment preceded by a comma.

The EJECT statement itself does not appear in the source listing.

SPACE -- SPACE LISTING

The SPACE instruction is used to insert one or more blank lines in the listing. The format of the SPACE instruction is as follows:

Name	Operation	Operand
Blank	SPACE	A decimal value up to 56, or blank

A decimal value is used to specify the number of blank lines to be inserted in the assembly listing. A blank operand causes one blank line to be inserted. If the specified value exceeds the number of lines remaining on the listing page, the statement will have the same effect as an EJECT statement. The SPACE statement itself does not appear in the source listing, unless it is found to be incorrect.

The SPACE instruction in the following example would cause three blank lines to appear in your source listing between the add instruction and the move instruction.

Name	Operation	Operand
	MVC	HALF,OLD
	SPACE	3
	AH	15,HALF

PRINT -- PRINT OPTIONAL DATA

The PRINT instruction is used to control printing of the assembly listing. The format of the PRINT instruction is:

Name	Operation	Operand
Blank	PRINT	One to three operands

Up to three operands may be used, that is, one out of each of the following groups: OFF or ON, GEN or NOGEN, DATA or NODATA.

OFF - No listing is printed. No execution of listing-control statements.

ON - A listing is printed.

GEN - All statements generated by macro instructions are printed.

NOGEN - Statements generated by macro instructions are not printed. However, the macro instruction itself and messages resulting from the MNOTE instruction, if used, will appear in the listing. (The MNOTE instruction is described under MNOTE -- Request for Error Message.) Any instruction that contains one or more Assembler-detected errors is also printed along with the appropriate diagnostic message(s).

DATA - Constants are fully printed out in the listing.

NODATA - Only the first eight bytes (16 hexadecimal digits) of the assembled data are printed in the listing.

A program may contain any number of PRINT statements. The condition set by a print statement remains in effect until another PRINT statement is encountered.

If an operand is omitted, its specification is assumed to remain in effect. If OFF is specified, GEN and DATA have no effect. If NOGEN is specified, DATA has no effect for generated DC instructions.

Until the first PRINT statement (if any) is encountered, the Assembler assumes that a PRINT instruction with the operands ON, NODATA, and GEN was given.

For example, if the statement DC XL32'00' appears in a program, 32 bytes of zeros are assembled. If the statement:

Name	Operation	Operand
	PRINT	DATA

is the last PRINT statement to appear before the DC statement, all 32 bytes of zeros are printed in the assembly listing. However, if:

Name	Operation	Operand
	PRINT	NODATA

is the last PRINT statement to appear before the DC statement, only eight bytes of zeros are printed in the assembly listing.

Program—Structure Control Instructions

The program-structure control instructions are used to influence the structure of the program to be assembled.

REPRO -- REPRODUCE FOLLOWING STATEMENT

The output of the Assembler program may be processed by the Linkage Editor program or the CMAINT program. Both programs require a socalled PHASE statement for operation. This statement must be included in the assembler output (the object deck). (See the SRL publication, Control and Service Programs, Form GC24-9006). Instead of waiting until you have the object deck, and then manually inserting the PHASE card, you may include it in your source deck if you use a REPRO statement immediately preceding the PHASE card.

The REPRO Assembler instruction causes the Assembler to punch a duplicate of any card immediately following the REPRO instruction. The punched cards resulting from REPRO instructions appear at the same point in the assembled text as they appeared in the source program. They are not, however, processed by the Assembler program.

If any REPRO instructions precede the START instruction or the implied start position (if no START instruction is used), the cards punched will precede the ESD cards for the assembly.

The format of the REPRO Assembler instruction is as follows:

Name	Operation	Operand
Blank	REPRO	Blank; or a comment preceded by a comma.

The following example illustrates the use of the REPRO instruction statement:

Name	Operation	Operand
	REPRO	
	PHASE	PROGA,A,X'1200'
	START	0
	•	
	•	
	•	

XFR -- GENERATE A TRANSFER CARD

The XFR instruction is provided to cause the generation of a transfer card at the same location the XFR instruction appears in the source program.

A transfer card is used by the loader of the TPS Basic Monitor and the TPS or DPS CMAINT and Linkage Editor program to define the transfer point or entry point of a phase, or subphase.

The format of the XFR instruction is as follows:

Name	Operation	Operand
Blank	XFR	A relocatable symbol

The symbol in the operand field must appear within the assembly, or be previously defined as either an entry point or an external symbol.

ORG -- SET LOCATION COUNTER

The ORG instruction is used to alter the setting of the location counter for the current control section. Each ORG statement causes a new output text card to be started. The format of the ORG instruction is:

Name	Operation	Operand
Blank	ORG	A relocatable expression or blank

Any symbols in the expression must have been previously defined. The unpaired relocatable symbol must be defined in the same control section in which the ORG statement appears.

The location counter is set to the value of the expression in the operand. If the operand is omitted, the location counter is set to a location that is one byte higher than the highest location assigned for the control section up to this point.

An ORG instruction must not be used to specify a location below the beginning of the control section in which it appears. For example, the instruction:

Name	Operation	Operand
	ORG	*-500

is invalid if it appears less than 500 bytes from the beginning of the current control section.

If you need to reset the location counter to a value that is one byte beyond the highest location yet assigned (in the control section), the following instruction would be used:

Name	Operation	Operand
	ORG	

If previous ORG statements have reduced the location counter for the purpose of redefining a portion of the current control section, a new ORG instruction without an operand can then be used to terminate the effects of such statements and restore the Location Counter to its highest setting in the control section.

LTORG -- BEGIN LITERAL POOL

The Assembler program places all literals encountered in a literal pool. This literal pool is automatically placed at the end of the first control section by the assembler. If you wish the literal pool to be placed at a different location (for example, when you use subphases within one control section), use an LTORG instruction.

The LTORG instruction causes all literals thus far encountered in the source program up to the LTORG statement (either from the beginning of the program or from a previous LTORG statement) to be assembled at appropriate boundaries starting at the first halfword boundary following the LTORG statement.

The format of the LTORG instruction is:

Name	Operation	Operand
Symbol or blank	LTORG	Blank; or a comment preceded by a comma.

The symbol represents the address of the first byte of the literal pool. It has a length attribute of one.

An LTORG instruction must not be used within a dummy section.

Special Addressing Considerations: Any literals used after the last LTORG statement in a program are placed at the end of the first control section. If there are no LTORG statements in a program, all literals used in the program are placed at the end of the first control section. Under these circumstances, you must ensure that the first control section is always addressable. This means that the base address register for the first control section should not be changed through usage in subsequent control sections. If you do not wish to reserve a register for this purpose, you may place an LTORG instruction at the end of each control section, thereby ensuring that all literals appearing in that section are addressable.

END -- END ASSEMBLY

The END instruction is required. It ends the assembly of a program. It may also designate a point in the program or in a separately assembled program to which control may be transferred after the program is loaded. The END instruction must always be the last statement of any source program.

The format of the END instruction statement is:

Name	Operation	Operand
Blank	END	A relocatable expression or blank

The operand specifies the point to which control is to be transferred when loading is completed. This point is usually the first machine instruction in the program, as shown in the following sequence.

Name	Operation	Operand
NAME	CSECT	
AREA	DS	50H
BEGIN	BASR	12,0
	USING	*,12
	.	
	.	
	.	
	END	BEGIN

If the END statement contains a symbolic address in the operand field, the Assembler automatically punches the transfer address into the END card.

Note: If the operand contains an external symbol, only a single-term relocatable expression is permitted.

Often it is desirable to divide a large program into several parts for execution. These parts are called phases. A phase may consist of one "head" phase and of up to nine subphases.

Phases of one program may either be assembled together or seperately.

A phase without subphases may consist of one or more control sections. If the object program created by the Assembler is to be processed by the Linkage Editor program, the beginning of a phase must coincide with the beginning of a control section. Two parts of a phase assembled separately may be combined to one phase by the Linkage Editor Program.

If you use the phasing technique you must use the FETCH or LOAD macro instruction. Its functions (and special considerations if you use IOCS in your program) are described in the SRL publication IBM System/360 Model 20, Disk Programming System, Input/Output Control System, Form GC24-9007. For information on the Tape IOCS refer to the publication IBM System/360 Model 20, Tape Programming System, Input/Output Control System, Form GC24-9003.

You must catalog a program phase in the core-image library under a unique name. A subphase can be cataloged only as part of its head phase.

The CMAINT (Core-Image Maintenance) program is available for cataloging program phases in the core-image library. You can load the cataloged phases into main storage for execution, one at a time, either consecutively or seperately. If a phase consists of a head phase and one or more subphases, the first subphase can be initiated only by the headphase and each subsequent subphase by its preceding subphase.

Overlay Using the FETCH Macro

CODING OF PHASES WITHOUT SUBPHASES

To code phases without subphases you must apply the following rules:

- Each phase must begin with a REPRO instruction followed by a PHASE statement. (For the first phase in an assembly these two statements must precede the START instruction).
- Issue a FETCH macro instruction with operand at the point in one phase where you want another phase to be loaded. The operand of the FETCH instruction specifies the name of the phase to be loaded.
- Each phase, except the last one in an assembly, must end with an XFR instruction. The last phase must end with an END instruction.
- If you use literals in your program you should issue a LTORG instruction in each phase to ensure that the literals are defined in the same phase in which they are used.

The following example demonstrates how to use the phasing technique. It is assumed that a Linkage Editor run is not required.

Name	Operation	Operand
	REPRO	
	PHASE	FIRST,A,4096
	START	4096
	USING	*,1,2
PHASE1	.	
	.	
EXIT1	FETCH	SECOND
	.	
	.	
	LTORG	
	XFR	PHASE1
*		
	REPRO	
	PHASE	SECOND,A,4386
	ORG	PHASE1+290
PHASE2	.	
	.	
EXIT2	FETCH	THIRD
	.	
	.	
	LTORG	
	END	PHASE2
	Second Assembly	
	REPRO	
	PHASE	THIRD,A,4386
	START	4386
	USING	*-290,1,2
PHASE3	.	
	.	
	END	PHASE3

When phase SECOND is loaded it overwrites phase FIRST except for the first 290

bytes. These 290 bytes may be used as data areas or to contain subroutines or both. Phase THIRD is fetched by phase SECOND and in turn overwrites it.

The next example shows an almost identical program. Only this time the object program generated by the Assembler must be processed by the Linkage Editor program before it can be cataloged.

```
|Name    |Operation|Operand                  |
|--------|---------|-------------------------|
|        |REPRO    |                         |
|        |PHASE    |FIRST,S,0                |
|CSECT1  |START    |0                        |
|PHASE1  |BASR     |12,0                     |
|        |USING    |*,12,13                  |
|        |LH       |13,=Y(PHASE1+4098)       |
|        |  .      |                         |
|        |  .      |                         |
|EXIT1   |FETCH    |SECOND                   |
|        |  .      |                         |
|        |  .      |                         |
|        |LTORG    |                         |
|        |XFR      |PHASE1                   |
|*       |         |                         |
|        |ORG      |CSECT1+290               |
|CSECT2  |CSECT    |                         |
|        |REPRO    |                         |
|        |PHASE    |SECOND, L, 290, CSECT1   |
|PHASE2  |BASR     |12,0                     |
|        |USING    |*,12                     |
|        |  .      |                         |
|        |  .      |                         |
|EXIT2   |FETCH    |THIRD                    |
|        |  .      |                         |
|        |  .      |                         |
|        |LTORG    |                         |
|        |END      |PHASE2                   |
|--------|---------|-------------------------|
|        Second Assembly                     |
|--------|---------|-------------------------|
|        |REPRO    |                         |
|        |PHASE    |THIRD,L,0,CSECT2         |
|CSECT3  |START    |0                        |
|PHASE3  |BASR     |12,0                     |
|        |USING    |*,12                     |
|        |  .      |                         |
|        |  .      |                         |
|        |END      |PHASE3                   |
```

CODING OF A PHASE WITH SUBPHASES

To code a phase with subphases apply the following rules:

- If your program must be processed by the Linkage Editor program before it can be cataloged, the head phase and the subphases must be contained in one control section.
- A REPRO instruction followed by a PHASE statement is required only for the head phase. If the beginning of the head phase coincides with the beginning of the assembly, these two statements must precede the START instruction.
- The load address for a subphase is derived from the load address contained in the first TXT-card of this subphase.
- The head phase and the subphases must end with a XFR instruction. If the end of the last subphase coincides with the end of the assembly, this subphase must end with an END instruction.
- If a Linkage Editor run is required before cataloging, issue a REPRO instruction followed by the Linkage Editor control statement ACTION DUP prior to the XFR instruction of the head phase. This ensures that the Linkage Editor does not ignore all subsequent XFR and END instructions.
- Issue a FETCH macro instruction without an operand at the point in the head and subphases where you want the subsquent subphase to be loaded into main storage.

The following example shows how to code a phase with subphases. It is assumed that a Linkage Editor run is not required.

```
|Name    |Operation|Operand                  |
|--------|---------|-------------------------|
|        |REPRO    |                         |
|        |PHASE    |PROGR1,A,4096            |
|        |START    |4096                     |
|        |USING    |*,1,2,3                  |
|BEGIN   |  .      |                         |
|        |  .      |                         |
|        |  .      |                         |
|EXITH   |FETCH    |                         |
|        |  .      |                         |
|        |  .      |                         |
|        |LTORG    |                         |
|        |XFR      |BEGIN                    |
|*       |         |                         |
|        |ORG      |BEGIN+4098               |
|SUBPH1  |  .      |                         |
|        |  .      |                         |
|EXIT1   |FETCH    |                         |
|        |  .      |                         |
|        |  .      |                         |
|        |LTORG    |                         |
|        |XFR      |SUBPH1                   |
|*       |         |                         |
|        |ORG      |BEGIN+4098               |
|SUBPH2  |  .      |                         |
|        |  .      |                         |
|EXIT2   |FETCH    |                         |
|        |  .      |                         |
|        |  .      |                         |
|        |END      |SUBPH2                   |
```

In the following example a Linkage Editor run is required before the phase can be cataloged.

```
Name     Operation  Operand
         REPRO
         PHASE      PROGR1,S,0
         START      0
BEGIN    BASR       11,0
         USING      *,11,12
         LH         12,=Y(BEGIN+X'1002')
          .
          .
EXITH    FETCH
          .
          .
         LTORG
         REPRO
         ACTION     DUP
         XFR        BEGIN
*
         ORG        BEGIN+X'1002'
SUBPH1    .
          .
         FETCH
          .
          .
         LTORG
         XFR        SUBPH1
         REPRO
         ACTION     NODUP
*
         ORG        BEGIN+X'1002'
SUBPH2    .
          .
         LTORG
         END        SUBPH2
```

Overlay Using the LOAD Macro

You can use the same technique with the LOAD macro as with the FETCH macro. The LOAD macro is used to load selfrelocatable phases or subphases; it differs from the FETCH macro in the following two points:

- the load address of the phase or subphase is specified in the operand of the LOAD instruction,

- after loading the phase or subphase, control is given to the next sequential instruction.

An example illustrating the use of the LOAD macro to load a phase (without subphases) is given below. It is assumed that no Linkage Editor run is required.

First assembly:

```
Name     Operation  Operand
         REPRO
         PHASE      ROOT,A,4096
         START      4096
         USING      *,1,2
ROOTPH    .
          .
         LOAD       MODULE,YMOD
CONT      .
          .
YMOD      .
          .
         END
```

Second Assembly:

```
Name     Operation  Operand
         REPRO
         PHASE      MODULE,A,0
         START      0
          .
          .
         END
```

When the program comes to the instruction LOAD, it loads the phase named MODULE (see second assembly) to the address of YMOD. After the phase MODULE has been loaded, the program continues with the instruction named CONT.

Macro Instructions

The Assembler includes a macro feature that can be used to reduce the amount of repetitive coding required for general, frequently used routines. For example, the routines for transferring records from magnetic tape to main storage, checking for accuracy, and deblocking to obtain a single record for processing are used for any logical input file on tape. Such routines involve many instructions that can be written once and, with modification, may be used repeatedly in any number of programs.

The macro feature is composed of two basic parts:

1. source-program macro instructions

2. a macro library of pre-written flexible routines called macro definitions.

A direct relationship exists between these two parts, i.e., a single macro instruction written in the source program is replaced, in the object program, by a routine taken from the macro library. The macro definition contained in the macro library consists of a series of instructions. Thus, many instructions are assembled for one macro instruction.

The same operation code is used in the macro instruction as in the macro definition. Therefore, the proper routine to be included in the object program is found by matching of operation codes.

As the instructions of a macro definition are assembled, they can be tailored to fit the particular problem program by a substitution process. The first statement of a macro definition (following the macro header) is the prototype statement. It defines the format of the macro instruction and contains various symbolic operands (called symbolic parameters) for which values may be substituted when the macro definition is used by a specific program.

The macro instruction in the source program specifies the values of the symbolic parameters (commonly called parameters) that are to be substituted in the macro definition when it is assembled. An example of this is:

```
&NAM ADD &S1,&S2,&SUM    Prototype statement in macro library
     .
     .
A    ADD RAT1,RAT2,TRAT  Example of a corresponding macro instruction
```

The example illustrates the prototype statement of an addition routine that could be used by any program to add any two terms and store the sum in a specified location. Program A might use the macro instruction to add RAT1 to RAT2 and store the result in a field named TRAT.

The parameters applicable to the specific job are specified in the macro instruction. The parameters are substituted for the symbolic parameters in the prototype statement when the macro routine is assembled. The parameters are also substituted in all the statements that follow the prototype statement to actually perform the addition. The statements following the prototype statement are called model statements.

For the above addition example, the complete macro definition routine might be:

```
     MACRO                  Header statement
&NAM ADD   &S1,&S2,&SUM     Prototype statem.
&NAM LH    13,&S1           Model statement
     AH    13,&S2           Model statement
     STH   13,&SUM          Model statement
     MEND                   Trailer statement
```

The & character preceding the symbolic name is part of the macro-language syntax as explained in the following sections.

IBM provides a number of pre-written macro definitions and specifies the macro instructions you can use to call these routines from the library. You can write your own macro definitions and store them in the macro library.

There are two groups of IBM-supplied macro definitions:

- IOCS macro definitions
- Monitor macro definitions

MACRO-INSTRUCTION FORMAT

The format of a macro instruction in a source-language statement must correspond to the format of the prototype statement in the macro definition. Therefore, the format of the prototype statement determines the form in which the macro instruction must be written in the source program.

The name field in the macro instruction may contain a name if the name field of the prototype statement contains a symbolic parameter.

The operation field in the macro instruction must contain exactly the same mnemonic operation code as the prototype statement, e.g., ADD. This may be any alphameric code with a maximum of five characters, the first of which must be alphabetic.

The operands in the operand field of a macro instruction must be written in the same format as the symbolic parameters in the operand field of the prototype statement. Either the positional format or the keyword format may be used.

POSITIONAL MACRO INSTRUCTIONS

The format of a positional macro instruction is as follows:

Name	Operation	Operand
A symbol or blank	Mnemonic operation code	Up to 49 operands, separated by commas, in the form described below

If the name field of a positional prototype statement contains a symbolic parameter, the name field of a positional macro instruction may either contain a symbol or be blank. If the name field is blank, the symbolic parameter in the macro definition is considered to be a null parameter. (Null parameters are described below.)

If the name field of a positional macro definition is blank, the name field of the positional macro instruction should be blank. If an entry is present it will be ignored.

If an entry is made in the name field of a macro instruction, the entry must conform to the format for a symbol, regardless of whether or not it will be used as a symbol by the macro definition.

The operation field of a macro instruction contains the same operation code that appears in the operation field of the corresponding prototype statement.

The placement and order of the operands in a positional macro instruction is determined by the placement and order of the symbolic parameters in the operand field of the prototype statement.

Any combination of up to eight characters may be used as a macro instruction operand if the following rules are observed:

1. Apostrophes must always occur in pairs.
2. Two apostrophes must be used to represent one apostrophe enclosed in paired apostrophes.
3. If an apostrophe is immediately preceded by the letter L, and immediately followed by a letter, the apostrophe is not considered in determining paired apostrophes.
4. Parentheses must always occur in pairs, left parenthesis then right parenthesis.
5. Nesting of parentheses is not permitted.
6. A parenthesis that occurs between paired apostrophes is not considered in determining paired parentheses.
7. An equal sign may occur only as the first character in an operand or within paired apostrophes.
8. A comma indicates the end of an operand unless placed between paired parentheses or paired apostrophes.
9. A blank indicates the end of the operand field unless placed between paired apostrophes.
10. Each group of consecutive ampersands must be of an even number.

The following are examples of valid macro instruction operands:

```
SYMBOL          A+2
123             L'WORKAR
*               =H'4096'
X'189A'         0(2,3)
```

Note: All characters are generated.

The following are invalid macro instruction operands, for the reasons stated:

I'NAME	Apostrophe not preceded by L
5A)B	Single parenthesis not enclosed in apostrophes
5,(0,3)	First comma not enclosed in parentheses or apostrophes
(15 B)	Blank does not occur between paired apostrophes
(TO,FROM)	More than eight characters

If no operand is specified for a symbolic parameter in the prototype statement, the comma that would have separated it from the next operand must not be omitted. If the last operand (or operands) are omitted from a macro instruction, the trailing comma is not required.

Any symbolic parameter for which a name or operand is not specified in the macro instruction becomes a null parameter.

The following example shows a macro instruction preceded by its corresponding prototype statement. The third and sixth operands of the macro instruction in this example are omitted and are therefore considered to be null parameters.

Name	Operation	Operand
	EXMPL	&A,&B,&C,&D,&E,&F
	EXMPL	17,**14,,AREA,FIELD6

If the symbolic parameter that corresponds to a null parameter is used in a model statement, a null character value replaces the symbolic parameter in the generated statement. The result will be the same as though the symbolic parameter did not appear in the statement.

For example, the first statement below is a model statement containing the symbolic parameter &C. If the operand that corresponds to &C is omitted from the macro instruction, the second statement is generated from the model statement.

Name	Operation	Operand
	MVC	TH&C,THIS
	MVC	TH,THIS

The positional prototype statement can be written in a format similar to the format used for other Assembler-language statements. To allow for the inclusion of up to 49 parameters in the prototype statement of a macro definition, use as many continuation cards as are required. The name field, if used, must begin in the begin column. The operation field followed by at least one blank must appear on the first card of the statement. The other rules are:

1. If the parameters in the operand field extend up to the end column and column 72 contains a nonblank character, the parameters may be continued in the continue column of the next card. A single parameter may be split between two cards.

2. A blank following a parameter signifies the end of all symbolic parameters.

3. Comments may appear after the blank that indicates the end of all parameters, up to and including column 71.

As many continuation cards as are required may be used in a positional macro instruction.

Unless changed by an ICTL instruction during assembly, the begin column for a macro instruction is assumed to be column 1, the end column is assumed to be column 71, and the continue column is assumed to be column 16.

This format may be changed by an ICTL instruction, the operand of which may be 25 or 25,71,38. If 25 is specified, column 25 is the begin column, and column 71 is the end column. No continuation cards will be recognized. If 25,71,38 is specified, the begin column is column 25, the end column is column 71, and the continue column (for macro instructions only) is column 38.

KEYWORD MACRO INSTRUCTIONS

The format of a keyword macro instruction is as follows:

Name	Operation	Operand
A symbol or blank	Mnemonic operation code	Up to 49 operands, separated by commas, in the form described below.

This format provides a direct association between the operands of the macro instruction and those of the corresponding prototype statement.

The very same parameters used in the prototype statement are specified (without the &) in the macro instruction, where they are equated to the value desired for the specific job. The parameters of a proto-

type statement are called keywords when they appear without the & in a macro instruction followed by an equality sign.

In the following example, the first line shows a prototype statement, the second line the corresponding macro instruction.

Name	Operation	Operand
	CHECK	&SUM=,&DIFF=
	CHECK	DIFF=25,SUM=PAY

Since the association of parameters is performed through the use of keywords, the operands in the macro instruction may appear in any order, and any parameters that are not needed may be omitted. If an operand is omitted, the comma that would have separated it from the next operand must not be written.

The rules for writing names and operation codes in keyword macro instructions are the same as those for positional macro instructions.

The begin, end, and continue columns for keyword macro instructions are the same as those for positional macro instructions.

Each macro instruction operand must consist of a keyword immediately followed by an equal sign and a value. Anything that can be used as an operand in a positional macro instruction may be used as a value in keyword a macro instruction.

The keyword part of each macro-instruction operand must correspond to one of the symbolic parameters that appears in the operand field of the prototype statement. A keyword corresponds to a symbolic parameter if the characters of the keyword are identical to the characters of the symbolic parameter that follow the ampersand.

Operands of a keyword macro instruction may appear on separate cards. A comma must follow every operand except the last, and the continuation column must contain a non-blank character. Comments may be contained on the separate cards that contain individual operands.

A symbolic parameter becomes a null parameter if:

1. A symbolic parameter appears in the name field of a prototype statement and the name field of the corresponding macro instruction is blank.
2. A keyword is specified in the operand field of a macro instruction and no value is associated with the keyword.
3. No value is associated with a keyword in the operand field of a prototype statement, and the keyword and its associated value are omitted from the operand field of a macro instruction.

The following rules are used to replace the symbolic parameters in the model statements of a keyword macro definition:

1. If a symbolic parameter appears in the name field of a prototype statement and the macro instruction is named, the symbolic parameter in the name field is replaced by the name.
2. The value associated with each parameter in the operand field of the prototype statement becomes the value of the symbolic parameter.
3. The value associated with each keyword specified in an operand of the macro instruction replaces the value obtained from the prototype statement for the symbolic parameter.

The following keyword macro definition (first box), keyword macro instruction (second box), and generated statements (third box) illustrate these rules:

Name	Oper-ation	Operand
	MACRO	
&NAM	MOVE	&REG=12,&AREA=SAVE,&TO=,&FROM=
&NAM	STH	&REG,&AREA
	LH	&REG,&FROM
	STH	&REG,&TO
	LH	&REG,&AREA
	MEND	
HERE	MOVE	TO=FLDA,FROM=FLDB,AREA=THERE
HERE	STH	12,THERE
	LH	12,FLDB
	STH	12,FLDA
	LH	12,THERE

Note that the keyword REG was omitted in the macro instruction and the standard value 12 obtained from the prototype statement was used in the generation wherever &REG appeared in the model statements.

If the entry FROM=FLDB is omitted from the macro instruction, the second model statement is generated as LH 12, which is an invalid statement to the Assembler.

ASSEMBLY OF MACRO INSTRUCTIONS

At program assembly time, the macro instruction specifies which definition is to be called from the macro library. The definition is extracted, tailored by the operands in the macro instruction, and inserted in the program. The complete program now consists of both source program statements and tailored model statements from the macro library in Assembler language.

In subsequent phases of the assembly, the entire program is processed to produce the machine-language object program.

Figure 10 illustrates the processing of a macro definition.

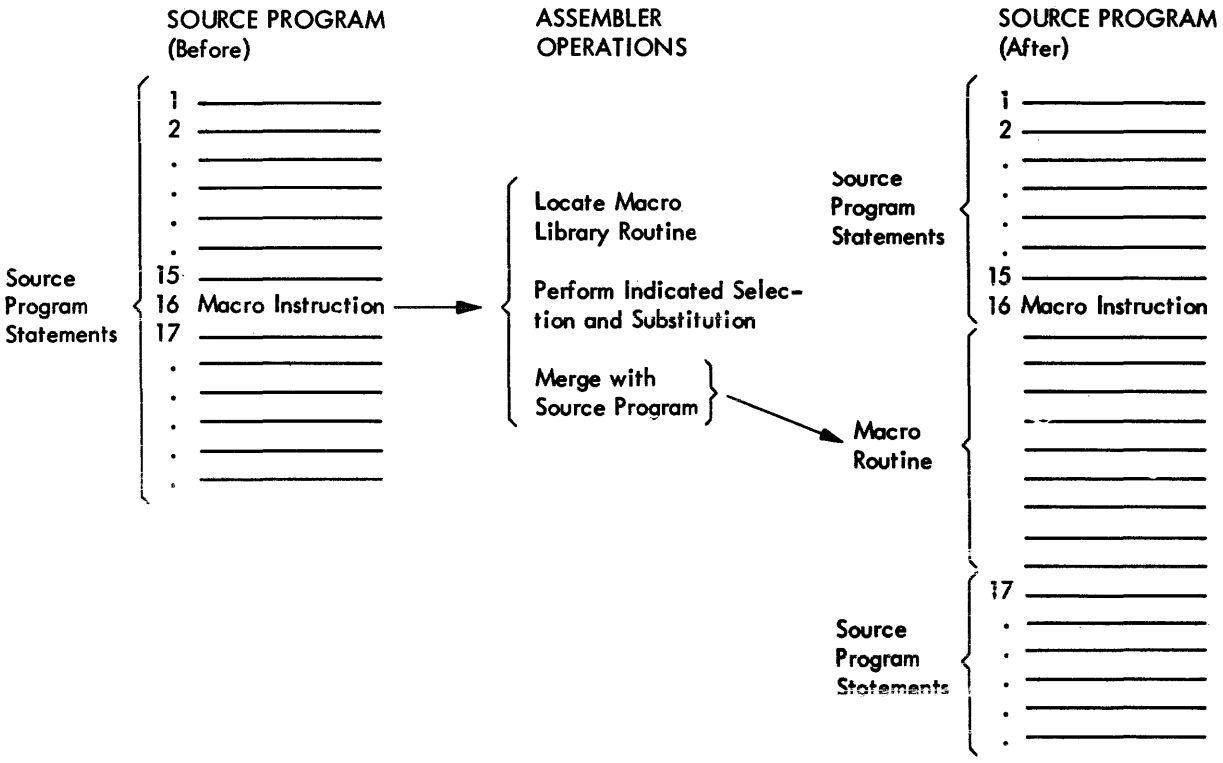


Figure 10. Schematic of Macro Processing

The macro language is an extension of the System/360 Model 20 Assembler language and is an aid in writing an Assembler-language program.

Before you can code a macro instruction, the series of statements that the macro instruction represents must be defined in a macro definition.

A macro definition is composed of a header statement, a prototype statement, one or more model statements, and a trailer statement, in this sequence. You may further include conditional-assembly instructions.

This section contains a description of the components of a positional macro definition and of the differences between it and a keyword macro definition. Furthermore, this section also contains an explanation of the model statements, the conditional assembly instructions, and the system variable symbols. Inner macro instructions as special model statements are also described. A sample macro definition and a step by step procedure for coding a macro definition is included.

Before you can use one of your own macro definitions you must include it in the macro library of your system. To this end, use the MMAINT (Macro Maintenance) program provided by IBM.

Positional Macro Definitions

To make a macro definition available to many programs place the macro definition in the macro library by means of a Macro Maintenance program (MMAINT). The MMAINT program enables you to delete or replace macro definitions in the macro library according to your needs.

When writing a macro definition, you cannot use the ICTL instruction to alter the normal format of the macro component statements. In a macro definition, the begin column is column 1, the end column is column 71, and the continue column for the prototype statement or an inner macro instruction is column 16.

Each macro definition includes (in the sequence indicated):

1. A header statement. This statement indicates the beginning of a macro definition.
2. A prototype statement. This statement indicates the various symbolic parameters of a macro definition and the format and the mnemonic operation code to be used in the macro instruction.
3. Model statements and conditional-assembly instructions and comments statements. Model statements define representations of the statements that will replace the macro instruction in the source program. Conditional-assembly instructions vary the sequence, number, and type of the statements generated, based on presence, absence or values of the operands given in a particular macro instruction (see Conditional-Assembly Instructions).
4. A trailer statement. It indicates the end of a macro definition.

MACRO -- HEADER STATEMENT

The header statement indicates to the MMAINT (Macro Maintenance program) that a macro definition follows. It must be the first statement in every macro definition. The format of this statement is:

Name	Operation	Operand
Blank	MACRO	Blank or vvmm*

*vvmm applies to DPS only. vv is the number of the program version; mm is the modification level. The operand, if present, is transferred into the macro directory (last two bytes of the corresponding entry).

PROTOTYPE STATEMENT

The prototype statement indicates the format and the mnemonic operation code of the positional macro instruction the Assembler is to interpret. It must be the second statement of every macro definition. The format of this statement is:

Name	Operation	Operand
A symbolic parameter or blank	A symbol	Up to 49 symbolic parameters, separated by commas

A symbolic parameter is an ampersand (&) followed by one to seven alphabetic and/or numeric characters, the first of which must be alphabetic.

You must not use any symbolic parameters that have &SYS as the first four characters.

Furthermore, symbolic parameters in the form &ALn, &AGn, &BLn, &BGn, &CLn, and &CGn, where n is one to five numeric characters, are not permitted. These symbols are reserved for internal use.

The following are valid symbolic parameters:

&READER	&LOOP2	&AGH
&A23456	&N	&BLC
&X4#F2	&S4	&CG6A

The following are invalid symbolic parameters for the reasons indicated:

IOAREA	First character is not an ampersand
&256B	First character after ampersand is not a letter
&AREA2456	More than seven characters after the ampersand
&BCD&34	Contains a special character other than initial &
&IN AREA	Contains an embedded blank
&SYSTEM	Contains &SYS as the first four characters
&AG15	Is in the form &AGn, where n is numeric
&BG28	Is in the form &BGn, where n is numeric
&CG215	Is in the form &CGn, where n is numeric

Name: The symbolic parameter in the name field is normally used to name the generated statements. It can also be used in model statements in the same way as symbolic parameters defined in the operand field.

Operation: The symbol in the operation field is the mnemonic operation code of the macro definition containing the prototype statement. The operation code consists of one to five alphabetic and/or numeric characters, the first of which must be alphabetic. The operation code in the operation field must be unique. It must differ from the operation code of any IBM-supplied macro definition, any machine and Assembler instruction, and the operation code of any other macro definition you defined yourself. A list of the IBM-supplied macro definitions is included in Appendix E.

Operand Field: The operand field may contain up to 49 symbolic parameters separated by commas. These symbolic parameters are used in model statements and replaced during generation by the corresponding operands of the macro instruction.

The following sample prototype statement contains three symbolic parameters: one in the name field and two in the operand field. The mnemonic operation code is MOVE.

Name	Operation	Operand
&NAME	MOVE	&TO,&FROM

Prototype Statement Format

To allow for the inclusion of up to 49 symbolic parameters in the prototype statement of a macro definition, use as many continuation cards as needed. The name field, if used, must begin in column 1. The operation field, preceded and followed by at least one blank, must appear on the first card of the statement. The other rules are:

- If the symbolic parameters in the operand field extend up to the end column, and if column 72 contains a non-blank character, the symbolic parameters may be continued in column 16 of the next card. A single symbolic parameter may be split between two cards.
- A blank following a symbolic parameter signifies the end of all symbolic parameters.
- Comments may appear after the blank that indicates the end of all symbolic parameters, up to and including column 71.

MODEL STATEMENTS

Model statements are representations of the statements that will replace the particular macro instruction in the source program.

A model statement that contains no symbolic parameters or variable symbols will appear in the source program in the same

format as it appears in the macro definition. If a model statement contains symbolic parameters or variable symbols, the Assembler replaces the symbolic parameters and variable symbols by the value specified in the macro instruction before the model statement is included in the source program.

A model statement consists of one to four fields (from left to right): name field, operation field, operand field, and comments field.

The operation field may contain the operation code of any machine or Assembler instruction except:

END, ICTL, ISEQ, LTORG, PRINT, and START.

It may also contain another inner macro instruction. The operation field must not contain a symbolic parameter. If REPRO is used as a model statement, the following card is not considered a model statement and therefore ignored by the Macro maintenance program.

The operand may consist of variable or non-variable symbols. For model statement fields, the rules for paired apostrophes, ampersands, or blanks in macro instruction operands must be followed.

Symbolic parameters used in a model statement must be defined in the prototype statement. Symbols used in a model statement must be defined within the macro definition or within the source program that calls the macro definition from the macro library.

In the following example, the symbol SAVEAREA is defined outside the macro definition.

The function of this macro definition is to move the contents of one storage area to another area in main storage.

	Name	Operation	Operand
Header		MACRO	
Prototype	&NAME	MOVE	&TO,&FROM
Model	&NAME	STH	12,SAVEAREA
Model		LH	12,&FROM
Model		STH	12,&TO
Model		LH	12,SAVEAREA
Trailer		MEND	

Note that each of the symbolic parameters used in the model statements of the preceding example appears in the prototype statement.

A model statement of a machine or Assembler instruction must not be continued on an additional card. If the model statement is a macro instruction (see Inner Macro Instructions), it may be continued on as many cards as needed.

During generation, each symbolic parameter in the name or operand field of a model statement is replaced by the characters of the macro instruction that correspond to the symbolic parameter in the prototype statement. The operand field of a generated model statement of a machine or Assembler instruction can contain 56 characters.

If a symbolic parameter or a system variable symbol appears in the comment field of a model statement, it is not replaced by the corresponding characters of the macro instruction.

In the following example, the characters HERE, FIELDA, and FIELDB of the macro instruction MOVE correspond to the symbolic parameters &NAME, &TO, and &FROM, respectively, of the prototype statement.

Name	Operation	Operand
HERE	MOVE	FIELDA,FIELDB

If the symbolic parameter &NAME appears in the name or operand field of a model statement, it will be replaced by the characters HERE. Similarly, the symbolic parameters &TO and &FROM will be replaced by the characters FIELDA and FIELDB, respectively. If the preceding macro instruction were used in a source program, the following Assembler-language statements would be generated.

Name	Operation	Operand
HERE	STH	12,SAVEAREA
	LH	12,FIELDB
	STH	12,FIELDA
	LH	12,SAVEAREA

You may use the same macro instruction more than once in the same program. The Assembler uses the same macro definition to interpret several occurrences of a macro instruction. The following example illustrates this.

```
             ┌──────┬──────┬──────────────┐
             │      │Oper- │              │
             │Name  │ation │Operand       │
             ├──────┼──────┼──────────────┤
Macro Instr. │HERE  │MOVE  │FIELDA,FIELDB │
             ├──────┼──────┼──────────────┤
Generated    │HERE  │STH   │12,SAVEAREA   │
Generated    │      │LH    │12,FIELDB     │
Generated    │      │STH   │12,FIELDA     │
Generated    │      │LH    │12,SAVEAREA   │
             ├──────┼──────┼──────────────┤
Macro Instr. │LABEL │MOVE  │INTO,OUTOF    │
             ├──────┼──────┼──────────────┤
Generated    │LABEL │STH   │12,SAVEAREA   │
Generated    │      │LH    │12,OUTOF      │
Generated    │      │STH   │12,INTO       │
Generated    │      │LH    │12,SAVEAREA   │
             └──────┴──────┴──────────────┘
```

In addition to denoting symbolic parameters, ampersands may appear in a character value or a self-defining value. Two ampersands must be used to represent a single ampersand in a character value or self-defining value. The first statement in the following example is a model statement; the second statement is the source statement generated from the model statement.

```
┌────┬─────┬──────────────────┬────────────┐
│Name│Oper-│Operand           │Comments    │
│    │ation│                  │            │
├────┼─────┼──────────────────┼────────────┤
│&SYM│DC   │C'&&SYM IS &SYM'  │&SYM IS NAME│
├────┼─────┼──────────────────┼────────────┤
│NAME│DC   │C'&SYM IS NAME'   │&SYM IS NAME│
└────┴─────┴──────────────────┴────────────┘
```

Combining Symbolic Parameters With Other Characters (Concatenation)

The characters represented by a symbolic parameter, SET symbols, system variable symbols, symbols, self-defining values, or character values may be concatenated as desired to produce symbols, self-defining values, and character values. (For a discussion of SET symbols and system variable symbols see the sections Conditional-Assembly Instructions and System Variable Symbols.) A symbolic parameter, a SET symbol, or a system variable symbol concatenated with a second symbolic parameter cannot produce a third symbolic parameter.

Concatenation can be performed in the name field and in the operand field, but is not permitted in the operation field. The following two points must be considered.

1. When a symbolic parameter, a SET symbol, or a system variable symbol is followed by an open parenthesis, a period, an alphabetic character, or a numeric character, a period must separate it from the character that follows.
2. When a symbolic parameter, a SET symbol, or a system variable symbol is followed by a single period, the period does not appear in the generated output.

The following examples illustrate these two points. In the examples, assume that &PARAM has the value A.

Macro Definition:	Generated Statement:
&PARAM.(BC)	A(BC)
&PARAM..BC	A.BC
&PARAM.BC	ABC
&PARAM.2BC	A2BC
&PARAM..2B	A.2B
BC&PARAM	BCA
BC.&PARAM	BC.A
B2&PARAM	B2A
&PARAM.&PARAM	AA
&PARAM&PARAM	AA
&PARAM..&PARAM	A.A

The following macro definition is a practical example of the preceding discussion. The function of the macro definition is to move the contents of one area in main storage to another area in main storage.

```
             ┌──────┬─────┬──────────────────┐
             │      │Oper-│                  │
             │Name  │ation│Operand           │
             ├──────┼─────┼──────────────────┤
Header       │      │MACRO│                  │
Prototype    │&NAME │MOVE │&PRE,&SAV,&REG,&NDX│
Model        │&NAME │STH  │&REG,&SAV.&NDX    │
Model        │      │LH   │&REG,&PRE+8       │
Model        │      │STH  │&REG,&PRE.A       │
Model        │      │LH   │&REG,&SAV&NDX     │
Trailer      │      │MEND │                  │
             ├──────┼─────┼──────────────────┤
Macro        │HERE  │MOVE │FIELD,AREA,12, 4  │
             ├──────┼─────┼──────────────────┤
Generated    │HERE  │STH  │12,AREA4          │
Generated    │      │LH   │12,FIELD+8        │
Generated    │      │STH  │12,FIELDA         │
Generated    │      │LH   │12,AREA4          │
             └──────┴─────┴──────────────────┘
```

Note that the first and fourth model statements have identical operands, except for the period between the two symbolic parameters in the operand field of the first model statement. The period is necessary in the third model statement to distinguish between the symbolic parameter &PRE and the symbol A. The period is unnecessary (but may be used) in the second model statement to distinguish between the symbolic parameter &PRE and +8.

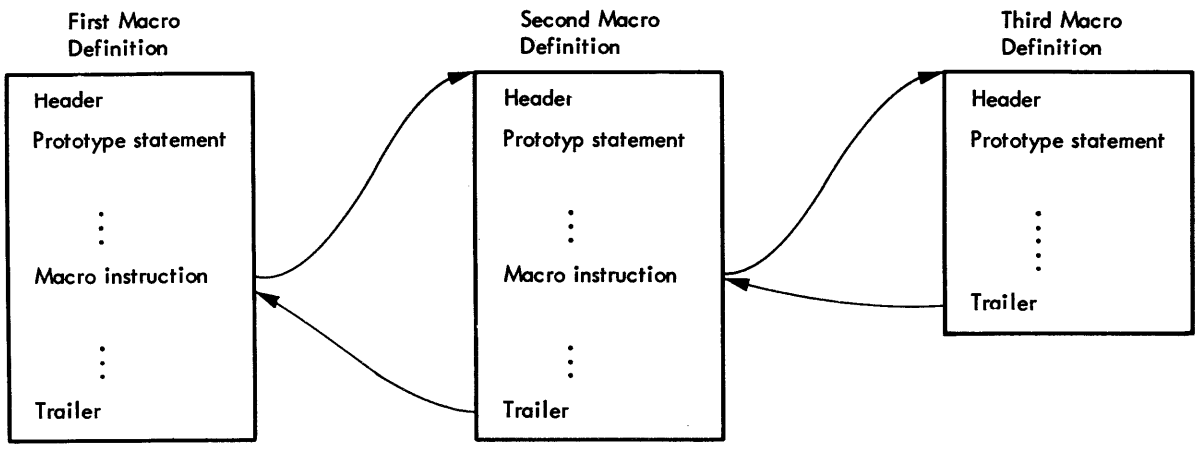


Figure 11. Schematic Representation of Nested Macro Instructions

Inner Macro Instructions

A macro definition may contain another macro instruction as a model statement. The containing macro instruction is called an outer macro instruction. The contained macro instruction is called an inner macro instruction.

The outer and inner macro instructions may be of the same or of different types. That is, both the inner and the outer macro instructions may be positional, they may both be keyword, or one may be positional and the other keyword.

When a macro definition contains a macro instruction, the macro instruction is said to be nested. The maximum depth of nesting is three. A macro definition (first level) may contain an inner macro instruction (second level). The definition of this inner macro instruction (second level) may again contain a macro instruction (third level).

Figure 11 shows a schematic representation of nested macro instructions.

The first-level macro definition can contain as many second-level macro instructions as are required. A second-level level macro definition can contain as many third-level macro instructions as are required. A third-level macro definition cannot contain a macro instruction.

Symbolic parameters that are part of the prototype statement, SET symbols, and system variable symbols may be used in an inner macro instruction. Each symbolic parameter, SET symbol, or system variable symbol is replaced by its value before the inner macro instruction is generated.

For example, the following macro definition is contained in the macro library.

```
|      | Oper-    |                          |
|Name  | ation    | Operand                  |
|------|----------|--------------------------|
|      | MACRO    |                          |
|      | ADD      | &N1,&N2,&N3,&REG,&AREA   |
|      | LH       | &REG,&N1                 |
|      | AH       | &REG,&N2                 |
|      | AH       | &REG,&N3                 |
|      | STH      | &REG,&AREA               |
|      | MEND     |                          |
```

The preceding ADD macro definition is used as an inner macro instruction in the COMP macro definition shown below. The macro instruction causing the generation of the model statements is given in the middle box. The generated statements are in the third box.

```
┌─────┬──────┬──────────────────────────────┐
│     │ Oper-│                              │
│Name │ ation│Operand                       │
├─────┼──────┼──────────────────────────────┤
│     │ MACRO│                              │
│     │ COMP │&AREA,&R1,&R2,&V1,&V2,&V3,&NA │
│     │ SR   │&R1,&R2                       │
│     │ CH   │&R1,&AREA                     │
│     │ BNE  │&NA                           │
│     │ ADD  │&V1,&V2,&V3,12,&AREA          │
│&NA  │ AH   │&R1,&AREA                     │
│     │ MEND │                              │
├─────┼──────┼──────────────────────────────┤
│     │ COMP │CHECK,10,11,X,Y,Z,CHNG        │
├─────┼──────┼──────────────────────────────┤
│     │ SR   │10,11                         │
│     │ CH   │10,CHECK                      │
│     │ BNE  │CHNG                          │
│     │ ADD  │X,Y,Z,12,CHECK                │
│     │ LH   │12,X                          │
│     │ AH   │12,Y                          │
│     │ AH   │12,Z                          │
│     │ STH  │12,CHECK                      │
│CHNG │ AH   │10,CHECK                      │
└─────┴──────┴──────────────────────────────┘
```

CONDITIONAL-ASSEMBLY INSTRUCTIONS

The information given in the preceding sections of this publication is sufficient to write a relatively simple macro definition. For each macro definition, the same sequence of statements is generated each time the macro definition is called by a macro instruction, except that the specific values and symbols in each statement may be different.

Frequently, it is desirable to vary the sequence, number, and type of instructions generated, based on the presence, absence, or values of the operands given in a particular macro instruction. Thus, the statements generated for two macro instructions calling the same macro definition might differ, while the functions performed by the statements are basically the same. To permit the writing of a more complex macro definition capable of producing a tailored set of generated statements based on the content of the macro instruction operands, two categories of special instructions are provided, the SET instructions and the conditional instructions.

The conditional-assembly instructions are: SETA, SETB, SETC, AGO, AGOB, AIF, AIFB, ANOP, MEXIT, and MNOTE.

The Set instructions SETA, SETC, and SETB perform arithmetic calculation, character manipulation, and set binary switches on the basis of logical and relational expressions.

The use of SET variable symbols in the operand field of model statements of macro definitions give you a high degree of flexibility in the application of macro definitions. For, by using the same symbol in the name field of a SET instruction, you may assign it a new value and thus alter the value of the operand in the model statement.

The results of the operations performed by the SET instructions are contained, during the generation of macro definitions, in a series of specially provided areas in main storage referred to by SET variable symbols. SET variable symbols can be used in model statements, SET instructions, and conditional instructions.

The AGO (Assembler GO) and AGOB (Assembler GO Back) instructions are similar to an unconditional branch instruction. They are used to indicate, by means of a sequence symbol, the next statement to be processed by the Assembler. (Sequence symbols are described in detail under Sequence Symbols).

The AIF (Assemble IF) and AIFB (Assemble IF Back) instructions are similar to a conditional branch instruction. They are used to indicate, by means of the logical value obtained from the operand and a sequence symbol, the next statement to be processed by the Assembler if the condition is TRUE.

The ANOP (Assemble NO Operation) instruction is used with the AGO, AGOB, AIF, and AIFB instructions if a sequence symbol cannot be used as the name of the next statement to be branched to.

The MEXIT (Macro EXIT) instruction is used to indicate to the macro generator that it is to terminate processing of a macro definition.

The MNOTE (Macro NOTE) instruction is used to generate messages in the output listing.

The functions of the SET instructions and the AGO, AGOB, AIF, and AIFB instructions are interrelated because the generated output is generally tailored by the use of AGO, AGOB, AIF, and AIFB instructions based on the results obtained from the values of SET instructions. While numerous examples of SET instructions are given in the section that explain the SETA, SETB, and SETC instructions, their use is shown in the sections describing the remaining conditional-assembly instructions.

SET VARIABLE SYMBOLS

The labels or symbols used in the name field of SET instructions are referred to as SET variable symbols or SET symbols.

The three types and formats of SET variable symbols are:

Symbol	Format
• SETA	&AGn or &ALn
• SETB	&BGn or &BLn
• SETC	&CGn

The n stands for an arithmetic value as described in the subsequent sections.

Three SET instructions are used to assign arithmetic, character, and logical values to SET symbols. The SETA instruction assigns an arithmetic value to a SETA symbol. The SETC instruction assigns a character value to a SETC symbol. A SETB instruction assigns a binary (or logical) value, TRUE (1) or FALSE (0), to a SETB symbol.

You should assign each SET symbol a specific value before the variable symbol is used in the operand field of a macro component. If you do not assign a value, the following assumed values are used.

- SETA symbols (arithmetic values) have an initial value of zero.
- SETC symbols (character values) have a null character value, zero bytes in length.
- SETB symbols (binary values) have an initial value of FALSE (0).

All SET symbols can be defined to be global. This means that once a value has been defined for a particular SET symbol, the value remains in effect for all references to it within the assembly. For example, if a source program contains three macro instructions, and a SETA symbol is given the value six in the macro definition called by the first macro instruction, the value six will be used when the particular SETA symbol appears within the macro definitions called by the other two macro instructions. You may, however, redefine the SETA symbol to a new value.

SETA and SETB symbols can be defined to be local, i.e., once a value has been defined for a particular SETA or SETB symbol, the value remains in effect for all references to it only within its macro definition. Once the macro instruction is assembled, the value of the SETA or SETB symbol is reset to zero. For example, if a source program contains two macro instructions, and a SETB symbol is assigned a value of one in the macro definition called by the first macro instruction, the SETB symbol is reset to zero after the macro instruction has been assembled.

When macro instructions are nested, local SETA and SETB symbols defined in the outer macro instruction are reset to zero immediately before the inner macro instruction is processed. After the inner macro instruction has been processed, the local variable symbols are reset to the values that were defined in the outer macro instruction.

SET symbols may be used with the following restrictions:

- They can only be used in the name or operand field of model statements or conditional-assembly instructions.
- They must not be used to generate a new sequence symbol, a SET symbol, a symbolic parameter, or a system variable symbol.
- The SETC symbol may be used in the operand field of a SETA statement only if the character string is composed of from one to five decimal digits.

For restrictions on SETB symbols, refer to Appendix E: Summary of Macro Language.

SETA -- SET ARITHMETIC

The SETA instruction may be used to assign an arithmetic value to a SETA symbol. Each arithmetic value is 5 digits in size, and each value is initially zero. You may change the value assigned to a SETA symbol by using another SETA instruction with the same variable symbol in the name field. The format of this instruction is:

Name	Operation	Operand
A SETA symbol	SETA	An arithmetic expression

You may use SETA symbols in the operand field or name field of model statements.

The SETA symbol in the name field may be either local or global. There are 16 different global and 16 different local SETA symbols.

A global SETA symbol has the form &AGn, where n = 0-15.

A local SETA symbol has the form &ALn, where n = 0-15.

The expression in the operand field may consist of one term, or as many as three terms connected by arithmetic operators.

The terms may be positive decimal self-defining values, symbolic parameters, SET symbols, or system variable symbols that represent positive decimal self-defining values. The arithmetic operators that can be used to combine terms are + (addition), - (subtraction), *(multiplication), and / (division).

The range of values that can be assigned to a SETA variable symbol is from 0 to 99999. Expressions are evaluated in storage using decimal arithmetic, which means that interim values can be in the range from 99999 to -99999.

For example, the expression 16215 + 16215 - 16215 is valid because neither the interim nor the final value exceeds 99999. The expression 65536*65536/65536 is invalid because even though the final value is equal to 65536, the interim value (4294967296) exceeds 99999.

The expression 65536*16384 is invalid, because the final value exceeds 99999.

The expression 3 + 4 - 9 is invalid because the final value is negative. The expression 3 - 5 + 6 is valid because, even if the interim value is negative, the final value is positive.

Division by zero results in a value of zero. In division, only the integer portion of the quotient is retained. For example, 97 divided by 25 gives the result of 3. The fractional portion of the quotient is dropped.

An expression must not contain two successive terms or two operators. An expression must not begin with an operator.

The following are examples of expressions that may be used in the operand field of a SETA instruction:

27	5*&AL12
&AG3+4	&AG6-&AL10+5

The following is not permitted in the operand field of a SETA instruction for the reasons stated:

&AG11+-12	Two successive operators
+14	Begins with an operator
&AG5-&AL8+8+&AG6	Expression consists of more than three terms
&AG18	&AG18 is not a valid SETA symbol since 18 > 15
&AG3*(&AL2+&AL1)	Grouping by use of parentheses is not permitted

The following procedure is used to evaluate the arithmetic expression in the operand field of a SETA instruction.

- Each term is given its decimal numerical value.
- The arithmetic operations are performed from left to right. However, multiplication and division are performed before addition and subtraction.
- The computed result is the value assigned to the SET symbol in the name field.

If the operand of a SETA instruction is found to be invalid, a value of zero is assigned to the SETA symbol in the name field.

The arithmetic value defined by a SETA instruction is represented in a model statement by the SETA symbol assigned. When a SETA symbol is detected during macro generation, it is replaced by the value of the symbol converted to a decimal self-defining value with any leading zeros dropped.

The example below illustrates this rule. The function of this macro definition is to move the contents of one storage area to another area in main storage. The symbol SAVEAREA is defined outside the macro definition.

Name	Operation	Operand
	MACRO	
&NAME	MOVE	&TO,&FROM
&AL1	SETA	10
&AL2	SETA	8
&AL3	SETA	&AL1-&AL2
&AL4	SETA	&AL1+&AL3
&NAME	STH	12,SAVEAREA
	LH	12,&FROM&AL3
	STH	12,&TO&AL4
	LH	12,SAVEAREA
	MEND	
HERE	MOVE	FIELDA,FIELDB
HERE	STH	12,SAVEAREA
	LH	12,FIELDB2
	STH	12,FIELDA12
	LH	12,SAVEAREA

If you have assigned an arithmetic value to a SETA symbol, you may change the assigned value by using the SETA symbol in the name field of another SETA instruction. If a SETA symbol has been used in the name field of more than one SETA statement, and the SETA symbol is used in the name or operand field of another model statement, the value substituted for the SETA symbol is the last value assigned to it.

The example below illustrates this rule. The function of this macro definition is

the same as that of the above example. The boxes contain again, respectively, the macro definition, macro instruction, and generated statements.

```
┌────────┬─────────┬──────────────────────┐
│        │ Oper-   │                      │
│Name    │ ation   │    Operand           │
├────────┼─────────┼──────────────────────┤
│        │ MACRO   │                      │
│&NAME   │ MOVE    │    &TO,&FROM         │
│&AL8    │ SETA    │    5                 │
│&NAME   │ STH     │    12,SAVEAREA       │
│        │ LH      │    12,&FROM&AL8      │
│&AL8    │ SETA    │    &AL8+3            │
│        │ STH     │    12,&TO&AL8        │
│        │ LH      │    12,SAVEAREA       │
│        │ MEND    │                      │
├────────┼─────────┼──────────────────────┤
│HERE    │ MOVE    │    FIELDA,FIELDB     │
├────────┼─────────┼──────────────────────┤
│HERE    │ STH     │    12,SAVEAREA       │
│        │ LH      │    12,FIELDB5        │
│        │ STH     │    12,FIELDA8        │
│        │ LH      │    12,SAVEAREA       │
└────────┴─────────┴──────────────────────┘
```

SETC -- SET CHARACTER

The SETC instruction may be used to assign a character value to a SETC symbol. Each global character value can vary from 0 to 8 bytes in size. Each character value is initially a null character value of zero bytes in length. You may change the character value assigned to a SETC symbol by using another SETC instruction with the same variable symbol in the name field. The format of this instruction is:

```
┌────────┬─────────┬──────────────────────┐
│        │ Oper-   │                      │
│Name    │ ation   │    Operand           │
├────────┼─────────┼──────────────────────┤
│A SETC  │ SETC    │    Up to 8 characters│
│symbol  │         │    enclosed by a pair│
│        │         │    of apostrophes    │
└────────┴─────────┴──────────────────────┘
```

SETC symbols in the name field are always global. They have the form &CG*n*, where *n* = 0-15.

You may use SETC symbols in the operand field or name field of model statements.

The value of the characters in the operand field is assigned to the SETC symbol in the name field. The characters in the operand field may consist of a string of characters, a SET symbol, symbolic parameters, system variable symbols, or any concatenation of the preceding values, enclosed within a pair of apostrophes. Length attributes cannot be substituted by the implicit length of the symbol.

The following statement assigns the character value AB%4 to the SET symbol &CG5:

```
┌────────┬─────────┬──────────────────────┐
│        │ Oper-   │                      │
│Name    │ ation   │    Operand           │
├────────┼─────────┼──────────────────────┤
│&CG5    │ SETC    │    'AB%4'            │
└────────┴─────────┴──────────────────────┘
```

More than one character value may be concatenated into a single character value by placing a period between the terminating apostrophe of one character value and the opening apostrophe of the next character value.

Either of the following statements may be used to assign the character value 2 AND 3 to the SETC symbol &CG14:

```
┌────────┬─────────┬──────────────────────┐
│        │ Oper-   │                      │
│Name    │ ation   │    Operand           │
├────────┼─────────┼──────────────────────┤
│&CG14   │ SETC    │    '2 AND 3'         │
│&CG14   │ SETC    │    '2'.' AND 3'      │
└────────┴─────────┴──────────────────────┘
```

Two apostrophes must be used to represent one apostrophe that is part of a character value enclosed in apostrophes.

The following statement assigns the character value L'SYMBOL to the SETC symbol &CG11 if &PARAM is substituted by SYMBOL:

```
┌──────┬──────┬──────────────────────────┐
│      │ Oper-│                          │
│Name  │ ation│ Operand                  │
├──────┼──────┼──────────────────────────┤
│&CG11 │ SETC │ 'L''&PARAM'              │
└──────┴──────┴──────────────────────────┘
```

Two ampersands must be used to represent one ampersand that is not part of a variable symbol. Both ampersands become part of the character value assigned to the SETC symbol. They are not replaced by a single ampersand.

The following statement assigns the character value HALF&& to the SETC symbol &CG4:

```
┌──────┬──────┬──────────────────────────┐
│      │ Oper-│                          │
│Name  │ ation│ Operand                  │
├──────┼──────┼──────────────────────────┤
│&CG4  │ SETC │ 'HALF&&'                 │
└──────┴──────┴──────────────────────────┘
```

SET symbols, symbolic parameters, and system variable symbols may be concatenated with other characters in the operand field of a SETC instruction according to the general rules for concatenation.

If &CG12 is assigned the character value AB%4, the following statement may be used to assign the character value AB%4RST to the SETC symbol &CG13:

```
|Name   |Oper-  |                      |
|       |ation  |   Operand            |
|&CG13  |SETC   |   '&CG12.RST'        |
```

If &CG12 has been assigned the character value AB%4, the following statement may be used to assign the character value RSTAB%4 to the SETC symbol &CG10:

```
|Name   |Oper-  |                      |
|       |ation  |   Operand            |
|&CG10  |SETC   |   'RST&CG12'         |
```

The character value that has been assigned to a SETC symbol is substituted for the SETC symbol when it is used in the name field or operand field of model statements. For example, consider the macro definition, macro instruction, and generated statements (shown in the boxes in this order) below.

The function of this macro definition is to move the contents of one storage area to another area in main storage. The symbol SAVEAREA is defined outside the macro definition.

```
|Name   |Oper-  |                      |
|       |ation  |   Operand            |
|       |MACRO  |                      |
|&NAME  |MOVE   |   &TO,&FROM          |
|&CG4   |SETC   |   'FIELD'            |
|&NAME  |STH    |   12,SAVEAREA        |
|       |LH     |   12,&CG4&FROM       |
|       |STH    |   12,&CG4&TO         |
|       |LH     |   12,SAVEAREA        |
|       |MEND   |                      |
|HERE   |MOVE   |   A,B                |
|HERE   |STH    |   12,SAVEAREA        |
|       |LH     |   12,FIELDB          |
|       |STH    |   12,FIELDA          |
|       |LH     |   12,SAVEAREA        |
```

If you have assigned a character value to a SETC symbol, you may change the value assigned by using the SETC symbol in the name field of another SETC instruction. If a SETC symbol has been used in the name field of more than one SETC instruction and the SETC symbol is used in the name or operand field of another model statement, the value substituted for the SETC symbol is the last value assigned to it.

The following example illustrates this rule:

```
|Name   |Oper-  |                      |
|       |ation  |   Operand            |
|       |MACRO  |                      |
|&NAME  |MOVE   |   &TO,&FROM          |
|&CG8   |SETC   |   'FIELD'            |
|&NAME  |STH    |   12,SAVEAREA        |
|       |LH     |   12,&CG8&FROM       |
|&CG8   |SETC   |   'AREA'             |
|       |STH    |   12,&CG8&TO         |
|       |LH     |   12,SAVEAREA        |
|       |MEND   |                      |
|HERE   |MOVE   |   A,B                |
|HERE   |STH    |   12,SAVEAREA        |
|       |LH     |   12,FIELDB          |
|       |STH    |   12,AREAA           |
|       |LH     |   12,SAVEAREA        |
```

If a SETA symbol is used in the operand field of a SETC instruction, it is replaced by the value of the SETA symbol converted to a decimal self-defining value with any leading zeros dropped.

A SETC symbol may be used in the operand field of SETA, SETB, SETC, AIF, and AIFB instructions.

Defining Substrings with SETC Instructions. A substring consists of a character value enclosed in apostrophes, immediately followed by two arithmetic terms separated by a comma and enclosed in parentheses.

The character value assigned to a SET symbol in a SETC instruction can be a substring. Substrings permit you to assign, to a SETC symbol, part of the value assigned to another SET symbol, a symbolic parameter, a self-defining character string, or any valid combination of the preceding values.

The arithmetic terms may consist of SETA symbols and self-defining decimal values with any leading zeros dropped. The first term indicates the first character in the substring, the second term the number of characters in the substring.

A character string from which a substring is extracted may contain up to 16 characters. The resulting substring that can be assigned to a SETC symbol may contain up to eight characters.

The following are examples of valid substring definitions in operand fields of SETC instructions:

```
'&CG6'(2,3)
'&CG10.XYZ'(4,&AG8)
'XYZ&CG10'(&AL4,6)
'&CG1.XYZ&AG2'(4,7)
'&PARAM'(3,2)
```

The following is not permitted in the operand field of a SETC instruction:

'&CG2' (4,6)	Blank between character value and arithmetic terms.
'&CG15'(8)	Only one arithmetic term.
'&CG4'(5 6)	Arithmetic terms not separated by a comma.
'CG5'3,4	Arithmetic terms not enclosed in parentheses.
'&CG5'(&CG4,2)	First term not arithmetic.

The following example illustrates the use of substrings. The macro instruction (HERE MOVE FIELDA,B) assigns the character value FIELDA to the symbol &TO. The SETC instruction assigns the value FIELD to the symbol &CG6. The &CG6 symbol is used in the LH model statement and is replaced in the generated statement by the value assigned to it.

```
|Name   |Operation |  Operand              |
|-------|----------|-----------------------|
|       |MACRO     |                       |
|&NAME  |MOVE      |  &TO,&FROM            |
|&CG6   |SETC      |  '&TO'(1,5)           |
|&NAME  |STH       |  12,SAVEAREA          |
|       |LH        |  12,&CG6&FROM         |
|       |STH       |  12,&TO               |
|       |LH        |  12,SAVEAREA          |
|       |MEND      |                       |
|-------|----------|-----------------------|
|HERE   |MOVE      |  FIELDA,B             |
|-------|----------|-----------------------|
|HERE   |STH       |  12,SAVEAREA          |
|       |LH        |  12,FIELDB            |
|       |STH       |  12,FIELDA            |
|       |LH        |  12,SAVEAREA          |
```

Substrings may be concatenated with other character values in the operand field of a SETC statement. If a substring follows a character value that is not a substring, the two may be concatenated by placing a period between the first character value and the substring.

For example, if &CG6 is assigned the character value AB%4, and &CG8 is assigned the character value ABCDEF, the following statement assigns &CG0 the character value AB%4BCD.

```
|Name   |Operation |  Operand              |
|-------|----------|-----------------------|
|&CG0   |SETC      |  '&CG6'.'&CG8'(2,3)   |
```

If a substring precedes another character value, the two may be concatenated by placing the terminating parentheses of the substring and the opening apostrophe of the next character value adjacent to one another.

If &CG2 is assigned the character value AB%4, and &CG3 is assigned the character value 5RS, any one of the following statements may be used to assign &CG4 the character value AB%45RS.

```
|       | Oper- |                         |
|Name   | ation | Operand                 |
|-------|-------|-------------------------|
|&CG4   | SETC  | '&CG2&CG3'              |
|&CG4   | SETC  | '&CG2'.'&CG3'           |
|&CG4   | SETC  | '&CG2.&CG3'             |
|&CG4   | SETC  | '&CG2'(1,4)'&CG3'       |
|&CG4   | SETC  | '&CG2'(1,4)'&CG3'(1,3)  |
```

If &CG2 contained AB%4XY and &CG3 contained 5RSTU, only the last instruction of the preceding example would produce the desired result of AB%45RS. The first four instructions would be in error because the result exceeds eight characters.

Assume &CG1 is assigned the character value ABCDE, &CG2 has been assigned the character value FGHIJKPQ, and &CG3 is assigned the character value LMNO. The following SETC instruction can be used to assign &CG4 the character value DEXYZFGM.

```
|    |Oper-|                                |
|Name|ation|Operand                         |
|----|-----|--------------------------------|
|&CG4|SETC |'&CG1.XYZ&CG2'(4,7)'&CG3'(2,1)|
```

The preceding example also illustrates how a character string from which a substring is extracted can contain up to 16 characters: '&CG1.XYZ&CG2' becomes 'ABCDEXYZFGHIJKPQ' before the substring DEXYZFG is extracted.

SETB -- SET BINARY

The SETB instruction assigns the value one (TRUE) or zero (FALSE) to a SETB symbol. The initial value is zero. You may change the value assigned to a SETB symbol by using another SETB instruction. The format of this instruction is:

Name	Oper-ation	Operand
A SETB symbol	SETB	A logical expression or a relational expression enclosed in parentheses

You may use SETB symbols in the operand field or name field of model statements.

The SETB symbol can be either local or global.

For DPS there are 256 different global and 256 different local SETB symbols.

For TPS there are 256 different global and 128 different local SETB symbols.

A global SETB symbol has the form &BG*n*, where *n* = 0-255.

A local SETB symbol has the form &BL*n*, where *n* = 0-255 for DPS and *n* = 0-127 for TPS.

The logical or relational expression in the operand field is evaluated to determine whether it is true or false, and the value one or zero, respectively, is assigned to the SETB symbol in the name field.

A logical expression may consist of a single term, or of two terms separated by a logical operator. If a logical expression consists of a single term, the term may be zero, one, or a SETB symbol. If a logical expression consists of two terms, each term must be a SETB symbol.

The logical operators are AND, OR, and NOT. The logical operator NOT may only be used to negate a SETB symbol.

A two-term logical expression is evaluated according to the following rules of Boolean logic:

- X AND Y is equivalent to X * Y, i.e., 0*0 = 0, 0*1 = 0, 1*0 = 0, and 1*1 = 1.
- X OR Y is equivalent to X + Y, i.e., 0+0 = 0, 0+1 = 1, 1+0 = 1, and 1+1 = 1.
- NOT X is equivalent to 1 - X, i.e., 1-0 = 1 and 1-1 = 0.

The following rules must be observed:

- A logical expression must not contain two terms in succession.
- A logical expression may contain two operators in succession but only in the combination AND NOT and OR NOT.
- A logical expression may begin with the operator NOT. It must not begin with the operators AND or OR.
- The logical operators must be separated by one blank from the terms they relate.
- The entire logical expression must be enclosed within parentheses.

The following are examples of logical expressions that may be used as the operand of a SETB instruction:

```
(NOT &BG9)
(&BG8)
(1)
(&BG13 AND &BL4)
(&BG8 AND NOT &BL6)
(NOT &BL22 AND &BG22)
(NOT &BL24 AND NOT &BL25)

(&BG12 OR &BL10)
(&BG25 OR NOT &BL25)
(NOT &BG10 OR &BG16)
(NOT &BG0 OR NOT &BG1)
```

The following is not permitted as the operand field of a SETB instruction, for the reasons stated:

&BG8	Not enclosed in parentheses.
(&BG6 &BL8)	Two terms in succession.
(&BG10 AND OR &BG12)	Two operators in succession; second one is OR.
(&BL10 NOT NOT &BL18)	Two operators in succession; first one is NOT.
(NOT 1)	Negated term is not a SETB symbol.
(AND &BG2 OR &BG3)	Expression begins with an operator other than NOT.
(&AG1 AND &AG3)	Not SETB variable symbols.

A relational expression is either an arithmetic relation or a character relation.

An arithmetic relation consists of two arithmetic expressions connected by a relational operator. An arithmetic expression can be a SETA symbol, a SETC symbol, or any valid operand of a SETA instruction. If a SETC symbol is used in an arithmetic relation, the SETC symbol must represent an arithmetic value. The arithmetic relation is enclosed within parentheses.

A character relation consists of two character values connected by a relational operator. In a character relation, each character value must be enclosed by apostrophes. A character value can be a SETA symbol, a SETC symbol, or any valid operand of a SETC instruction, except substrings. If a SETA symbol is used in a character relation, the SETA symbol is treated as a character value. The maximum length of any character value used in a character relation is eight. If two character values in a character relation are of unequal length, the longer value is always considered greater, regardless of the content of the two values. The character relation is enclosed within parentheses.

The relational operators are:

EQ (equal),
NE (not equal),
LT (less than),
GT (greater than),
LE (less than or equal to),
GE (greater than or equal to).

A relational expression must not contain two values in succession. A relational expression must not contain two operators in succession. The relational operators must be separated from the values they relate by one blank.

Relational operators and logical operators must not appear in the same SETB instruction.

The following are examples of valid operand fields of SETB instructions with a relational operator:

```
('FIELD' NE '&CG4')
(12 EQ &AL4)
(&AL10 GT &AG6)
('&CG8' LT '&CG4')
('&CG5.X9' EQ '&CG2')
(&AL9+&AL4*7 LT 16*&AG1+4)
(&BG4 EQ 1)
```

The following is not permitted in the operand field of a SETB instruction, for the reasons stated:

&BG8	Not enclosed in parentheses.
(&BG6 &BL8)	Two terms in succession.
(&BG10 GT EQ &BG16)	Two operators in succession.
(LE &BL20 EQ &BL21)	Expression begins with an operator.
(&AG3 EQ '&AG4')	Arithmetic value equated to character value.

The logical value that has been assigned to a SETB symbol is substituted for the SETB symbol when it is used in the operand field of a SETB, AIF, or AIFB instruction. (A detailed description of the AIF and AIFB instructions is given in the sections AIF -- Conditional Branch and AIFB -- Conditional Branch Backward.) If the SETB symbol is used in any other Assembler-language statement, the logical value is converted to an integer. The logical value TRUE is converted to the integer one, and the logical value FALSE is converted to the integer zero.

If you have assigned a logical value to a SETB symbol, you may change the value assigned by using the SETB symbol in the name field of another SETB statement. If a SETB symbol has been used in the name field of more than one SETB statement, and the SETB symbol is used in the name or operand field of another model statement, the value substituted for the SETB symbol is the last value assigned to it.

The following example illustrates this rule. '&TO' GT 'AAAAAA' has the logical value TRUE because FIELDA has a greater binary value than AAAAAA.

The function of this macro definition is to move the contents of one storage area to another area in main storage. The boxes contain respectively, the macro definition, macro instruction, and generated statements.

Name	Operation	Operand
	MACRO	
&NAME	MOVE	&TO,&FROM
&BG8	SETB	('&TO' GT 'AAAAAA')
&NAME	STH	12,SAVEAREA
	LH	12,&FROM&BG8
&BG8	SETB	(NOT &BG8)
	STH	12,&TO&BG8
	LH	12,SAVEAREA
	MEND	
HERE	MOVE	FIELDA,FIELDB
HERE	STH	12,SAVEAREA
	LH	12,FIELDB1
	STH	12,FIELDA0
	LH	12,SAVEAREA

Testing for Null Parameters. A null parameter is a symbolic parameter defined in a positional prototype statement, but undefined in the macro instruction calling the macro definition.

The SETB instruction can be used to test for the presence of a null parameter. This is accomplished by placing the symbolic parameter to be tested in the operand field of a SETB instruction and equating it to a null character string. A null character string is represented by two apostrophes. If the parameter is present in the calling macro instruction, the result is FALSE or zero. If the parameter is not present in the calling macro instruction, the result is TRUE or one.

For example, if the prototype statement is:

```
    &NAME ADD &FROM1,&FROM2,&SUM
```

and the macro instruction is:

```
    FIRST ADD FIELD1,,FIELD3
```

the result of the SETB instruction

```
    &BG10 SETB ('&FROM1' EQ '')
```

is FALSE (0), while the result of the SETB instruction

```
    &BG8 SETB ('&FROM2' EQ '')
```

is TRUE (1).

When the same prototype statement and the same macro instruction are used, the result of the SETB instruction

```
    &BG10 SETB ('&FROM1' NE '')
```

is TRUE (1), while the result of the SETB instruction

```
    &BG8 SETB ('&FROM2' NE '')
```

is FALSE (0).

SEQUENCE SYMBOLS

Sequence symbols are used in the operand fields of AGO, AGOB, AIF, AIFB instructions and in the name field of model statements and conditional assembly instructions. They indicate to the Assembler the sequence of source statements to be generated.

A sequence symbol consists of a period (.) followed by one to seven alphabetic and/or numeric characters. The first character must always be alphabetic.

The following example illustrates the use of sequence symbols as a "branching address".

```
Name    Operation  Operand
        MACRO
        .
        .
        AGO        .DOWN
        .
        .
.UP     ANOP
LOOP    AH         7,FOUR
        .
        .
        AGO        .OUT
        .
        .
.DOWN   .
        AGO        .AGAIN
        .
        .
.AGAIN  LH         9,DATA
        .
        .
        AGOB       .UP
        .
        .
.OUT    MEND
```

To ensure proper generation, all sequence symbols used in a macro definition must be unique.

The following are valid sequence symbols:

```
.READER     .A23456     .AG4
.LOOP2      .X4F2       .SYSTEM
.N          .S4         .BL16
```

The following are invalid sequence symbols, for the reasons stated:

IOAREA	First character is not a period.
.246B	First character after period is not a alphabetic.
.AREA2456	More than seven characters after period.
.IN AREA	Contains an embedded blank.
.TWO.A5	Contains a special character other than initial period.

A sequence symbol may be used in the name field of any statement within a macro definition that does not require a symbol or SET symbol, except a header or a prototype statement.

If a sequence symbol appears in the name field of an inner macro instruction in a macro definition and the corresponding prototype statement contains a symbolic parameter in the name field, the sequence symbol does not replace the symbolic parameter in the model statement.

A sequence symbol appearing in the name field of a model statement does not appear in the generated statement.

AIF -- CONDITIONAL BRANCH

The AIF instruction may be used to skip one or more statements in your macro definition. The format of this instruction is:

Name	Oper-ation	Operand
A sequence symbol or blank	AIF	A logical or relational expression enclosed in parentheses, followed by a sequence symbol defined in a following statement

Any logical or relational expression that may be used in the operand field of a SETB instruction may also be used in the operand field of an AIF instruction. As in the SETB instruction, the logical or relational expression must be enclosed in parentheses. The sequence symbol in the operand field must immediately follow the closing parenthesis of the logical or relational expression. It must also appear in the name field of a statement <u>following</u> the AIF instruction.

The following are examples of valid contents of the operand fields of AIF instructions:

```
(&BG12 AND &BL10).LOOP
(&AL10 EQ &AG6).LAST
```

The following examples are invalid as the operand field of an AIF instruction, for the reasons stated:

(&BG8 AND NOT &BG9)	No sequence symbol.
.X4F2	No logical or relational expression.
(&BG8 AND NOT &BG9) <u>.XF2</u>	Blank between logical expression and sequence symbol.

The logical or relational expression in the operand field is evaluated to determine whether it is TRUE or FALSE. If the expression is TRUE, the statement named by the sequence symbol in the operand field is the next statement processed by the Assembler. If the expression is FALSE, the next sequential statement is processed by the Assembler.

The following example illustrates the use of the AIF conditional-assembly instruction. It also illustrates the use of global SET symbols to carry values between macro instructions in the same assembly.

The function of this macro definition is to move the contents of one storage area to another area in main storage.

The first time the macro instruction appears in an assembly, a save area is defined. The generated instructions of all additional appearances of this macro instruction in an assembly use the save area and the register specified in the first appearance of the macro instruction.

The boxes in the example below contain respectively: the macro definition, the first macro instruction, the statements generated as a result of the first macro instruction, the second macro instruction, and the statements generated because of it.

Name	Oper-ation	Operand
	MACRO	
	MOVE	&TO,&FROM,&REG,&SAVE
	AIF	(&BG1).A
&BG1	SETB	(1)
&CG1	SETC	'&SAVE'
&CG2	SETC	'&REG'
	B	&CG1+2
&CG1	DC	H'0'
.A	STH	&CG2,&CG1
	LH	&CG2,&FROM
	STH	&CG2,&TO
	LH	&CG2,&CG1
	MEND	
	MOVE	TAX,DEDUCT,9,WORK1
	B	WORK1+2
WORK1	DC	H'0'
	STH	9,WORK1
	LH	9,DEDUCT
	STH	9,TAX
	LH	9,WORK1
	MOVE	FICA,DEDUCT,7,WORK6
	STH	9,WORK1
	LH	9,DEDUCT
	STH	9,FICA
	LH	9,WORK1

The B and DC statements are not generated for the second macro instruction, for when the first macro instruction was assembled, &BG1 was set to one. The third and fourth parameters in the second MOVE macro instruction are ignored. &CG1 is used to assign a name to the DC model statement.

```
+--------+------+--------------------------+
|        |Oper- |                          |
|Name    |ation |Operand                   |
+--------+------+--------------------------+
|        |MACRO |                          |
|        |MOVE  |&NOCHAR,&TO,&FROM         |
|&AL2    |SETA  |&NOCHAR                   |
|        |AIF   |(&AL2 LE 256).LSTMOV      |
|.LOOP   |MVC   |&TO+&AL1.(256),&FROM+&AL1 |
|&AL1    |SETA  |&AL1+256                  |
|&AL2    |SETA  |&NOCHAR-&AL1              |
|        |AIFB  |(&AL2 GT 256).LOOP        |
|.LSTMOV |MVC   |&TO+&AL1.(&AL2),&FROM+&AL1|
|        |MEND  |                          |
+--------+------+--------------------------+
|        |MOVE  |540,OUT,INPUT             |
+--------+------+--------------------------+
|        |MVC   |OUT+0(256),INPUT+0        |
|        |MVC   |OUT+256(256),INPUT+256    |
|        |MVC   |OUT+512(28),INPUT+512     |
+--------+------+--------------------------+
|        |MOVE  |97,OUT+540,RESULT         |
+--------+------+--------------------------+
|        |MVC   |OUT+540+0(97),RESULT+0    |
+--------+------+--------------------------+
```

The value of the local variable symbol &AL1 is initially zero.

AIFB -- CONDITIONAL BRANCH BACKWARD

The AIFB instruction may be used to conditionally alter the sequence in which source statements are processed by the macro generator. The format of this instruction is:

```
+--------+-----+--------------------------+
|        |Oper-|                          |
|Name    |ation|Operand                   |
+--------+-----+--------------------------+
|A se-   |AIFB |A logical or relational ex-|
|quence  |     |pression enclosed in paren-|
|symbol  |     |theses, followed by a se-  |
|or      |     |quence symbol defined in a |
|blank   |     |preceding statement        |
+--------+-----+--------------------------+
```

The AIFB statement is identical to the AIF statement, except that the sequence symbol in the operand field must be in the name field of a statement preceding the AIFB statement.

The following example illustrates the use of the AIFB instruction. The function of the macro definition is to move a specified number of bytes of information from one location in main storage to another. The first operand represents the number of bytes to be moved. The second operand specifies the first position of the field to be filled. The third operand specifies the location of the first byte to be moved.

The boxes in the example below contain respectively: the macro definition, the first macro instruction, the statements generated as a result of the first macro instruction, the second macro instruction, and the statements generated because of it.

AGO -- UNCONDITIONAL BRANCH

The AGO instruction may be used to alter the sequence in which source statements are processed by the Assembler. The format of this instruction is:

```
+--------+---------+----------------------+
|        | Oper-   |                      |
|Name    | ation   | Operand              |
+--------+---------+----------------------+
|A se-   | AGO     | A sequence symbol    |
|quence  |         | defined in a following|
|symbol  |         | statement            |
|or      |         |                      |
|blank   |         |                      |
+--------+---------+----------------------+
```

The sequence symbol in the operand field may be in the name field of a statement following the AGO statement. The statement named by the sequence symbol in the operand field is the next statement processed by the Assembler.

The following example illustrates the use of the AGO instruction. The function of this macro instruction is to move a specified number of bytes from one location in main storage to another. The MOVE macro definition shown in the section AIFB -- Conditional Branch Backward is used as an inner macro instruction in this example.

The boxes in the example below contain respectively: the macro definition, the first macro instruction, the statements generated as a result of the first macro instruction, the second macro instruction, and the statements generated because of it.

```
┌──────┬─────────┬──────────────────────────┐
│      │ Oper-   │                          │
│Name  │ ation   │  Operand                 │
├──────┼─────────┼──────────────────────────┤
│      │ MACRO   │                          │
│      │ MOVEN   │  &NOCHAR,&TO,&FROM       │
│      │ AIF     │  ('&NOCHAR' EQ '').A     │
│      │ AIF     │  (&NOCHAR NE 2).B        │
│.A    │ STH     │  12,SAVEAREA             │
│      │ LH      │  12,&FROM                │
│      │ STH     │  12,&TO                  │
│      │ LH      │  12,SAVEAREA             │
│      │ AGO     │  .C                      │
│.B    │ MOVE    │  &NOCHAR,&TO,&FROM       │
│.C    │ MEND    │                          │
├──────┼─────────┼──────────────────────────┤
│      │ MOVEN   │  ,FIELDA,WORK            │
├──────┼─────────┼──────────────────────────┤
│      │ STH     │  12,SAVEAREA             │
│      │ LH      │  12,WORK                 │
│      │ STH     │  12,FIELDA               │
│      │ LH      │  12,SAVEAREA             │
├──────┼─────────┼──────────────────────────┤
│      │ MOVEN   │  97,OUT+540,RESULT       │
├──────┼─────────┼──────────────────────────┤
│      │ MVC     │  OUT+540+0(97),RESULT+0  │
└──────┴─────────┴──────────────────────────┘
```

AGOB -- UNCONDITIONAL BRANCH BACKWARD

The AGOB instruction may be used to alter the sequence in which source statements are processed by the Assembler. The format of this instruction is:

```
┌─────────┬─────────┬──────────────────────┐
│         │Oper-    │                      │
│Name     │ation    │  Operand             │
├─────────┼─────────┼──────────────────────┤
│A se-    │AGOB     │  A sequence symbol   │
│quence   │         │  defined in a        │
│symbol   │         │  preceding statement │
│or blank │         │                      │
└─────────┴─────────┴──────────────────────┘
```

The AGOB instruction is identical to the AGO statement except that the sequence symbol in the operand field must be in the name field of a instruction preceding the AGOB instruction.

The following illustrates the use of the AGOB instruction. The macro definition in this example is functionally the same as the macro definition in the section AIFB--Conditional Branch Backward.

The boxes in the example below contain respectively: the macro definition, the macro instruction, and the statements generated as a result of the macro instruction.

```
┌──────┬──────┬──────────────────────────┐
│      │Oper- │                          │
│Name  │ation │Operand                   │
├──────┼──────┼──────────────────────────┤
│      │MACRO │                          │
│      │MOVE  │&NOCHAR,&TO,&FROM         │
│&AL2  │SETA  │&NOCHAR                   │
│.LOOP │AIF   │(&AL2 LE 256).LSTMOV      │
│      │MVC   │&TO+&AL1.(256),&FROM+&AL1 │
│&AL1  │SETA  │&AL1+256                  │
│&AL2  │SETA  │&NOCHAR-&AL1              │
│      │AGOB  │.LOOP                     │
│.LSTMOV│MVC  │&TO+&AL1.(&AL2),&FROM+&AL1│
│      │MEND  │                          │
├──────┼──────┼──────────────────────────┤
│      │MOVE  │540,OUT,INPUT             │
├──────┼──────┼──────────────────────────┤
│      │MVC   │OUT+0(256),INPUT+0        │
│      │MVC   │OUT+256(256),INPUT+256    │
│      │MVC   │OUT+512(28),INPUT+512     │
└──────┴──────┴──────────────────────────┘
```

ANOP -- NO OPERATION

The ANOP instruction may be used to facilitate conditional and unconditional branching to statements named by symbols or SET symbols. The format of this instruction is:

```
┌─────────┬─────────┬──────────────────────┐
│         │Oper-    │                      │
│ Name    │ation    │    Operand           │
├─────────┼─────────┼──────────────────────┤
│ A se-   │ANOP     │    Blank             │
│ quence  │         │                      │
│ symbol  │         │                      │
└─────────┴─────────┴──────────────────────┘
```

If you want to use an AGO, AGOB, AIF, or AIFB instruction to branch to a instruction that has a symbol or SET symbol in the name field, place an ANOP statement before the instruction you want to branch to, and branch to the ANOP instruction.

The following example illustrates the use of the ANOP statement. This example allows a field of any length to be moved. The source and destination fields need not be on a halfword boundary. The name field contains the symbolic name of the first instruction of the macro routine.

The boxes in the example below contain respectively: the macro definition, the macro instruction, and the statements generated as a result of the macro instruction.

```
┌────────┬──────┬──────────────────────────┐
│        │Oper- │                          │
│Name    │ation │Operand                   │
├────────┼──────┼──────────────────────────┤
│        │MACRO │                          │
│&NAME   │MOVE  │&NOCHAR,&FROM,&TO         │
│&AL2    │SETA  │&NOCHAR                   │
│&CG1    │SETC  │'&NAME'                   │
│.LOOP   │AIF   │(&AL2 LE 256).LSTMOV      │
│&CG1    │MVC   │&TO+&AL1.(256),&FROM+&AL1 │
│&AL1    │SETA  │&AL1+256                  │
│&AL2    │SETA  │&NOCHAR-&AL1              │
│&CG1    │SETC  │''                        │
│        │AGOB  │.LOOP                     │
│.LSTMOV │ANOP  │                          │
│&CG1    │MVC   │&TO+&AL1.(&AL2),&FROM+&AL1│
│        │MEND  │                          │
├────────┼──────┼──────────────────────────┤
│FIRST   │MOVE  │540,INPUT,OUT             │
├────────┼──────┼──────────────────────────┤
│FIRST   │MVC   │OUT+0(256),INPUT+0        │
│        │MVC   │OUT+256(256),INPUT+256    │
│        │MVC   │OUT+512(28),INPUT+512     │
└────────┴──────┴──────────────────────────┘
```

Note that the value of the local variable symbol &AL1 is initially zero.

MEXIT -- MACRO DEFINITION EXIT

The MEXIT instruction can be used to indicate to the Assembler to terminate processing of a macro definition. The format of this instruction is:

```
┌─────────┬─────────┬──────────────────────┐
│         │Oper-    │                      │
│ Name    │ation    │   Operand            │
├─────────┼─────────┼──────────────────────┤
│ A se-   │MEXIT    │   Blank              │
│ quence  │         │                      │
│ symbol  │         │                      │
│ or      │         │                      │
│ blank   │         │                      │
└─────────┴─────────┴──────────────────────┘
```

The MEXIT instruction may be used in a macro definition when you wish that only a certain portion of the definition be generated. For example, a definition contains two sequences of operations. The first sequence is to be generated if a specified condition is met and the second sequence is to be generated if another specified condition is met. The use of the MEXIT instruction after the first sequence will terminate generation, just the same as the MEND instruction will do when placed after the second sequence.

The MEXIT instruction should not be confused with the MEND instruction. The MEND instruction indicates the end of a macro definition to the macro generating phase of the Assembler, as well as signifying the end of generation. Every macro definition must contain a MEND instruction even if the definition contains one or more MEXIT instructions.

The following example illustrates the use of the MEXIT instruction. The function of the macro definition is to move a specified number of bytes of information from one location in main storage to another. The definition is essentially the same as the macro definition shown in the section AGO -- Unconditional Branch. However, the use of the MEXIT instruction reduces the time required for assembling the macro instruction if the first routine is used.

The boxes in the example below contain respectively: the macro definition, the first macro instruction, the statements generated as a result of the first macro instruction, the second macro instruction, and the statements generated because of it.

```
┌────────┬──────┬──────────────────────────┐
│        │Oper- │                          │
│Name    │ation │Operand                   │
├────────┼──────┼──────────────────────────┤
│        │MACRO │                          │
│        │MOVE  │&NOCHAR,&TO,&FROM         │
│        │AIF   │('&NOCHAR' EQ '').A       │
│        │AIF   │(&NOCHAR NE 2).B          │
│.A      │STH   │12,SAVEAREA               │
│        │LH    │12,&FROM                  │
│        │STH   │12,&TO                    │
│        │LH    │12,SAVEAREA               │
│        │MEXIT │                          │
│.B      │ANOP  │                          │
│&AL2    │SETA  │&NOCHAR                   │
│.LOOP   │AIF   │(&AL2 LE 256).LSTMOV      │
│        │MVC   │&TO+&AL1.(256),&FROM+&AL1 │
│&AL1    │SETA  │&AL1+256                  │
│&AL2    │SETA  │&NOCHAR-&AL1              │
│        │AGOB  │.LOOP                     │
│.LSTMOV │MVC   │&TO+&AL1.(&AL2),&FROM+&AL1│
│        │MEND  │                          │
├────────┼──────┼──────────────────────────┤
│        │MOVE  │2,OUT,INPUT               │
├────────┼──────┼──────────────────────────┤
│        │STH   │12,SAVEAREA               │
│        │LH    │12,INPUT                  │
│        │STH   │12,OUT                    │
│        │LH    │12,SAVEAREA               │
├────────┼──────┼──────────────────────────┤
│        │MOVE  │540,OUT,INPUT             │
├────────┼──────┼──────────────────────────┤
│        │MVC   │OUT+0(256),INPUT+0        │
│        │MVC   │OUT+256(256),INPUT+256    │
│        │MVC   │OUT+512(28),INPUT+512     │
└────────┴──────┴──────────────────────────┘
```

MNOTE -- REQUEST FOR A MESSAGE

The MNOTE instruction may be used to request the Assembler to generate a message. The format of this instruction is:

Name	Operation	Operand
A sequence symbol or blank	MNOTE	Any combination of characters enclosed in apostrophes. A severity code, as used in Assembler languages for higher models, is ignored.

When an MNOTE statement is processed by the Assembler, the characters in the operand field are printed in the program listing in the same way error messages are printed in the program listing. The outside apostrophes are not printed.

If variable symbols are used in the operand field, they are replaced by the values they represent.

The following example illustrates the use of the MNOTE statement. This macro definition tests for the presence of the three parameters in the macro instruction. If any parameter is missing, an appropriate message is printed and assembly of the macro instruction is terminated.

```
|      |Oper-  |                                |
|Name  |ation  |Operand                         |
|------|-------|--------------------------------|
|      |MACRO  |                                |
|      |MOVE   |&NOCHAR,&TO,&FROM               |
|      |AIF    |('&NOCHAR' NE '').N0            |
|      |MNOTE  |'FIRST PARAMETER OMITTED'       |
|&BL1  |SETB   |(1)                             |
|.N0   |AIF    |('&TO' NE '').N1                |
|      |MNOTE  |'SECOND PARAMETER OMITTED'      |
|&BL1  |SETB   |(1)                             |
|.N1   |AIF    |('&FROM' NE '').N2              |
|      |MNOTE  |'THIRD PARAMETER OMITTED'       |
|.N3   |MNOTE  |'GENERATION TERMINATED'         |
|      |MEXIT  |                                |
|.N2   |AIFB   |(&BL1).N3                       |
|&AL2  |SETA   |&NOCHAR                         |
|.LOOP |AIF    |(&AL2 LE 256).LSTMO             |
|      |MVC    |&TO+&AL1.(256),&FROM+&AL1       |
|&AL1  |SETA   |&AL1+256                        |
|&AL2  |SETA   |&NOCHAR-&AL1                    |
|      |AGOB   |.LOOP                           |
|.LSTMO|MVC    |&TO+&AL1.(&AL2),&FROM+&AL1      |
|      |MEND   |                                |
```

Comments Statements

Comments statements may be interspersed in the model statements of a macro definition. Two types of comments statements are permitted.

The first type of comments statement has an asterisk (*) in column 1, followed by the comment. This type is included in a macro definition. The Assembler generates this type of comments statement into any source program that uses the particular macro definition.

The second type of comments statement has a period (.) in column 1, immediately followed by an asterisk (*), followed by the comment. This type of comments statement documents the macro definition and is not included in the macro definition.

MEND -- TRAILER STATEMENT

The trailer statement indicates to the Assembler that a macro definition is complete. It must be the last statement in every macro definition. The format of this statement is:

Name	Operation	Operand
A sequence symbol or blank	MEND	Blank

A sequence symbol consists of a period followed by one to seven alphabetic and/or numeric characters, the first of which must be alphabetic. Sequence symbols are discussed in detail under Sequence Symbols.

Keyword Macro Definitions

This section describes the differences between a keyword macro definition and a positional macro definition.

A keyword macro definition is used in cases where the number or type of operands is such that a positional macro instruction becomes confusing or cumbersome. It allows the values specified by each parameter to be used with a predefined keyword. A keyword macro definition allows the operands to be specified in any desired order.

The keyword format has two additional advantages: (1) it is possible to limit the number of operands in a given card and (2) it allows the specification of a standard value in the prototype statement. If an operand is missing in the macro instruction, the standard value from the prototype statement replaces any occurrences of that symbolic parameter in the model statements.

Each keyword macro definition must include: a header statement, a prototype statement, model statements, and a trailer statement.

Like a positional macro definition, a keyword macro definition may contain comment and conditional-assembly statements, all of which are described in preceding sections. Keyword macro definitions may include the &SYSNDX and &SYSECT system variable symbols, but no &SYSLIST system variable symbols. (See System Variable Symbols).

The general format of a keyword prototype statement is:

Name	Operation	Operand
A symbolic parameter or blank	A symbol	Up to 49 operands, separated by commas, of the form described below

A keyword prototype statement differs from a positional prototype statement only in the operand field.

Each operand must consist of a symbolic parameter followed by an equal sign. Symbolic parameters are described under Positional Prototype Statement.

The equal sign may be followed by a standard value to be substituted for the symbolic parameter in case the parameter is not contained as a keyword in the operand field of a macro instruction.

If a standard value is either not desired or unnecessary, the equal sign is followed by the comma that separates the parameter from the next parameter. In the case of the last parameter, the equal sign may be followed by a blank.

Anything that can be used as an operand in a macro instruction may be used as a standard value in a keyword prototype statement, including null values.

The following are valid keyword prototype statement operands:

```
&TO=234
&LOOP2=SYMBOL
&S4=H'4096'
&FROM=
```

The following are invalid keyword prototype statement operands, for the reasons stated:

=CARDAREA	No symbolic parameter.
&TYPE	No equal sign.
&IN 256B	Standard value used, but no equal sign.
&TWO =123	Equal sign does not immediately follow symbolic parameter.

System Variable Symbols

System variable symbols are local variable symbols that are assigned values by the Assembler. They may be used in the name or operand field of macro definition statements. They are not permitted in the name field of conditional-assembly instructions.

If a system variable symbol is used in the name or operand field of a statement that is part of a macro definition, the value substituted for the variable symbol is the value the Assembler has assigned to the variable symbol.

&SYSNDX -- MACRO INSTRUCTION INDEX

The system variable symbol &SYSNDX may be concatenated to other characters to create unique symbols for generated statements. &SYSNDX is assigned the decimal value 0001 for the first macro instruction that is assembled. The value assigned to &SYSNDX for any other macro instruction is one plus the value assigned to &SYSNDX for the previous macro instruction. High-order zeros are not suppressed.

Throughout one use of a macro definition, the value of &SYSNDX may be considered a constant, independent of any inner macro instruction in that definition. If &SYSNDX is used in the name or operand field of a statement that is part of a macro definition, the value substituted for &SYSNDX is the value assigned to it for the macro instruction being interpreted.

One use of the &SYSNDX system variable symbol is shown in the following macro definition. The function of this macro definition is to move the contents of one storage area to another area in main storage.

In the example, A&SYSNDX provides a unique symbol in the name field for branching to a particular instruction within the macro definition. The content of a field is not moved if the first byte of the field is a binary zero.

The function of this macro definition is to move the contents of one storage area to another area in main storage.

```
|Name       | Operation |  Operand          |
|-----------|-----------|-------------------|
|           | MACRO     |                   |
|           | MOVE      | &TO,&FROM         |
|           | CLI       | &FROM,X'00'       |
|           | BE        | A&SYSNDX          |
|           | STH       | 12,SAVEAREA       |
|           | LH        | 12,&FROM          |
|           | STH       | 12,&TO            |
|           | LH        | 12,SAVEAREA       |
|A&SYSNDX   | EQU       | *                 |
|           | MEND      |                   |
```

If the following macro instructions were the 106th and the 107th macro instructions interpreted by the macro generator, the following statements would be generated.

```
| Name    |Operation |   Operand           |
|---------|----------|---------------------|
|         |MOVE      |   FIELDA,FIELDB     |
|---------|----------|---------------------|
|         |CLI       |   FIELDB,X'00'      |
|         |BE        |   A0106             |
|         |STH       |   12,SAVEAREA       |
|         |LH        |   12,FIELDB         |
|         |STH       |   12,FIELDA         |
|         |LH        |   12,SAVEAREA       |
| A0106   |EQU       |   *                 |
|---------|----------|---------------------|
|         |MOVE      |   FIELDC,FIELDD     |
|---------|----------|---------------------|
|         |CLI       |   FIELDD,X'00'      |
|         |BE        |   A0107             |
|         |STH       |   12,SAVEAREA       |
|         |LH        |   12,FIELDD         |
|         |STH       |   12,FIELDC         |
|         |LH        |   12,SAVEAREA       |
| A0107   |EQU       |   *                 |
```

&SYSECT -- CURRENT CONTROL SECTION

The system variable symbol &SYSECT may be used to represent the name of the control section in which a macro instruction appears. For each macro instruction processed by the Assembler, &SYSECT is assigned a value that is the name of the control section in which the macro instruction appears.

When &SYSECT is used in a macro definition, the value substituted for &SYSECT is the name of the last CSECT, DSECT, or START statement that occurs before the macro instruction. If no named CSECT, DSECT, or START statement occurs before a macro instruction, &SYSECT is assigned a null-character value for that macro instruction.

CSECT or DSECT statements processed in a macro definition affect the value for &SYSECT for any subsequent inner macro instructions in that definition, and for any other following macro instructions.

Throughout the use of a macro definition, the value of &SYSECT may be considered a constant, independent of any CSECT or DSECT statements or inner macro instructions in that definition.

The example below illustrates these rules. (In the example, model statements not required for explanation have been omitted.)

```
    |Name      |Operation |Operand          |
    |----------|----------|-----------------|
    |          |MACRO     |                 |
    |          |INNER     |&INCSECT         |
 1  |&INCSECT  |CSECT     |                 |
 2  |          |DC        |Y(&SYSECT)       |
    |          |MEND      |                 |
    |----------|----------|-----------------|
    |          |MACRO     |                 |
    |          |OUTR1     |                 |
 3  |CSOUT1    |CSECT     |                 |
    |          |DS        |100C             |
 4  |          |INNER     |INA              |
 5  |          |INNER     |INB              |
 6  |          |DC        |Y(&SYSECT)       |
    |          |MEND      |                 |
    |----------|----------|-----------------|
    |          |MACRO     |                 |
    |          |OUTR2     |                 |
 7  |          |DC        |Y(&SYSECT)       |
    |          |MEND      |                 |
    |----------|----------|-----------------|
 8  |MAINPROG  |CSECT     |                 |
    |          |DS        |200C             |
 9  |          |OUTR1     |                 |
10  |          |OUTR2     |                 |
    |----------|----------|-----------------|
    |MAINPROG  |CSECT     |                 |
    |          |DS        |200C             |
    |CSOUT1    |CSECT     |                 |
    |          |DS        |100C             |
    |INA       |CSECT     |                 |
    |          |DC        |Y(CSOUT1)        |
    |INB       |CSECT     |                 |
    |          |DC        |Y(INA)           |
    |          |DC        |Y(MAINPROG)      |
    |          |DC        |Y(INB)           |
```

Statement 8 is the last CSECT, DSECT, or START instruction processed before statement 9 is processed. Therefore, &SYSECT is assigned the value MAINPROG for macro instruction OUTR1 in statement 9. MAINPROG is substituted for &SYSECT when it appears in statement 6.

Statement 3 is the last CSECT, DSECT, or START instruction processed before statement 4 is processed. Therefore &SYSECT is assigned the value CSOUT1 for macro instruction INNER in statement 4. CSOUT1 is substituted for &SYSECT when it appears in statement 2.

Statement 1 is used to generate a CSECT instruction for statement 4. This is the last CSECT, DSECT, or START statement that appears before statement 5. Therefore, &SYSECT is assigned the value INA for macro instruction INNER in statement 5. INA is substituted for &SYSECT when it appears in statement 2.

Statement 1 is used to generate a CSECT statement for statement 5. This is the last CSECT, DSECT, or START statement that appears before statement 10. Therefore, &CSECT is assigned the value INB for macro instruction OUTR2 in statement 10. INB is substituted for &SYSECT when it appears in statement 7.

```
|      |Oper-|                                |
|Name  |ation|Operand                         |
|------|-----|--------------------------------|
|      |MACRO|                                |
|&NAME |ADD  |&F1,&F2,&F3,&F4,&F5             |
|&NAME |STH  |12,SAVEAREA                     |
|      |LH   |12,&F1                          |
|&AL1  |SETA |2                               |
|.ADD  |AH   |12,&SYSLIST(&AL1)               |
|&AL1  |SETA |&AL1+1                          |
|&AL2  |SETA |&AL1+1                          |
|      |AIFB |('&SYSLIST(&AL2)' NE '').ADD    |
|      |STH  |12,&SYSLIST(&AL1)               |
|      |LH   |12,SAVEAREA                     |
|      |MEND |                                |
|------|-----|--------------------------------|
|      |ADD  |FTAX,FICA,STAX,BONDS,DEDUCT     |
|------|-----|--------------------------------|
|      |STH  |12,SAVEAREA                     |
|      |LH   |12,FTAX                         |
|      |AH   |12,FICA                         |
|      |AH   |12,STAX                         |
|      |AH   |12,BONDS                        |
|      |STH  |12,DEDUCT                       |
|      |LH   |12,SAVEAREA                     |
|------|-----|--------------------------------|
|DEUTOT|ADD  |REGHRS,OTHRS,TOTHRS             |
|------|-----|--------------------------------|
|DEUTOT|STH  |12,SAVEAREA                     |
|      |LH   |12,REGHRS                       |
|      |AH   |12,OTHRS                        |
|      |STH  |12,TOTHRS                       |
|      |LH   |12,SAVEAREA                     |
```

&SYSLIST(n) -- MACRO INSTRUCTION OPERAND FIELD

The system variable symbol &SYSLIST(n) provides you with an alternate way to refer to the nth operand of a positional macro instruction. n may be a decimal self-defining value or a SETA symbol. The &SYSLIST(n) system variable symbol is not permitted in a keyword macro definition.

&SYSLIST(n) and symbolic parameters may be used in the same macro definition.

The self-defining value following &SYSLIST(n) may be any value between 1 and 49, regardless of the number of symbolic parameters in the prototype statement. If the corresponding symbolic parameter is not contained in the prototype statement, it is treated as a null parameter.

The following example illustrates the use of the &SYSLIST(n) system variable symbol.

The function of this macro definition is to add the contents of the fields specified in the operand field of the macro instruction and store the sum in the field specified by the last operand of the macro instruction. Depending on the number of operands included in the macro instruction, 2, 3, or 4 fields are added together. The result is stored in the last field specified in the macro instruction operand.

```
|      |Oper-|                                |
|Name  |ation|Operand                         |
|------|-----|--------------------------------|
|      |MACRO|                                |
|&LABEL|MOVE |&A,&T1,&F1,&T2,&F2,&T3,&F3      |
|&LABEL|STH  |12,&A                           |
|&AL3  |SETA |2                               |
|&AL4  |SETA |3                               |
|.LO   |LH   |12,&SYSLIST(&AL4)               |
|      |STH  |12,&SYSLIST(&AL3)               |
|&AL3  |SETA |&AL3+2                          |
|&AL4  |SETA |&AL4+2                          |
|      |AIFB |('&SYSLIST(&AL3)' NE '').LO     |
|      |LH   |12,&A                           |
|      |MEND |                                |
|------|-----|--------------------------------|
|MULMOV|MOVE |AREA,A,B,X,Y                    |
|------|-----|--------------------------------|
|MULMOV|STH  |12,AREA                         |
|      |LH   |12,B                            |
|      |STH  |12,A                            |
|      |LH   |12,Y                            |
|      |STH  |12,X                            |
|      |LH   |12,AREA                         |
```

The preceding example further illustrates the use of the &SYSLIST(n) system variable symbol. In this macro definition, a multiple move is accomplished. The numb-

er of fields to be moved depends upon the number of symbolic parameters included in the prototype statement and the number of entries in the operand field of the macro instruction.

Sample Macro Definition

This section contains a sample macro definition named GMOVE. Figure 12 is a flowchart that describes the logic of the macro definition. The flowchart is an example of one which you might draw in preparing to code a macro definition. Figure 13 is the actual coding of the macro definition.

The section further contains a set of instructions for using the macro definition in a source program. The set of instructions illustrates the rules for writing macro instructions in a source program.

The GMOVE macro instruction causes the Assembler to generate instructions to move a source field to a destination field regardless of the length of the field. It can also move up to ten source fields of any length to consecutive locations in a destination field. A typical use of this multi-source or gather-move function is to build an output record.

If the same move is to be used in several places within the same control section, a facility is provided to generate the move instructions as a closed subroutine and link to the subroutine rather then generate them repeatedly in-line. Use of the subroutine facility saves main storage.

IN-LINE USE OF THE GMOVE MACRO INSTRUCTION

To generate the appropriate move instructions without establishing a subroutine:

1. Leave the name field blank.
2. Punch the operation field as GMOVE.
3. State in the operand field:
   a. The first operand as the name of the destination field.
   b. The second operand as the name of the first source field.
   c. The third operand as the length of the first source field.
   d. If only one source field is used, enter no more operands.
   e. If several source fields are used, punch two additional operands for each additional source field up to a maximum of ten source fields. Each pair of operands consists of the symbol of the source field followed by the length of the source field.

Code Generated for the GMOVE Macro Instruction used In-line

The GMOVE macro instruction generates one or more MVC instructions each time it appears, as illustrated by the three examples below.

```
┌──────┬──────────────────────────────────┐
│Oper- │                                  │
│ation │Operand                           │
├──────┼──────────────────────────────────┤
│GMOVE │FIELDA,FIELDB,17                  │
├──────┼──────────────────────────────────┤
│MVC   │FIELDA+0(17),FIELDB+0             │
├──────┼──────────────────────────────────┤
│GMOVE │FIELDY,FIELDX,540                 │
├──────┼──────────────────────────────────┤
│MVC   │FIELDY+0(256),FIELDX+0            │
│MVC   │FIELDY+256(256),FIELDX+256        │
│MVC   │FIELDY+512(28),FIELDX+512         │
├──────┼──────────────────────────────────┤
│GMOVE │OUTPUT,NAME,20,ADDRESS,75,MANNUM,5│
├──────┼──────────────────────────────────┤
│MVC   │OUTPUT+0(20),NAME+0               │
│MVC   │OUTPUT+20(75),ADDRESS+0           │
│MVC   │OUTPUT+95(5),MANNUM+0             │
└──────┴──────────────────────────────────┘
```

Address Adjustment of Fields

An operand within the macro instruction containing the symbolic name of a source or destination field may be address-adjusted provided that the total length of the operand does not exceed eight characters. For example:

```
┌──────┬──────────────────────────────────┐
│Oper- │                                  │
│ation │ Operand                          │
├──────┼──────────────────────────────────┤
│GMOVE │ OUT+100,CITY,35,STATE,20,JOB+12,4│
├──────┼──────────────────────────────────┤
│MVC   │ OUT+100+0(35),CITY+0             │
│MVC   │ OUT+100+35(20),STATE+0           │
│MVC   │ OUT+100+55(4),JOB+12+0           │
└──────┴──────────────────────────────────┘
```

The address of the destination field in the second MVC instruction is higher than that of the first destination field by the length of the first source field. Likewise, the address of the destination field of the third MVC instruction statement is higher than that of the second MVC instruction by the length of the second source field.

RESERVING SPACE IN THE DESTINATION FIELD

Occasionally it is impossible to enter one or more fields in an output record at the time the rest of the record is being built. With the GMOVE macro instruction, a facility is provided to leave a space between the fields being moved. This is accomplished by entering a zero as the symbol for a source field with the length given as the number of bytes to be skipped before the next source field to be moved. For example:

```
|         |                          |Col. |
|Operation|Operand                   |72   |
|---------|--------------------------|-----|
|GMOVE    |OUT,NAME,20,ADDRESS,30,0, |-    |
|         |54,SERNO,5,JOB,16         |     |
|---------|--------------------------|-----|
|MVC      |OUT+0(20),NAME+0          |     |
|MVC      |OUT+20(30),ADDRESS+0      |     |
|MVC      |OUT+104(5),SERNO+0        |     |
|MVC      |OUT+109(16),JOB+0         |     |
```

In the example above, the operand field of the GMOVE macro instruction contains the addresses of five source fields: NAME, ADDRESS,0,SERNO and JOB. The symbol 0 for the source field together with the length specification of 54 cause the bytes to be skipped from address OUT+50 to address OUT+103 inclusively.

USE OF THE SUBROUTINE FACILITY OF THE GMOVE MACRO DEFINITION

To obtain a generated subroutine with the GMOVE macro instruction, write the macro instruction in the same manner as if a subroutine were not desired and write a unique name in the name field of the macro instruction. The length of the name must not exceed seven characters. The symbolic name preceded by an E must also be unique.

An entry in the name field will cause the generation of the subroutine. The subroutine, once established, can be used by:

- Coding the unique symbolic name in the name field.
- Coding GMOVE in the operation field.

Up to five closed subroutines can be generated within each control section.

A subroutine thus generated can only be used in the control section in which it is generated. Generation of a subroutine in one control section will cause the macro generator to "forget" all subroutines generated in previous control sections.

The entry in the name field of a GMOVE macro instruction which generates a subroutine is used as the name of the entry point of the routine. The name preceded by an E is equated to the next sequential instruction following the macro instruction. Register 9 is used to link a generated subroutine. The previous contents of register 9 are lost. If you wish to save the contents of register 9, it is your responsibility to save and restore it.

The operands of a GMOVE macro instruction that link to an established subroutine are ignored and may be omitted. It may be desirable to include all operands in each usage to ensure that the operands are present in the first occurrence during an assembly.

The following example shows the subroutine facility of the GMOVE macro.

The first and the third box show macro instructions as they might be coded in a source program. The second and fourth box show the source statements generated on account of the macro instructions.

```
|    |Oper-|                          |Col. |
|Name|ation|Operand                   |72   |
|----|-----|--------------------------|-----|
|COL |GMOVE|OUTREC,SERNO,4,NAME,20,AD |-    |
|    |     |DRESS,30,CITY,25,STATE,15 |-    |
|    |     |,0,25,INPUT1,540,INPUT2,2 |-    |
|    |     |60                        |     |
|----|-----|--------------------------|-----|
|    |LH   |9,ECOL                    |     |
|*   |     |START OF GMOVE SUBROUTINE |     |
|COL |MVC  |OUTREC+0(4),SERNO+0       |     |
|    |MVC  |OUTREC+4(20),NAME+0       |     |
|    |MVC  |OUTREC+24(30),ADDRESS+0   |     |
|    |MVC  |OUTREC+54(25),CITY+0      |     |
|    |MVC  |OUTREC+79(15),STATE+0     |     |
|    |MVC  |OUTREC+119(256),INPUT1+0  |     |
|    |MVC  |OUTREC+375(256),INPUT1+256|     |
|    |MVC  |OUTREC+631(28),INPUT1+512 |     |
|    |MVC  |OUTREC+659(256),INPUT2+0  |     |
|    |MVC  |OUTREC+915(4),INPUT2+256  |     |
|    |BR   |9                         |     |
|*   |     |END OF GMOVE SUBROUTINE   |     |
|ECOL|DC   |Y(*+2)                    |     |
|----|-----|--------------------------|-----|
|COL |GMOVE|                          |     |
|----|-----|--------------------------|-----|
|    |BAS  |9,COL                     |     |
```

Note that subroutines can be established without using the subroutine facility as shown in the example. This can be accomplished by using the macro instruction to generate in-line coding within a closed subroutine.

MAIN-STORAGE CONSIDERATIONS FOR GMOVE SUBROUTINES

In the preceding example, 72 bytes of main storage were used for the two macro instructions. If the subroutine had not been used, 120 bytes of main storage would have been required. Eight additional bytes of main storage are used when a subroutine is established and each request for the subroutine requires four bytes.

The amount of main storage that is saved is a function of how many MVC instructions are generated in the subroutine. The use of a GMOVE subroutine containing only one MVC instruction does not save main storage unless it is used six times (including the initial time). It would actually take extra main storage if used less than five times, even if no additional instructions were required to save the contents of register 9.

ERROR CHECKING

Error checking is performed on the operands in a GMOVE macro instruction.

Generation is always terminated if either the destination field or the first source field and its length are not specified. An appropriate error message is generated.

If an invalid source field length (zero or non-numeric) is specified or if the length of any other source field is omitted, generation is terminated. An appropriate error message is generated.

If an attempt is made to generate the sixth subroutine in the same control section, the GMOVE routine is generated in-line. An appropriate error message is generated.

Note that subroutines can be established without using the subroutine facility of the GMOVE macro. This can be accomplished by using the GMOVE macro instruction to generate in-line coding within a closed subroutine.

USE OF GLOBAL SET SYMBOLS WITHIN THE GMOVE MACRO DEFINITION

To ensure a correct assembly of a macro instruction, you must avoid any conflict in the use of SET symbols. The GMOVE macro instruction makes use of global SET symbols as described below.

1. The SETC symbol &CG1 is used in the GMOVE macro definition, and must not be used to communicate a value past the occurrence of the GMOVE macro instruction. In addition, if one or more subroutines are generated, the SETC symbols &CG10 to &CG15 are used.

2. If no subroutines are generated, the GMOVE macro instruction is properly generated regardless of the setting of any global SET symbols, either before or between occurrences of the GMOVE macro instruction.

3. If one or more subroutines are generated in one control section, the SETC symbols &CG11 to &CG15 are used by the Assembler to store subroutine names. They must not be used between the generation of the GMOVE subroutine routine and the last use of the GMOVE macro definition to link to a subroutine routine. The SETC symbol &CG10 is used to store the name of the control section, and therefore:

   a. &CG10 must not contain the name of the control section before the first subroutine is created.

   b. &CG10 must not be changed until all subroutines and linkages to subroutines in a control section have been generated.

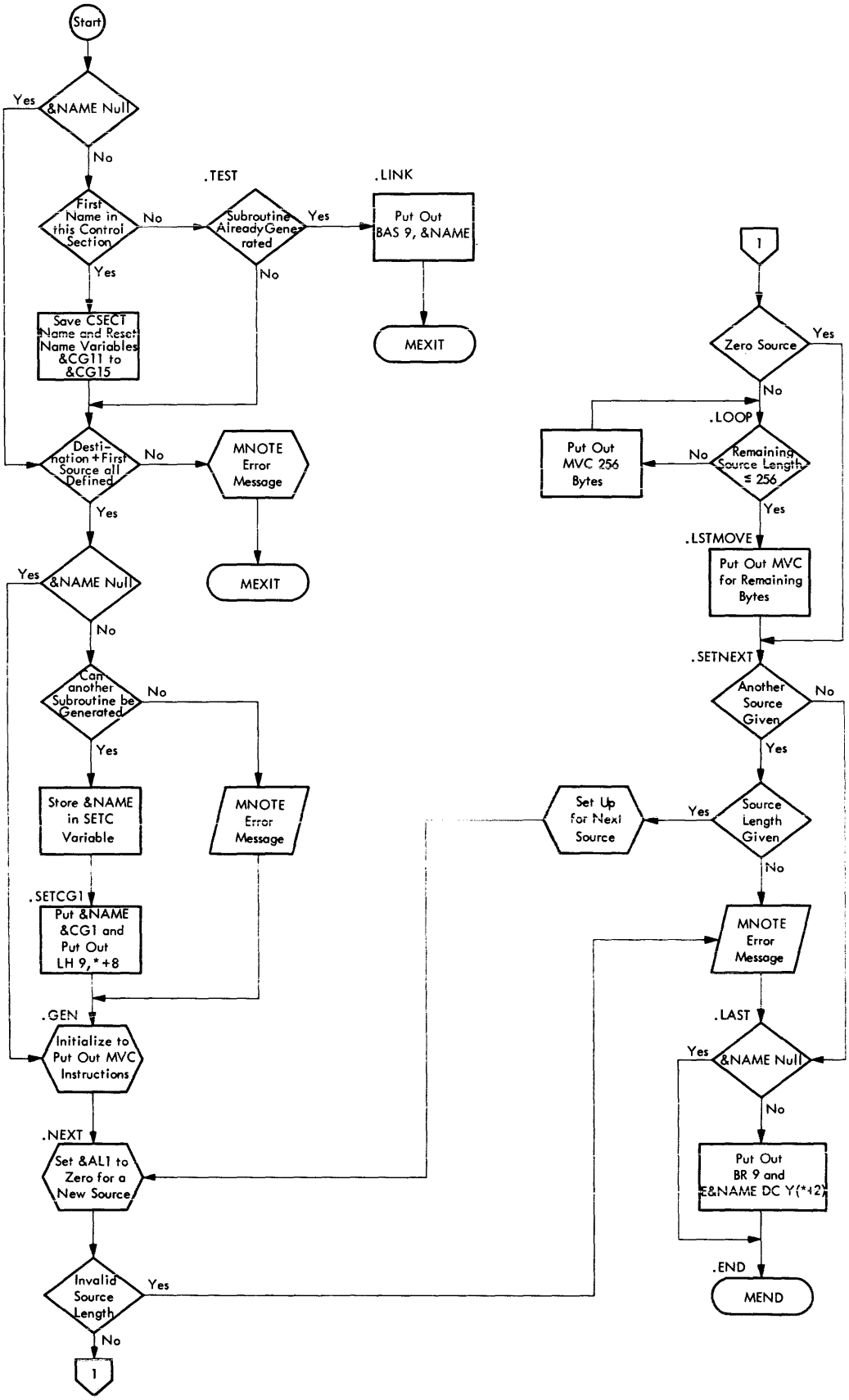


Figure 12. Flowchart of the GMOVE Macro Instruction

IBM System/360 Assembler Coding Form

X28-6509-4 U/M025 Printed in U.S.A.

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 5 | CARD ELECTRO NUMBER *

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
	MACRO			
&NAME	GMOVE	&TO,&F1,&LF1,&F2,&LF2,&F3,&LF3,&F4,&LF4,&F5,&LF5,&F6,&LF-		
		6,&F7,&LF7,&F8,&LF8,&F9,&LF9,&F10,&LF10		
	AIF	('&NAME' EQ '').NONAME		
	AIF	('&CG10' EQ '&SYSECT').TEST		
.*			FIRST USE OF GMOVE WITH SUBROUTINE REQUESTED	
.*			IN THIS CONTROL SECTION	
&CG10	SETC	'&SYSECT'		
&CG11	SETC	''		
&CG12	SETC	''		
&CG13	SETC	''		
&CG14	SETC	''		
&CG15	SETC	''		
	AGO	.NONAME		
.*			CHECK FOR SUBROUTINE ALREADY GENERATED	
.TEST	AIF	('&NAME' EQ '&CG11').LINK		
	AIF	('&NAME' EQ '&CG12').LINK		
	AIF	('&NAME' EQ '&CG13').LINK		
	AIF	('&NAME' EQ '&CG14').LINK		
	AIF	('&NAME' EQ '&CG15').NONAME		
.*			LINK TO PREVIOUSLY GENERATED SUBROUTINE	
.LINK	BAS	9,&NAME		
	MEXIT			
.*			CHECK FOR PRESENCE OF THE MINIMUM PARAMETERS	
.NONAME	AIF	('&TO' NE '').A		

* A standard card form, IBM electro 6509, is available for punching source statements from this form.
Instructions for using this form are in any IBM System/360 Assembler Reference Manual.
Address comments concerning this form to IBM Corporation, Programming Publications, Department 232, San Jose, California 95114.

IBM System/360 Assembler Coding Form

X28-6509-4 U/M025 Printed in U.S.A.

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 2 OF 5 | CARD ELECTRO NUMBER *

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
	MNOTE	'DESTINATION PARAMETER OMITTED'		
&BL1	SETB	(1)		
.A	AIF	('&F1' NE '').B		
	MNOTE	'FIRST SOURCE PARAMETER OMITTED'		
&BL1	SETB	(1)		
.B	AIF	('&LF1' NE '').C		
	MNOTE	'LENGTH OF FIRST SOURCE PARAMETER OMITTED'		
.D	MNOTE	'GENERATION TERMINATED'		
	MEXIT			
.C	AIFB	(&BL1).D		
.*			SUFFICENT PARAMETERS PRESENT FOR GENERATION	
&CG1	SETC	''		
	AIF	('&NAME' EQ '').GEN		
.*			CHECK IF ANOTHER SUBROUTINE CAN BE GENERATED	
	AIF	('&CG11' NE '').S		
&CG11	SETC	'&NAME'		
	AGO	.SETCG1		
.S	AIF	('&CG12' NE '').T		
&CG12	SETC	'&NAME'		
	AGO	.SETCG1		
.T	AIF	('&CG13' NE '').U		
&CG13	SETC	'&NAME'		
	AGO	.SETCG1		
.U	AIF	('&CG14' NE '').V		
&CG14	SETC	'&NAME'		

* A standard card form, IBM electro 6509, is available for punching source statements from this form.
Instructions for using this form are in any IBM System/360 Assembler Reference Manual.
Address comments concerning this form to IBM Corporation, Programming Publications, Department 232, San Jose, California 95114.

Figure 13. Coding of the GMOVE Macro Instruction, Part 1 of 3

IBM System/360 Assembler Coding Form

X28-6509-4 U/M025
Printed in U.S.A.

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 3 OF 5 | CARD ELECTRO NUMBER *

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
	AGO	.SETCG1		
.V	AIF	('&CG1S' NE '').ALLSUB		
&CG15	SETC	'&NAME'		
.*		GENERATE LINKAGE FOR FIRST OCCURRENCE		
.SETCG1	LH	9,E&NAME		
&CG1	SETC	'&NAME'		
&BL2	SETB	(1)		
*		START OF GMOVE SUBROUTINE		
	AGO	.GEN		
.ALLSUB	MNOTE	'NO FURTHER SUBROUTINES IN THIS CONTROL SECTION'		
	MNOTE	'GENERATION PROCEDES WITHOUT SUBROUTINE'		
.*		GENERATION OF ACTUAL MOVE INSTRUCTIONS		
.GEN	ANOP			
&AL3	SETA	&LF1		
&AL4	SETA	2		
.NEXT	ANOP			
&AL1	SETA	0		
	AIF	(&AL3 EQ 0).ERRLGTH		
	AIF	('&SYSLIST(&AL4)' EQ '0').SETNEXT		
.*		ABOVE STATEMENT RESERVES SPACE IN DESTINATION		
.*		FOR EACH SOURCE ENTERED AS ZERO		
.LOOP	ANOP			
&AL2	SETA	&AL3-&AL1		

* A standard card form, IBM electro 6509, is available for punching source statements from this form.
Instructions for using this form are in any IBM System/360 Assembler Reference Manual.
Address comments concerning this form to IBM Corporation, Programming Publications, Department 232, San Jose, California 95114.

IBM System/360 Assembler Coding Form

X28-6509-4 U/M025
Printed in U.S.A.

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 4 OF 5 | CARD ELECTRO NUMBER *

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
&AL6	SETA	&AL5+&AL1		
	AIF	(&AL2 LE 256).LSTMOVE		
&CG1	MVC	&TO+&AL6.(256),&SYSLIST(&AL4)+&AL1		
&AL1	SETA	&AL1+256		
&CG1	SETC	''		
	AGOB	.LOOP		
.LSTMOVE	ANOP			
&CG1	MVC	&TO+&AL6.(&AL2),&SYSLIST(&AL4)+&AL1		
&CG1	SETC	''		
.*		ALL MOVES GENERATED FOR THIS SOURCE		
.*				
.*		CHECK FOR PRESENCE OF NEXT SOURCE		
.SETNEXT	ANOP			
&AL4	SETA	&AL4+2		
	AIF	('&SYSLIST(&AL4)' EQ '').LAST		
&AL7	SETA	&AL4+1		
	AIF	('&SYSLIST(&AL7)' NE '').SETUP		
.ERRLGTH	MNOTE	'MISSING OR INVALID LENGTH FOR SOURCE &SYSLIST(&AL4)'		
	MNOTE	'NO FURTHER SOURCES PROCESSED'		
	AGO	.LAST		
.*		SET UP FOR GENERATION OF NEXT SOURCE		
.SETUP	ANOP			
&AL5	SETA	&AL5+&AL3		
&AL3	SETA	&SYSLIST(&AL7)		
	AGOB	.NEXT		

* A standard card form, IBM electro 6509, is available for punching source statements from this form.
Instructions for using this form are in any IBM System/360 Assembler Reference Manual.
Address comments concerning this form to IBM Corporation, Programming Publications, Department 232, San Jose, California 95114.

Figure 13. Coding of the GMOVE Macro-Instruction, Part 2 of 3

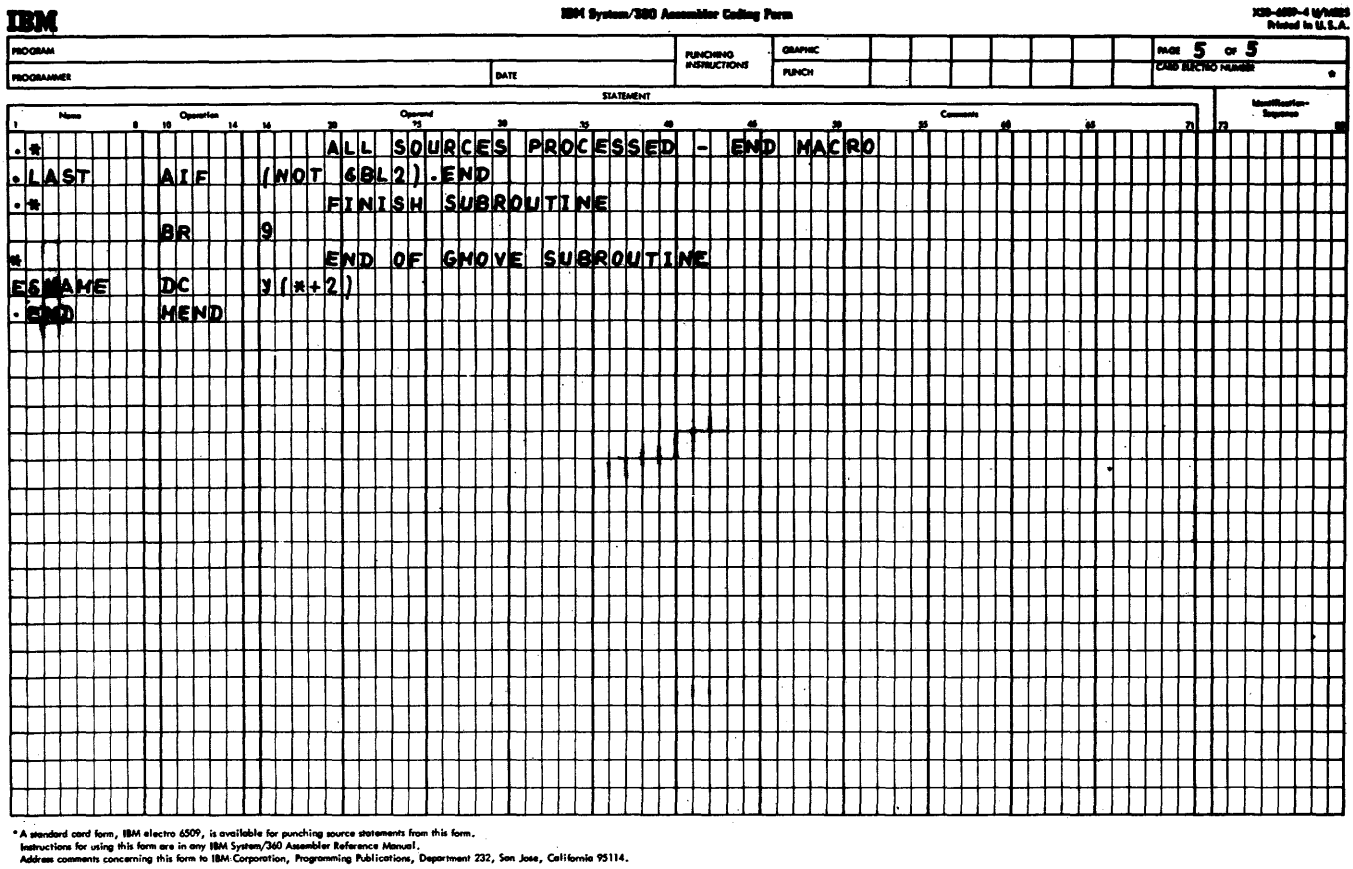


Figure 13. Coding of the GMOVE Macro-Instruction, Part 3 of 3

Assembly of a Program (DPS/TPS)

DPS only:

Assembler source programs can be either assembled and executed in one job (assemble-and-execute function) or assembled and executed in separate jobs. In an assemble-and-execute job, the object program (in addition to being executed immediately) may be punched into cards or written onto magnetic tape.

When the assemble-and-execute function is to be used, the following conditions must be fulfilled:

- The program must not require linking and/or relocation.
- The Core-Image Maintenance program must be contained in the core-image library of the disk-resident system.
- A relocatable area must be available in the disk-resident system.

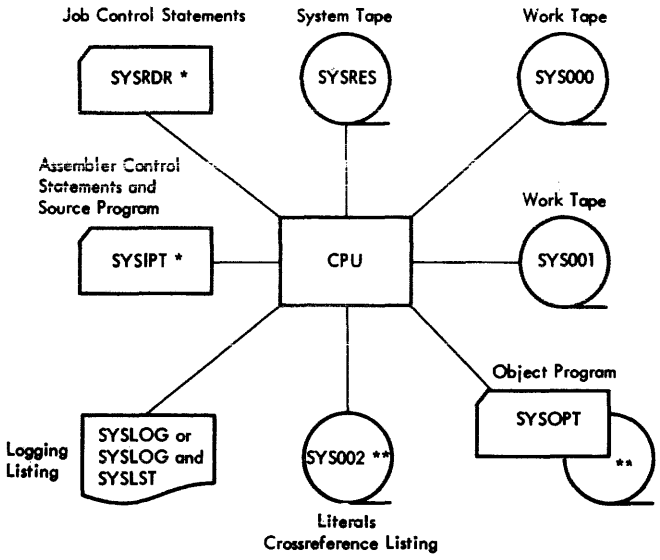
Object programs that require linking and/or relocation must be processed by the Linkage Editor program before they can be executed and/or included in the core-image library.

DPS/TPS:

Assembler object programs that do not require linking and/or relocation can be executed directly under control of the disk-resident (tape-resident) system. Use the execute-loader function for this purpose. They can also be included in the core-image library of the disk-resident (tape-resident) system, from where they can then be loaded and executed.

The files and corresponding input/output devices used for a program assembly are as shown in Figures 14 and 15.

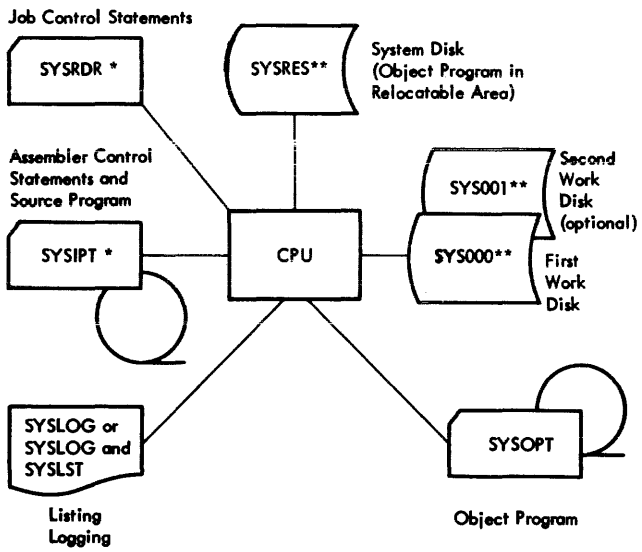
Note that for TPS the literal Workfile and Text Output file may be assigned to the same tape unit.



* The same unit may be used for both reading control cards and the source program.

** The same tape drive may be used for both object-program output, literals, and Crossreference Listing.

Figure 14. Input/Output Devices Used for a Program Assembly with the TPS Assembler



* The same unit may be used for reading control cards and the source program.

** The same disk drive may be used for SYSRES and SYS000 or SYSRES and SYS001 but not for SYS00 and SYS001.

Figure 15. Input/Output Devices Used for a Program Assembly with the DPS Assembler

Job Control Statements

Device assignments are normally given at system generation time. If, however, these assignments were not given at that time or if you wish to alter any of the assignments, ASSGN job-control statements as shown below may be used to make the desired assignments.

// JOB ASSEMB
or
// JOB ASSEMB,p.-name
	Required if the source program is to be assembled only. Required if the assemble-and-execute function is to be used. (DPS only).

// DATE ...	Required if the assembly is the first job in the system run.
// ASSGN SYSLOG,...	Optional. Refers to the printer, which lists job control statements if you include a LOG control statement.
// ASSGN SYSLST,...	Required. Refers to the printer, which prints the program listing and other information.
// ASSGN SYSIPT,...	Required. Refers to the card reading device (or magnetic tape drive) (DPS only) on which the program control statements (if any), the Assembler source program, and (if card input) an end-of-file card are read.
// ASSGN SYSOPT,...	Required if an object program is to be produced in cards or on magnetic tape. Refers to the card punching device or magnetic tape drive on which the object program is to be produced.
// ASSGN SYS000,...	Required. Refers to the disk (tape) drive containing workfile1.
// ASSGN SYS001,...	Required. TPS: Required. DPS: Required if two workfiles are used see (AWORK). Refers to the disk (tape) drive containing workfile2.
// ASSGN SYS002,...	TPS Optional. For literals. Maybe same as for SYSOPT.

// VOL SYS000,WORK1
// DLAB ...
// XTENT ...
	DPS
	Required for workfile 1.

// VOL SYS001,WORK2
// DLAB ...
// XTENT ...
	Required if two workfiles are used. (See AWORK).

// EXEC	Required.

Program Control Statements

Program control statements are supplied for use by the Assembler program. These statements indicate which of the Assembler processing options the Assembler program is to perform or provide. The Assembler control statements are:

- AWORK - Assembler Work File statement,
- AOPTN - Assembler Option statement,
- ICTL - Input Format Control statement,
- ISEQ - Input Sequence Checking statement

The AWORK statement is used for the disk-resident Assembler only. All four control statements have the same format as Assembler language instructions. AOPTN, AWORK, ICTL, and ISEQ appear in the operation field, while the various options are specified as operands.

AWORK -- ASSEMBLER WORKFILE STATEMENT

An AWORK statement indicates the number of work areas the DPS Assembler is to use in processing source-program statements. The Assembler can use either one or two work areas. Two work areas are provided only if two disk drives are available. In this case, one work area can be assigned to each disk drive. Using two work areas on separate disk drives shortens the processing time required by the Assembler.

The format of the AWORK statement is:

Name	Operation	Operand
	AWORK	n

The AWORK statement requires one operand(n) which must be either the digit 1 or the digit 2. If the digit 1 appears as the operand, the Assembler assumes one work area. If the digit 2 is used, the Assembler assumes two work areas. If no AWORK statement is provided, the Assembler assumes one work area.

A work area must always be assigned to a disk area consisting of contiguous storage positions. This is accomplished by using the proper job-control (XTENT) statements. For details concerning XTENT statements, refer to the SRL publication IBM System/360 Model 20, Disk Programming System, Control and Service Programs, Form GC24-9006.

The AWORK statement must precede any AOPTN statements used and the source statements.

AOPTN (ASSEMBLER OPTION) STATEMENTS

AOPTN statement(s) may be used if the normal Assembler output is to be altered. AOPTN statement(s), must be written preceding any other source-program statements, even an ICTL statement. Figure 16 shows the option indicators that can be used.

The normal Assembler output consists of two major files: the object program and program listing. The object program consists of three types of information: the External Symbol Dictionary (ESD), Text (TXT), and the Relocation Dictionary (RLD). The program listing consists of five lists of information: ESD listing, source and object program listing, RLD listing, error listing, and symbol table.

The format of the AOPTN statement is:

Name	Operation	Operand
	AOPTN	option symbol(s)

The option(s) that may be supplied in the AOPTN statement are shown in Figure 16; each option is identified by a symbol which is used in the AOPTN statement. Each option of the AOPTN statement may be specified in a different AOPTN statement, or they may appear as multiple operands (separated by commas) in a single statement.

OPTION SYMBOL	MEANING
NODECK	Object program (ESD, TXT, and RLD data) not produced in cards or tape. (Their appearance in the program listing is not affected.)
NOESD	No ESD data will appear in the object program or the program listing.
NORLD	No RLD data will appear in the object program or the program listing. Thus, the object program will be absolute.
NOLIST	The program listing will not appear. (A statement indicating the number of errors in the program will, however, be printed.)
NOERR	The error listing will not appear in the program listing. (A statement indicating the number of errors will, however, be printed.)
NOSYM	Neither a Table of Defined Symbols nor a Crossreference List will appear in the program listing.
NOVERIFY*	Intermediate write operations on disk are not verified.
CROSSREF*	A cross-reference listing will appear instead of the symbol table listing. The cross-reference listing contains all the symbols used in the program and the number of the statement in which they were used (see Appendix G). Note: The total number of symbol definitions and references must not exceed 12,288.
LITERAL**	Literals will be processed. If LITERAL is omitted, literals will not be processed. Note: The user must be sure he has enough tape units for literal processing.
ENTRY	An ENTRY card will be produced at the end of the output text.

* For the disk-resident Assembler only.

** For the tape-resident Assembler only. (Not required for DPS.)

Figure 16. AOPTN Card Option Indicators

Note: No object deck will be produced in case of the LIMIT EXCEEDED error regardless of the option specified. (See Appendix H. Assembler Diagnostic Messages). However, a listing will appear.

The AOPTN statements must precede the source statements and follow the AWORK statement, if any.

ICTL -- INPUT FORMAT CONTROL

The ICTL instruction allows you to alter the normal format of your program statements. An ICTL instruction statement, if any, must precede all other statements in the source program except the AWORK and AOPTN statements, if any, and must not be used more than once. Only the following two formats of the ICTL instruction are allowed:

either:

Name	Operation	Operand
Blank	ICTL	25

or:

Name	Operation	Operand
Blank	ICTL	25,71,38

In the first case, the operand specifies that the begin column of the coding format is 25. Since the end column is not specified, it is assumed to be column 71. No continuation lines are permitted.

In the second case, the begin column is column 25, the end column is column 71, and the continue column for macro instructions is column 38.

Non-standard operand specifications other than the two above are not allowed.

ISEQ -- INPUT SEQUENCE CHECKING

The ISEQ instruction is used to check the sequence of input cards. The format of the ISEQ instruction statement is as follows:

Name	Operation	Operand
Blank	ISEQ	Two decimal values of the form l,r; or blank

The operands l and r, respectively, specify the leftmost and rightmost columns of the field in the input cards to be checked. Operand r must be equal to, or greater than, operand l. Operand l must be greater than 72. The field specified by operands l and r must not be greater than seven bytes.

Sequence checking begins with the first card following the ISEQ statement. Comparison of adjacent cards makes use of the eight-bit internal collating sequence (see Appendix J). The input cards are said to be in sequence if the value in the columns checked in the second card is greater than or equal to that of the first card. If a statement is found to be out of sequence a warning is given but no error message appears in the diagnostics- listing.

An ISEQ statement with a blank operand terminates the checking operation. The next ISEQ statement initiates a new check.

Sequence checking is only performed on statements contained in the source program. Statements generated by a macro instruction are not checked for sequence (see Appendix J).

Cataloging a Macro Definition

The macro library is maintained by the Macro Maintenance Program (MMAINT). Use this program to add or delete any number of macro definitions to or from the macro library.

Job Control Statements (DPS)

The job control statements required for a MMAINT run are:

// JOB MMAINT	Required.
// DATE ...	Required if first job after IPL.
// ASSGN SYSLOG,...	Optional.
// ASSGN SYSLST,...	Optional.
// ASSGN SYSIPT,...	Required.
// VOL SYSIPT,SYSIF	Required only if SYSIPT refers to a disk drive.
// DLAB ...	Required only if SYSIPT refers to a disk drive.
// XTENT ...	Required only if SYSIPT refers to a disk drive.
// EXEC	Required.

Job Control Statements (TPS)

The job control statements required for an MMAINT run are:

// JOB MMAINT	Required.
// DATE ...	Required if first job after IPL.
// ASSGN SYSLOG,...	Optional.
// ASSGN SYSLST,...	Optional.
// ASSGN SYSIPT,...	Required.
// ASSGN SYS000	Work tape.
// EXEC	Required.

Program Control Statements

The program control statements required to catalog a macro definition in source format are:

```
// CATAL
// END
```

You may use any number of macro definitions in a MMAINT run, but each macro definition must be preceded by a // CATAL statement and the last definition must be followed by the // END statement.

For other (optional) program control statements see the SRL publications IBM System/360 Model 20, Disk Programming System, Control and Service Programs, Form GC24-9006, or IBM System/360 Model 20, Tape Programming System, Control and Service Program, Form GC24-9000.

If listing is specified, a list is printed during the assembly run. The form of the lists and the type of information contained therein are shown in Appendix G.

Source Program and Data Checking (Assembler)

Diagnostic messages are printed if incorrectly coded instructions are detected by the Assembler program during an assembly run. Both the number of the related source-program statement and the action taken by the Assembler program are printed together with each message. If more than one error is detected within a source-program statement, the diagnostic messages for all errors detected are listed. The Assembler program then takes the action for the severest error detected, and the appropriate action is printed with the diagnostic messages.

Notification of the types of Assembler actions that may occur during an assembly run are listed below in the order of their severity with the severest type first:

ASSEMBLY IN ERROR (AIE)

The user's program cannot be executed. The assembly run is completed, but only diagnostic messages preceding the AIE message may be considered valid.

STATEMENT TREATED AS COMMENT (STC)

STATEMENT INCOMPLETELY ASSEMBLED (SIA)

If the statement is a machine instruction, the storage area required for the instruction is reserved and filled with zeros.

STATEMENT ASSEMBLED (SA)

The individual diagnostic messages are listed in Appendix H.

Source Program and Data Checking (MMAINT)

A diagnostic message is added to an incorrect statement. Only one error can be detected within a statement.

If an error has been detected within a macro definition, sequence symbols are not processed and the macro definition is not catalogued. If an error has been detected within a prototype statement, the macro definition is not checked for further errors.

The TPS macro maintenance program performs a check to determine whether the contents of columns 73 through 80 of a statement are out of sequence. In case of a sequence error, the statement is flagged with the letter S. If no other errors are contained in the macro definition, it is catalogued.

The individual diagnostic messages are listed in Appendix I.

Language Compatibility

The IBM System/360 Model 20 DPS/TPS Assembler and macro Languages are closely patterned after the Basic Programming Support (BPS) and Disk and Tape Operating Systems (DOS/TOS) Assembler and macro languages except where differences in machine design require specific instructions for Model 20. These differences are as follows:

- There are seven Model 20 machine instructions that are not contained in the instructions sets of higher System/360 models:
  CIO, XIO, TIOB, SPSW, BAS, BASR, HPR.

  If programs that were written for the Model 20 are to be executed on higher System/360 models, the use of these instructions can be avoided as follows:

  a. CIO, XIO, TIOB, and SPSW instructions may be avoided by using IOCS macro instructions for input/output operations.

  b. BAS and BASR instructions must be replaced by BAL and BALR, respectively, with consideration given to the functional differences between Model 20 and higher System/360 models.

  c. HPR instructions must be replaced by branch instructions.

- The Model 20 Program Status Word (PSW) has a length of only four bytes.

- The Model 20 Channel Command Word has a length of only 6 bytes and must be aligned at a half-word boundary. The Model 20 Assembler instruction is DCCW instead of CCW. The use of the DCCW instruction may be avoided by using IOCS.

- The number and length of Model 20 registers differ from higher System/360 models.

  a. Model 20 has only eight registers (8-15). In addition, it has eight pseudo base registers (0-7) with fixed contents. The pseudo base registers contain 0, 4096, 8192, 12288, 16384, 20480, 24576, and 28672, respectively, thus permitting direct addressing.

  b. Model 20 registers have a length of only 2 bytes.

Note: Y-type address constants two bytes in length must not be specified for System/360 models having a storage capacity of more than 65535 bytes. No Y-type constants must be used in programs to be executed under control of the Operating System/360.

Programs written in the Model 20 Basic Assembler Language can be assembled by the Model 20 DPS/TPS Assembler program unless blank operands are used in Assembler or machine instructions.

The instructions LCLA, LCLB, GBLA, GBLB, and GBLC which are discussed in the Disk and Tape Operating Systems, Assembler Specifications are ignored. The use of LCLC is illegal because local SETC symbols are not allowed.

The relationships between the individual Assembler languages are as shown in Appendix F.

Glossary

Absolute Address:
1. An address that is permanently assigned by the machine designer to a storage location.
2. A pattern of characters that identifies a unique storage location without further modification.

Access Method: Any of the data management techniques available to the user for transferring data between main storage and an input/output device.

Access Time:
1. The time interval between the instant at which data is called for from a storage device and the instant delivery is completed, i.e., the read time.
2. The time interval between the instant at which data is requested to be stored and the instant at which storage is completed, i.e., the write time.

Address:
1. An identification (name, label, or number) for a register, location in main storage, or any other data source.
2. Any part of an instruction that specifies the location of an operand for the instruction.
3. (v.t.) In BSCA IOCS a technique by which the CPU prepares a remote station to receive a message.

Address Constant: A value, or an expression representing a value, interpreted as a main storage address.

Allocate: To assign storage locations or areas of storage for a specific job.

Allocated Variable: A variable with which storage has been associated.

Alphabetic Character: Any of the characters #, $, @, and the characters of the alphabet (A through Z).

Alternate Drive: When two drives are given for one multi-volume file, the first drive is the primary drive and the second drive is the alternate drive. Tape reels or disk packs are mounted such that the first is on the primary drive, the second on the alternate drive, the third on the primary drive, etc.

Arithmetic Data: Numeric values used in arithmetic operations (add, subtract, multiply, and divide).

Arithmetic Operators: Any of the prefix operators (+ and -) or the infix operators (+, -, *, /, and **).

+ = plus	/ = divide
- = minus	** = raise to a power
* = multiply	

Ascending Order: A sequence of records such that the control fields of each successive record collate equal to or higher than those of the preceding record.

Ascending Sequence: See Ascending Order.

ASCII: See USASCII.

Assemble: To prepare a machine-language program from a symbolic-language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

Assemble-and-Execute: A job setup which provides for an assembly of a source program followed immediately by the execution of the assembled program.

Assemble-and-Go: See Assemble-and-Execute.

Assembler: A program that assembles.

Assembler Language: A symbolic language (used to write source programs) which enables the programmer to use all machine functions as if he were coding in machine language.

Attribute: A characteristic; for example, attributes of data include record length, record format, data file name, associated device type and volume identification, use, creation date, etc.

Backup and Restore Program (BACKUP): A DPS service program that can be used to

1. create a backup tape from one or more disk files and one or more card files,
2. create a backup disk from one or more disk files, and
3. restore each backup file to its original medium.
4. Change the volume and file serial numbers.

Base:
1. A reference value.
2. A number that is multiplied by itself as many times as indicated by an exponent.

3. the number system in terms of which a value is represented. Examples: the decimal base (ten), the binary base (two), the hexadecimal base (sixteen).

Base Address: A given address from which an effective address is derived by combination with a relative address. (See Displacement).

Binary:
1. A characteristic or property involving a selection, choice, or condition in which there are two possibilities.
2. The numeration system with a radix of two.

Binary Coded Decimal: A decimal notation in which the individual decimal digits are each represented by a group of binary digits, e.g., in the 8-4-2-1 binary coded decimal notation, the number twenty-three is represented as 0010 0011 whereas, in binary notation, twenty-three is represented as 10111.

Binary Digit: The smallest unit of information. It can have either of the two binary values zero or one.

Binary Search: A search in which a set of items is divided into two parts, where one part is rejected, and the process is repeated on the accepted part until the item with the desired property is found.

Bit: See Binary Digit.

Buffer: An area in main storage used as intermediate storage in I/O operations. During input, data is read into a buffer; during output, data is written from a buffer.

Byte: The smallest addressable unit of information in System/360. Every byte consists of eight bits, each having a value of zero or one.

Card-Resident System: Consists of the card control programs: Initial Program Loader, (Basic) Monitor, and Job Control. Used for the execution of object programs contained in punched cards.

Catalog: (v.t.) The action of including an object program or program phase in the core-image library as a temporary or a permanent entry.

Chaining File: See Chaining.

Chaining: A record retrieval technique. The control information contained in records of one (the chaining) file is used to access a record in another (the chained) file. The chained file must be organized indexed-sequentially.

Character:
1. One of a set of elementary signals which may include decimal digits 0 through 9, the letters A through Z, punctuation marks and any other symbols acceptable to a computer for reading, writing, or storing.
2. An 8-bit (1-byte) code that can be manipulated in main storage.

Character Set: An ordered set of unique representation called characters.

CMAINT: See Core-Image Maintenance Program.

Code:
1. (v.t.) To represent data or machine instructions in a symbolic form that can be accepted by an appropriate processor program.
2. Machine instructions produced on the bases of coded instructions (see Object program).

Collating Sequence: The relative order of characters on which a sort or merge is based.

Comment: A string of characters used for documentation.

Communication Region: An area of the (Basic) Monitor. Contains date, storage-capacity specification, UPSI byte, user areas 1 and 2, program-name area, and various control bits used by the system. Provides for intra-program and inter-program communication.

Compile-and-Execute: A job setup which provides for a compilation of a source program followed immediately by the execution of the compiled program.

Compile-and-Go: See Compile and Execute.

Compiler: A program which translates a program written in a problem-oriented (RPG, PL/I, etc.) language into object code.

Compiler Control Statement: Any one of the control statements in the input stream that defines the requirements and options of a job to the compiler.

Concatenation: The operation that connects two character strings in the order indicated to form one string whose length is equal to the sum of the lengths of the two strings.

Control Programs: A set of programs which provide the management functions necessary for continuous operation of a computing system.

Control Section: The smallest unit of a program that can be separately assembled or compiled. All elements of a control section are in constant relationship to each other.

Control Statement: Any of the statements in the input to a specific job that define the requirements of the job or its options.

Conversion: The process of changing from one form of representation to another.

Copy System Disk Program (COPSYS): A DPS service program used to copy the system file stored on the system disk pack onto another disk pack.

Copy System Tape Program: A TPS service program contained in punched cards. Copies user's tape-resident system from one tape onto another.

Core-Image Directory: A table on the system disk pack used as directory to the program (core-image) library. Each directory entry contains information about a program phase and its location in the library.

Core-Image Format: A data format identical to that used in main storage. Programs or program phases stored in the core-image library constitute data in core-image format. Such programs or program phases are ready for direct loading from the core-image library into main storage.

Core-Image Library: An external-storage area containing the Job Control program, other IBM-supplied programs (except the [Basic] Monitor and the IPL), and user's problem programs. Permits retrieval of programs or program phase by the Monitor.

Core-Image Maintenance Program: A system service program. Updates the core-image library and directory. Is used to add and/or replace and/or delete phases and the (Basic) Monitor.

Core-Image Program: A system service program that permits the printing, writing and/or punching of one or more entries of the core-image library.

CPSYS: See Copy System Tape Program.

CSERV: See Core-Image Service Program.

Data:
1. A representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or by automatic means.
2. Any representations such as characters to which meaning is, or might be, assigned.

Data File: A collection of related records treated as a unit consisting of data in one of several prescribed arrangements and described by control information to which the system has access.

Data Format Item: Specifications in the program that describe data items in the stream. Such data items may be characters or arithmetic values in character form.

Data Item: A single unit of data.

Data Set: See data file.

Data Transmission: The sending of data from an external storage device to main storage and vice-versa.

Decimal: The number system based on the value 10.

Decimal Digit: One of the characters 0 through 9 in a decimal number. For example, in the number 567, each of the numeric characters 5, 6, and 7 is a decimal digit.

Decimal Point: The radix point in decimal representation.

Delimiter: Any valid special character or combination of special characters used to separate items of data, such as identifiers, constants, and statements.

Descending Order: A sequence of records such that the control fields of each successive record collate equal to or lower than those of the preceding record.

Descending Sequence: See Descending Order.

Device Address: See Physical Device Address and Symbolic Device Address.

Device Independence: The ability to request input/output operations without regard to the characteristics of the input/output devices.

Direct Access: Retrieval or storage of data by a reference to its location on a volume rather than relative to the previously retrieved or stored data.

Direct Address: An address that specifies the location of an operand without need of modification such as adding a base address value.

Directory: See Core-Image Directory or Macro Directory

Directory Entry: A unit of information in the core-image or macro directory. (Phase header or macro identifier.)

Directory Service Program: A system service program. Causes printing of the core-image and/or macro directory and/or system directory.

Disk-Resident System: Contains the Monitor, the disk-resident portion of the IPL, and the Job Control program. May contain any one or a combination of the following: IBM-supplied or user-written programs, macro definitions, and a relocatable area.

Displacement: A value, or an expression representing a value, which is added to a base address to obtain the effective address.

DPS Control Programs: A collective term used to refer to the Initial Program Loader, the Monitor, and the Job Control program.

DSERV: See Directory Service Program.

Dump:
1. (v.t.) To copy the contents of all or part of main storage or an external storage onto an output device, so that it can be examined.
2. (n.) The data resulting from 1.
3. (n.) A routine that will accomplish 1.

EBCDIC: (Extended Binary Coded Decimal Interchange Code) A specific set of eight-bit codes standard throughout System/360.

Edit: To modify the form or format of data, e.g., to insert characters such as page numbers or decimal points or to delete characters such as leading zeros.

Edit Pattern: A field composed of characters of a special significance. These characters control such editing functions as zero suppression, insertion of a floating dollar sign, etc.

Effective Address: The absolute address that is derived by applying any specified indexing (base address value) or indirect addressing rules to the specified address. The derived effective address is actually used to identify the current operand.

Entry Name: The symbolic address of an entry point.

Entry Point: In a routine, any place to which control can be passed.

EOF Card: End-of-file card which signals the end of a logical set of input cards (/*b in columns 1-3, where b = blank).

EOF Record: End-of-file record which signals the end of a logical set of input records (/*b in columns 1-3, where b = blank).

ESD: See External Symbol Dictionary.

ESID: See External Symbol Identification.

Exceptional Condition: An occurrence which can cause a program interrupt or an unexpected situation such as an overflow error, or an occurrence of an expected situation such as an end-of-file condition that occurs at an unpredictable time.

EXEC statement: See Execute statement.

Executable Object Program: The set of machine instructions produced by a language translater and prepared for loading into main storage either by link-editing or by a a CMAINT run if an installation uses a disk-resident system; the set of machine instructions produced by a language translator without further preparation if an installation uses a card-resident system.

Execute: (v.t.) To carry out an instruction or perform a routine.

Execute-Loader Function: The function of executing an object program that is not cataloged in the core-image library. The object program may be read from either a card-reading device or a tape drive or from the relocatable area.

Execute Statement: A Job Control statement that designates a job by identifying the load module to be fetched and executed.

Explicit addressing: An addressing technique which requires the specification of all elements of on address (base and displacement) by means of absolute values.

Exponent: In a floating-point constant a decimal integer that specifies the power to which the base of the floating-point constant is to be raised.

Expression: An operand entry that consists of a single term or an arithmetic combination of terms, normally representing an address value.

External Storage: A storage device outside the computer capable of storing information in a form acceptable to the computer; for example, cards or magnetic tapes.

External Symbol: A control section name, entry point name, or external reference; a symbol contained in the external symbol dictionary.

External Symbol Dictionary (ESD): Control information associated with an object or load module which identifies the external symbols in the module.

External Symbol Identification (ESID): ESID numbers are assembler-assigned pointers that are used by the Linkage Editor program to correctly recompute the address constants referred to in RLD entries.

Feature: A function of a program, or a particular circuitry in a system device, that can be used to perform specific operations.

Fetch:
1. (v.t.) To read into main storage and pass control to phases or subphases.
2. (n.) The name of a control routine of the (Basic) Monitor that accomplishes 1.

Field: In a record or in a data stream, a specified area used for a particular category of data, for example, a number of character positions used to represent a wage rate or a number of bytes in main storage used to express the address of data in main storage.

File: See Data File.

Fixed-Point: Pertaining to a number system in which the location of the (decimal) point is fixed with respect to one end of the numerals according to some convention.

Flag: Any of various types of indicators used for identification, normally a bit.

Floating Point: Pertaining to a numeration system in which the position of the point does not remain fixed with respect to one end of the numerals.

Format: The general makeup of data, a control statement, or a record.

Graphic: The visual representation of a character or symbol.

Half-Byte: The leftmost or rightmost four bits of an eight-bit byte. Can contain representation of a digit or the sign of a number.

Halfword: Two adjacent bytes where the left byte is on a halfword boundary.

Halfword Boundary: Any even-numbered addressable byte position in main storage.

Hexadecimal: A number system using the equivalent of the decimal number 16 as a base. The values 0-15 are represented by the digits 0-9 and the aphabetic characters A-F.

High-Order Digit: Leftmost digit of a decimal number.

Identifier: A symbol whose purpose is to identify, indicata, or name a body of data.

Implicit Addressing: An addressing technique that allows the specification of symbol addresses.

Index Register: A register whose content is added to the operand address prior to or during the execution of an instruction.

Indirect Address: An address that specifies a storage location which contains either a direct address or another indirect address.

Infix Operator: An operator that defines an operation between two operands.

Initialize: To set counters, switches, and addresses to zero, blank, or other starting values at the beginning of, or at prescribed points in, a computer routine.

Initial Program Loader: A system control program. Loads (Basic) Monitor into main storage. Is used to assign physical I/O device addresses to symbolic addresses SYSRDR and/or SYSRES. Places name of Job Control program into communication region of (Basic) Monitor. The program must be executed at the beginning of a system run.

Initial Value: A value placed into a register or a storage area at the beginning of an operation and used during the operation for count purposes or control purposes or both.

Inner Macro Instruction: A macro instruction contained in a macro definition.

Input: The transfer of data from an external storage device to main storage.

Input Job Stream: A sequence of Job-Control statements entering the system, which may also include input data.

Input/Output Control System: A group of macro definitions which are contained in the macro library of the programming system. These macro definitions can be retrieved from the library and tailored to the input/output requirements of the user.

Inquiry Program: A program whose execution is initiated by an inquiry request on the printer-keyboard attached to the Model 20 system. When such a request is made, routines of the Monitor cause the current contents of main storage to be rolled out, the inquiry program to be loaded and executed and, on execution of that program, the original contents to be rolled in again.

Installation: A particular computing system in the context of the overall function it serves and the individuals who manage it, operate it, apply it to problems, service it, and use the results it produces.

Integer Digit: A digit to the left of the decimal point.

Inter-Program Communication: The exchange of data between two or more programs.

Interrupt: (v.t.) To stop a process in such a way that it can be resumed.

Intra-Program Communication: The exchange of data between two or more phases of a multi-phase program. Facilitated by the communication region.

I/O: Input or output or both.

I/O Area: An area (portion) of main storage into which data is read or from which data is written.

I/O Overlap: A system feature that permits an input/output operation to be performed simultaneously with other I/O operations or with processing or both.

I/O Time: the time interval between the instant at which data is called for from an external storage device and the instant delivery is completed (read time); the time interval between the instant at which data is requested to be stored in an external storage device and the instant at which storage is completed (write time).

IOSC: See Input/Output Control System.

IPL: See Initial Program Loader.

Job: An externally specified unit of work for the computing system from the standpoint of installation accounting and operating system control.

Job Control Program: A system control program. Resides in main storage between jobs and provides for automatic job-to-job transition. Processes Job Control statements in the input stream.

Job Control Statement: Any one of the control statements (in the input stream) that identify a job or define its requirements and options.

Job Stream: See Input Job Stream.

K Bytes:
1024 bytes. For example: nK = n x 1024 bytes.

Key: One or more characters within an item of data used to identify that data or to control its use.

Keyword: A mnemonic in a keyword macro instruction.

Keyword Macro Instruction: A macro instruction whose operands must each consist of a mnemonic (keyword), an equal sign, and a specification. The operands need not be in a predetermined order.

Language Translator: A program or a routine that accepts statements in one language and produces equivalent statements in another language such as an assembler or a compiler.

LDSYS: See Load System Program.

Library: A collection of objects (e.g., files, volumes, card decks) associated with a particular use, and the location of which is identified in a directory of some type.

Library Allocation Organization Program: A system service program. Used to redefine the limits of one or a combination of the following: core-image library, core-image directory, macro library, macro directory, and relocatable area.

Library Management Program: System service programs such as Core-Image Maintenance, Macro Maintenance, Directory Service, and Library Allocation Organization programs.

Library Work Area: An area on the system disk pack used by the CMAINT program for updating the Monitor or the IPL. In assemble-and-go or compile-and-go runs, the CMAINT program uses this area for the storing of tape label information.

Linkage: Machine instructions that connect separately assembled control sections.

Linkage Editor: A system service program. Relocates programs or phases and links separately assembled programs or phases.

Link-Editing: The function of combining a program control section with one or more other, separately assembled program control sections into one executable object program.

Load:
1. To read a program or a program phase into main storage.
2. To initially write a data file onto disk.

Load System Disk Program: A system service program that creates a tape- or disk-resident system from card input.

Logical Unit Block: An entry in the Logical Unit Table.

Logical Unit Table: Part of the (Basic) Monitor. It has logical unit blocks, each of which refers to one specific symbolic I/O address and contains the address of a physical unit block. These symbolic addresses are related to physical I/O device addresses by means of ASSGN control statements.

Loop: A sequence of instructions that is executed repeatedly a specified number of times or until a condition is brought about that ends this repeated execution.

Low-Order Digit: The rightmost digit of a decimal number.

LUB Table: See Logical Unit Table.

Machine Instruction: An Assembler-language statement, or its functional equivalent in machine language, that instructs the computing system to perform one specific operation, such as add, subtract, compare, etc.

Macro Definition: A set of statements in the macro library used by the DPS/TPS Assembler program to expand a macro instruction specified in the source program into a series of machine instructions.

Macro Directory: An area of the macro library section of a tape-resident system, a table on the system disk pack of a disk-resident system. Is used with programs written in the Assembler language.
The TPS version has four priority sections, each of which contains the identifiers for the macro definitions contained in the corresponding section of the macro library.
The DPS version lists the names, begin addresses, and lengths of macro definitions contained in the macro library. Is used with programs written in the Assembler language.
The Macro directory can be listed on a printer by means of the Directory Service program.

Macro Instruction: A statement used in a source program and replaced by a specific sequence of machine instructions in the associated object program.

Macro Library (DPS): A disk area containing the macro definitions for the macro instructions issued in user-written programs. Contains source statements needed to generate frequently used routines.

Macro Library (TPS): An area of the macro library section of the system tape. Has four priority sections, each of which contains the macro definitions for the macro instructions in user programs. Contains source statements needed to generate frequently used routines.

Macro Maintenance Program: A system service program. Updates the macro library and directory. Is used to add and/or delete macro definitions.

Macro Name: An entry in the macro directory that identifies and points to the corresponding macro definition in the macro library.

Macro Service Program: A system service program that permits the printing, punching, and/or writing of one or more macro definitions from the macro library.

Main Storage: All addressable internal storage of the CPU (central processing unit). It holds the program(s) under whose control internal manipulation of data is performed.

MMAINT: See Macro Maintenance Program.

Mnemonic: A contraction or abbreviation whose characters are suggestive of the full expression.

Model Statement: Any one of the statements in a macro definition that may be selected and/or altered (usually according to the operands specified in the macro instruction) and become part of the code generated into the source program.

Monitor: The main control program in DPS. Resident in main storage throughout a system run. Loads programs into main storage and causes their execution.

Monitor I/O Area: An area of main storage within the Monitor used as a buffer by various Monitor routines when they read data into main storage or transfer data to an output device.

MSERV: See Macro Service Program.

Name: A set of one or more characters that identifies a statement, file, module, etc., and that is usually associated with the location of that which it identifies.

Nesting: The occurrence of a macro instruction in a macro definition.

Object Program: The output of a single execution of an assembler or compiler.

Odd-Even Check: See Parity Check.

Operand:
1. A value or a unit of data that is operated on.
2. The information needed to define and/or locate 1.

Operation:
1. A program step undertaken by a computer in execution of a machine instruction such as add, multiply, compare, etc.
2. The execution of a series of instructions for the purpose of having one specific function performed, e.g., the transfer of data between main storage and an I/O device.

Operation Code: A mnemonic that represents an operation.

Operational Expression: An expression containing operators.

Operator:
1. A person who operates a machine.
2. A symbol specifying an operation to be performed (see Arithmetic Operator and Comparison Operator).

Option: A specification in a program or a control statement that may be used by the programmer to influence the execution of the program or any of its statements.

Output: The transfer of data from main storage to an external storage device.

Overflow:
1. That portion of the result which exceeds the capacity of the particular unit of storage
2. Page end on a printer.

Overlap: (v.t.) To do something at the same time something else is being done; for example, to perform an I/O operation while instructions of a program are being executed by the CPU.

Overlay: To place a phase or subphase into main storage locations occupied by another phase or subphase that has already been processed.

Pack: (v.t.) A storage technique where by two digits or one digit and sign are stored per byte.

Packed Decimal: A data format in which two digits or one digit and sign are stored per byte.

Parameter: A variable that is given a constant value for a specific purpose or process.

Parity Bit: A binary digit appended to an array of bits to make the sum of all the bits always odd or always even.

Parity Check: A check that tests whether the number of ones (or zeros) in an array of binary digits is odd or even.

Phase: A program or a portion of a program executed as one main-storage load if it is not devided up into subphases. Loading a phase, which is stored in the core-image library, is initiated by a set of Job Control statements, a FETCH or a LOAD in a preceding phase. May be output of Assembler, RPG, PL/I, or Linkage Editor program.

Physical Device Address: A code used by the CPU to select an I/O device.

Physical and Logical Unit Tables Service Program: A system service program. This program (PSERV) is used to display and/or change the permanent device assignments and/or to change the configuration byte of the (Basic) Monitor on the system disk pack.

Physical IOCS: A set of routines contained in the Monitor program. These routines control the transfer of data from the CPU to attached tape and/or disk drives and to the printer-keyboard, if present. The routines also control all data transfer from the aforementioned I/O devices to the CPU.

Physical Unit Block: An entry in the Physical Unit Table.

Physical Unit Table: A table contained in the (Basic) Monitor. It has a number of physical unit blocks, each of which contains an actual device address. Pointers to these blocks are inserted into the logical unit table by means of ASSGN control statements.

Point Alignment: Alignment of arithmetic data in a variable depending on the location of the assumed or actual decimal point.

Position Macro Instruction: A macro instruction whose operands consist of only the values specified by the programmer. They must be specified in a predetermined order.

Prefix Operator: An operator that precedes, and is associated with, a single operand. The prefix operators are + and -.

Priority Level: Classifies macro definitions by frequency usage in TPS. Four levels are used in the macro library section of the tape resident system.

Priority Section: An area of the TPS macro directory or library. Each priority section is assigned to a specific priority level.

Problem Data: Arithmetic or logical (character) data that is processed under control of the problem program in main storage.

Problem Program: Any program that is not part of the programming system or of the card programming support.

Processing Program: Any program that is not a control program.

Program: A series of machine instructions that, when executed, cause the necessary processing to achieve the desired result(s).

Program Library: A collective term used to refer to core-image directory and library.

Program Library Section: The section of the system tape that contains the program library.

Program Section: See Control Section.

Prototype Statement: The first instruction (following the macro header) in a macro definition. It defines the format of the macro instruction and contains various symbolic parameters for which values are substituted when the macro routine is used by a specific program.

PSERV: See Physical and Logical Unit Tables Service Program.

PUB: See Physical Unit Block.

Read: To acquire data from an external storage device.

Read/Compute, Write/Compute Overlap Feature: A feature of the IBM System/360 Model 20, Submodel 5, that permits data transfer from or to magnetic-tape and disk I/O device to be overlapped with processing.

Read Time: See I/O Time.

Receiving Field: Any field to which a value may be moved or assigned.

Relative Address: The number that specifies the difference between the absolute (effective) address and the base address (see Displacement).

Relational Operators: The following operators used in the macro language of the DPS/TPS Assembler: EQ (equal to) GE (greater than or equal to) GT (greater than) LE (less than or equal to) LT (less than) NE (not equal).

Relocatable Address: An address that can be modified by adding a relocation factor, i.e., a base address value.

Relocatable Area: An area on the system disk pack used to temporarily hold an object program (or phase) thus permitting a program or program phase to be assembled and executed in one job.

Relocation: The modification of address constants required to compensate for a change or origin of a phase or subphase.

Report Program Generator: A program which generates report-writing programs in accordance with specifications describing the characteristics of the files involved, the processing to be performed, and the desired output.

Restart: To re-establish the status of a job using the information recorded at a checkpoint.

RPG: See Report Program Generator.

RWC Feature: See Read/Compute, Write/Compute Overlap Feature.

Second-Level Macro Definition: A macro definition that is called by an inner (second-level or nesting) macro instruction.

Sense Byte(s): One or more bytes in main storage. The individual bits of a sense byte (or bytes) are used to indicate the status of I/O devices.

Service Program: Any of the system programs that assist in the use of a computing system and in the successful execution of problem programs, without contributing directly to the control of the system or production of results.

Significant Digit: A digit that contributes to the accuracy or precision of a numeral. The number of significant digits is counted beginning with the digit contributing the most value, called the most significant digit, and ending with the one contributing the least value, called the least significant digit.

Simple Statement: See Statement.

Source Program: A series of statements in the symbolic language of an assembler or compiler, which constitutes the entire input to a single execution of the assembler or compiler.

Source Statement: A statement written in a source language (e.g. Assembler language).

Sort/Merge: A descriptive term meaning "sort or merge". This term is frequently used in connection with a generalized program from which types of sort or merge programs may be defined.

Special Character: Any character that is neither alphabetic nor numeric.

Statement: A meaningful expression or generalized instruction in a source language.

Storage Allocation: The assignment of blocks of storage to blocks of data.

Stream:
1. The flow of data from an external storage medium to main storage; the flow of data from main storage to an external storage medium.
2. See also Input Job Stream.

Subfield: The subdivision of a field.

Subphase: A portion of a program executed as one main-storage load. Loading a subphase from the core-image library, in which it is stored, is initiated by appropriate instructions either within the phase of which the subphase is a part or within the preceding subphase.

Symbolic Address: An address represented by one or more symbols convenient to the programmer.

Symbolic Device Address: A symbol used in IBM-supplied and user-written programs to refer to an I/O device (e.g., SYSRES, SYSIPT, SYS005). This address is related to an actual address by means of the logical unit table.

SYSIPT: See System Input Device.

SYSLOG: See System Output Printer.

SYSLST: See System Output Printer.

SYSOPT: See System Output Device.

SYSRDR: See System Reader.

SYSRES: See System Residence Device.

System Control Program: A collective term used to refer to the Monitor, the Job Control program, and the Initial Program Loader.

System Directory: A table on the system disk pack listing the addresses and sizes of the core-image library and directory, the macro library and directory, and the relocatable area.

System Disk Pack: The disk pack which contains the user's disk-resident system.

System Input Device: An I/O device specified as a source of an input job stream, excluding Job Control statements.

System Output Device: An I/O device used as output device for system programs or problem programs or both (symbolic device address is SYSOPT).

System Output Printer: A printer used to list the output of system programs (symbolic device address is SYSLST) or to log control statements (symbolic device address is SYSLOG) or both (same printer is assigned to both symbolic device addresses).

System Reader: An I/O device used by the system control programs to read the Job Control statements.

System Resident Device: For DPS the disk drive that contains the system disk pack if a disk-resident system is used; the disk drive on whose pack the Job Control program writes label information if a card-resident system is used. For TPS the tape drive that contains the magnetic-tape volume with the system programs.

System Service Programs: A collective term used to refer to the Library Management programs, the PSERV program, the Linkage Editor, and the Load System program.

System Tape: The reel of magnetic tape on which the user's tape-resident system is located.

Tape Error Recovery Routine: A system routine that controls the execution of error recovery procedures in the case of magnetic tape I/O errors.

Tape Error Statistics Routine: A system routine that analyzes magnetic-tape read/write errors and noise records that may occur during the execution of a program. In addition, the routine records the number of erase gap commands that are issued during the execution of the program.

Tape-Resident System: (Also referred to as "user's tape-resident system.") Contains the Basic Monitor program, the Job Control program, and may contain any IBM-supplied and/or user-written programs and/or macro definitions. Consists of three sections: Monitor section, program library section, and macro library section. Is created and updated by means of maintenance programs.

TER: See Tape Error Recovery Routine.

TES: See Tape Error Statistics Routine.

Third-Level Macro Definition: A macro definition that is called by an inner macro instruction of the third level.

Throughput: A measure of system efficiency: the rate at which work can be handled by a computing system.

Transient Area: An area in main storage into which the Monitor loads transient routines for execution.

Truncation: The process of cutting short a data entity either on the right or on the left.

Unpack: (v.t.) To convert numeric data stored in the packed format to unpacked decimal format.

User Routine: A routine written and supplied by the user and incorporated into a system program as a modification. Each user-written routine is accessed through a program exit.

User's Tape-Resident System: See Tape-Resident System.

Utility Programs: Programs that perform frequently recurring jobs such as copying files from one data carrier to another, initializing a disk pack or a reel of magnetic tape, etc.

Volume: That portion of a single unit of storage media that is accessible to a single read/write mechanism. For example, a reel of magnetic tape for a 2415 magnetic tape drive, or one 1316 Disk Pack for a 2311 Disk Storage Drive.

Volume Label. A record which uniquely identifies a volume of magnetic tape or a disk pack by its volume serial number and other information.

Volume Table of Contents (VTOC): A table stored on disk pack. The table contains the labels of all files contained on the same disk pack.

VTOC: See Volume Table of Contents.

Write: To record data on either disk or on magnetic tape.

Write Time: See I/O time.

Zoned Decimal: See Unpacked Decimal.

Appendix A. Machine—Instruction Mnemonic Codes

The following list is an alphabetical listing of the mnemonic operation codes of all the machine instructions that can be represented in the Model 20 DPS/TPS Assembler Language. The column headings in the list and the information each column provides are as follows:

Mnemonic Code: This column gives the mnemonic operation code for the machine instruction.
Instruction: This column contains the name of the instruction associated with the mnemonic.
Operation Code: This column contains the hexadecimal equivalent of the actual machine operation code.
Basic Machine Format: This column gives the basic machine format of the instruction: RR, RX, SI or SS.
Operand Field Format: This column shows the symbolic format of the operand field for the particular mnemonic.

Mnemonic Code	Instruction	Operation Code	Basic Machine Format	Operand Field Format
AH	Add Halfword	4A	RX	$R_1,D_2(0,B_2)$
AP	Add Decimal	FA	SS	$D_1(L_1,B_1),D_2(L_2,B_2)$
AR	Add	1A	RR	R_1,R_2
BAS	Branch and Store	4D	RX	$R_1,D_2(0,B_2)$
BASR	Branch and Store Register	0D	RR	R_1,R_2
BC	Branch on Condition	47	RX	$M_1,D_2(0,B_2)$
BCR	Branch on Condition	07	RR	M_1,R_2
CH	Compare Halfword	49	RX	$R_1,D_2(0,B_2)$
CIO	Control I/O	9B	SI	$D_1(B_1),UF$
CLC	Compare Logical	D5	SS	$D_1(L,B_1),D_2(B_2)$
CLI	Compare Logical Immediate	95	SI	$D_1(B_1),I_2$
CP	Compare Decimal	F9	SS	$D_1(L_1,B_1),D_2(L_2,B_2)$
DP	Divide Decimal	FD	SS	$D_1(L_1,B_1),D_2(L_2,B_2)$
ED	Edit	DE	SS	$D_1(L,B_1),D_2(B_2)$
HPR	Halt and Proceed	99	SI	$D_1(B_1),I_2$
LH	Load Halfword	48	RX	$R_1,D_2(0,B_2)$
MP	Multiply Decimal	FC	SS	$D_1(L_1,B_1),D_2(L_2,B_2)$
MVC	Move Characters	D2	SS	$D_1(L,B_1),D_2(B_2)$
MVI	Move Immediate	92	SI	$D_1(B_1),I_2$
MVN	Move Numerics	D1	SS	$D_1(L,B_1),D_2(B_2,$
MVO	Move with Offset	F1	SS	$D_1(L_1,B_1),D_2(L_2,B_2)$
MVZ	Move Zones	D3	SS	$D_1(L,B_1),D_2(B_2)$
NI	AND Logical Immediate	94	SI	$D_1(B_1),I_2$
OI	OR Logical Immediate	96	SI	$D_1(B_1),I_2$
PACK	Pack	F2	SS	$D_1(L_1,B_1),D_2(L_2,B_2)$
SH	Subtract Halfword	4B	RX	$R_1,D_2(0,B_2)$
SP	Subtract Decimal	FB	SS	$D_1(L_1,B_1),D_2(L_2,B_2)$
SPSW	Set Program Status Word	81	SI	$D_1(B_1)$
SR	Subtract	1B	RR	R_1,R_2
STH	Store Halfword	40	RX	$F_1,D_2(0,B_2)$
TIOB	Test I/O and Branch	9A	SI	$D_1(B_1),UF$
TM	Test Under Mask	91	SI	$D_1(B_1),I_2$
TR	Translate	DC	SS	$D_1(L,B_1),D_2(B_2)$
UNPK	Unpack	F3	SS	$D_1(L_1,B_1),D_2(L_2,,B_2)$
XIO	Execute I/O	D0	SS	$D_1(UF,B_1),D_2(B_2)$
ZAP	Zero and Add Decimal	F8	SS	$D_1(L_1,B_1),D_2(L_2,B_2)$

Extended Mnemonic Instruction Codes

EXTENDED CODE	OPERAND	MEANING	MACHINE INSTRUCTION
B	$D_2(0,B_2)$	Branch Unconditional	BC 15,$D_2(0,B_2)$
BR	R_2	Branch Unconditional (RR Format)	BCR 15,R_2
NOP	$D_2(0,B_2)$	No Operation	BC 0,$D_2(0,B_2)$
NOPR	R_2	No Operation (RR Format)	BCR 0,R_2
		USED AFTER COMPARE INSTRUCTIONS	
BH	$D_2(0,B_2)$	Branch on High	BC 2,$D_2(0,B_2)$
BL	$D_2(0,B_2)$	Branch on Low	BC 4,$D_2(0,B_2)$
BE	$D_2(0,B_2)$	Branch on Equal	BC 8,$D_2(0,B_2)$
BNH	$D_2(0,B_2)$	Branch on Not High	BC 13,$D_2(0,B_2)$
BNL	$D_2(0,B_2)$	Branch on Not Low	BC 11,$D_2(0,B_2)$
BNE	$D_2(0,B_2)$	Branch on Not Equal	BC 7,$D_2(0,B_2)$
		USED AFTER ARITHMETIC INSTRUCTIONS	
BO	$D_2(0,B_2)$	Branch on Overflow	BC 1,$D_2(0,B_2)$
BP	$D_2(0,B_2)$	Branch on Plus	BC 2,$D_2(0,B_2)$
BM	$D_2(0,B_2)$	Branch on Minus	BC 4,$D_2(0,B_2)$
BZ	$D_2(0,B_2)$	Branch on Zero	BC 8,$D_2(0,B_2)$
		USED AFTER TEST UNDER MASK INSTRUCTION	
BO	$D_2(0,B_2)$	Branch if Ones	BC 1,$D_2(0,B_2)$
BM	$D_2(0,B_2)$	Branch if Mixed	BC 4,$D_2(0,B_2)$
BZ	$D_2(0,B_2)$	Branch if Zeros	BC 8,$D_2(0,B_2)$

Appendix B. Machine—Instruction Format

	Basic Machine Format								Assembler Operand Field Format	Applicable Instructions
RR	8 Operation Code	4 R_1	4 R_2						R_1,R_2 (See note 1)	AR,BASR,SR
	8 Operation Code	4 M_1	4 R_2						M_1,M_2 (See notes 1 and 4)	BCR
RX	8 Operation Code	4 R_1	4 X_2	4 B_2	12 D_2				R_1,D_2(0,B_2) R_1,S_2 (See notes 1,2,3,and 7)	STH,LH,CH,AH,SH, BAS
	8 Operation Code	4 M_1	4 X_2	4 B_2	12 D_2				M_1,D_2(0,B_2) M_1,S_2 (See notes 2,3,4,and 7)	BC
SI	8 Operation Code	8 I_2		4 B_1	12 D_1				D_1(B_1),I_2 S_1,I_2 (See notes 2,3,6,and 7)	CLI,MVI,NI,OI,TM,HPR
	8 Operation Code	8 --		4 B_1	12 D_1				D_1(B_1) S_1 (See notes 2,3,7,and 8)	SPSW
	8 Operation Code	8 UF		4 B_1	12 D_1				D_1(B_1),UF S_1,UF (See notes 2,3,and 7-9)	TIOB CIO (D_1(B_1) detailed specification)
SS	8 Operation Code	4 L_1	4 L_2	4 B_1	12 D_1	4 B_2	12 D_2		D_1(L_1,B_1),D_2(L_2,B_2) S_1(L_1),S_2(L_2) (See notes 2,3,5,and 7)	PACK,UNPK,MVO,AP, CP,DP,MP,SP,ZAP
	8 Operation Code	8 L		4 B_1	12 D_1	4 B_2	12 D_2		D_1(L,B_1),D_2(B_2) S_1(L),S_2 (See notes 2,3,5,and 7)	CLC,MVC,MVN, MVZ,TR,ED,
	8 Operation Code	8 UF		4 B_1	12 D_1	4 B_2	12 D_2		D_1(UF,B_1),D_2(B_2) S_1(UF),S_2 (See notes 2,3,7,and 9)	XIO (D_2(B_2) detailed specification)

Notes to Appendix B:

1. R_1 and R_2 are absolute expressions that specify general registers. The general register numbers are 8 through 15.

2. D_1 and D_2 are absolute expressions that specify displacements. A value of 0 - 4095 may be specified.

3. B_1 and B_2 are absolute expressions that specify base registers. Register numbers range between 0 and 15.

4. M_1 is an absolute expression representing a condition code.

5. L, L_1, and L_2 are absolute expressions that specify field lengths. An L expression can specify a value of 1 - 256. L_1 and L_2 expressions can specify a value of 1 - 16. In all cases, the assembled value will be one less than the specified value.

6. I and I_2 are absolute expressions that provide immediate data. The value of the expression may be 0 - 255.

7. S_1 and S_2 are absolute or relocatable expressions that specify an address.

8. SI instruction fields that are crossed out in the machine formats are not examined during instruction execution. The fields are not written in the symbolic operand, but are assembled as binary zeros.

9. UF is an absolute expression representing an input/output unit address and a function.

Appendix C. Assembler—Instructions

Mnemonic	Name Field	Operand Field
CSECT	An optional symbol	Blank; or a comment preceded by a comma.
DC	An optional symbol	One operand
DCCW	An optional symbol	Four operands, separated by commas
DROP	Blank	One to four absolute expressions, separated by commas
DS	An optional symbol	One operand
DSECT	A required symbol	Blank or a comment preceded by a comma.
EJECT	Blank	Blank or a comment preceded by a comma.
END	Blank	A relocatable expression or blank
ENTRY	Blank	One relocatable symbol
EQU	A required symbol	An absolute or relocatable expression
EXTRN	Blank	One relocatable symbol
ICTL	Blank	25 (or 25,71,38 when using macro-instructions)
ISEQ	Blank	A blank, or two decimal values separated by a comma
LTORG	An optional symbol	Blank or a comment preceded by a comma.
ORG	Blank	A relocatable expression or blank
PRINT	Blank	One to three operands
REPRO	Blank	Blank or a comment preceded by a comma.
SPACE	Blank	A decimal term or blank
START	An optional symbol	A self-defining term or blank
TITLE	0-4 characters	Up to 62 characters, enclosed in apostrophes
USING	Blank	A relocatable expression followed by one to four absolute expressions, separated by commas
XFR	Blank	A relocatable symbol

Appendix D. Summary of Constants

TYPE	IMPLIED LENGTH** (BYTES)	ALIGN-MENT	LENGTH MODIFIER RANGE	SPECIFIED by	TRUNCATION/ PADDING SIDE
C	as needed	byte	1 to 32	characters	right
X	as needed	byte	1 to 32	hexadecimal digits	left
B	as needed	byte	1 to 8	binary digits	left
H	2	halfword*	1 to 2	decimal digits	left
P	as needed	byte	1 to 16	decimal digits	left
Z	as needed	byte	1 to 16	decimal digits	left
Y	2	halfword*	1 to 2	any expression	left

*Unless an explicit length is specified.
**The implied length must not exceed the maximum modifier length.

Appendix E. Summary of Macro Language

Expressions in Macro Language

Type	Arithmetic	Character	Logical	Relational
Can contain	Positive decimal self-defining terms SETA and SETB symbols SETC symbols if the value assigned is a positive decimal self-defining term Symbolic parameters if the corresponding operand is a positive decimal self-defining term &SYSLIST(n) if the corresponding operand is a positive decimal self-defining term &SYSNDX	Up to 8 characters enclosed by apostrophes Any SET symbol, system variable symbol, or symbolic parameter enclosed by apostrophes Any concatenation enclosed by apostrophes	0, 1, or SETB symbols Maximum of two SETB symbols 0 and 1 can be used only in single-term expressions (which must not be preceded by the operator NOT)	Two arithmetic expressions Two character expressions
Operators	+ - * /	Concatenating by using a period (.) and/or substringing	AND, OR and NOT	EQ, NE, LT, GT, LE and GE
Range of values	0 to 99999	Zero to eight characters	0 (FALSE) or 1 (TRUE)	0 (FALSE) or 1 (TRUE)
Can be used in	SETA operands Relational expressions	SETC operands Relational expressions	SETB operands AIF operands AIFB operands	SETB operands AIF operands AIFB operands

Name and Operand Field of Instructions

Instruction	Name Field	Operand Field
AGO	A sequence symbol or blank	A sequence symbol defined in a following statement
AGOB	A sequence symbol or blank	A sequence symbol defined in a preceding statement
AIF	A sequence symbol or blank	A logical or relational expression enclosed by parentheses, immediately followed by a sequence symbol defined in a following statement
AIFB	A sequence symbol or blank	A logical or relational expression enclosed by parentheses, immediately followed by a sequence symbol defined in a preceding statement
ANOP	A sequence symbol	Blank
MACRO	Blank	For DPS: the version code (2 cols.) and the modification code (2 cols.), or blank; for TPS: blank.
MEND	A sequence symbol or blank	Blank
MEXIT	A sequence symbol or blank	Blank
MNOTE	A sequence symbol or blank	A combination of characters enclosed by apostrophes
SETA	&AGn or &ALn, where n is 0-15	An arithmetic expression with a maximum of three terms
SETB	&BGn where n is 0-255 for both DPS and TPS. &BLn where n is 0-255 for DPS and 0-127 for TPS	A logical expression or a relational expression enclosed by parentheses
SETC	&CGn, where n is 0-15	Up to eight characters enclosed by a pair of apostrophes --substrings and concatenation permitted.
Model Statement (any Assembler language mnemonic operation code, except END, ICTRL, ISEQ, LTORG, PRINT, and START)	A symbol, a variable symbol, a sequence symbol, a symbolic parameter, or a concatenation representing a symbol	Any valid combination of characters(including variable symbols)
Prototype Statement	A symbolic parameter or blank	Up to 49 symbolic parameters separated by commas
Macro Instruction	Valid symbol or blank	Up to 49 operands, separated by commas

*When the IOCS is used, the following symbols must not be used:
&BG0 - &BG19, &BG21, &BG27, &BG28, &BG69, and &BG80 - &BG88.

Symbolic Parameters and Variable Symbols in Expressions

Symbol	Defined By	Initialized or Set to	Value Changed By	Can be Used In
Symbolic parameter	Prototype statement	Corresponding macro instruction operand	Constant throughout definition	Arithmetic expressions if operand is an unsigned decimal self-defining term Character expressions Model statements
SETA	Predefined	0 (at assembly start for global, at macro call for local)	SETA instruction	Arithmetic expressions Character expressions Model statements
SETB	Predefined	0 (at assembly start for global, at macro call for local)	SETB instruction	Arithmetic expressions Character expressions Logical expressions Model statements
SETC	Predefined	Null character value (at assembly start)	SETC instruction	Arithmetic expressions if operand is an unsigned decimal self-defining term Character expressions Model statements
&SYSNDX	The Assembler	Macro instruction index	Constant throughout definition; unique for each macro instruction	Arithmetic expressions Character expressions Model statements
&SYSECT	The Assembler	Control section in which macro instruction appears	Constant throughout definition; set by CSECT, DSECT, and START	Character expressions Model statements
&SYSLIST(n) Where n is a SETA symbol or a decimal self-defining value	The Assembler	Corresponding positional macro-instruction operand	Constant throughout definition for a given value of n	Arithmetic expressions if operand is an unsigned decimal self-defining term Character expressions Model statements

IBM—Supplied Macro Definitions

```
List of IBM-Supplied Macro Definitions (DPS)

ASSGN ATENT

CDIPL CLOSE CNTRL CNVRT COMRG CRDPR CREAD

DCCB  DCNT  DCSCT DSENG DSKA  DTFBG DTFBN DTFBT DTFBU DTFBV DTFBW DTFBX DTFBY DTFCF
DTFCG DTFDA DTFDC DTFDF DTFDO DTFDR DTFEN DTFIA DTFID DTFIN DTFIQ DTFIR DTFIS DTFIT
DTFIV DTFLC DTFLD DTFM3 DTFMM DTFMT DTFMU DTFMV DTFMW DTFMX DTFMY DTFM1 DTFM2 DTFNA
DTFNB DTFNC DTFND DTFNE DTFNF DTFP  DTFPA DTFPC DTFPD DTFPE DTFPK DTFPL DTFPM DTFPN
DTFPO DTFPQ DTFPR DTFRG DTFSC DTFSD DTFSE DTFSF DTFSG DTFSH DTFSI DTFSJ DTFSK DTFSL
DTFSN DTFSR DTFST DTFSU DTFSV DTFSW DTFSX DTFSY DTFSZ DTFTC DTFTL DTFTO DTFYR DTFYW

ENDFL ENDMT EOJ EOM ESETL EXITB

FEOV FETCH

GET

I$BG0 I$BG1 I$BG3 IB249 IE001 IICMA INTRD I0001 IPIQO IQIPT

LBRET LOAD LOM LXITB

MACRO MAINT MCIPL MDERP MFET  MINQ  MJOB1 MJOB2 MJOB3 MJOB4 MJOB5 MJOB6 MONTR MPPK
MRIN  MROUT MSCED MSC00 MSC10 MSC11 MTRAN MVCOM

NOINQ

OPEN

PREQ PRTOV PUT

READ  RELSE RETRN RJBGN RJCHK RJCMP RJCON RJCRD RJCTR RJDRD RJDRH RJDWE RJE   RJEMM
RJEND RJECT RJERR RJEUT RJEXP RJHDR RJLER RJMSG RJOUT RJPCH RJPRD RJPTR RJREQ RJRQM
RJRQP RJTRA RJTRC RJTRD RJTWC RJTWE RJWNQ RPGEQ SDRTB RAPQP TRATB

SETFL SETL

TRUNC

WAIT WAITB WAITC WAITF WRITE


List of IBM-Supplied Macro Definitions (TPS)

Priority 1
CNTRL COMRG CRDPR DSENG EOJ   EOM   EXITB FEOV  FETCH GET   LBRET LOM   LXITB MVCOM
PRTOV PUT   READ  RELSE RJCMP RJEXP TRUNC WAITB WAITC WRITE

Priority 2
CLOSE DTFBG DTFBT DTFBU DTFBV DTFBW DTFBX DTFBY DTFCF DTFCG DTFMM DTFMT DTFMU DTFMV
DTFMW DTFMX DTFMY DTFNA DTFNB DTFNC DTFND DTFNE DTFNF DTFPA DTFPC DTFPD DTFPF DTFSR
DTFST DTFSU DTFSV DTFSW DTFSX DTFSY DTFSZ OPEN

Priority 3
DTFBN DTFSN

Priority 4
DTFEN
```

Appendix F. Assembler—Language Features

Feature	Model 20 Basic Assembler	Model 20 DPS/TPS Assembler	SUPPAK	BPS Basic Assembler	BPS/BOS Assembler	DOS/TOS	OS/360
No. of continuation cards/statement (excl. of macro constructions)	0	0	0	0	1	1	2
Input character code	EBCDIC	EBCDIC	BCD or EBCDIC	EBCDIC	EBCDIC	EBCDIC	EBCDIC
ELEMENTS:							
Maximum characters/ symbol	4	8	6	6	8	8	8
Character self-defining terms	1 char. only	1 char. only	X	1 char. only	X	X	X
Binary self-defining terms	--	8 bits max.	--	--	X	X	X
Length-attribute reference	--	X	--	--	X	X	X
Literals	--	see DC	--	--	X	X	X
Extended Mnemonics	--	X	X	--	X	X	X
Maximum location-counter value	$2^{14}-1$	$2^{15}-1$	$2^{24}-1$	$2^{16}-1$	$2^{24}-1$	$2^{24}-1$	$2^{24}-1$
Multiple control sections per assembly	--	max. of 8	--	--	X	X	X
EXPRESSIONS:							
Operators	+-	+-*	+-*/	+-*	+-*/	+-*/	+-*/
Number of terms	3	3	16	3	3	8	16
Number of parentheses	--	1 level	--	--	1 level	3 levels	5 levels
Complex relocatability	--	X	--	--	X	X	X
ASSEMBLER INSTRUCTIONS:							
DC and DS Expressions in modifiers	--	--	--	--	--	X	X
Multiple operands	--	--	--	--	--	--	X
Multiple constants/ operand	--	--	--	--	Except address cons.	X	X
Bit length specification	--	--	--	--	--	--	X
Scale modifier	--	--	--	--	X	X	X
Exponent modifier	--	--	--	--	X	X	X
DC types	Only C, X, H, Y	C, X, B, H, P, Z, Y	Except B, V	Except B, P, V Z, Y, S	X	X	X
DC duplication factor	--	X	X	Except A	Except S	X	X
DC duplication factor of zero	--	--	--	--	Except S	X	X
DC length modifier	Except H, Y	see Appx. D	X	Except H, E, D	X	X	X

Feature	Model 20 Basic Assembler	Model 20 DPS/TPS Assembler	SUPPAK	BPS Basic Assembler	BPS/BOS Assembler	DOS/TOS	OS/360
DS maximum length modifier	256	256	256	256	256	65,535	65,535
DS constant subfield permitted	--	--	--	--	X	X	X
DS type	only H,C	only H,C	only C,H, F,D	only C,H, F,D	X	X	X
DS length modifier	only C	only C	only C	only C	X	X	X
COPY	--	--	--	--	--	X	X
CSECT	--	8 max.	--	--	X	X	X
DSECT	--	7 max.	--	--	X	X	X
ISEQ	--	X	--	--	X	X	X
LTORG	--	X	--	--	X	X	X
PRINT	--	X	--	--	X	X	X
TITLE	--	X	X	--	X	X	X
COM	--	--	--	--	--	X	X
ICTL	--	1 opnd. 25 only or 3 opnds. 25,71,38	1 opnd.	1 opnd. 1 or 25 only	X	X	X
USING	2 opnds. 1st opnd reloc. only	5 opnds. 1st opnd reloc. only	2-17 opnds. 1st opnd reloc. only	2 opnds. reloc. only	6 opnds.	X	X
DROP	1 opnd. only	4 opnds. only	X	1 opnd. only	5 opnds.	X	X
CCW	--	4 opnds. (DCCW)	X	opnd. 2 reloc. only	X	X	X
ORG	No blank opnd.	X	No blank opnd.	No blank opnd.	X	X	X
ENTRY	1 opnd. only	1 opnd. only	1 opnd. only	1 opnd. only	1 opnd. only	X	X
EXTRN	1 opnd. only	1 opnd. only	1 opnd. only	max. 14 1 opnd. only	1 opnd. only	X	X
CNOP	--	--	2 dec. digits	2 dec. digits	2 dec. digits	X	X
PUNCH	--	--	--	--	--	X	X
REPRO	--	X	--	--	X	X	X

Macro Instructions	360/20 IOCS only	**	--	--	X	X	X
Operand Sublists		--			--	X	X
Attributes of macro-instruction operands inside macro definitions and symbols used in conditional-assembly instructions outside macro definitions		--			--	X	X
Subscripted SET symbols		--			--	X	X
Maximum number of parameters		49			49	200	200
Conditional-assembly instructions outside macro definitions		--			--	X	X
Maximum number of SET symbols							
Global SETA		16			16	*	*
Global SETB		256[1]			128	*	*
Global SETC		16			16	*	*
Local SETA		16			16	*	*
Local SETB		256[2]			128	*	*
Local SETB		128[3]					
Local SETC		0			0	*	*

Legend:

* The number of SET symbols permitted by the Operating System/360 and DOS/TOS Assembler is variable, dependent upon available main storage.

X Implemented as specified in IBM System/360 Assembler Language SRL.

-- Not implemented.

** See the SRL publication IBM System/360 Model 20, Disk Programming System, Performance Estimates, Form GC33-6003.

[1] For DPS and TPS
[2] For DPS only
[3] For TPS only

Appendix G. Output Listings (Assembler and MMAINT)

Assembler Program

FIELD	PRINT POSITIONS	MEANING
	External Symbol Dictionary (ESD)	
SYMBOL	01-08	External name.
TYPE	11-12	Symbol type.
ID	15-16	ESD entry number.
ADDR	19-22	Address of symbol before relocation.
LENGTH	26-29	Length attribute of control section.
LD ID	34-35	ESD entry number of control section where name appears.
	Instructions	
LOCATN	02-05	Location of assembled instruction (hexadecimal).
OBJECT CODE	07-20, or	Assembled instruction (hexadecimal).
	07-22	Constant generated (hexadecimal).
ADD1	22-25	Effective address of 1st operand (hexadecimal).
ADD2	27-30	Effective address of 2nd operand (hexadecimal).
STMNT	32-35	Statement number (decimal).
SOURCE STATEMENT	37-116	Source statement (card image). Column 36 contains a plus sign if a source statement is generated.
	118-120	Number of TXT card (decimal). Note: If NODECK was specified, the number is that of a NOESD option.
WRN	03-05	Flag: Possible error in previous statement.
SEQ	08-10	Flag: Sequence error in previous statement.
ERR	13-15	Flag: Error in previous statement.
CAT	18-20	Flag: Catastrophic error in previous statement.
	Relocation Dictionary (RLD)	
POS.ID	03-04	ESD ID number of control section containing the constant.
REL.ID	10-11	ESD ID number of control section containing the address in the constant.
FLGS	16-17	Type of relocation.
ADDR	20-23	Address of constant before relocation.
	Diagnostics	
STATEMENT NO.	05-08	Listing sequence number of statement in error.
ERROR MESSAGES	20-79	Explanation of error.
ACTION	80-120	Action taken by Assembler.
	Table of Defined Symbols	
SYMBOL	01-08 65-72	Name assigned to source statement.
LEN	11-13 75-77	Length attribute (decimal notation).
VALUE	17-20 81-84	Value attribute (hexadecimal notation).
TYPE	24 88	Type attribute.

Crossreference List* (for DPS Assembler only)		
SYMBOL	01-08	Source Label
LEN	10-13	Length attribute (decimal)
VALUE	17-20	Value attribute (hexadecimal)
DEF	23-26	Listing sequence number of statement which defines label
CROSS-	31-34	Listing sequence numbers of statements
REFERENCE	37-40	which contain the label
	43-46	
	49-52	
	55-58	
	61-64	
	67-70	
	73-76	
	79-82	
	85-88	
	91-94	
	97-100	
	103-106	
	109-112	
	115-118	
*Replaces Table of Defined Symbols if CROSSREF is specified in AOPTN statement.		

```
                                        EXTERNAL SYMBOL DICTIONARY

 SYMBOL  TYPE ID  ADDR  LENGTH LD IF

          SD  01  0100   0048


 EXAMPLE

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT

 0100                          0002          START 256
 0000                          0003          USING *-256,0
 0008                          0004 R8       EQU   8
 0009                          0005 R9       EQU   9
 0100 4890 013E       013E     0006 BEGIN    LH    R9,ADDR          LOAD ADDRESS OF TABLE
                               0007          MVA   SWITCH,0         SET SWITCH OFF
            ERR
 0104 D201 0140 0144  0140 0144 0008          MVC   LASTAD(2),H0     CLEAR TABLE
 010A 4D80 0000       0000     0009          BAS   R8,SUBR          GO TO SUBROUTINE
            ERR
 010E 9500 0142       0142     0010          CLI   SWITCH,0         IF SWITCH OFF
 0112 4780 011E       011E     0011          BE    EXIT             LEAVE ROUTINE
 0116 4A90 0146       0146     0012          AH    R9,H2            ELSE INCREASE ADDRESS
 011A 47F0 0104       0104     0013          B     BEGIN+4          AND BRANCH BACK
 011E 4D80 00C0       00C0     0014 EXIT     BAS   R8,192
 0122                          0015 AREA     DS    4H
 012A                          0016 TABLE    DS    10H
 013E 0122                     0017 ADDR     DC    Y(AREA)
 0140 012A                     0018 LASTAD   DC    Y(TABLE)
 0142 00                       0019 SWITCH   DC    X'00'
 0144 0000                     0020 H0       DC    H'0'
 0146 0002                     0021 H2       DC    H'2'
                               0022          END

0002 STATEMENTS FLAGGED


                                        RELOCATION DICTIONARY

POS.ID REL.ID FLGS ADDR

 01     01     04  013E
 01     01     04  0140


                                        DIAGNOSTICS

STATEMENT NO.      ERROR MESSAGES                    ACTION
    0007           UNDEFINED OPERATION CODE          STATEMENT TREATED AS COMMENT
    0009           UNDEFINED SYMBOL                  STATEMENT INCOMPLETELY ASSEMBLED


                                        TABLE OF DEFINED SYMBOLS

 SYMBOL   LEN  VALUE   TYPE

ADDR        2   013E    Y
AREA        2   0122    H
BEGIN       4   0100    I
EXIT        4   011E    I
H0          2   0144    H
H2          2   0146    H
LASTAD      2   0140    Y
R8          1   0008
R9          1   0009
SWITCH      1   0142    X
TABLE       2   012A    H
```

Figure 17. Sample Listing of Assembler Output

FIELD	PRINT POSITION	MEANING
STMNT NO.	1-8	Statement number
MACRO DEFINITION STATEMENT	12-91	Macro definition statement
DIAGNOSTIC	95-120	Diagnostic message

```
                    MACRO
STMT NO.    MACRO DEFINITION STATEMENT                          DIAGNOSTIC

            &NAME    ADD    &F1,&F2,&F3,&F4,&F5

ADD  001    *
ADD  002    &NAME    STH    12,SAVEAREA
ADD  003             LH     12,&F1
ADD  004    &AL1     SETA   2
ADD  005    .ADD     AH     12,&SYSLIST(&AL1)
ADD  006    &AL1     SETA   &AL1+1
ADD  007    &AL2     SETA   &AL1+1
ADD  008             AIFB   ('&SYSLIST(&AL2)' NE '').ADD
ADD  009             STH    12,&SYSLIST(&AL1)
ADD  010             LH     12,SAVEAREA
ADD  011             MEND
```

Figure 18. Sample Listing of Macro Maintenance-Program Output

Appendix H. Assembler Diagnostic Messages

The associated actions (see Output Listings) are abbreviated as:

SA = Statement assembled
STC= Statement treated as comment
SIA= Statement incompletely assembled
AIE= Assembly in Error

Diagnostic Message	Meaning	Associated Action	Notes
ASSEMBLER CONTROL STATEMENT - AWORK INVALID	The operand of an AWORK statement is not a single operand of either 1 or 2.	STC	
ASSEMBLER CONTROL OR MACRO STATEMENT - INVALID NAME FIELD	An AOPTN statement has been detected with an entry in the name field. The name field has been ignored, but the remainder of the statement has been processed.	SA	
ASSEMBLER CONTROL STATEMENT - INVALID OR MISSING OPERAND	An AOPTN statement has been encountered with no operands or with an operand function that is not one of the valid options. The valid operand and any other operands following are ignored.	SIA	
BAD DATA	Erroneous data found in DC or DCCW statement.	SIA	
CONSTANT TRUNCATED	Specified constant length is less than the actual length of a constant.	SA	
ILLEGAL CONTINUATION LINE	A statement is continued on the subsequent line (in the subsequent card).	STC	
ILLEGAL FORMAT	1. Invalid delimiter. 2. Missing or extra field(s) in statement operand. 3. First operand is a literal. 4. Blank operand in a machine instruction. 5. Parentheses are not paired. 6. Invalid symbol in L'SYMBOL. 7. Illegal double indexing in RX format. 8. Illegal type of self-defining term.	SIA	
ILLEGAL MODIFIER	Incorrect modifier in a DC or DS statement.	SIA	
IMPROPER START VALUE	The value in a START statement is not an integer multiple of two. (The value is increased to the next higher integer multiple of two).	SA	
INVALID CONDITION CODE SPECIFICATION	Condition code mask specified in BC or BCR is other than 0-15.	SIA	

Diagnostic Message	Meaning	Associated Action	Notes
INVALID EXPRESSION	1. For all instructions except DC and EQU, the value of the expression is negative. 2. The expression contains more than three terms. 3. The expression is complex relocatable, but it is not allowed for this instruction. 4. The terms in a multiplication are not absolute. 5. An arithmetic operator begins or ends an expression. 6. Parentheses are not paired. 7. During expression evaluation a value greater than $2^{15}-1$ or less than -2^{15} has been reached. 8. Invalid delimiter sequence. 9. The operand in an ORG statement is not relocatable within the section. 10. The operand in an END or XFR statement is not relocatable.	SIA/STC	
INVALID IMMEDIATE DATA	The immediate data is an invalid self-defining term (e.g., more than one byte).	SIA	
INVALID LENGTH VALUE	1. For SS type instructions. a. Length is greater than 256 b. For two length instructions, L1 or L2 is greater than 16. 2. Length is specified as a relocatable term.		
INVALID LITERAL SPECIFICATION	Literals have been specified, although an AOPTN LITERAL statement is not included. (TPS Assembler only).	SIA	
INVALID NAME FIELD	1. For all instructions except TITLE: The statement name begins with a non-alphabetic character. ($, @, and # are considered alphabetic characters). 2. The statement name is longer than eight characters. 3. Non-alphanumeric characters appear within the statement name. 4. A statement name is present in a statement which must not have a name. 5. A DSECT statement has no name. 6. No name or an invalid name in an EQU statement.	SA/STC	
INVALID OCCURENCE OF ASSEMBLER STATEMENT	1. Program has more than one START card. 2. A START card is improperly placed in the program. 3. An LTORG appears within a dummy control section.	STC	

Diagnostic Message	Meaning	Associated Action	Notes
INVALID REGISTER CONTENTS	1. Specified register contents are not relocatable or they exceed $2^{15}-1$. 2. Specified contents for registers 0 - 7 are not correct.	SIA/SA	
INVALID REGISTER SPECIFICATION	1. Register specified is other than 0 to 15. 2. Register specified as relocatable term.	SIA/STC	
INVALID SELF-DEFINING TERM	The self-defining term: 1. is too large 2. is too long 3. contains an invalid character.	SIA	
INVALID OPERATION CODE	1. The operation code begins with a non-alphabetic character. ($, @, and # are considered alphabetic characters). 2. Non-alphanumeric characters appear within the operation code.	STC	
LIMIT ERROR	Storage required for a constant exceeds $2^{15}-1$.	STC	
LIMIT EXCEEDED	1. The value of the location counter has exceeded $2^{15}-1$. 2. The value of the location counter set by the ORG statement has gone below the initial value of the control section. 3. The total number of CSECT, DSECT, and EXTRN statements exceeds 31. 4. The total number of CSECT and DSECT statements exceeds 8. 5. Total number of ENTRY statements must not exceed 20.	AIE	
MACRO - GENERATION TERMINATED	Maximum number (999 for TPS; 4999 for DPS) of executed AGO, AGOB, AIF, and AIFB statements exceeded.	STC	1, 2
MACRO - GENERATION TERMINATED BY OPERATOR'S INTERVENTION	The generation is terminated after the operator has assigned the value 'FF' to the core storage position '00CE' via the console.	STC	
MACRO - INNER MACRO NESTING DEPTH EXCEEDED	An inner macro instruction has been given within a third level. (This macro instruction will not be expanded).	STC	1, 2
MACRO INSTRUCTION - INVALID OCCURANCE OF DATA	The columns up to the continue column of a continuation line are not blank.	SA	
MACRO INSTRUCTION - INVALID OPERAND	For example, operand longer than 7 characters, or operand has an equal sign in other than the first position.	STC	

Diagnostic Message	Meaning	Associated Action	Notes
MACRO INSTRUCTION - KEYWORD MULTIPLE SPECIFIED	A keyword occurs more than once in a macro instruction operand.	SA	
MACRO INSTRUCTION - TOO MANY OPERANDS	More operands in a macro instruction than specified in the prototype statement. The extra operands are ignored.	SIA	
MACRO INSTRUCTION - UNDEFINED KEYWORD	Keyword in macro instruction does not match any keyword defined in the prototype. This operand is ignored; all other operands are processed. Note: Only one message appears when more than one undefined keyword appears in the same card of a macro instruction.	SIA	
MACRO - INVALID DATA IN ARITHMETIC OPERATION	Non-numeric character encountered in an AIF, AIFB, or SETB statement with an arithmetic relation or in a SETA statement.	STC	1, 2
MACRO - INVALID RESULT IN ARITHMETIC OPERATION	For an AIF, AIFB, or SETB statement with an arithmetic relation or for a SETA STATEMENT: 1. Result is negative 2. Result is greater than 99999.	STC	1, 2
MACRO - INVALID SUBSTRING	Specified substring not wholly contained in the character string of a SETC instruction.	STC	1, 2
MACRO - INVALID SYSLIST REFERENCE	SYSLIST reference to a parameter number is less than 1 or greater than 49.	STC	1, 2
MACRO - LONG FINAL RESULT CHARACTER OPERATION	Character string result has exceeded eight characters in an AIF, AIFB, or SETB statement with a character relation or a SETC statement.	STC	1, 2
MACRO - LONG INTERMEDIATE RESULT IN CHARACTER OPERATION	Character string has exceeded sixteen characters for a SETC statement.	STC	1, 2
MACRO - STATEMENT TRUNCATED	A generated model statement exceeds column 71	SA	
MACRO - UNDEFINED OPERATION CODE	An instruction with an operation code which is not recognized as a valid System/360 Model 20 operation code and is not contained in the macro library was found during macro generation.	STC	2
MISSING UF IN XIO	Missing unit and function specification in XIO statement.	SIA	
MNOTE	A message from the macro coder to the macro user, generally identifying an error in the macro instruction.	None	

Diagnostic Message	Meaning	Associated Action	Notes
MULTIPLE DEFINITION	1. Identical symbols appear in the name fields of two or more statements. 2. For an EXTRN statement: a. Operand is identical to the name field of another statement b. Two or more statements have identical operands 3. The name fields of CSECT and/or DSECT statements are identical. The statement encountered second is considered unnamed.	SA/STC	
NOT ADDRESSABLE	1. No base register specified. 2. An absolute displacement is greater than 4095 3. Base register(s) specified in USING statement(s) cannot be applied, (no coincidental relocatability).	SIA	
RELOCATION ERROR	1. Base register specified in relocatable operand. 2. Relocatable expression in YL1( ). 3. Symbol in operand of ENTRY statement is defined in an unnamed control section.	SA	
STATEMENT FORMAT CANNOT BE ANALYZED	1. Erroneous operand in an instruction. 2. Blank operand in an Assembler instruction. 3. For DC or DS: a. First character in operand field is not alphabetic or numeric. b. No alphabetic character follows the duplication factor. c. No terminating apostrophe or close parenthesis. d. Terminating apostrophe or close parenthesis followed by a non-blank character. e. Length specification is non-numeric. f. Invalid constant type. 4. An operation code and/or an operand is not contained in columns 2 through 71. 5. Operation code has more than 5 characters. 6. DC literal has no fourth sub-field. 7. No initial apostrophe in TITLE statement. 8. More than four registers are specified in a USING or DROP statement.	STC	
SYMBOL NOT PREVIOUSLY DEFINED	A symbol in the operand of an ORG or EQU statement is not defined in a previously encountered statement.	STC	

Diagnostic Message	Meaning	Associated Action	Notes
TOO MANY DIGITS	Too many digits in a decimal value or a self-defining term.	SA	
UNDEFINED OPERATION CODE	Mnemonic operation code is not recognized as a valid IBM System/360 Model 20 operation code and is not contained in the macro library.	SIC	
UNDEFINED SYMBOL	A symbol has been referenced but it is not defined in the name field of any instruction.	SIA	
UNPAIRED AMPERSAND	Odd number of ampersands encountered in a constant. (Two ampersands must be specified for every ampersand wanted in a constant.)	SA	

Notes:

1. These messages refer to statements in the assembly listing which contain additional information. This additional information is: the macro-instruction name, and a pointer to the instruction in error within the generated macro routine.

2. Diagnostic messages for macro instructions may be caused by improper data in the macro instruction (for example, alphabetic characters supplied for a length specification).

Appendix I. Diagnostic Messages of the Macro Maintenance Program

MESSAGE	MEANING
CHARACTER VALUE TOO LONG	A character value of this statement is too long.
CHARACTER STRING TOO LONG	A character string of this statement is too long.
COLS 1 THRU 15 NOT BLANK	Columns 1 through 15 must be blank.
FORMAT ERROR IN NAME FLD	The name field of this statement has the wrong format.
ILLEGAL CONTIN PUNCH	Column 72 must be blank for this statement.
ILLEGAL OPERAND	An operand is not allowed for this statement.
ILLEGAL OPERATION CODE	The operation code of this statement is not allowed within a macro definition, or the operation code specified in a prototype statement is a machine instruction or an assembler statement.
ILLEGAL STATEMENT FORMAT	The format of the statement is illegal (has no operation code).
ILLEGAL CONTINUATION LINE	The continuation line is not permitted.
ILLEGAL SUBSTRING	A substring specification is not permitted for this statement.
INCORRECT CONT LINE	The continuation line is incorrect (column 16 of a continuation line is blank).
INV ARITHM TERM OF SUBSTR	The arithmetic term of a substring is invalid.
INV CHAR STR TERMINATION	The termination of the character string is incorrect.
INV FORMAT OF ARITHM EXPR	Arithmetic expression has an invalid format.
INV FORMAT OF LOGICAL EXP	The format of the logical expression is invalid.
INV OPERAND TERMINATION	The termination of the operand is incorrect.
INV PROTOTYPE STATEMENT	The prototype statement is incorrect.
INV RELATIONAL OPERATOR	An invalid relational operator (EQ, NE, GT, GE, LT, LE) is specified.
INV SET STATEMENT NAME	The name of a SET statement is invalid.
INV SUBSTR TERMINATION	The termination of a substring is incorrect.
INV SYMBOL TERMINATION	The termination of a symbol is incorrect.
INV SYMB PAR IN OPERAND	The operand contains an incorrect symbolic parameter.
INV SYMB PAR IN NAME FLD	The name field contains an incorrect symbolic parameter.
INV SYSVARSYM IN NAME FLD	The name field contains an incorrect system variable symbol.
INV SYSVARSYM IN OPERAND	The operand contains an incorrect system variable symbol.

MESSAGE	MEANING
INVALID ARITHMETIC TERM	The statement contains an arithmetic term which is invalid.
INVALID LOGICAL OPERATOR	An invalid logical operator is specified.
INVALID OPERATION CODE	The operation code for this statement is invalid.
INVALID MACRO STATEMENT	The macro statement specified is incorrect.
INVALID MNOTE OPERAND	The operand of an MNOTE statement is invalid.
INVALID OPERAND TYPE	Positional prototype statement with keyword or vice versa.
INVALID SEQUENCE SYMBOL	The sequence symbol of this statement is invalid.
INVALID STANDARD VALUE	The standard value specified for a keyword is incorrect.
MORE THAN 3 ATITHM TERMS	This statement contains more than 3 arithmetic terms.
MULTI DEFINED SYMB PARAM	A parameter is defined more than once.
NO END PAREN IN OPERAND	The terminating parenthesis of this operand is missing.
NO INIT ' IN CHAR STRING	The initial apostrophe of a character string is missing.
NO INIT ' IN 2ND CHAR VAL	The second character value has no initial apostrophe.
NO INIT PAREN IN OPERAND	The initial parenthesis of this operand is missing.
NO SEQ SYMB IN NAME FLD	The name field of an ANOP statement does not contain a sequence symbol.
OPERAND MISSING	The operand of this statement is missing.
OPERAND NOT CONTINUED	The continuation of an operand is missing.
OPERAND OVERFLOWS COL 71	The operand extends beyond column 71.
S *	The contents of columns 73 through 80 of this statement are out of sequence.
Sequence Symbol MULTI DEFINED	A sequence symbol is defined more than once.
Sequence Symbol NOT DEFINED	A sequence symbol should be defined.
TOO MANY SYMB PARAMETERS	This statement contains too many symbolic parameters.

* TPS only

Appendix J. Character Codes

This appendix lists all System/360 card codes to which a printer graphic is assigned. (The printer graphic may vary according to the national character set.)

EBCDIC CODE	CARD PUNCH COMBINATION	PRINTER GRAPHIC	DECIMAL	HEXADECIMAL
00000000	12,0,9,8,1		0	00
00000001	12,9,1		1	01
00000010	12,9,2		2	02
00000011	12,9,3		3	03
00000100	12,9,4		4	04
00000101	12,9,5		5	05
00000110	12,9,6		6	06
00000111	12,9,7		7	07
00001000	12,9,8		8	08
00001001	12,9,8,1		9	09
00001010	12,9,8,2		10	0A
00001011	12,9,8,3		11	0B
00001100	12,9,8,4		12	0C
00001101	12,9,8,5		13	0D
00001110	12,9,8,6		14	0E
00001111	12,9,8,7		15	0F
00010000	12,11,9,8,1		16	10
00010001	11,9,1		17	11
00010010	11,9,2		18	12
00010011	11,9,3		19	13
00010100	11,9,4		20	14
00010101	11,9,5		21	15
00010110	11,9,6		22	16
00010111	11,9,7		23	17
00011000	11,9,8		24	18
00011001	11,9,8,1		25	19
00011010	11,9,8,2		26	1A
00011011	11,9,8,3		27	1B
00011100	11,9,8,4		28	1C
00011101	11,9,8,5		29	1D
00011110	11,9,8,6		30	1E
00011111	11,9,8,7		31	1F
00100000	11,0,9,8,1		32	20
00100001	0,9,1		33	21
00100010	0,9,2		34	22
00100011	0,9,3		35	23
00100100	0,9,4		36	24
00100101	0,9,5		37	25
00100110	0,9,6		38	26
00100111	0,9,7		39	27
00101000	0,9,8		40	28
00101001	0,9,8,1		41	29
00101010	0,9,8,2		42	2A
00101011	0,9,8,3		43	2B
00101100	0,9,8,4		44	2C
00101101	0,9,8,5		45	2D
00101110	0,9,8,6		46	2E
00101111	0,9,8,7		47	2F

EBCDIC CODE	CARD PUNCH COMBINATION	PRINTER GRAPHIC	DECIMAL	HEXADECIMAL
00110000	12,11,0,9,8,1		48	30
00110001	9,1		49	31
00110010	9,2		50	32
00110011	9,3		51	33
00110100	9,4		52	34
00110101	9,5		53	35
00110110	9,6		54	36
00110111	9,7		55	37
00111000	9,8		56	38
00111001	9,8,1		57	39
00111010	9,8,2		58	3A
00111011	9,8,3		59	3B
00111100	9,8,4		60	3C
00111101	9,8,5		61	3D
00111110	9,8,6		62	3E
00111111	9,8,7		63	3F
01000000		blank	64	40
01000001	12,0,9,1		65	41
01000010	12,0,9,2		66	42
01000011	12,0,9,3		67	43
01000100	12,0,9,4		68	44
01000101	12,0,9,5		69	45
01000110	12,0,9,6		70	46
01000111	12,0,9,7		71	47
01001000	12,0,9,8		72	48
01001001	12,8,1		73	49
01001010	12,8,2		74	4A
01001011	12,8,3	.	75	4B
01001100	12,8,4	<	76	4C
01001101	12,8,5	(	77	4D
01001110	12,8,6	+	78	4E
01001111	12,8,7	|	79	4F
01010000	12	&	80	50
01010001	12,11,9,1		81	51
01010010	12,11,9,2		82	52
01010011	12,11,9,3		83	53
01010100	12,11,9,4		84	54
01010101	12,11,9,5		85	55
01010110	12,11,9,6		86	56
01010111	12,11,9,7		87	57
01011000	12,11,9,8		88	58
01011001	11,8,1		89	59
01011010	11,8,2	!	90	5A
01011011	11,8,3	$	91	5B
01011100	11,8,4	*	92	5C
01011101	11,8,5	)	93	5D
01011110	11,8,6	;	94	5E
01011111	11,8,7	¬	95	5F

EBCDIC CODE	CARD PUNCH COMBINATION	PRINTER GRAPHIC	DECIMAL	HEXADECIMAL
01100000	11	-	96	60
01100001	0,1	/	97	61
01100010	11,0,9,2		98	62
01100011	11,0,9,3		99	63
01100100	11,0,9,4		100	64
01100101	11,0,9,5		101	65
01100110	11,0,9,6		102	66
01100111	11,0,9,7		103	67
01101000	11,0,9,8		104	68
01101001	0,8,1		105	69
01101010	12,11		106	6A
01101011	0,8,3	,	107	6B
01101100	0,8,4	%	108	6C
01101101	0,8,5	_	109	6D
01101110	0,8,6	>	110	6E
01101111	0,8,7	?	111	6F
01110000	12,11,0		112	70
01110001	12,11,0,9,1		113	71
01110010	12,11,0,9,2		114	72
01110011	12,11,0,9,3		115	73
01110100	12,11,0,9,4		116	74
01110101	12,11,0,9,5		117	75
01110110	12,11,0,9,6		118	76
01110111	12,11,0,9,7		119	77
01111000	12,11,0,9,8		120	78
01111001	8,1		121	79
01111010	8,2	:	122	7A
01111011	8,3	#	123	7B
01111100	8,4	@	124	7C
01111101	8,5	'	125	7D
01111110	8,6	=	126	7E
01111111	8,7	"	127	7F
10000000	12,0,8,1		128	80
10000001	12,0,1		129	81
10000010	12,0,2		130	82
10000011	12,0,3		131	83
10000100	12,0,4		132	84
10000101	12,0,5		133	85
10000110	12,0,6		134	86
10000111	12,0,7		135	87
10001000	12,0,8		136	88
10001001	12,0,9		137	89
10001010	12,0,8,2		138	8A
10001011	12,0,8,3		139	8B
10001100	12,0,8,4		140	8C
10001101	12,0,8,5		141	8D
10001110	12,0,8,6		142	8E
10001111	12,0,8,7		143	8F

EBCDIC CODE	CARD PUNCH COMBINATION	PRINTER GRAPHIC	DECIMAL	HEXADECIMAL
10010000	12,11,8,1		144	90
10010001	12,11,1		145	91
10010010	12,11,2		146	92
10010011	12,11,3		147	93
10010100	12,11,4		148	94
10010101	12,11,5		149	95
10010110	12,11,6		150	96
10010111	12,11,7		151	97
10011000	12,11,8		152	98
10011001	12,11,9		153	99
10011010	12,11,8,2		154	9A
10011011	12,11,8,3		155	9B
10011100	12,11,8,4		156	9C
10011101	12,11,8,5		157	9D
10011110	12,11,8,6		158	9E
10011111	12,11,8,7		159	9F
10100000	11,0,8,1		160	A0
10100001	11,0,1		161	A1
10100010	11,0,2		162	A2
10100011	11,0,3		163	A3
10100100	11,0,4		164	A4
10100101	11,0,5		165	A5
10100110	11,0,6		166	A6
10100111	11,0,7		167	A7
10101000	11,0,8		168	A8
10101001	11,0,9		169	A9
10101010	11,0,8,2		170	AA
10101011	11,0,8,3		171	AB
10101100	11,0,8,4		172	AC
10101101	11,0,8,5		173	AD
10101110	11,0,8,6		174	AE
10101111	11,0,8,7		175	AF
10110000	12,11,0,8,1		176	B0
10110001	12,11,0,1		177	B1
10110010	12,11,0,2		178	B2
10110011	12,11,0,3		179	B3
10110100	12,11,0,4		180	B4
10110101	12,11,0,5		181	B5
10110110	12,11,0,6		182	B6
10110111	12,11,0,7		183	B7
10111000	12,11,0,8		184	B8
10111001	12,11,0,9		185	B9
10111010	12,11,0,8,2		186	BA
10111011	12,11,0,8,3		187	BB
10111100	12,11,0,8,4		188	BC
10111101	12,11,0,8,5		189	BD
10111110	12,11,0,8,6		190	BE
10111111	12,11,0,8,7		191	BF

EBCDIC CODE	CARD PUNCH COMBINATION	PRINTER GRAPHIC	DECIMAL	HEXADECIMAL
11000000	12,0		192	C0
11000001	12,1	A	193	C1
11000010	12,2	B	194	C2
11000011	12,3	C	195	C3
11000100	12,4	D	196	C4
11000101	12,5	E	197	C5
11000110	12,6	F	198	C6
11000111	12,7	G	199	C7
11001000	12,8	H	200	C8
11001001	12,9	I	201	C9
11001010	12,0,9,8,2		202	CA
11001011	12,0,9,8,3		203	CB
11001100	12,0,9,8,4		204	CC
11001101	12,0,9,8,5		205	CD
11001110	12,0,9,8,6		206	CE
11001111	12,0,9,8,7		207	CF
11010000	11,0		208	D0
11010001	11,1	J	209	D1
11010010	11,2	K	210	D2
11010011	11,3	L	211	D3
11010100	11,4	M	212	D4
11010101	11,5	N	213	D5
11010110	11,6	O	214	D6
11010111	11,7	P	215	D7
11011000	11,8	Q	216	D8
11011001	11,9	R	217	D9
11011010	12,11,9,8,2		218	DA
11011011	12,11,9,8,3		219	DB
11011100	12,11,9,8,4		220	DC
11011101	12,11,9,8,5		221	DD
11011110	12,11,9,8,6		222	DE
11011111	12,11,9,8,7		223	DF
11100000	0,8,2		224	E0
11100001	11,0,9,1		225	E1
11100010	0,2	S	226	E2
11100011	0,3	T	227	E3
11100100	0,4	U	228	E4
11100101	0,5	V	229	E5
11100110	0,6	W	230	E6
11100111	0,7	X	231	E7
11101000	0,8	Y	232	E8
11101001	0,9	Z	233	E9
11101010	11,0,9,8,2		234	EA
11101011	11,0,9,8,3		235	EB
11101100	11,0,9,8,4		236	EC
11101101	11,0,9,8,5		237	ED
11101110	11,0,9,8,6		238	EE
11101111	11,0,9,8,7		239	EF

EBCDIC CODE	CARD PUNCH COMBINATION	PRINTER GRAPHIC	DECIMAL	HEXADECIMAL
11110000	0	0	240	F0
11110001	1	1	241	F1
11110010	2	2	242	F2
11110011	3	3	243	F3
11110100	4	4	244	F4
11110101	5	5	245	F5
11110110	6	6	246	F6
11110111	7	7	247	F7
11111000	8	8	248	F8
11111001	9	9	249	F9
11111010	12,11,0,9,8,2		250	FA
11111011	12,11,0,9,8,3		251	FB
11111100	12,11,0,9,8,4		252	FC
11111101	12,11,0,9,8,5		253	FD
11111110	12,11,0,9,8,6		254	FE
11111111	12,11,0,9,8,7		255	FF

Appendix K. Minimum and Maximum System Configuration

Minimum System Configuration

The following is the minimum system configuration required to perform an assembly.

Submodel 2

An IBM 2020 Central Processing Unit Model C2 for the TPS (8,192 bytes of main storage), or BC2 for the DPS (12,288 bytes of main storage);

an IBM 2415 Magnetic Tape Unit Model 2 or 5 (with at least one 9-track drive) for the TPS, or

an IBM 2311 Disk Storage Drive Model 11 or 12 for the DPS;

one of the following card reading devices:

IBM 2501 Card Reader Model A1 or A2,
IBM 2520 Card Read-Punch Model A1,
IBM 2560 Multi-Function Card Machine Model A1;

one of the following printers:

IBM 1403 Printer Model N1, 2, or 7,
IBM 2203 Printer Model A1.

Submodel 4

An IBM 2020 Central Processing Unit Model BC4 (12,288 bytes of main storage);

an IBM 2311 Disk Storage Drive Model 12;

an IBM 2560 Multi-Function Card Machine Model A2;

an IBM 2203 Printer Model A2.

Submodel 5

An IBM 2020 Central Processing Unit Model C5 for the TPS (8,192 bytes of main storage), or BC5 for the DPS (12,288 bytes of main storage);

an IBM 2415 Magnetic Tape Unit Model 2 or 5 (with at least one 9-track drive) for the TPS, or

an IBM 2311 Disk Storage Drive Model 11 or 12 for the DPS;

one of the following card reading devices:

IBM 2501 Card Reader Model A1 or A2,
IBM 2520 Card Read-Punch Model A1,
IBM 2560 Multi-Function Card Machine Model A1;

one of the following printers:

IBM 1403 Printer Model N1, 2, or 7,
IBM 2203 Printer Model A1.

Maximum System Configuration

Assembler object programs may be produced for the following maximum system configuration.

Submodel 2

An IBM 2020 Central Processing Unit Model D2 (16,384 bytes of main storage); with or without IBM Binary Synchronous Communications Adapter, Feature No. 2074;

two IBM 2311 Disk Storage Drives Model 11 or 12 (both must be the same model);

an IBM 2415 Magnetic Tape Unit Model 1 through 6;

an IBM 2501 Card Reader Model A1 or A2;

an IBM 1442 Card Punch Model 5;

one of the following card units:

IBM 2520 Card Read-Punch Model A1,
IBM 2520 Card Punch Model A2 or A3,
IBM 2560 Multi-Function Card Machine Model A1;

one of the following printers:

IBM 1403 Printer Model N1, 2, or 7,
IBM 2203 Printer Model A1;

one of the following magnetic character readers:

IBM 1419 Magnetic Character Reader Model 1 or 31,

IBM 1259 Magnetic Character Reader Model 1, 31, or 32;

an IBM 2152 Printer-Keyboard.

Submodel 4

An IBM 2020 Central Processing Unit Model D4 (16,384 bytes of main storage); with or without IBM Binary Synchronous Communications Adapter, Feature No. 2074;

two IBM 2311 Disk Storage Drives Model 12;

an IBM 2560 Multi-Function Card Machine Model A2;

an IBM 2203 Printer Model A2.

an IBM 2152 Printer-Keyboard.

Submodel 5

An IBM 2020 Central Processing Unit Model E5 (32,768 bytes of main storage); with or without IBM Binary Synchronous Communications Adapter, Feature No. 2074;

four IBM 2311 Disk Storage Drives Model 11 or 12;

an IBM 2415 Magnetic Tape Unit Model 1 through 6;

an IBM 2501 Card Reader Model A1 or A2;

an IBM 1442 Card Punch Model 5;

one of the following card units:

IBM 2520 Card Read-Punch Model A1,
IBM 2520 Card Punch Model A2 or A3,
IBM 2560 Multi-Function Card Machine Model A1;

one of the following printers:

IBM 1403 Printer Model N1, 2, or 7,
IBM 2203 Printer Model A1;

one of the following magnetic character readers:

IBM 1419 Magnetic Character Reader Model 1 or 31,
IBM 1259 Magnetic Character Reader Model 1, 31, or 32;

an IBM 2152 Printer-Keyboard.

Appendix L. Hexadecimal—Decimal Number Conversion Table

The table in this appendix provides for direct conversion of decimal and hexadecimal numbers between 0000 and 4095 (hexadecimal 000 and FFF).

For numbers outside the range of the table, add the following values to the table figures:

Hexadecimal	Decimal	Hexadecimal	Decimal
1000	4096	9000	36864
2000	8192	A000	40960
3000	12288	B000	45056
4000	16384	C000	49152
5000	20480	D000	53248
6000	24576	E000	57344
7000	28672	F000	61440
8000	32768		

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3756	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	2828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

Appendix M. Sample Programs

DPS Assembler Language Program

The purpose of this sample program is to give a short demonstration of the application of the Assembler language.

Description of the Program

The program reads data cards from the 2501 Card Reader. Only the first 13 columns of each card are read. Input-card format is as follows:

Column	Contents
1	A,B or C (card-type indication)
2 - 4	a 3-digit, positive decimal number (Field1)
5 - 7	a 3-digit, positive decimal number (Field2)
8 - 10	a 3-digit, positive decimal number (Field3)
11 - 13	a 3-digit, positive decimal number (Field4)

The program first checks the input cards for correct format. If any one of the columns does not contain a decimal digit, the program halts and displays the number and digit in error by executing an HPR instruction. (Refer to the halt routines in part 11 of Figure 19). Then the next card is read in.

If the data is correct, the desired calculations are carried out. The calculations vary depending on the type of card as designated in column 1:

Type	Arithmetic Operation
A	Field1 + Field2 - Field3 * Field4
B	Field1 - Field2 * Field3 + Field4
C	Field1 * Field2 + Field3 - Field4

As each calculation is performed, a line of output is printed. The format of the printed line is:

Print Positions	Contents of Print Field
11 - 19	CARD TYPE
21	Type (A,B or C)
26 - 45	Arithmetic Operation
49	Equal Sign
52 - 60	Result

Program Organization

The program consists of two separate control sections, PART1 and PART2. The first section, PART1 (see parts 2 and 3 of Figure 19), contains all the input/output (IOCS) routines. These routines consist of the instructions required to read the data cards (READ) and to print the results (PRINT). The IOCS routines will not be discussed in detail here. For further information, refer to the SRL publication IBM System/360 Model 20, Disk Programming System, Input/Output Control System, Form GC24-9007.

The second section, PART2 (see part 6 of Figure 19), contains a number of small routines that test the contents of the cards and perform the desired calculations. These routines, which make up the main program, are quite similar to each other. Compare, for example, the routines beginning with the names TSTRICOL, TSTMICOL, TSTLECOL and TRICOL, TMICOL, TLECOL (parts 7 and 8 of Figure 19). Although the source code is different for each of these routines, the Assembler produces the same object code. This is demonstrated even more clearly by the statements 0202, 0234 and 0264, which read as follows:

```
0202:  OI  59(PRINTRG),X'F0'
0234:  OI  DPRINTAR+59,B'11110000'
0264:  OI  RESZ-L'RESZ-1,C'0'
```

Here, both operands are expressed differently in each of the three source instructions. However, their object code is exactly the same.

Linking PART1 and PART2

The two sections of the program, PART1 and PART2, are assembled separately. They are brought together to form a single program by means of the Linkage Editor Program. Hence, the symbols which are common to both sections must be defined by ENTRY instructions in the section in which they appear as a name, and by EXTRN instructions in the section in which they are referred to.

If we take the name OPENF as an example, we find that it is defined in the first Assembly as

```
ENTRY  OPENF
```

The address, as shown in the External Symbol Dictionary, is hexadecimal 017C. In the second Assembly, this name is defined as

```
EXTRN  OPENF
```

In this assembly, the Assembler uses the value of hexadecimal 0000 for OPENF. The Linkage Editor then inserts the actual value. This is the address from the ESD of the first assembly, plus the relocation factor, which can be found in the Linkage Editor listing output in column REL-FR (part 17 of Figure 19). Thus, the actual address inserted by the Linkage Editor is

```
        X'017C'  ESD
        X'0DBE'  relocation factor (REL-FR)
OPENF - X'0F3A'
        =======
```

In addition to the ENTRY and EXTRN instructions, the Link register (LINKRG), which is used for branching between the main program (PART2) and the IOCS routines (PART1), must be defined in both sections. The same register must be specified in each section (in our case, register 13).

Control Statements

The Job Control cards needed for the Linkage Editor run are produced before PART1 of the program. This is done by means of REPRO statements at assembly time. These cards also include the PHASE card. The letter S following the phase name causes the program to be loaded immediately behind the Monitor at the time of program execution. (Refer to the publication IBM System/360 Model 20, Disk Programming System, Control and Service Programs, Form GC24-9006.

Since the four cards that are produced by the Assembler come before the START card in the source deck, they are also in front of the ESD (External Symbol Dictionary) cards in the object deck produced by the Assembler.

Addressing in the Main Program

In the main program (PART2) two base registers are used - BASERG1 and BASERG2 - in order to show you the use of the USING and DROP instructions.

From BEGIN (statement 0052) to TYPEA (statement 0090) the Assembler uses register 10 (BASERG1) as a base register and assumes the base address BASEA11. In the routine TYPEA, register 10 is still used as the base register, but its contents are changed to the base address BASEA12.

For routine TYPEB, the Assembler has both base registers at its disposal, but it uses only one of them, BASERG2, because this provides the smaller displacement.

Routine CEXPRA uses BASERG1 again because BASERG2 has been made unavailable by means of a DROP instruction.

When using the DROP instruction, take care that the register to be dropped is no longer needed in that routine. Otherwise, an addressability error will occur. If the register that has been dropped is required again at a later point in the program, the Assembler must be informed of this by means of a further USING instruction.

In routine CEXPRB exactly this has been done; BASERG2 has been made available again. Since this base register provides a smaller displacement, it is used by the Assembler until the end of the program, with the exception of the instructions for branching back to the GETCARD routine (statements 308 - 310).

GETCARD was defined before the base addresses BASEA12 and BASEA21. Therefore its address is lower than that contained in the two base registers at this point. Hence, BASERG1 must be loaded with the original base address BASEA11 before a branch to GETCARD takes place.

Loading and re-loading of the base registers takes place when the program is executed. Both base registers are loaded initially in the routine headed LOAD BASE REGISTERS with the addresses BASEA11 and BASEA21, respectively. The contents of BASERG2 are not changed during the whole of the program. On the other hand, BASERG1 is loaded with the address BASEA12 in the TYPEA routine and re-loaded with its original value, BASEA11, in the BGETCARD routine.

Addressing the Input and Output Areas

To address the card and print areas, the registers CARDRG and PRINTRG (general registers 12 and 15), respectively, are used.

The two areas can be utilized either by explicit addressing (specifying base and displacement), as in the routine CEXPRA, or by implicit addressing, as in the routines CEXPRB and CEXPRC. In the latter case the areas are defined in a Dummy Control Section. The Assembler is informed via a USING instruction that register CARDRG is used to address the first Dummy Control Section (DCS1) and that register PRINTRG is used to address the second Dummy Control

Section (DCS2). This is done at the beginning of the main program (statements 0056 and 0057). Of course, CARDRG and PRINTRG must be loaded when the program is executed.

Formatting the Instruction Listing

To make the program listings more readable, related instructions have been grouped into routines and subroutines and spaced out accordingly by using SPACE, EJECT, and TITLE cards. Compare the listing of the source deck (Figure 20) with the assembly listing (Figure 19) to see the effect of the respective source statement.

For example, statements SP1 058 and SP2 087 in Figure 20 are not listed in Figure 19 but the appropriate spacing was performed by the Assembler during execution. Likewise, the TITLE statement of source statement SP2 045 in Figure 20 is not listed in the assembly listing but the effect of the execution of this formatting statement is obvious.

Look at Figure 19 part 5; the last statement listed is SP2 044. The first statement listed in Figure 19 part 6 is SP2 046. The TITLE statement caused a skip to the next page with the new heading printed on top of that page. An EJECT statement causes a skip to the next page without changing the heading. Compare statements SP2 075 - SP2 077 in Figure 20 with the same statements in Figure 19.

Cross Reference List

The Cross Reference List contains useful information such as: the lengths of storage areas; the address of a particular symbol; the numbers of those statements which refer to the symbol in question. The information contained in the table is listed under headings as follows:

SYMBOL: Under this heading, the Assembler lists, in alphabetical order, all symbols in the program.

LEN: In this column you can find the appropriate length attribute of the symbol.

VALUE: This column contains the hexadecimal value of each symbol.

DEF: This column shows the number of the statement that defines the particular symbol. You can find this number under the heading STMT of the instruction listing

CROSS REFERENCE: Under this heading, the Assembler gives a list of the statement numbers, in which the symbol in question is referred to. You can find these numbers under the heading STMT of the instruction listing.

Below are three examples to explain the use of the Cross Reference List.

Example 1. Let us assume you wish to find out from which points in the program a branch is made to the entry point BGETCARD (statement 0308). To find this symbol, look down the column SYMBOL in the Cross Reference List until you find BGETCARD. The column DEF confirms that the symbol was defined in statement 0308. Now look under CROSS REFERENCE. There you find that the symbol was used in statements 0290 and 0295.

Column LEN in the Cross Reference List contains the number 4 for the symbol BGETCARD. The Assembler has assigned BGETCARD a length attribute of 4 because the machine instruction at statement 0308 takes up four bytes of main storage.

The value X'02E6' for BGETCARD under VALUE is the relative storage address of the first byte of the machine instruction at statement 0308.

Example 2. Suppose you want to know which register is meant by BASERG2 (refer to statement 0206) and where it is loaded.

You look for BASERG2 in the column SYMBOL of the Cross Reference List and find in the column VALUE hexadecimal 000B. This means that the symbol BASERG2 stands for register 11.

The symbol is defined in statement 0026, as you can find in column DEF. Under CROSS-REFERENCE you can find that the symbol is used in statements 0055, 0120, 0174 and 0206.

If you look up these statements you will find that register 11 is loaded by the machine instruction at statement 0055.

Example 3. Statement 0263 uses symbol RESZ as its first operand and RESP as its second operand. Both operands use an implied length. What lengths are inserted by the Assembler?

Now look for RESZ under SYMBOL in the Cross Reference List. The column LEN gives a length attribute of 9 since the area defined by RESZ is nine bytes long. We can check this by looking at the statement in which RESZ is defined, i.e., statement 0375.

Similarly, we find that the symbol RESP has a length attribute of 5.

```
// LOG
// JOB ASSEMB
// DATE 70020
// ASSGN SYSIPT,X'100',R4          2501
// ASSGN SYSOPT,X'300',P2          1442
// ASSGN SYS000,X'803',D4
// VOL SYS000,WORK1
// DLAB  'SYSTEM/360 MOD 20 DPS                          1202020',P       C
               0001,67150,67150
// XTENT 1,000,0153000,0199009,'202020',SYS000
// EXEC
MOD 20 DPS ASSEMBLER VERSION 03  MOD-LEVEL 02
```

EXTERNAL SYMBOL DICTIONARY

SYMBOL	TYPE	ID	ADDR	LENGTH	LD ID
PART1	SD	01	0000	01F8	
OPENF	LD		017C		01
READ	LD		0186		01
PRINT	LD		018E		01
CLOSEF	LD		0196		01
CARDAR	LD		01AF		01
PRINTAR	LD		01BC		01
EOCF	ER	02			

Figure 19. DPS Sample Program, Part 1 of 17

```
SLE                                                                    01/20/70  PAGE  001

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                        DPS ASSEMB 03/02

                              0001          AOPTN NOSYM                                    SP1 001
                              0002 *                                                       SP1 002
                              0003 ************************************************************ SP1 003
                              0004 *    PREPARE CONTROL CARDS FOR LINKAGE EDITOR RUN       SP1 004
                              0005 ************************************************************ SP1 005
                              0006 *                                                       SP1 006
                              0007          REPRO                                          SP1 007
                                   // JOB LNKEDT                                          SP1 008
                              0008          REPRO                                          SP1 009
                                   // ASSGN SYSOPT,UA          OUTPUT ONLY IN RELOCATABLE AREA SP1 010
                              0009          REPRO                                          SP1 011
                                   // EXEC                                                SP1 012
                              0010          REPRO                                          SP1 013
                                    PHASE SAMPLE,S,0          START JUST BEHIND THE MONITOR  SP1 014
```

```
SLE    ASSEMBLER SAMPLE, PART 1, IOCS                                  01/20/70  PAGE  002

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                        DPS ASSEMB 03/02

                              0012 ************************************************************ SP1 016
 0000                         0013 PART1    START 0                  FIRST PART OF THE SAMPLE  SP1 017
                              0014 ************************************************************ SP1 018


                              0016 ************************************************************ SP1 020
                              0017 *    DEFINE ENTRIES AND EXTERNAL SYMBOLS                 SP1 021
                              0018 ************************************************************ SP1 022
                              0019 *                                                       SP1 023
 017C                         0020          ENTRY OPENF              OPEN ROUTINE          SP1 024
 0186                         0021          ENTRY READ               READ ROUTINE          SP1 025
 018E                         0022          ENTRY PRINT              PRINT ROUTINE         SP1 026
 0196                         0023          ENTRY CLOSEF             CLOSE ROUTINE         SP1 027
 01AF                         0024          ENTRY CARDAR             AREA WHERE CARD IS STORED SP1 028
 01BC                         0025          ENTRY PRINTAR            PRINT AREA            SP1 029
                              0026 *                                                       SP1 030
                              0027          EXTRN EOCF               END OF CARD FILE      SP1 031


                              0029 ************************************************************ SP1 033
                              0030 *    SYMBOLIC REGISTER DEFINITIONS                       SP1 034
                              0031 ************************************************************ SP1 035
                              0032 *                                                       SP1 036
 000D                         0033 LINKRG   EQU   13                 LINK-REGISTER         SP1 037


                              0035 ************************************************************ SP1 039
                              0036 *    DEFINE THE FILES                                    SP1 040
                              0037 ************************************************************ SP1 041
                              0038 *                                                       SP1 042
                              0039          PRINT NOGEN              DON'T PRINT GENERATED STATEMENTS SP1 043
                              0040 *                                                       SP1 044
                              0041 READER   DTFSR TYPEFLE=INPUT,BLKSIZE=13,DEVICE=READ01,EOFADDR=EOCF,  CSP1 045
                              0042              IOAREA1=INAREA,OVERLAP=NO,WORKA=YES          SP1 046
                              0092 *                                                       SP1 047
                              0093 PRINTER  DTFSR TYPEFLE=OUTPUT,WORKA=YES,DEVICE=PRINTER,BLKSIZE=60 SP1 048
                              0132 *                                                       SP1 049
                              0133          DTFEN                                          SP1 050


                              0281 ************************************************************ SP1 052
                              0282 *    OPEN THE FILES                                      SP1 053
                              0283 ************************************************************ SP1 054
                              0284 *                                                       SP1 055
                              0285 OPENF    OPEN  READER,PRINTER     OPEN READER AND PRINTER SP1 056
 0184 07FD                    0291          BR    LINKRG             RETURN                SP1 057  048


                              0293 ************************************************************ SP1 059
                              0294 *    READ A CARD                                         SP1 060
                              0295 ************************************************************ SP1 061
                              0296 *                                                       SP1 062
                              0297 READ     GET   READER,CARDAR      READ A CARD AND MOVE IT TO CARDAR SP1 063
 018C 07FD                    0302          BR    LINKRG             RETURN                SP1 064  049
```

Figure 19. DPS Sample Program, Part 2 of 17

```
SLE    ASSEMBLER SAMPLE, PART 1, IOCS                                                    01/20/70  PAGE  003

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                                  DPS ASSEMB 03/02

                                0304 ********************************************************************** SP1 066
                                0305 *    PRINT A LINE                                                      SP1 067
                                0306 ********************************************************************** SP1 068
                                0307 *                                                                      SP1 069
                                0308 PRINT    PUT   PRINTER,PRINTAR     PRINT A LINE                      SP1 070
019A 07FD                       0313          BR    LINKRG              RETURN                            SP1 071  050

                                0315 ********************************************************************** SP1 073
                                0316 *    CLOSE THE FILES AND GO BACK TO MONITOR (END OF JOB)               SP1 074
                                0317 ********************************************************************** SP1 075
                                0318 *                                                                      SP1 076
                                0319 CLOSEF   CLOSE READER,PRINTER      CLOSE READER AND PRINTER          SP1 077
                                0325 *                                                                      SP1 078
                                0326          PRINT GEN                 PRINT GENERATED STATEMENTS        SP1 079
                                0327 *                                                                      SP1 080
                                0328          EOJ   ,                   GIVE CONTROL BACK TO MONITOR      SP1 081
019E 47F0 00C2       00C2       0329+         BC    15,194(0,0)                                           EOJ  012 053

                                0331 ********************************************************************** SP1 083
                                0332 *    AREAS                                                             SP1 084
                                0333 ********************************************************************** SP1 085
                                0334 *                                                                      SP1 086
01A2                            0335 INAREA   DS    13C                 INPUT AREA FOR IOCS               SP1 087
01AF                            0336 CARDAR   DS    CL13                CARD IS STORED HERE AFTER GET     SP1 088
01BC                            0337 PRINTAR  DS    CL60                PRINT AREA                        SP1 089

                                0339          END                                                           SP1 091  055
```

NO STATEMENT FLAGGED

Figure 19. DPS Sample Program, Part 3 of 17

```
                                    RELOCATION DICTIONARY

 POS.ID REL.ID FLGS ADDR

   01     01    04  0000
   01     01    04  0002
   01     02    04  0004
   01     01    04  0008
   01     01    04  0010
   01     01    04  0034
   01     01    04  0046
   01     01    04  0068
   01     01    04  00AC
   01     01    04  00CA
   01     01    04  00CE
   01     01    04  00D2
   01     01    04  00D6
   01     01    04  00E2
   01     01    04  0136
   01     01    04  015A
   01     01    04  016C
   01     01    04  0170
   01     01    04  017E
   01     01    04  01B2
   01     01    04  017E
   01     01    04  0182
   01     01    04  0188
   01     01    04  018A
   01     01    04  0190
   01     01    04  0192
   01     01    04  0198
   01     01    04  019C
```

```
          END OF JOB
// JOB ASSEMB                        SECOND ASSEMBLY
// EXEC
MOD 20 DPS ASSEMBLER VERSION 03  MOD-LEVEL 02
```

```
                                    EXTERNAL SYMBOL DICTIONARY

 SYMBOL   TYPE ID  ADDR  LENGTH LD ID

PART2     SD  01  0000   0318
OPENF     ER  02
READ      ER  03
PRINT     ER  04
CLOSEF    ER  05
CARDAR    ER  06
PRINTAR   ER  07
EOCF      LD      02EE           01
```

Figure 19. DPS Sample Program, Part 4 of 17

```
 SLE                                                                    01/20/70  PAGE  001
LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                DPS ASSEMB 03/02
                              0001          AOPTN CROSSREF,ENTRY                    SP2 001
```

```
 SLE    ASSEMBLER SAMPLE, PART 2, DEFINITIONS                           01/20/70  PAGE  002
LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                DPS ASSEMB 03/02
                              0003 ************************************************************************ SP2 003
0000                          0004 PART2    START 0                   SECOND PART OF SAMPLE                  SP2 004
                              0005 ************************************************************************ SP2 005


                              0007 ************************************************************************ SP2 007
                              0008 *    DEFINE ENTRIES AND EXTERNAL SYMBOLS                                  SP2 008
                              0009 ************************************************************************ SP2 009
                              0010 *                                                                         SP2 010
                              0011          EXTRN OPENF               OPEN ROUTINE                         SP2 011
                              0012          EXTRN READ                READ ROUTINE                         SP2 012
                              0013          EXTRN PRINT               PRINT ROUTINE                        SP2 013
                              0014          EXTRN CLOSEF              CLOSE ROUTINE                        SP2 014
                              0015          EXTRN CARDAR              AREA WHERE CARD IS STORED            SP2 015
                              0016          EXTRN PRINTAR             PRINT AREA                           SP2 016
                              0017 *                                                                         SP2 017
02EE                          0018          ENTRY EOCF                END OF CARD FILE                     SP2 018


                              0020 ************************************************************************ SP2 020
                              0021 *    SYMBOLIC REGISTER DEFINITIONS                                        SP2 021
                              0022 ************************************************************************ SP2 022
                              0023 *                                                                         SP2 023
0008                          0024 REG8     EQU   8                   REG. 8                               SP2 024
000A                          0025 BASERG1  EQU   10                  BASE-REGISTER 1                      SP2 025
000B                          0026 BASERG2  EQU   11                  BASE-REGISTER 2                      SP2 026
000D                          0027 LINKRG   EQU   13                  LINK-REGISTER                        SP2 027
000E                          0028 BRANCHRG EQU   14                  BRANCH-REGISTER                      SP2 028
000C                          0029 CARDRG   EQU   12                  REGISTER FOR CARD AREA               SP2 029
000F                          0030 PRINTRG  EQU   15                  REGISTER FOR PRINT AREA              SP2 030


                              0032 ************************************************************************ SP2 032
                              0033 *    EQUATE SYMBOLS                                                       SP2 033
                              0034 ************************************************************************ SP2 034
                              0035 *                                                                         SP2 035
0008                          0036 EQUAL    EQU   8                   CONDITION  EQUAL                     SP2 036
000B                          0037 NOTLOW   EQU   11                  CONDITION  EQUAL AND HIGH            SP2 037
                              0038 *                                                                         SP2 038
00C1                          0039 TA       EQU   C'A'                TYPE A                               SP2 039
00C2                          0040 TB       EQU   C'B'                TYPE B                               SP2 040
00C3                          0041 TC       EQU   C'C'                TYPE C                               SP2 041
00F0                          0042 CHAR0    EQU   C'0'                CHARACTER 0                          SP2 042
00F9                          0043 CHAR9    EQU   C'9'                CHARACTER 9                          SP2 043
0040                          0044 BLANK    EQU   C' '                CHARACTER 'BLANK'                    SP2 044
```

Figure 19. DPS Sample Program, Part 5 of 17

```
SLE    ASSEMBLER SAMPLE, PART 2, MAIN PROGRAM                                    01/20/70  PAGE  003

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                          DPS ASSEMB 03/02

                             0046 ********************************************************************** SP2 046
                             0047 *    MAIN PROGRAM                                                      SP2 047
                             0048 ********************************************************************** SP2 048
                             0049 *                                                                      SP2 049
                             0050 *    LOAD BASE REGISTER                                                SP2 050
                             0051 *                                                                      SP2 051
0000 0DA0                    0052 BEGIN    BASR  BASERG1,0          LOAD BASE-REGISTER 1 AND          SP2 052  009
0002                         0053          USING *,BASERG1          INFORM ASSEMBLER                  SP2 053
0002                         0054 BASEA11  EQU   *                  BASE-ADDRESS 1,1                  SP2 054
0002 48B0 A108       010A    0055          LH    BASERG2,=Y(BASEA21) LOAD BASE-REGISTER 2              SP2 055  009
0000                         0056          USING DCARDAR,CARDRG             FOR IMPLICIT ADDRESSING    SP2 056
0000                         0057          USING DPRINTAR,PRINTRG           FOR IMPLICIT ADDRESSING    SP2 057


                             0059 *    OPEN CARD AND PRINTER FILE                                        SP2 059
                             0060 *                                                                      SP2 060
0006 48E0 A10A       010C    0061          LH    BRANCHRG,=Y(OPENF)  LOAD REG. WITH BRANCH ADDRESS     SP2 061  009
000A 0DDE                    0062          BASR  LINKRG,BRANCHRG     OPEN READER AND PRINTER AND RETURN SP2 062  009
000C 47F0 A2AE       02B0    0063          B     PRNTLINE            PRINT A BLANK LINE                SP2 063  009


                             0065 *    READ A CARD                                                       SP2 065
                             0066 *                                                                      SP2 066
0010 48E0 A10C       010E    0067 GETCARD  LH    BRANCHRG,=Y(READ)   LOAD REG. WITH BRANCH ADDRESS     SP2 067  009
0014 0DDE                    0068          BASR  LINKRG,BRANCHRG     BRANCH TO READ ROUTINE AND RETURN SP2 068  009


                             0070 *    BLANK PRINTAREA AND RESET PACKED RESULT FIELD                     SP2 070
                             0071 *                                                                      SP2 071
0016 48F0 A10E       0110    0072          LH    PRINTRG,=Y(PRINTAR) LOAD ADDRESS OF PRINT AREA        SP2 072  009
001A 9240 F000       0000    0073          MVI   0(PRINTRG),C' '     BLANK IN FIRST BYTE OF PRINT AREA  SP2 073  009
001E D23A F001 F000  0001 0000 0074          MVC   1(L'DPRINTAR-1,PRINTRG),0(PRINTRG) BLANK THE REST   SP2 074  009
0024 F840 A306 A118  0308 011A 0075          ZAP   RESP,=P'0'          INITIALIZE RESULT FIELD           SP2 075  009
```

Figure 19. DPS Sample Program, Part 6 of 17

```
LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                                  DPS ASSEMB 03/02

                             0077 *    TEST TYPE OF CARD                                          SP2 077
                             0078 *                                                               SP2 078
002A 48C0 A110       0112    0079          LH    CARDRG,=Y(CARDAR)   INITIALIZE REGISTER         SP2 079  009
002E 95C1 C000       0000    0080          CLI   0(CARDRG),TA        TEST IF TYPE A              SP2 080  009
0032 4780 A048       004A    0081          BC    8,TYPEA             YES                         SP2 081  009
0036 95C2 C000       0000    0082          CLI   0(CARDRG),TB        IF TYPE B                   SP2 082  010
003A 4780 A0A2       00A4    0083          BC    EQUAL,TYPEB         YES                         SP2 083  010
003E 95C3 C000       0000    0084          CLI   0(CARDRG),TC        IF TYPE C                   SP2 084  010
0042 4780 A0AA       00AC    0085          BE    TYPEC               YES                         SP2 085  010
0046 47F0 A2B8       02BA    0086          BC    15,HALT0001         GO TO HALT ROUTINES         SP2 086  010


                             0088 *    TEST FOR DEC. NUMBERS IN THE CARD ENTRIES                  SP2 088
                             0089 *                                                               SP2 089
004A 0DA0                    0090 TYPEA    BASR  BASERG1,0           CHANGE CONTENTS OF BASE-REGISTER 1  SP2 090  010
004C                         0091          USING *,BASERG1         AND INFORM THE ASSEMBLER    SP2 091
004C                         0092 BASEA12  EQU   *                 BASE-ADDRESS 1,2            SP2 092
                             0093 *                                                               SP2 093
004C 4880 A0C8       0114    0094          LH    8,=H'1'             INITIALIZE COUNTER          SP2 094  010
0050 95F9 C003       0003    0095 TSTRICOL CLI   3(CARDRG),249       COMPARE RIGHT COL WITH 9    SP2 095  010
0054 4720 A28E       02DA    0096          BH    HALT003N            INVALID IF HIGH             SP2 096  010
0058 95F0 C003       0003    0097          CLI   3(CARDRG),240       COMPARE WITH 0              SP2 097  010
005C 4740 A28E       02DA    0098          BL    HALT003N            INVALID IF LOW              SP2 098  010
                             0099 *                                                               SP2 099
0060 95F9 C002       0002    0100 TSTMICOL CLI   2(CARDRG),C'9'      COMPARE MIDDLE COL WITH 9   SP2 100  010
0064 4720 A282       02CE    0101          BH    HALT002N            INVALID IF HIGH             SP2 101  010
0068 95F0 C002       0002    0102          CLI   2(CARDRG),C'0'      COMPARE WITH 0              SP2 102  010
006C 47B0 A02C       0078    0103          BNL   TSTLECOL            O. K. IF NOT LOW            SP2 103  010
0070 9540 C002       0002    0104          CLI   2(CARDRG),C' '      IF BLANK                    SP2 104  011
0074 4770 A282       02CE    0105          BNE   HALT002N            NO, INVALID                 SP2 105  011
                             0106 *                                                               SP2 106
0078 95F9 C001       0001    0107 TSTLECOL CLI   1(CARDRG),X'F9'     COMPARE LEFT COL. WITH 9    SP2 107  011
007C 4720 A276       02C2    0108          BH    HALT001N            INVALID IF HIGH             SP2 108  011
0080 95F0 C001       0001    0109          CLI   1(CARDRG),X'F0'     COMPARE WITH 0              SP2 109  011
0084 47B0 A044       0090    0110          BNL   TSTIFIN             O. K. IF NOT LOW            SP2 110  011
0088 9540 C001       0001    0111          CLI   1(CARDRG),X'40'     IF BLANK                    SP2 111  011
008C 4770 A276       02C2    0112          BNE   HALT001N            NO, INVALID                 SP2 112  011
                             0113 *                                                               SP2 113
0090 4980 A0CA       0116    0114 TSTIFIN  CH    8,=H'4'             ALL ENTRIES CHECKED         SP2 114  011
0094 47B0 A0D0       011C    0115          BNL   CEXPRA              YES, COMPUTE EXPRESSION TYPE A  SP2 115  011
0098 4A80 A0C8       0114    0116          AH    8,=H'1'             INCREMENT COUNTER BY 1      SP2 116  011
009C 4AC0 A0CC       0118    0117          AH    CARDRG,=H'3'        NEXT ENTRY                  SP2 117  011
00A0 47F0 A004       0050    0118          B     TSTRICOL            LOOP                        SP2 118  011
```

Figure 19. DPS Sample Program, Part 7 of 17

```
SLE    ASSEMBLER SAMPLE, PART 2, MAIN PROGRAM                                    01/20/70  PAGE  005

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                          DPS ASSEMB 03/02

 00B4                          0120          USING BASEA21,BASERG2                                  SP2 120
 00A4 4DD0 B000       00B4     0121 TYPEB    BAS   LINKRG,TESTENTR  BRANCH TO SUBROUTINE AND RETURN SP2 121  011
 00A8 47F0 B0DA       018E     0122          B     CEXPRB           BRANCH TO SUBROUTINE            SP2 122  012
                               0123 *                                                               SP2 123
                               0124 *                                                               SP2 124
 00AC 4DD0 B000       00B4     0125 TYPEC    BAS   LINKRG,TESTENTR  BRANCH TO SUBROUTINE AND RETURN SP2 125  012
 00B0 47F0 B14C       0200     0126          B     CEXPRC           BRANCH TO SUBROUTINE            SP2 126  012
                               0127 *                                                               SP2 127
                               0128 *                                                               SP2 128
 00B4                          0129 BASEA21  EQU   *                BASE-ADDRESS 2,1                SP2 129
                               0130 *                                                               SP2 130
 00B4 4880 B240       02F4     0131 TESTENTR LH    REG8,H1          INITIALIZE REGISTER             SP2 131  012
 00B8 95F9 C003       0003     0132 TRICOL   CLI   ENTRY+2,CHAR9    COMPARE RIGHT COL WITH 9        SP2 132  012
 00BC 4720 B226       02DA     0133          BH    HALT003N         HIGH                            SP2 133  012
 00C0 95F0 C003       0003     0134          CLI   ENTRY+2,CHAR0    WITH 0                          SP2 134  012
 00C4 4740 B226       02DA     0135          BL    HALT003N         LOW                             SP2 135  012
                               0136 *                                                               SP2 136
 00C8 95F9 C002       0002     0137 TMICOL   CLI   ENTRY+1,CHAR9    COMPARE MIDDLE COL WITH 9       SP2 137  012
 00CC 4720 B21A       02CE     0138          BH    HALT002N         HIGH                            SP2 138  012
 00D0 95F0 C002       0002     0139          CLI   ENTRY+1,CHAR0    WITH 0                          SP2 139  012
 00D4 47B0 B02C       00E0     0140          BNL   TLECOL           O. K.                           SP2 140  012
 00D8 9540 C002       0002     0141          CLI   ENTRY+1,BLANK    IF BLANK                        SP2 141  012
 00DC 4770 B21A       02CE     0142          BNE   HALT002N         NO                              SP2 142  012
                               0143 *                                                               SP2 143
 00E0 95F9 C001       0001     0144 TLECOL   CLI   ENTRY,CHAR9      COMPARE LEFT COL WITH 9         SP2 144  013
 00E4 4720 B20E       02C2     0145          BH    HALT001N         HIGH                            SP2 145  013
 00E8 95F0 C001       0001     0146          CLI   ENTRY,CHAR0      WITH 0                          SP2 146  013
 00EC 47B0 B044       00F8     0147          BNL   TIFIN            O. K.                           SP2 147  013
 00F0 9540 C001       0001     0148          CLI   ENTRY,BLANK      IF BLANK                        SP2 148  013
 00F4 4770 B20E       02C2     0149          BNE   HALT001N         NO                              SP2 149  013
                               0150 *                                                               SP2 150
 00F8 4980 B244       02F8     0151 TIFIN    CH    REG8,H4          ALL ENTRIES CHECKED             SP2 151  013
 00FC 07BD                     0152          BCR   NOTLOW,LINKRG    YES                             SP2 152  013
 00FE 4A80 B240       02F4     0153          AH    REG8,H1          INCREMENT COUNTER BY 1          SP2 153  013
 0102 4AC0 B242       02F6     0154          AH    CARDRG,H3        NEXT ENTRY                      SP2 154  013
 0106 47F0 B004       00B8     0155          B     TRICOL           LOOP                            SP2 155  013


                               0157 *    LITERAL POOL                                               SP2 157
                               0158 *                                                               SP2 158
 010A                          0159          LTORG                                                  SP2 159
 010A 00B4                     0160     #  Y(BASEA21)                                                      013
 010C 0000                     0161     #  Y(OPENF)                                                        013
 010E 0000                     0162     #  Y(READ)                                                         013
 0110 0000                     0163     #  Y(PRINTAR)                                                      013
 0112 0000                     0164     #  Y(CARDAR)                                                       013
 0114 0001                     0165     #  H'1'                                                            013
 0116 0004                     0166     #  H'4'                                                            013
 0118 0003                     0167     #  H'3'                                                            014
 011A 0C                       0168     #  P'0'                                                            014
```

Figure 19. DPS Sample Program, Part 8 of 17

```
LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                                        DPS ASSEMB 03/02

                             0170 *    COMPUTE EXPRESSIONS, TYPE A ...  ENTRY1 + ENTRY2 - ENTRY3 * ENTRY4 SP2 161
                             0171 *                         TYPE B ...  ENTRY1 - ENTRY2 * ENTRY3 + ENTRY4 SP2 162
                             0172 *                         TYPE C ...  ENTRY1 * ENTRY2 + ENTRY3 - ENTRY4 SP2 163
                             0173 *                                                                SP2 164
                             0174          DROP  BASERG2                                           SP2 165
                             0175 *                                                                SP2 166
 011C 48C0 A2B2      02FE    0176 CEXPRA   LH    CARDRG,YCARDAR            INITIALIZE REGISTER   SP2 167  014
 0120 48F0 A2B4      0300    0177          LH    PRINTRG,YPRINTAR          INITIALIZE REGISTER   SP2 168  014
                             0178 *                                                                SP2 169
 0124 D208 F00A A2C3 000A 030F 0179          MVC   10(9,PRINTRG),MESS        MESSAGE TO PRINT AREA SP2 170  014
 012A 92C1 F014      0014    0180          MVI   20(PRINTRG),TA            TYPE TO PRINTAREA     SP2 171  014
                             0181 *                                                                SP2 172
 012E D202 F019 C001 0019 0001 0182          MVC   25(3,PRINTRG),1(CARDRG)   NUMBER 1 TO PRINT AREA SP2 173  014
 0134 F212 A2BF C001 030B 0001 0183          PACK  RESP+3(2),1(3,CARDRG)     PACK NUMBER 1 (N1)    SP2 174  014
                             0184 *                                                                SP2 175
 013A 924E F01D      001D    0185          MVI   29(PRINTRG),C'+'          SIGN TO PRINT AREA    SP2 176  014
 013E D202 F01F C004 001F 0004 0186          MVC   31(3,PRINTRG),4(CARDRG)   NUMBER 2 TO PRINT AREA SP2 177  014
 0144 F212 A2C1 C004 030D 0004 0187          PACK  PFLD,4(3,CARDRG)          PACK NUMBER 2 (N2)    SP2 178  014
 014A FA21 A2BE A2C1 030A 030D 0188          AP    RESP+2(3),PFLD            N1 + N2               SP2 179  014
                             0189 *                                                                SP2 180
 0150 9260 F023      0023    0190          MVI   35(PRINTRG),C'-'          SIGN TO PRINT AREA    SP2 181  015
 0154 D202 F025 C007 0025 0007 0191          MVC   37(3,PRINTRG),7(CARDRG)   NUMBER 3 TO PRINT AREA SP2 182  015
 015A F212 A2C1 C007 030D 0007 0192          PACK  PFLD,7(3,CARDRG)          PACK NUMBER 3 (N3)    SP2 183  015
 0160 FB21 A2BE A2C1 030A 030D 0193          SP    RESP+2(3),PFLD            N1 + N2 - N3          SP2 184  015
                             0194 *                                                                SP2 185
 0166 925C F029      0029    0195          MVI   41(PRINTRG),C'*'          SIGN TO PRINT AREA    SP2 186  015
 016A D202 F02B C00A 002B 000A 0196          MVC   43(3,PRINTRG),10(CARDRG)  NUMBER 4 TO PRINT AREA SP2 187  015
 0170 F212 A2C1 C00A 030D 000A 0197          PACK  PFLD,10(3,CARDRG)         PACK NUMBER 4 (N4)    SP2 188  015
 0176 FC41 A2BC A2C1 0308 030D 0198          MP    RESP,PFLD                 N1 + N2 - N3 * N4     SP2 189  015
                             0199 *                                                                SP2 190
 017C 927E F030      0030    0200          MVI   48(PRINTRG),C'='          EQUAL SIGN TO PRINT AREA SP2 191 015
 0180 F384 F033 A2BC 0033 0308 0201          UNPK  51(9,PRINTRG),RESP        RESULT TO PRINT AREA  SP2 192  015
 0186 96F0 F03B      003B    0202          OI    59(PRINTRG),X'F0'         CHANGE SIGN IN ZONE TO F SP2 193 016
 018A 47F0 A224      0270    0203          B     EDIT                                            SP2 194  016
                             0204 *                                                                SP2 195
                             0205 *                                                                SP2 196
 00B4                        0206          USING BASEA21,BASERG2                                   SP2 197
                             0207 *                                                                SP2 198
 018E 48C0 B24A      02FE    0208 CEXPRB   LH    CARDRG,YCARDAR            INITIALIZE REGISTER   SP2 199  016
 0192 48F0 B24C      0300    0209          LH    PRINTRG,YPRINTAR          INITIALIZE REGISTER   SP2 200  016
                             0210 *                                                                SP2 201
 0196 D208 F00A B25B 000A 030F 0211          MVC   DPRINTAR+10(L'MESS),MESS  MESSAGE TO PRINT AREA SP2 202  016
 019C 92C2 F014      0014    0212          MVI   DPRINTAR+20,TB            TYPE TO PRINT AREA    SP2 203  016
                             0213 *                                                                SP2 204
 01A0 D202 F019 C001 0019 0001 0214          MVC   DPRINTAR+25(L'NUMB1),NUMB1  NUMBER 1 TO PRINT AREA SP2 205 016
 01A6 F212 B257 C001 030B 0001 0215          PACK  RESP+L'RESP-LNUMBP(LNUMBP),NUMB1 PACK NUMBER 1 (N1) SP2 206 016
                             0216 *                                                                SP2 207
 01AC 9260 F01D      001D    0217          MVI   DPRINTAR+29,C'-'          '-' TO PRINT AREA     SP2 208  016
 01B0 D202 F01F C004 001F 0004 0218          MVC   DPRINTAR+31(L'NUMB2),NUMB2  NUMBER 2 TO PRINT AREA SP2 209 016
 01B6 F212 B259 C004 030D 0004 0219          PACK  PFLD,NUMB2                PACK NUMBER 2 (N2)    SP2 210  016
 01BC FB11 B257 B259 030B 030D 0220          SP    RESP+L'RESP-LDIF(LDIF),PFLD N1 - N2               SP2 211  017
                             0221 *                                                                SP2 212
 01C2 925C F023      0023    0222          MVI   DPRINTAR+35,C'*'          '*' TO PRINT AREA     SP2 213  017
 01C6 D202 F025 C007 0025 0007 0223          MVC   DPRINTAR+37(L'NUMB3),NUMB3  NUMBER 3 TO PRINT AREA SP2 214 017
 01CC F212 B259 C007 030D 0007 0224          PACK  PFLD,NUMB3                PACK NUMBER 3 (N3)    SP2 215  017
 01D2 FC41 B254 B259 0308 030D 0225          MP    RESP,PFLD                 N1 - N2 * N3          SP2 216  017
```

Figure 19. DPS Sample Program, Part 9 of 17

```
SLE    ASSEMBLER SAMPLE, PART 2, MAIN PROGRAM                                    01/20/70  PAGE  007

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                                  DPS ASSEMB 03/02

                                0226 *                                                                SP2 217
01D8 924E F029       0029       0227          MVI   DPRINTAR+41,C'+'          '+' TO PRINT AREA      SP2 218  017
01DC D202 F02B C00A  002B 000A  0228          MVC   DPRINTAR+43(L'NUMB4),NUMB4 NUMBER 4 TO PRINT AREA SP2 219  017
01E2 F212 B259 C00A  030D 000A  0229          PACK  PFLD,NUMB4                PACK NUMBER 4 (N4)     SP2 220  017
01E8 FA41 B254 B259  0308 030D  0230          AP    RESP,PFLD                 N1 - N2 * N3 + N4      SP2 221  017
                                0231 *                                                                SP2 222
01EE 927E F030       0030       0232          MVI   DPRINTAR+48,C'='          '=' TO PRINT AREA      SP2 223  017
01F2 F384 F033 B254  0033 0308  0233          UNPK  DPRINTAR+51(L'RESP*2-1),RESP RESULT TO PR AREA  SP2 224  017
01F8 96F0 F03B       003B       0234          OI    DPRINTAR+59,B'11110000'   CHANGE SIGN IN ZONE TO F SP2 225  018
01FC 47F0 B1BC       0270       0235          B     EDIT                                              SP2 226  018
                                0236 *                                                                SP2 227
                                0237 *                                                                SP2 228
0200 48C0 B24A       02FE       0238 CEXPRC LH    CARDRG,YCARDAR            INITIALIZE REGISTER     SP2 229  018
0204 48F0 B24C       0300       0239        LH    PRINTRG,YPRINTAR          INITIALIZE REGISTER     SP2 230  018
                                0240 *                                                                SP2 231
0208 D208 F00A B25B  000A 030F  0241        MVC   DMESS,MESS                MESSAGE AND             SP2 232  018
020E D200 F014 C000  0014 0000  0242        MVC   PTYPE,TYPE                TYPE TO PRINT AREA      SP2 233  018
                                0243 *                                                                SP2 234
0214 D202 F019 C001  0019 0001  0244        MVC   PNUMB1,NUMB1                                      SP2 235  018
021A F212 B257 C001  030B 0001  0245        PACK  RESPRI,NUMB1              N1                      SP2 236  018
                                0246 *                                                                SP2 237
0220 925C F01D       001D       0247        MVI   SIGN1,C'*'                                        SP2 238  018
0224 D202 F01F C004  001F 0004  0248        MVC   PNUMB2,NUMB2                                      SP2 239  018
022A F212 B259 C004  030D 0004  0249        PACK  PFLD,NUMB2                N2                      SP2 240  018
0230 FC41 B254 B259  0308 030D  0250        MP    RESP,PFLD                 N1 * N2                 SP2 241  019
                                0251 *                                                                SP2 242
0236 924E F023       0023       0252        MVI   SIGN2,C'+'                                        SP2 243  019
023A D202 F025 C007  0025 0007  0253        MVC   PNUMB3,NUMB3                                      SP2 244  019
0240 F212 B259 C007  030D 0007  0254        PACK  PFLD,NUMB3                N3                      SP2 245  019
0246 FA41 B254 B259  0308 030D  0255        AP    RESP,PFLD                 N1 * N2 + N3            SP2 246  019
                                0256 *                                                                SP2 247
024C 9260 F029       0029       0257        MVI   SIGN3,C'-'                                        SP2 248  019
0250 D202 F02B C00A  002B 000A  0258        MVC   PNUMB4,NUMB4                                      SP2 249  019
0256 F212 B259 C00A  030D 000A  0259        PACK  PFLD,NUMB4                N4                      SP2 250  019
025C FB41 B254 B259  0308 030D  0260        SP    RESP,PFLD                 N1 * N2 + N3 - N4       SP2 251  019
                                0261 *                                                                SP2 252
0262 927E F030       0030       0262        MVI   EQSIGN,C'='                                        SP2 253  019
0266 F384 F033 B254  0033 0308  0263        UNPK  RESZ,RESP                 RESULT TO PRINT AREA    SP2 254  020
026C 96F0 F03B       003B       0264        OI    RESZ+L'RESZ-1,C'0'        CHANGE SIGN TO F        SP2 255  020


                                0266 *    EDIT RESULT                                                 SP2 257
                                0267 *                                                                SP2 258
0270 4880 B240       02F4       0268 EDIT     LH    REG8,H1             INITIALIZE REGISTER           SP2 259  020
0274 48F0 B24E       0302       0269          LH    PRINTRG,YRESZ       LOAD ADDRESS OF RESULT FIELD ZONED SP2 260  020
0278 95F0 F000       0000       0270 TSTIF0   CLI   0(PRINTRG),C'0'      TEST IF 0                     SP2 261  020
027C 4770 B1E4       0298       0271          BNE   TSTSIGN             NO                            SP2 262  020
0280 4980 B250       0304       0272          CH    REG8,YLRESZ         ALL BYTES CHECKED             SP2 263  020
0284 47B0 B1FC       02B0       0273          BNL   PRNTLINE            YES                           SP2 264  020
0288 9240 F000       0000       0274          MVI   0(PRINTRG),C'       MOVE BLANK                    SP2 265  020
028C 4AF0 B240       02F4       0275          AH    PRINTRG,H1          NEXT BYTE                     SP2 266  020
0290 4A80 B240       02F4       0276          AH    REG8,H1             INCREMENT COUNTER BY 1        SP2 267  020
0294 47F0 B1C4       0278       0277          B     TSTIF0              LOOP                          SP2 268  020
                                0278 *                                                                SP2 269
0298 4BF0 B240       02F4       0279 TSTSIGN  SH    PRINTRG,H1          ONE STEP BACK                 SP2 270  020
029C 9101 B258       030C       0280          TM    RESP+L'RESP-1,B'00000001'  TEST SIGN             SP2 271  020
```

Figure 19. DPS Sample Program, Part 10 of 17

```
SLE    ASSEMBLER SAMPLE, PART 2, MAIN PROGRAM                                    01/20/70  PAGE  008

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                          DPS ASSEMB 03/02

 02A0 4710 B1F8       02AC      0281           BC    MINUS                                          SP2 272  021
 02A4 924E F000       0000      0282 PLUS      MVI   0(PRINTRG),C'+'    MOVE PLUS SIGN              SP2 273  021
 02A8 47F0 B1FC       02B0      0283           B     PRNTLINE                                       SP2 274  021
 02AC 9260 F000       0000      0284 MINUS     MVI   0(PRINTRG),C'-'    MOVE MINUS SIGN             SP2 275  021


                                0286 *     PRINT A LINE                                           SP2 277
                                0287 *                                                            SP2 278
 02B0 48E0 B246       02FA      0288 PRNTLINE LH     BRANCHRG,YPRINT          LOAD BRANCH-REGISTER SP2 279  021
 02B4 0DDE                      0289          BASR   LINKRG,BRANCHRG          PRINT A LINE AND RETURN SP2 280  021
 02B6 47F0 B232       02E6      0290          B      BGETCARD                 READ NEXT CARD         SP2 281  021


                                0292 *     HALT ROUTINES                                          SP2 283
                                0293 *                                                            SP2 284
 02BA 9900 0001       0001      0294 HALT0001 HPR    1,0                INVALID CARD TYPE           SP2 285  021
 02BE 47F0 B232       02E6      0295          B      BGETCARD           NEXT CARD                   SP2 286  021
                                0296 *                                                            SP2 287
 02C2 4080 B230       02E4      0297 HALT001N STH    8,HPRI+2           STORE ENTRY NUMBER (N = 1, 2, 3 OR 4 SP2 288  021
 02C6 9610 B231       02E5      0298          OI     HPRI+3,X'10'       'OR' COLUMN NUMBER (COL. 1) SP2 289  021
 02CA 47F0 B22E       02E2      0299          B      HPRI                                           SP2 290  021
                                0300 *                                                            SP2 291
 02CE 4080 B230       02E4      0301 HALT002N STH    REG8,HPRI+2        ENTRY NUMBER (N = 1, 2, 3 OR 4) SP2 292  021
 02D2 9620 B231       02E5      0302          OI     HPRI+3,X'20'       NUMBER OF COL               SP2 293  021
 02D6 47F0 B22E       02E2      0303          B      HPRI                                           SP2 294  022
                                0304 *                                                            SP2 295
 02DA 4080 B230       02E4      0305 HALT003N STH    REG8,HPRI+2        ENTRY NUMBER (N = 1, 2, 3 OR 4) SP2 296  022
 02DE 9630 B231       02E5      0306          OI     HPRI+3,X'30'       COL 3                       SP2 297  022
 02E2 9900 0000       0000      0307 HPRI      HPR    0,0                WILL BE UPDATED BY HALT ROUTINES SP2 298  022
 02E6 48A0 B252       0306      0308 BGETCARD LH     BASERG1,YBASEA11   LOAD BASE REGISTER 1 WITH ORIG VALUE SP2 299  022
 0002                           0309          USING  BASEA11,BASERG1                                 SP2 300
 02EA 47F0 A00E       0010      0310          B      GETCARD            NEXT CARD                   SP2 301  022


                                0312 *     END OF CARD FILE                                       SP2 303
                                0313 *                                                            SP2 304
 02EE 48E0 B248       02FC      0314 EOCF LH BRANCHRG,YCLOSEF LOAD ADDRESS OF CLOSE ROUTINE                FREE   SP2 305  022
 02F2 07FE                      0315  BR BRANCHRG CLOSE THE FILES                                        FORMAT SP2 306  022
```

Figure 19. DPS Sample Program, Part 11 of 17

```
SLE    ASSEMBLER SAMPLE, PART 2, DATA DEFINITIONS                                    01/20/70  PAGE  009

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                            DPS ASSEMB 03/02

                              0317 *     DATA DEFINITIONS                                  SP2 308
                              0318 *                                                       SP2 309
02F4 0001                     0319 H1        DC    H'1'                                    SP2 310  022
02F6 0003                     0320 H3        DC    H'3'                                    SP2 311  022
02F8 0004                     0321 H4        DC    H'4'                                    SP2 312  022
                              0322 *                                                       SP2 313
02FA 0000                     0323 YPRINT    DC    Y(PRINT)          ADDRESS OF PRINT ROUTINE      SP2 314  022
02FC 0000                     0324 YCLOSEF   DC    Y(CLOSEF)         ADDRESS OF CLOSE ROUTINE      SP2 315  022
02FE 0000                     0325 YCARDAR   DC    Y(CARDAR)         ADDRESS OF CARDAREA           SP2 316  022
0300 0000                     0326 YPRINTAR  DC    Y(PRINTAR)        ADDRESS OF PRINTAREA          SP2 317  024
0302 0033                     0327 YRESZ     DC    Y(PRINTAR+51)     ADDRESS OF RESULT FIELD ZONED SP2 318  024
0304 0009                     0328 YLRESZ    DC    Y(L'RESZ)         LENGTH OF RESZ                SP2 319  024
0306 0002                     0329 YBASEA11  DC    Y(BASEA11)        BASE ADDRESS 1 1              SP2 320  024
                              0330 *                                                       SP2 321
0308 000000000C               0331 RESP      DC    PL5'0'            RESULT PACKED                 SP2 322  024
030D 000C                     0332 PFLD      DC    PL2'0'            PACK FIELD                    SP2 323  024
                              0333 *                                                       SP2 324
0308                          0334           ORG   RESP              SET LOC. COUNTER BACK TO RESP SP2 325
0308 000000                   0335           DC    X'000000'         FIRST PART OF RESP            SP2 326  026
030B 000C                     0336 RESPRI    DC    PL2'0'            RIGHT PART OF RESP            SP2 327  026
0002                          0337 LNUMBP    EQU   L'PFLD            LENGTH OF PACKED NUMBER       SP2 328
0002                          0338 LDIF      EQU   LNUMBP            LENGTH OF DIFFERENCE          SP2 329
                              0339 *                                                       SP2 330
030F                          0340           ORG                     , TO PREVENT OVERLOADING OF PFLD SP2 331
030F C3C1D9C440E3E8D7         0341 MESS      DC    C'CARD TYPE'                            SP2 332  027
                              0342 *                                                       SP2 333
```

Figure 19. DPS Sample Program, Part 12 of 17

```
 SLE    ASSEMBLER SAMPLE, PART 2, DUMMY CONTROL SECTIONS                                01/20/70  PAGE  010

LOCATN  OBJECT CODE  ADD1 ADD2 STMT     SOURCE STATEMENT                                DPS ASSEMB 03/02

 0000                          0344 DCS1     DSECT ,                 DUMMY CARDAREA          SP2 335
                               0345 *                                                        SP2 336
 0000                          0346 DCARDAR  DS    0CL13                                   SP2 337
 0000                          0347 TYPE     DS    C                                       SP2 338
 0001                          0348 NUMB1    DS    CL3                                     SP2 339
 0004                          0349 NUMB2    DS    CL3                                     SP2 340
 0007                          0350 NUMB3    DS    CL3                                     SP2 341
 000A                          0351 NUMB4    DS    CL3                                     SP2 342
 0001                          0352 ENTRY    EQU   NUMB1                                   SP2 343
 000D                          0353 CAEND    EQU   *                                       SP2 344


 0000                          0355 DCS2     DSECT ,                 DUMMY PRINTAREA         SP2 346
                               0356 *                                                        SP2 347
 0000                          0357 DPRINTAR DS    0CL60                                   SP2 348
 000A                          0358         ORG   DPRINTAR+10                             SP2 349
 000A                          0359 DMESS    DC    C'CARD TYPE'                            SP2 350
 0014                          0360         ORG   DPRINTAR+20                             SP2 351
 0014                          0361 PTYPE    DS    C                                       SP2 352
 0019                          0362         ORG   DPRINTAR+25                             SP2 353
 0019                          0363 PNUMB1   DS    CL3                                     SP2 354
 001D                          0364 SIGN1    EQU   *+1                                     SP2 355
 001F                          0365         ORG   *+3                                     SP2 356
 001F                          0366 PNUMB2   DS    CL3                                     SP2 357
 0023                          0367 SIGN2    EQU   *+1                                     SP2 358
 0025                          0368         ORG   *+3                                     SP2 359
 0025                          0369 PNUMB3   DS    CL3                                     SP2 360
 0029                          0370 SIGN3    EQU   *+1                                     SP2 361
 002B                          0371         ORG   *+3                                     SP2 362
 002B                          0372 PNUMB4   DS    CL3                                     SP2 363
 0030                          0373 EQSIGN   EQU   *+2                                     SP2 364
 0033                          0374         ORG   *+5                                     SP2 365
 0033                          0375 RESZ     DS    CL9                                     SP2 366
 003C                          0376 PAEND    EQU   *                                       SP2 367


 0000                          0378         END   BEGIN                                   SP2 369  028
```

NO STATEMENT FLAGGED

RELOCATION DICTIONARY

POS.ID	REL.ID	FLGS	ADDR
01	01	04	010A
01	02	04	010C
01	03	04	010E
01	07	04	0110
01	06	04	0112
01	04	04	02FA
01	05	04	02FC
01	06	04	02FE
01	07	04	0300
01	07	04	0302
01	01	04	0306

Figure 19. DPS Sample Program, Part 13 of 17

```
                                CROSS-REFERENCE LIST

 SYMBOL    LEN  VALUE  DEF     CROSS-REFERENCE

BASEA11      1  0002   0054    0309  0329
BASEA12      1  004C   0092
BASEA21      1  00B4   0129    0055  0120  0206
BASERG1      1  000A   0025    0052  0053  0090  0091  0308  0309
BASERG2      1  000B   0026    0055  0120  0174  0206
BEGIN        2  0000   0052    0378
BGETCARD     4  02E6   0308    0290  0295
BLANK        1  0040   0044    0141  0148
BRANCHRG     1  000E   0028    0061  0062  0067  0068  0288  0289  0314  0315
CAEND        1  000D   0353
CARDAR       1  0000   0015    0079  0325
CARDRG       1  000C   0029    0056  0079  0080  0082  0084  0095  0097  0100  0102  0104  0107  0109  0111  0117
                                0154  0176  0182  0183  0186  0187  0191  0192  0196  0197  0208  0238
CEXPRA       4  011C   0176    0115
CEXPRB       4  018E   0208    0122
CEXPRC       4  0200   0238    0126
CHAR0        1  00F0   0042    0134  0139  0146
CHAR9        1  00F9   0043    0132  0137  0144
CLOSEF       1  0000   0014    0324
DCARDAR     13  0000   0346    0056
DCS1         1  0000   0344
DCS2         1  0000   0355
DMESS        9  000A   0359    0241
DPRINTAR    60  0000   0357    0057  0074  0211  0212  0214  0217  0218  0222  0223  0227  0228  0232  0233  0234
                                0358  0360  0362
EDIT         4  0270   0268    0203  0235
ENTRY        3  0001   0352    0132  0134  0137  0139  0141  0144  0146  0148
EOCF         4  02EE   0314    0018
EQSIGN       1  0030   0373    0262
EQUAL        1  0008   0036    0083
GETCARD      4  0010   0067    0310
HALT0001     4  02BA   0294    0086
HALT001N     4  02C2   0297    0108  0112  0145  0149
HALT002N     4  02CE   0301    0101  0105  0138  0142
HALT003N     4  02DA   0305    0096  0098  0133  0135
HPRI         4  02E2   0307    0297  0298  0299  0301  0302  0303  0305  0306
H1           2  02F4   0319    0131  0153  0268  0275  0276  0279
H3           2  02F6   0320    0154
H4           2  02F8   0321    0151
LDIF         1  0002   0338    0220  0220
LINKRG       1  000D   0027    0062  0068  0121  0125  0152  0289
LNUMBP       1  0002   0337    0215  0215  0338
MESS         9  030F   0341    0179  0211  0211  0241
MINUS        4  02AC   0284    0281
NOTLOW       1  000B   0037    0152
NUMB1        3  0001   0348    0214  0214  0215  0244  0245  0352
NUMB2        3  0004   0349    0218  0218  0219  0248  0249
NUMB3        3  0007   0350    0223  0223  0224  0253  0254
NUMB4        3  000A   0351    0228  0228  0229  0258  0259
OPENF        1  0000   0011    0061
PAEND        1  003C   0376
PART2        1  0000   0004
```

Figure 19. DPS Sample Program, Part 14 of 17

CROSS-REFERENCE LIST

SYMBOL	LEN	VALUE	DEF	CROSS-REFERENCE													
PFLD	2	030D	0332	0187	0188	0192	0193	0197	0198	0219	0220	0224	0225	0229	0230	0249	0250
				0254	0255	0259	0260	0337									
PLUS	4	02A4	0282														
PNUMB1	3	0019	0363	0244													
PNUMB2	3	001F	0366	0248													
PNUMB3	3	0025	0369	0253													
PNUMB4	3	002B	0372	0258													
PRINT	1	0000	0013	0323													
PRINTAR	1	0000	0016	0072	0326	0327											
PRINTRG	1	000F	0030	0057	0072	0073	0074	0074	0177	0179	0180	0182	0185	0186	0190	0191	0195
				0196	0200	0201	0202	0209	0239	0269	0270	0274	0275	0279	0282	0284	
PRNTLINE	4	02B0	0288	0063	0273	0283											
PTYPE	1	0014	0361	0242													
READ	1	0000	0012	0067													
REG8	1	0008	0024	0131	0151	0153	0268	0272	0276	0301	0305						
RESP	5	0308	0331	0075	0183	0188	0193	0198	0201	0215	0215	0220	0220	0225	0230	0233	0233
				0250	0255	0260	0263	0280	0280	0334							
RESPRI	2	030B	0336	0245													
RESZ	9	0033	0375	0263	0264	0264	0328										
SIGN1	1	001D	0364	0247													
SIGN2	1	0023	0367	0252													
SIGN3	1	0029	0370	0257													
TA	1	00C1	0039	0080	0180												
TB	1	00C2	0040	0082	0212												
TC	1	00C3	0041	0084													
TESTENTR	4	00B4	0131	0121	0125												
TIFIN	4	00F8	0151	0147													
TLECOL	4	00E0	0144	0140													
TMICOL	4	00C8	0137														
TRICOL	4	00B8	0132	0155													
TSTIFIN	4	0090	0114	0110													
TSTIF0	4	0278	0270	0277													
TSTLECOL	4	0078	0107	0103													
TSTMICOL	4	0060	0100														
TSTRICOL	4	0050	0095	0118													
TSTSIGN	4	0298	0279	0271													
TYPE	1	0000	0347	0242													
TYPEA	2	004A	0090	0081													
TYPEB	4	00A4	0121	0083													
TYPEC	4	00AC	0125	0085													
YBASEA11	2	0306	0329	0308													
YCARDAR	2	02FE	0325	0176	0208	0238											
YCLOSEF	2	02FC	0324	0314													
YLRESZ	2	0304	0328	0272													
YPRINT	2	02FA	0323	0288													
YPRINTAR	2	0300	0326	0177	0209	0239											
YRESZ	2	0302	0327	0269													

Figure 19. DPS Sample Program, Part 15 of 17

```
         END OF JOB
// PAUSE INSERT ASSEMB OUTPUT DECKS OF BOTH ASSEMBLIES FOR LNKEDT RUN
         B40
// JOB LNKEDT
// ASSGN SYSOPT,UA                 OUTPUT ONLY IN RELOCATABLE AREA
// EXEC

MOD 20 DPS  LINKAGE EDITOR    VERS 03/01


        INPUT

   PHASE SAMPLE,S,0               START JUST BEHIND THE MONITOR        SP1 014
        TXT LOADP X'0000'
        ACTION DUP                                                     PDR4547
        XFR       X'017C'                                              SLE 047
        ACTION NODUP                                                   PDR4550
        TXT LOADP X'017C'
        END                                                            SLE 055
        TXT LOADP X'0000'
        END       X'0000'                                              SLE 028
 ENTRY                                                                 SLE 029


        OUTPUT

 PHASE SAMPLE,A,X'0D4A',,0000                                          SLE0001
        TXT LOADP X'0D4A'
        ACTION DUP                                                     SLE0027
        XFR       X'0EC6'                                              SLE0028
        ACTION NODUP                                                   SLE1001
        TXT LOADP X'0EC6'
        END       X'0F42'                                              SLE1024
```

Figure 19.. DPS Sample Program, Part 16 of 17

```
01/20/70  PHASE     LOCORE  HICORE  XFR-ADD ESD TYPE  LABEL     LOADED  REL-FR


          SAMPLE    0D4A    1259    0F42

                                          CSECT     PART1     0D4A    0D4A
                                           ENTRY     OPENF    0EC6
                                           ENTRY     READ     0ED0
                                           ENTRY     PRINT    0ED8
                                           ENTRY     CLOSEF   0EE0
                                           ENTRY     CARDAR   0EF9
                                           ENTRY     PRINTAR  0F06

                                          CSECT     PART2     0F42    0F42
                                           ENTRY     EOCF     1230


     NUMBER OF UNDEFINED RLD REFERENCE -  000

     NUMBER OF TXT OR REP CARDS OUTSIDE PHASE LIMITS-  000
```

```
          END OF JOB
// JOB CMAINT                        CORE IMAGE MAINTENANCE RUN
// EXEC R
```

```
DPS CMAINT PROGRAM VERSION 06  MOD-LEVEL 00
// CATAL
PHASE SAMPLE  SUBPHASE 0  LOADPOINT 0D4A  ENTRYPOINT 0EC6  REPLACED AN OLD ONE
PHASE SAMPLE  SUBPHASE 1  LOADPOINT 0EC6  ENTRYPOINT 0F42  REPLACED AN OLD ONE
// END
```

```
SYSTEM D       FIRST SECTOR  LAST S ALLOC  LAST S OCC    SECT ALLOC  SECT OCC  SECT AVAIL        01/20/70

CORE IMAGE D   004 4 0       005 1 9       005 1 6         00080       00077      00003

CORE IMAGE L   005 2 0       060 1 9       059 7 7         05500       05458      00042
```

```
          END OF JOB
// JOB SAMPLE                        ASSEMBLER SAMPLE
// EXEC

          CARD TYPE A      1 +   2 -   3 *   4  =          0
          CARD TYPE B      1 -   2 *   3 +   4  =         +1
          CARD TYPE C      1 *   2 +   3 -   4  =         +1
          CARD TYPE A      5 +   6 -   7 *   8  =        +32
          CARD TYPE B      5 -   6 *   7 +   8  =         +1
          CARD TYPE C      5 *   6 +   7 -   8  =        +29
          CARD TYPE A     45 +  71 -  28 *  39  =      +3432
          CARD.TYPE B     45 -  71 *  28 +  39  =       -689
          CARD TYPE C     45 *  71 +  28 -  39  =      +3184
          CARD TYPE A    248 + 935 - 627 * 310  =    +172360
          CARD TYPE B    248 - 935 * 627 + 310  =    -430439
          CARD TYPE C    248 * 935 + 627 - 310  =    +232197
```

```
          END OF JOB
```

Figure 19. DPS Sample Program, Part 17 of 17

```
// LOG
// JOB ASSEMB
// DATE 70020
// ASSGN SYSIPT,X'100',R4          2501
// ASSGN SYSOPT,X'300',P2          1442
// ASSGN SYS000,X'803',D4
// VOL SYS000,WORK1
// DLAB  'SYSTEM/360 MOD 20 DPS                        1202020',P       C
            0001,67150,67150
// XTENT 1,000,0153000,0199009,'202020',SYS000
// EXEC
         AOPTN NOSYM                                                    SP1 001
*                                                                       SP1 002
************************************************************************ SP1 003
*    PREPARE CONTROL CARDS FOR LINKAGE EDITOR RUN                       SP1 004
************************************************************************ SP1 005
*                                                                       SP1 006
         REPRO                                                          SP1 007
// JOB LNKEDT                                                          SP1 008
         REPRO                                                          SP1 009
// ASSGN SYSOPT,UA                 OUTPUT ONLY IN RELOCATABLE AREA      SP1 010
         REPRO                                                          SP1 011
// EXEC                                                                SP1 012
         REPRO                                                          SP1 013
   PHASE SAMPLE,S,0                START JUST BEHIND THE MONITOR        SP1 014
SLE      TITLE 'ASSEMBLER SAMPLE, PART 1, IOCS'                         SP1 015
************************************************************************ SP1 016
PART1    START 0                 FIRST PART OF THE SAMPLE               SP1 017
************************************************************************ SP1 018
         SPACE 2                                                        SP1 019
************************************************************************ SP1 020
*    DEFINE ENTRIES AND EXTERNAL SYMBOLS                                SP1 021
************************************************************************ SP1 022
*                                                                       SP1 023
         ENTRY OPENF             OPEN ROUTINE                           SP1 024
         ENTRY READ              READ ROUTINE                           SP1 025
         ENTRY PRINT             PRINT ROUTINE                          SP1 026
         ENTRY CLOSEF            CLOSE ROUTINE                          SP1 027
         ENTRY CARDAR            AREA WHERE CARD IS STORED              SP1 028
         ENTRY PRINTAR           PRINT AREA                             SP1 029
*                                                                       SP1 030
         EXTRN EOCF              END OF CARD FILE                       SP1 031
         SPACE 2                                                        SP1 032
************************************************************************ SP1 033
*    SYMBOLIC REGISTER DEFINITIONS                                      SP1 034
************************************************************************ SP1 035
*                                                                       SP1 036
LINKRG   EQU   13                LINK-REGISTER                          SP1 037
         SPACE 2                                                        SP1 038
************************************************************************ SP1 039
*    DEFINE THE FILES                                                   SP1 040
************************************************************************ SP1 041
*                                                                       SP1 042
         PRINT NOGEN             DON'T PRINT GENERATED STATEMENTS       SP1 043
*                                                                       SP1 044
READER   DTFSR TYPEFLE=INPUT,BLKSIZE=13,DEVICE=READ01,EOFADDR=EOCF,     CSP1 045
               IOAREA1=INAREA,OVERLAP=NO,WORKA=YES                      SP1 046
*                                                                       SP1 047
PRINTER  DTFSR TYPEFLE=OUTPUT,WORKA=YES,DEVICE=PRINTER,BLKSIZE=60       SP1 048
*                                                                       SP1 049
         DTFEN                                                          SP1 050
         SPACE 2                                                        SP1 051
************************************************************************ SP1 052
*    OPEN THE FILES                                                     SP1 053
************************************************************************ SP1 054
*                                                                       SP1 055
OPENF    OPEN  READER,PRINTER    OPEN READER AND PRINTER                SP1 056
         BR    LINKRG            RETURN                                 SP1 057
         SPACE 2                                                        SP1 058
************************************************************************ SP1 059
*    READ A CARD                                                        SP1 060
************************************************************************ SP1 061
*                                                                       SP1 062
READ     GET   READER,CARDAR     READ A CARD AND MOVE IT TO CARDAR      SP1 063
         BR    LINKRG            RETURN                                 SP1 064
         SPACE 2                                                        SP1 065
************************************************************************ SP1 066
*    PRINT A LINE                                                       SP1 067
************************************************************************ SP1 068
```

Figure 20. Source Deck Listing, Part 1 of 7

```
*                                                                      SP1 069
PRINT    PUT   PRINTER,PRINTAR     PRINT A LINE                        SP1 070
         BR    LINKRG              RETURN                              SP1 071
         SPACE 2                                                       SP1 072
************************************************************************ SP1 073
*    CLOSE THE FILES AND GO BACK TO MONITOR (END OF JOB)               SP1 074
************************************************************************ SP1 075
*                                                                      SP1 076
CLOSEF   CLOSE READER,PRINTER      CLOSE READER AND PRINTER            SP1 077
*                                                                      SP1 078
         PRINT GEN                 PRINT GENERATED STATEMENTS          SP1 079
*                                                                      SP1 080
         EOJ   ,                   GIVE CONTROL BACK TO MONITOR        SP1 081
         SPACE 2                                                       SP1 082
************************************************************************ SP1 083
*    AREAS                                                             SP1 084
************************************************************************ SP1 085
*                                                                      SP1 086
INAREA   DS    13C                 INPUT AREA FOR IOCS                 SP1 087
CARDAR   DS    CL13                CARD IS STORED HERE AFTER GET       SP1 088
PRINTAR  DS    CL60                PRINT AREA                          SP1 089
         SPACE 2                                                       SP1 090
         END                                                           SP1 091
/*
// JOB ASSEMB                      SECOND ASSEMBLY
// EXEC
         AOPTN CROSSREF,ENTRY                                          SP2 001
SLE      TITLE 'ASSEMBLER SAMPLE, PART 2, DEFINITIONS'                 SP2 002
************************************************************************ SP2 003
PART2    START 0                   SECOND PART OF SAMPLE               SP2 004
************************************************************************ SP2 005
         SPACE 2                                                       SP2 006
************************************************************************ SP2 007
*    DEFINE ENTRIES AND EXTERNAL SYMBOLS                               SP2 008
************************************************************************ SP2 009
*                                                                      SP2 010
         EXTRN OPENF               OPEN ROUTINE                        SP2 011
         EXTRN READ                READ ROUTINE                        SP2 012
         EXTRN PRINT               PRINT ROUTINE                       SP2 013
         EXTRN CLOSEF              CLOSE ROUTINE                       SP2 014
         EXTRN CARDAR              AREA WHERE CARD IS STORED           SP2 015
         EXTRN PRINTAR             PRINT AREA                          SP2 016
*                                                                      SP2 017
         ENTRY EOCF                END OF CARD FILE                    SP2 018
         SPACE 2                                                       SP2 019
************************************************************************ SP2 020
*    SYMBOLIC REGISTER DEFINITIONS                                     SP2 021
************************************************************************ SP2 022
*                                                                      SP2 023
REG8     EQU   8                   REG. 8                              SP2 024
BASERG1  EQU   10                  BASE-REGISTER 1                     SP2 025
BASERG2  EQU   11                  BASE-REGISTER 2                     SP2 026
LINKRG   EQU   13                  LINK-REGISTER                       SP2 027
BRANCHRG EQU   14                  BRANCH-REGISTER                     SP2 028
CARDRG   EQU   12                  REGISTER FOR CARD AREA              SP2 029
PRINTRG  EQU   15                  REGISTER FOR PRINT AREA             SP2 030
         SPACE 2                                                       SP2 031
************************************************************************ SP2 032
*    EQUATE SYMBOLS                                                    SP2 033
************************************************************************ SP2 034
*                                                                      SP2 035
EQUAL    EQU   8                   CONDITION  EQUAL                    SP2 036
NOTLOW   EQU   11                  CONDITION  EQUAL AND HIGH           SP2 037
*                                                                      SP2 038
TA       EQU   C'A'                TYPE A                              SP2 039
TB       EQU   C'B'                TYPE B                              SP2 040
TC       EQU   C'C'                TYPE C                              SP2 041
CHAR0    EQU   C'0'                CHARACTER 0                         SP2 042
CHAR9    EQU   C'9'                CHARACTER 9                         SP2 043
BLANK    EQU   C' '                CHARACTER 'BLANK'                   SP2 044
         TITLE 'ASSEMBLER SAMPLE, PART 2, MAIN PROGRAM'                SP2 045
************************************************************************ SP2 046
*    MAIN PROGRAM                                                      SP2 047
************************************************************************ SP2 048
*                                                                      SP2 049
*    LOAD BASE REGISTER                                                SP2 050
*                                                                      SP2 051
BEGIN    BASR  BASERG1,0           LOAD BASE-REGISTER 1 AND            SP2 052
         USING *,BASERG1           INFORM ASSEMBLER                    SP2 053
```

Figure 20. Source Deck Listing, Part 2 of 7

```
BASEA11  EQU   *                    BASE-ADDRESS 1,1                    SP2 054
         LH    BASERG2,=Y(BASEA21) LOAD BASE-REGISTER 2                SP2 055
         USING DCARDAR,CARDRG               FOR IMPLICIT ADDRESSING    SP2 056
         USING DPRINTAR,PRINTRG             FOR IMPLICIT ADDRESSING    SP2 057
         SPACE 2                                                       SP2 058
*    OPEN CARD AND PRINTER FILE                                        SP2 059
*                                                                      SP2 060
         LH    BRANCHRG,=Y(OPENF)  LOAD REG. WITH BRANCH ADDRESS       SP2 061
         BASR  LINKRG,BRANCHRG     OPEN READER AND PRINTER AND RETURN  SP2 062
         B     PRNTLINE            PRINT A BLANK LINE                  SP2 063
         SPACE 2                                                       SP2 064
*    READ A CARD                                                       SP2 065
*                                                                      SP2 066
GETCARD  LH    BRANCHRG,=Y(READ)   LOAD REG. WITH BRANCH ADDRESS       SP2 067
         BASR  LINKRG,BRANCHRG     BRANCH TO READ ROUTINE AND RETURN   SP2 068
         SPACE 2                                                       SP2 069
*    BLANK PRINTAREA AND RESET PACKED RESULT FIELD                     SP2 070
*                                                                      SP2 071
         LH    PRINTRG,=Y(PRINTAR) LOAD ADDRESS OF PRINT AREA          SP2 072
         MVI   0(PRINTRG),C' '     BLANK IN FIRST BYTE OF PRINT AREA   SP2 073
         MVC   1(L'DPRINTAR-1,PRINTRG),0(PRINTRG) BLANK THE REST       SP2 074
         ZAP   RESP,=P'0'          INITIALIZE RESULT FIELD             SP2 075
         EJECT                                                         SP2 076
*    TEST TYPE OF CARD                                                 SP2 077
*                                                                      SP2 078
         LH    CARDRG,=Y(CARDAR)   INITIALIZE REGISTER                 SP2 079
         CLI   0(CARDRG),TA        TEST IF TYPE A                      SP2 080
         BC    8,TYPEA             YES                                 SP2 081
         CLI   0(CARDRG),TB        IF TYPE B                           SP2 082
         BC    EQUAL,TYPEB         YES                                 SP2 083
         CLI   0(CARDRG),TC        IF TYPE C                           SP2 084
         BE    TYPEC               YES                                 SP2 085
         BC    15,HALT0001         GO TO HALT ROUTINES                 SP2 086
         SPACE 2                                                       SP2 087
*    TEST FOR DEC. NUMBERS IN THE CARD ENTRIES                         SP2 088
*                                                                      SP2 089
TYPEA    BASR  BASERG1,0           CHANGE CONTENTS OF BASE-REGISTER 1  SP2 090
         USING *,BASERG1           AND INFORM THE ASSEMBLER            SP2 091
BASEA12  EQU   *                   BASE-ADDRESS 1,2                    SP2 092
*                                                                      SP2 093
         LH    8,=H'1'             INITIALIZE COUNTER                  SP2 094
TSTRICOL CLI   3(CARDRG),249       COMPARE RIGHT COL WITH 9            SP2 095
         BH    HALT003N            INVALID IF HIGH                     SP2 096
         CLI   3(CARDRG),240       COMPARE WITH 0                      SP2 097
         BL    HALT003N            INVALID IF LOW                      SP2 098
*                                                                      SP2 099
TSTMICOL CLI   2(CARDRG),C'9'      COMPARE MIDDLE COL WITH 9           SP2 100
         BH    HALT002N            INVALID IF HIGH                     SP2 101
         CLI   2(CARDRG),C'0'      COMPARE WITH 0                      SP2 102
         BNL   TSTLECOL            O. K. IF NOT LOW                    SP2 103
         CLI   2(CARDRG),C' '      IF BLANK                            SP2 104
         BNE   HALT002N            NO, INVALID                         SP2 105
*                                                                      SP2 106
TSTLECOL CLI   1(CARDRG),X'F9'     COMPARE LEFT COL. WITH 9            SP2 107
         BH    HALT001N            INVALID IF HIGH                     SP2 108
         CLI   1(CARDRG),X'F0'     COMPARE WITH 0                      SP2 109
         BNL   TSTIFIN             O. K. IF NOT LOW                    SP2 110
         CLI   1(CARDRG),X'40'     IF BLANK                            SP2 111
         BNE   HALT001N            NO, INVALID                         SP2 112
*                                                                      SP2 113
TSTIFIN  CH    8,=H'4'             ALL ENTRIES CHECKED                 SP2 114
         BNL   CEXPRA              YES, COMPUTE EXPRESSION TYPE A      SP2 115
         AH    8,=H'1'             INCREMENT COUNTER BY 1              SP2 116
         AH    CARDRG,=H'3'        NEXT ENTRY                          SP2 117
         B     TSTRICOL            LOOP                                SP2 118
         EJECT                                                         SP2 119
         USING BASEA21,BASERG2                                         SP2 120
TYPEB    BAS   LINKRG,TESTENTR     BRANCH TO SUBROUTINE AND RETURN     SP2 121
         B     CEXPRB              BRANCH TO SUBROUTINE                SP2 122
*                                                                      SP2 123
*                                                                      SP2 124
TYPEC    BAS   LINKRG,TESTENTR     BRANCH TO SUBROUTINE AND RETURN     SP2 125
         B     CEXPRC              BRANCH TO SUBROUTINE                SP2 126
*                                                                      SP2 127
*                                                                      SP2 128
BASEA21  EQU   *                   BASE-ADDRESS 2,1                    SP2 129
*                                                                      SP2 130
TESTENTR LH    REG8,H1             INITIALIZE REGISTER                 SP2 131
TRICOL   CLI   ENTRY+2,CHAR9       COMPARE RIGHT COL WITH 9            SP2 132
```

Figure 20. Source Deck Listing, Part 3 of 7

```
         BH    HALT003N            HIGH                                    SP2 133
         CLI   ENTRY+2,CHAR0       WITH 0                                  SP2 134
         BL    HALT003N            LOW                                     SP2 135
*                                                                          SP2 136
TMICOL   CLI   ENTRY+1,CHAR9       COMPARE MIDDLE COL WITH 9               SP2 137
         BH    HALT002N            HIGH                                    SP2 138
         CLI   ENTRY+1,CHAR0       WITH 0                                  SP2 139
         BNL   TLECOL              O. K.                                   SP2 140
         CLI   ENTRY+1,BLANK       IF BLANK                                SP2 141
         BNE   HALT002N            NO                                      SP2 142
*                                                                          SP2 143
TLECOL   CLI   ENTRY,CHAR9         COMPARE LEFT COL WITH 9                 SP2 144
         BH    HALT001N            HIGH                                    SP2 145
         CLI   ENTRY,CHAR0         WITH 0                                  SP2 146
         BNL   TIFIN               O. K.                                   SP2 147
         CLI   ENTRY,BLANK         IF BLANK                                SP2 148
         BNE   HALT001N            NO                                      SP2 149
*                                                                          SP2 150
TIFIN    CH    REG8,H4             ALL ENTRIES CHECKED                     SP2 151
         BCR   NOTLOW,LINKRG       YES                                     SP2 152
         AH    REG8,H1             INCREMENT COUNTER BY 1                  SP2 153
         AH    CARDRG,H3           NEXT ENTRY                              SP2 154
         B     TRICOL              LOOP                                    SP2 155
         SPACE 2                                                           SP2 156
*    LITERAL POOL                                                          SP2 157
*                                                                          SP2 158
         LTORG                                                            SP2 159
         EJECT                                                            SP2 160
*    COMPUTE EXPRESSIONS, TYPE A ...  ENTRY1 + ENTRY2 - ENTRY3 * ENTRY4   SP2 161
*                        TYPE B ...  ENTRY1 - ENTRY2 * ENTRY3 + ENTRY4   SP2 162
*                        TYPE C ...  ENTRY1 * ENTRY2 + ENTRY3 - ENTRY4   SP2 163
*                                                                          SP2 164
         DROP  BASERG2                                                    SP2 165
*                                                                          SP2 166
CEXPRA   LH    CARDRG,YCARDAR              INITIALIZE REGISTER           SP2 167
         LH    PRINTRG,YPRINTAR            INITIALIZE REGISTER           SP2 168
*                                                                          SP2 169
         MVC   10(9,PRINTRG),MESS          MESSAGE TO PRINT AREA         SP2 170
         MVI   20(PRINTRG),TA              TYPE TO PRINTAREA             SP2 171
*                                                                          SP2 172
         MVC   25(3,PRINTRG),1(CARDRG)     NUMBER 1 TO PRINT AREA        SP2 173
         PACK  RESP+3(2),1(3,CARDRG)       PACK NUMBER 1 (N1)            SP2 174
*                                                                          SP2 175
         MVI   29(PRINTRG),C'+'            SIGN TO PRINT AREA            SP2 176
         MVC   31(3,PRINTRG),4(CARDRG)     NUMBER 2 TO PRINT AREA        SP2 177
         PACK  PFLD,4(3,CARDRG)            PACK NUMBER 2 (N2)            SP2 178
         AP    RESP+2(3),PFLD              N1 + N2                       SP2 179
*                                                                          SP2 180
         MVI   35(PRINTRG),C'-'            SIGN TO PRINT AREA            SP2 181
         MVC   37(3,PRINTRG),7(CARDRG)     NUMBER 3 TO PRINT AREA        SP2 182
         PACK  PFLD,7(3,CARDRG)            PACK NUMBER 3 (N3)            SP2 183
         SP    RESP+2(3),PFLD              N1 + N2 - N3                  SP2 184
*                                                                          SP2 185
         MVI   41(PRINTRG),C'*'            SIGN TO PRINT AREA            SP2 186
         MVC   43(3,PRINTRG),10(CARDRG)    NUMBER 4 TO PRINT AREA        SP2 187
         PACK  PFLD,10(3,CARDRG)           PACK NUMBER 4 (N4)            SP2 188
         MP    RESP,PFLD                   N1 + N2 - N3 * N4             SP2 189
*                                                                          SP2 190
         MVI   48(PRINTRG),C'='            EQUAL SIGN TO PRINT AREA      SP2 191
         UNPK  51(9,PRINTRG),RESP          RESULT TO PRINT AREA          SP2 192
         OI    59(PRINTRG),X'F0'           CHANGE SIGN IN ZONE TO F      SP2 193
         B     EDIT                                                       SP2 194
*                                                                          SP2 195
*                                                                          SP2 196
         USING BASEA21,BASERG2                                            SP2 197
*                                                                          SP2 198
CEXPRB   LH    CARDRG,YCARDAR              INITIALIZE REGISTER           SP2 199
         LH    PRINTRG,YPRINTAR            INITIALIZE REGISTER           SP2 200
*                                                                          SP2 201
         MVC   DPRINTAR+10(L'MESS),MESS    MESSAGE TO PRINT AREA         SP2 202
         MVI   DPRINTAR+20,TB              TYPE TO PRINT AREA            SP2 203
*                                                                          SP2 204
         MVC   DPRINTAR+25(L'NUMB1),NUMB1  NUMBER 1 TO PRINT AREA        SP2 205
         PACK  RESP+L'RESP-LNUMBP(LNUMBP),NUMB1 PACK NUMBER 1 (N1)       SP2 206
*                                                                          SP2 207
         MVI   DPRINTAR+29,C'-'            '-' TO PRINT AREA             SP2 208
         MVC   DPRINTAR+31(L'NUMB2),NUMB2  NUMBER 2 TO PRINT AREA        SP2 209
         PACK  PFLD,NUMB2                  PACK NUMBER 2 (N2)            SP2 210
         SP    RESP+L'RESP-LDIF(LDIF),PFLD N1 - N2                       SP2 211
```

Figure 20. Source Deck Listing, Part 4 of 7

```
*                                                                      SP2 212
         MVI   DPRINTAR+35,C'*'            '*' TO PRINT AREA           SP2 213
         MVC   DPRINTAR+37(L'NUMB3),NUMB3  NUMBER 3 TO PRINT AREA      SP2 214
         PACK  PFLD,NUMB3                  PACK NUMBER 3 (N3)          SP2 215
         MP    RESP,PFLD                   N1 - N2 * N3                SP2 216
*                                                                      SP2 217
         MVI   DPRINTAR+41,C'+'            '+' TO PRINT AREA           SP2 218
         MVC   DPRINTAR+43(L'NUMB4),NUMB4  NUMBER 4 TO PRINT AREA      SP2 219
         PACK  PFLD,NUMB4                  PACK NUMBER 4 (N4)          SP2 220
         AP    RESP,PFLD                   N1 - N2 * N3 + N4           SP2 221
*                                                                      SP2 222
         MVI   DPRINTAR+48,C'='            '=' TO PRINT AREA           SP2 223
         UNPK  DPRINTAR+51(L'RESP*2-1),RESP RESULT TO PR AREA          SP2 224
         OI    DPRINTAR+59,B'11110000'     CHANGE SIGN IN ZONE TO F    SP2 225
         B     EDIT                                                     SP2 226
*                                                                      SP2 227
*                                                                      SP2 228
CEXPRC LH    CARDRG,YCARDAR              INITIALIZE REGISTER           SP2 229
       LH    PRINTRG,YPRINTAR            INITIALIZE REGISTER           SP2 230
*                                                                      SP2 231
       MVC   DMESS,MESS                  MESSAGE AND                   SP2 232
       MVC   PTYPE,TYPE                  TYPE TO PRINT AREA            SP2 233
*                                                                      SP2 234
       MVC   PNUMB1,NUMB1                                              SP2 235
       PACK  RESPRI,NUMB1                N1                            SP2 236
*                                                                      SP2 237
       MVI   SIGN1,C'*'                                                SP2 238
       MVC   PNUMB2,NUMB2                                              SP2 239
       PACK  PFLD,NUMB2                  N2                            SP2 240
       MP    RESP,PFLD                   N1 * N2                       SP2 241
*                                                                      SP2 242
       MVI   SIGN2,C'+'                                                SP2 243
       MVC   PNUMB3,NUMB3                                              SP2 244
       PACK  PFLD,NUMB3                  N3                            SP2 245
       AP    RESP,PFLD                   N1 * N2 + N3                  SP2 246
*                                                                      SP2 247
       MVI   SIGN3,C'-'                                                SP2 248
       MVC   PNUMB4,NUMB4                                              SP2 249
       PACK  PFLD,NUMB4                  N4                            SP2 250
       SP    RESP,PFLD                   N1 * N2 + N3 - N4             SP2 251
*                                                                      SP2 252
       MVI   EQSIGN,C'='                                               SP2 253
       UNPK  RESZ,RESP                   RESULT TO PRINT AREA          SP2 254
       OI    RESZ+L'RESZ-1,C'0'          CHANGE SIGN TO F              SP2 255
         SPACE 2                                                       SP2 256
*    EDIT RESULT                                                       SP2 257
*                                                                      SP2 258
EDIT     LH    REG8,H1             INITIALIZE REGISTER                SP2 259
         LH    PRINTRG,YRESZ       LOAD ADDRESS OF RESULT FIELD ZONED SP2 260
TSTIF0   CLI   0(PRINTRG),C'0'     TEST IF 0                          SP2 261
         BNE   TSTSIGN             NO                                 SP2 262
         CH    REG8,YLRESZ         ALL BYTES CHECKED                  SP2 263
         BNL   PRNTLINE            YES                                SP2 264
         MVI   0(PRINTRG),C' '     MOVE BLANK                         SP2 265
         AH    PRINTRG,H1          NEXT BYTE                          SP2 266
         AH    REG8,H1             INCREMENT COUNTER BY 1             SP2 267
         B     TSTIF0              LOOP                               SP2 268
*                                                                      SP2 269
TSTSIGN  SH    PRINTRG,H1          ONE STEP BACK                      SP2 270
         TM    RESP+L'RESP-1,B'00000001'  TEST SIGN                   SP2 271
         BO    MINUS                                                  SP2 272
PLUS     MVI   0(PRINTRG),C'+'     MOVE PLUS SIGN                     SP2 273
         B     PRNTLINE                                               SP2 274
MINUS    MVI   0(PRINTRG),C'-'     MOVE MINUS SIGN                    SP2 275
         SPACE 2                                                      SP2 276
*    PRINT A LINE                                                      SP2 277
*                                                                      SP2 278
PRNTLINE LH    BRANCHRG,YPRINT                LOAD BRANCH-REGISTER    SP2 279
         BASR  LINKRG,BRANCHRG                PRINT A LINE AND RETURN SP2 280
         B     BGETCARD                       READ NEXT CARD          SP2 281
         SPACE 2                                                      SP2 282
*    HALT ROUTINES                                                     SP2 283
*                                                                      SP2 284
HALT0001 HPR   1,0                INVALID CARD TYPE                  SP2 285
         B     BGETCARD           NEXT CARD                          SP2 286
*                                                                      SP2 287
HALT001N STH   8,HPRI+2           STORE ENTRY NUMBER (N = 1, 2, 3 OR 4 SP2 288
         OI    HPRI+3,X'10'       'OR' COLUMN NUMBER (COL. 1)        SP2 289
         B     HPRI                                                   SP2 290
```

Figure 20. Source Deck Listing, Part 5 of 7

```
*                                                                      SP2 291
HALT002N STH   REG8,HPRI+2          ENTRY NUMBER (N = 1, 2, 3 OR 4)    SP2 292
         OI    HPRI+3,X'20'         NUMBER OF COL                      SP2 293
         B     HPRI                                                    SP2 294
*                                                                      SP2 295
HALT003N STH   REG8,HPRI+2          ENTRY NUMBER (N = 1, 2, 3 OR 4)    SP2 296
         OI    HPRI+3,X'30'         COL 3                              SP2 297
HPRI     HPR   0,0                  WILL BE UPDATED BY HALT ROUTINES   SP2 298
BGETCARD LH    BASERG1,YBASEA11     LOAD BASE REGISTER 1 WITH ORIG VALUE SP2 299
         USING BASEA11,BASERG1                                         SP2 300
         B     GETCARD              NEXT CARD                          SP2 301
       SPACE 2                                                         SP2 302
*    END OF CARD FILE                                                  SP2 303
*                                                                      SP2 304
EOCF LH BRANCHRG,YCLOSEF LOAD ADDRESS OF CLOSE ROUTINE          FREE   SP2 305
 BR BRANCHRG CLOSE THE FILES                                    FORMAT SP2 306
         TITLE 'ASSEMBLER SAMPLE, PART 2, DATA DEFINITIONS'             SP2 307
*    DATA DEFINITIONS                                                  SP2 308
*                                                                      SP2 309
H1       DC    H'1'                                                    SP2 310
H3       DC    H'3'                                                    SP2 311
H4       DC    H'4'                                                    SP2 312
*                                                                      SP2 313
YPRINT   DC    Y(PRINT)             ADDRESS OF PRINT ROUTINE           SP2 314
YCLOSEF  DC    Y(CLOSEF)            ADDRESS OF CLOSE ROUTINE           SP2 315
YCARDAR  DC    Y(CARDAR)            ADDRESS OF CARDAREA                SP2 316
YPRINTAR DC    Y(PRINTAR)           ADDRESS OF PRINTAREA               SP2 317
YRESZ    DC    Y(PRINTAR+51)        ADDRESS OF RESULT FIELD ZONED      SP2 318
YLRESZ   DC    Y(L'RESZ)            LENGTH OF RESZ                     SP2 319
YBASEA11 DC    Y(BASEA11)           BASE ADDRESS 1 1                   SP2 320
*                                                                      SP2 321
RESP     DC    PL5'0'               RESULT PACKED                      SP2 322
PFLD     DC    PL2'0'               PACK FIELD                         SP2 323
*                                                                      SP2 324
         ORG   RESP                 SET LOC. COUNTER BACK TO RESP      SP2 325
         DC    X'000000'            FIRST PART OF RESP                 SP2 326
RESPRI   DC    PL2'0'               RIGHT PART OF RESP                 SP2 327
LNUMBP   EQU   L'PFLD               LENGTH OF PACKED NUMBER            SP2 328
LDIF     EQU   LNUMBP               LENGTH OF DIFFERENCE               SP2 329
*                                                                      SP2 330
         ORG                        , TO PREVENT OVERLOADING OF PFLD   SP2 331
MESS     DC    C'CARD TYPE'                                            SP2 332
*                                                                      SP2 333
         TITLE 'ASSEMBLER SAMPLE, PART 2, DUMMY CONTROL SECTIONS'       SP2 334
DCS1     DSECT ,                    DUMMY CARDAREA                     SP2 335
*                                                                      SP2 336
DCARDAR  DS    0CL13                                                   SP2 337
TYPE     DS    C                                                       SP2 338
NUMB1    DS    CL3                                                     SP2 339
NUMB2    DS    CL3                                                     SP2 340
NUMB3    DS    CL3                                                     SP2 341
NUMB4    DS    CL3                                                     SP2 342
ENTRY    EQU   NUMB1                                                   SP2 343
CAEND    EQU   *                                                       SP2 344
         SPACE 2                                                       SP2 345
DCS2     DSECT ,                    DUMMY PRINTAREA                    SP2 346
*                                                                      SP2 347
DPRINTAR DS    0CL60                                                   SP2 348
         ORG   DPRINTAR+10                                             SP2 349
DMESS    DC    C'CARD TYPE'                                            SP2 350
         ORG   DPRINTAR+20                                             SP2 351
PTYPE    DS    C                                                       SP2 352
         ORG   DPRINTAR+25                                             SP2 353
PNUMB1   DS    CL3                                                     SP2 354
SIGN1    EQU   *+1                                                     SP2 355
         ORG   *+3                                                     SP2 356
PNUMB2   DS    CL3                                                     SP2 357
SIGN2    EQU   *+1                                                     SP2 358
         ORG   *+3                                                     SP2 359
PNUMB3   DS    CL3                                                     SP2 360
SIGN3    EQU   *+1                                                     SP2 361
         ORG   *+3                                                     SP2 362
PNUMB4   DS    CL3                                                     SP2 363
EQSIGN   EQU   *+2                                                     SP2 364
         ORG   *+5                                                     SP2 365
RESZ     DS    CL9                                                     SP2 366
PAEND    EQU   *                                                       SP2 367
         SPACE 2                                                       SP2 368
         END   BEGIN                                                   SP2 369
```

Figure 20. Source Deck Listing, Part 6 of 7

```
/*                                                                    SP2 370
// PAUSE INSERT ASSEMB OUTPUT DECKS OF BOTH ASSEMBLIES FOR LNKEDT RUN
// JOB CMAINT                        CORE IMAGE MAINTENANCE RUN
// EXEC R
// CATAL
// END
// JOB SAMPLE                        ASSEMBLER SAMPLE
// EXEC
A  1  2  3  4
B  1  2  3  4
C  1  2  3  4
A  5  6  7  8
B  5  6  7  8
C  5  6  7  8
A 45 71 28 39
B 45 71 28 39
C 45 71 28 39
A248935627310
B248935627310
C248935627310
/*
```

Figure 20. Source Deck Listing, Part 7 of 7

TPS Assembler Language Program

You may use the source deck from the DPS sample program in a tape-oriented system. However, to run the program under TPS you must make the following changes:

- replace the job control statements
- replace the AOPTN CROSSREF,ENTRY statement (SP2 001) by an AOPTN ENTRY statement.

The TPS job control statements with which you must replace the corresponding DPS job control statements are listed below.

```
Job control cards for first assembly
// LOG
// JOB ASSEMB
// ASSGN SYS000,X'780'T2
// ASSGN SYS001,X'781'T2
// ASSGN SYS002,X'782',T2
// ASSGN SYSOPT,X'300'P2
// DATE 70020
// EXEC

Job control cards for second assembly are
identical to those for DPS

Job control cards for Linkage Editor run
// JOB LNKEDT
// ASSGN SYSOPT,X'782',T2
// EXEC

Job control cards for CMAINT run
// JOB CMAINT
// ASSGN SYSIPT,X'782',T2
// FILES SYSIPT,REW
// ASSGN SYSOPT,X'781',T2
// EXEC
// CATAL
// END
```

Now you must perform a new IPL with the system tape created in the CMAINT run above. IPL is followed by

```
// LOG
// JOB SAMPLE
// DATE 70020
// EXEC
Data cards
```

You can study the general and detailed organization of the program in the description of the DPS sample program. The only differences which occur are in the Job Control, Linkage Editor and CMAINT programs, of which you can find a description in the SRL publication IBM System/360 Model 20, Tape Programming System, Control and Service Programs, Form GC24-9000.

For information on the Tape IOCS refer to the publication IBM System/360 Model 20, Tape Programming System, Input/Output Control System, Form GC24-9003.

Index

Indexes to System/360 Model 20 SRL publications are consolidated in the publication IBM System/360 Model 20, Disk Programming System: Master Index, Form GC33-6008.

(Where more than one page reference is given, major reference appears first.)

IBM System/360 Model 20 DPS/TPS Assembler Language GC24-9002-5

IBM®

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/360 Model 20
DPS/TPS Assembler Language

GC24-9002-5

- How did you use this publication?

  As a reference source ..... ☐
  As a classroom text ..... ☐
  As a self-study text ..... ☐

- Based on your own experience, rate this publication . . .

	.....	.....	.....	.....	.....
As a reference source:	Very Good	Good	Fair	Poor	Very Poor
As a text:	Very Good	Good	Fair	Poor	Very Poor

- What is your occupation? .....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . .

This SRL manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

CUT ALONG THIS LINE

Fold Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Department 813 BP

Fold Fold

IBM

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]