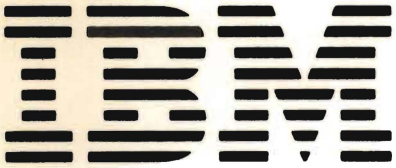
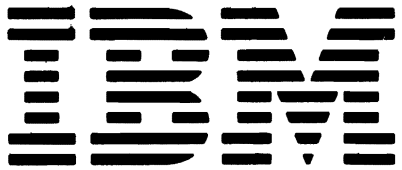


SC21-7705-3

File No. S34-21

IBM System/34
Basic Assembler and Macro Processor Reference Manual

Program Number 5726-AS1



SC21-7705-3

File No. S34-21

IBM System/34
Basic Assembler and Macro Processor Reference Manual

Program Number 5726-AS1

Fourth Edition (January 1982)

This is a major revision of, and obsoletes, SC21-7705-2 and Technical Newsletters SN21-8019 and SN21-8175. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

This edition applies to release 8, modification 0 of the IBM System/34 Basic Assembler and Macro Processor Program Product (Program 5726-AS1) and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Changes are periodically made to the information herein; changes wll be reported in technical newsletters or in new editions of this publication.

Use this publication only for the purposes stated in the *Preface*.

Publications are not stocked at the address below. Requests for copies of IBM publications and for technical information about the system should be made to your IBM representative or to the branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. Use the Reader's Comment Form at the back of this publication to make comments about this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 532, Rochester, Minnesota 55901. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

PURPOSE OF THE MANUAL

The *Basic Assembler and Macro Processor Reference Manual* is a reference manual for the programmer writing assembler programs for the IBM System/34. This manual is not intended to teach an inexperienced assembler programmer how to code assembler programs.

Readers are expected to use the manuals listed under *Related Publications* in this *Preface* for further information on how to code basic assembler language programs. For example, for a complete description of the formats of System/34 machine instruction statements that can be used in assembler programs, see the *IBM System/34 Functions Reference Manual.*

This program provides ideographic support when used with the ideographic version of the SSP and with the hardware devices that version supports.

HOW THIS MANUAL IS ORGANIZED

This publication is organized as follows:

- Chapter 1 explains the assembler functions and how these functions are executed.
- Chapter 2 presents the assembler language coding conventions and programming conventions. The assembler language format is described, as well as the three types of terms used to code the statements.
- Chapter 3 describes the assembler instruction statements.
- Chapter 4 describes the machine instruction statements and lists their mnemonic operation codes.
- Chapter 5 explains the macro processor and the coding of macroinstruction definitions.
- Chapter 6 describes the IBM-supplied macroinstructions and the general rules for coding macroinstructions.
- Chapter 7 contains programming considerations, information about assembler control statements, data files, OCL, and data management considerations.
- Chapter 8 lists and explains all printed messages issued by the assembler and the macro processor.
- Appendix A shows a sample program. Appendix A also shows two IBM-supplied macroinstruction definitions and related macroinstruction expansions.
- Appendix B shows the coded character set for EBCDIC (extended binary coded decimal interchange code).

A glossary provides a list of both new and familiar terms.

Note: Because the manual is arranged for reference purposes, certain terms appear, of necessity, earlier in the manual than the discussions explaining them. The reader who encounters a problem of this sort should refer to the index, which will direct the reader to the term's definition or explanation.

SYSTEM REQUIREMENTS

Refer to the *IBM System/34 Planning Guide*, GC21-5154, for the System/34 assembler compiler requirements.

PREREQUISITE PUBLICATIONS

- *IBM System/34 Introduction*, GC21-5153
- *IBM System/34 Planning Guide*, GC21-5154
- *IBM System/34 System Support Reference Manual*, SC21-5155

RELATED PUBLICATIONS

- *IBM System/34 Assembler Reference Summary*, GX21-7674
- *IBM System/34 Concepts and Design Guide*, SC21-7742
- *IBM System/34 Installation and Modification Reference Manual: Program Products and Physical Setup*, SC21-7689
- *IBM System/34 System Data Areas and Diagnostic Aids Manual*, LY21-0049
- *IBM System/34 Data Communications Reference Manual*, SC21-7703
- *IBM System/34 Interactive Communications Feature Reference Manual*, SC21-7751
- *IBM System/34 Sort Reference Manual*, SC21-7658
- *IBM System/34 Ideographic Sort Reference Manual*, SC21-7850
- *IBM System/34 1255 Magnetic Character Reader Reference Manual*, SC21-7740
- *IBM System/34 Scientific Macroinstruction Reference Manual*, SA21-9275
- *IBM System/34 Overlay Linkage Editor Reference Manual*, SA21-7707
- *IBM System/34 Functions Reference Manual*, SA21-9243
- *IBM System/34 Bibliography*, GH30-0231
- *IBM System/34 Displayed Messages Guide*, SC21-5159
- *IBM System/34 Master Index*, SC21-7739

ASSEMBLER CODING MATERIAL

IBM System/34 Basic Assembler Coding Form, GX21-9279

Contents

Chapter 1. Introduction

The IBM System/34 Basic Assembler and Macro Processor Program Product consists of two distinct parts: the assembler processor and the macro processor. The macro processor is the first to scan the source program. When it encounters a macroinstruction statement, the macro processor refers to a previously coded and stored macroinstruction definition and uses the information in that definition and the parameters coded in the macroinstruction statement to expand the macroinstruction statement into a series of assembler instruction statements and/or machine instruction statements. These statements are inserted in the source program and the original macroinstruction statement is modified to appear as a comment.

The IBM-supplied macroinstructions perform both system services and input/output device support.

After the macro processor has expanded each macroinstruction statement in the source program, the assembler receives control. The assembler translates the machine instruction statements into a form usable by the IBM System/34 and assigns relative storage addresses to all statements, constants, and storage areas.

Thus, the principal function of the IBM System/34 Basic Assembler and Macro Processor program product is to translate assembler language source programs into machine language object programs. Therefore, to write source programs to be assembled by the program product, you must be familiar with the basic assembler language.

IBM SYSTEM/34 BASIC ASSEMBLER LANGUAGE

The IBM System/34 Basic Assembler language is a symbolic programming language and must be translated into a form usable by the computer before execution. This computer-usable form is called machine language or object code. The IBM System/34 Basic Assembler language provides a convenient method for representing, on a one-for-one basis, machine instruction statements and related data necessary to write a program for execution by any model of the IBM System/34.

This one-for-one relationship to machine language makes the assembler language versatile. Further versatility is available because the assembler programmer can refer to instructions, data areas, and other program elements by symbolic names, as well as actual machine addresses. Also available are the EBCDIC bit pattern, binary arithmetic capabilities, and access to SSP blocks such as DTFs and IOBs. The only restrictions to be considered are machine restrictions. It is possible to write some programs that will execute faster because in unique situations a programmer may see ways to code more efficiently than the routine procedures a compiler would generate.

Assembler Language Statements

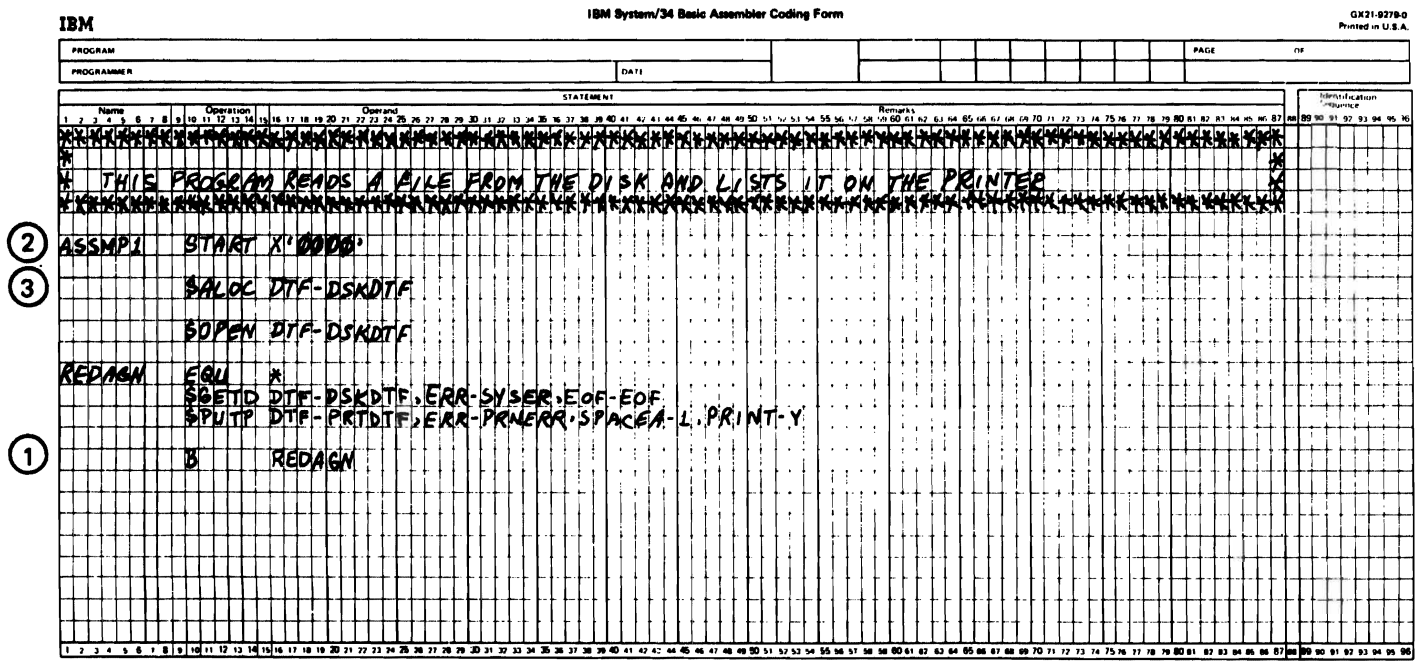
The basic assembler language is composed of assembler language statements that use symbols, called mnemonics, to represent the operation codes of three types of assembler language statements. The three types of assembler language statements are as follows:

1. Machine instruction statements represent machine language instructions on a one-for-one basis. The symbolically represented machine instruction statements are translated into executable machine language code by the assembler processor.

2. Assembler instruction statements control the functions of the assembler. Each assembler instruction statement causes the assembler to perform a specific operation during the assembly process but is not translated into executable machine language code by the assembler processor.

3. Macroinstruction statements represent a sequence of machine and/or assembler instruction statements. Each macroinstruction statement causes the macro processor to select and/or modify assembler language statements found in the definition of the macroinstruction.

Figure 1-1 shows an example of each type of assembler language statement.



(1) Machine instruction statement.

(2) Assembler instruction statement.

(3) Macroinstruction statement.

Figure 1-1. Example of Assembler Language Statement Types

Chapter 2. Assembler Language

In order to code in assembler language, the programmer must become familiar with certain definitions, coding conventions, instructions, and other features of the language. This chapter deals with these items.

ASSEMBLER LANGUAGE SOURCE PROGRAM RECORDS

A source program is a sequence of assembler language statements. The body of each record is composed of two parts. The first part contains the assembler language statement, which is normally in columns 1-87. The column following the assembler language statement, column 88, must always be blank. The second part of the record contains the optional identification-sequence number that is normally in columns 89-96.

The input format control (ICTL) assembler instruction can change the columns defined as the assembler language statement. The input sequence checking (ISEQ) assembler instruction can change the columns defined as identification sequence number in the source program. These assembler instructions are explained in Chapter 3.

CHARACTER SET

Source statements are written using the following characters:

Letters	A through Z, and $, #, @
Digits	0 through 9
Special characters	+ - , . * () ' blank

In addition, any valid character available on the input device may be designated between single quotes, and in remarks and comments. Note that not all print belts available on System/34 have all the special characters listed above. The assembler will accept all characters as input, but those characters not on the print belt will not be printed.

CODING CONVENTIONS

A coding sheet that contains suggested columns for each entry is provided. The coding form is shown in Figure 2-1. Space is provided at the top for program identification and instructions to the operator; this information does not become part of the source program. The coding examples in this book do not show this part of the form.

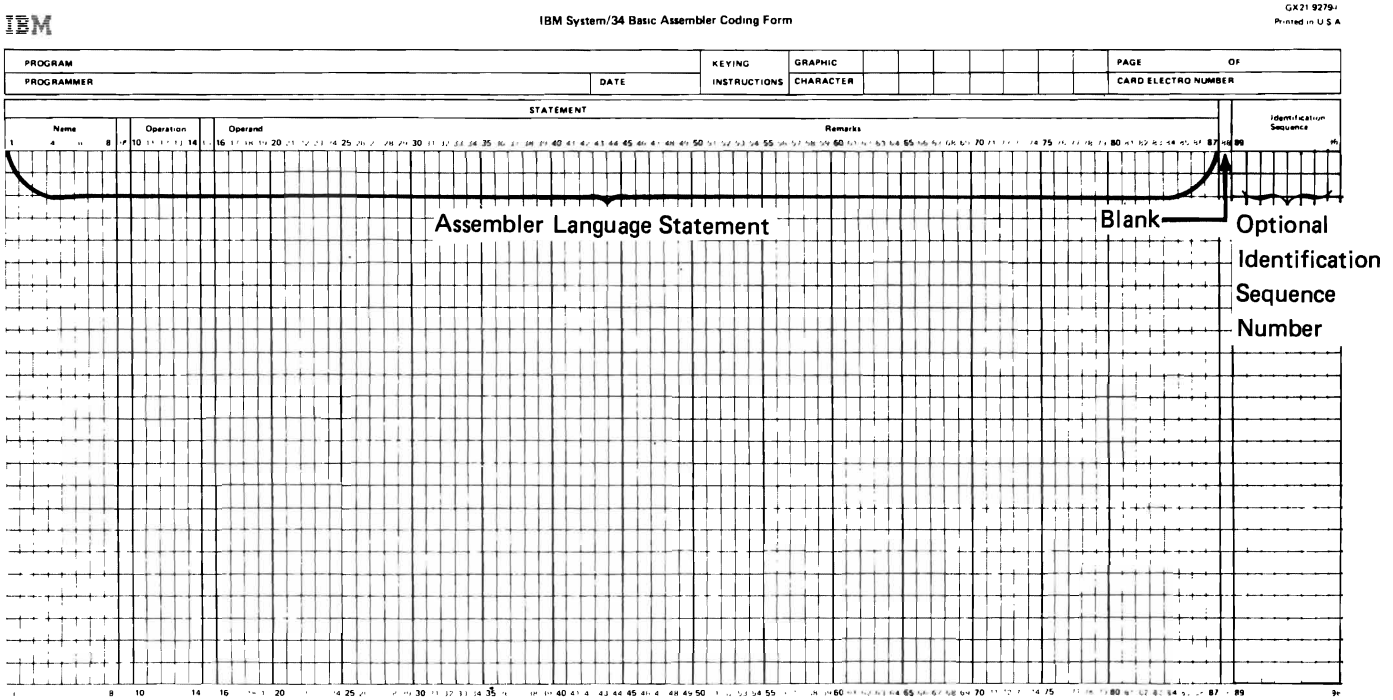
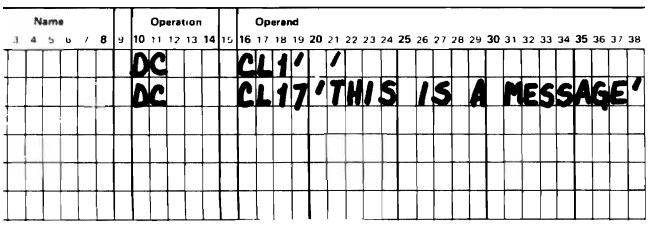


Figure 2-1. Sample Coding Sheet.

Assembler Language Statement Entries

Assembler language statements may consist of from one to four entries. They are from left to right: a name entry, an operation entry, an operand entry, and a remarks entry. See Figure 2-2 for an example of each.

The delimiter that separates individual entries is one or more blanks. Operands can contain a blank only when a constant is defined and enclosed in apostrophes. For example:



Name Entry

A name is a symbol created by the programmer to identify an assembler language statement, storage area, or value. The name entry, if allowed, is optional except on the equate (EQU) instruction. The symbol may be from one to eight characters with the exception of the name entry used in a START instruction. The name entry in a START instruction is limited to six characters. See *START Instruction* in Chapter 3 for more information.

The first character of the name entry must appear in column 1 and must be a letter. (See *Character Set* earlier in this chapter.) The remaining characters may be letters or digits, but not special characters. The name entry, if used, must be followed by at least one blank. If column 1 is blank, the assembler program assumes no name has been entered. (See Figure 2-2.)

Operation Entry

The operation entry is the mnemonic operation code specifying the machine instruction, assembler instruction, or macroinstruction desired. An operation entry is mandatory and may not start in column 1. Valid mnemonic operation codes consist of from one to five characters. The operation entry must be followed by at least one blank. (See Figure 2-2.)

Operand Entry

Operand entries identify and describe data to be acted upon by the operation, by indicating such things as storage locations, registers, affected masks, storage area lengths, or types of data. Operands are required for all machine instruction statements and for certain assembler instruction statements. An operand is defined as a term or an arithmetic combination of terms. (See *Terms* in this chapter.)

Operand entries must be separated by commas, and no blanks may intervene between operands and the commas that separate them. The last operand must be followed by at least one blank.

Remarks Entry

Remarks are descriptive items of information about the program. Any valid character available on the input device may be used in writing a remark. The remark follows the operand and must be separated from it by at least one blank. Remarks cannot extend beyond the end column. (See Figure 2-2.)

Comment Statements

An entire statement field, columns 1-87, may be used for a comment by placing an asterisk (*) in column 1. Extensive comment entries may be written by using a series of lines with an asterisk in column 1 of each line. Comments may appear anywhere in the source program and are printed in the assembly listing, but do not have any effect on the execution or main storage requirements of the program. (See Figure 2-2.)

Identification Sequence Entry

The optional identification sequence entry is used to enter program identification and/or statement sequence characters. To aid in keeping source statements in order, the programmer may number the records in this field. These characters are placed in their respective records and during the assembly the programmer may request the assembler to verify this sequence by use of the input sequence (ISEQ) assembler instruction.

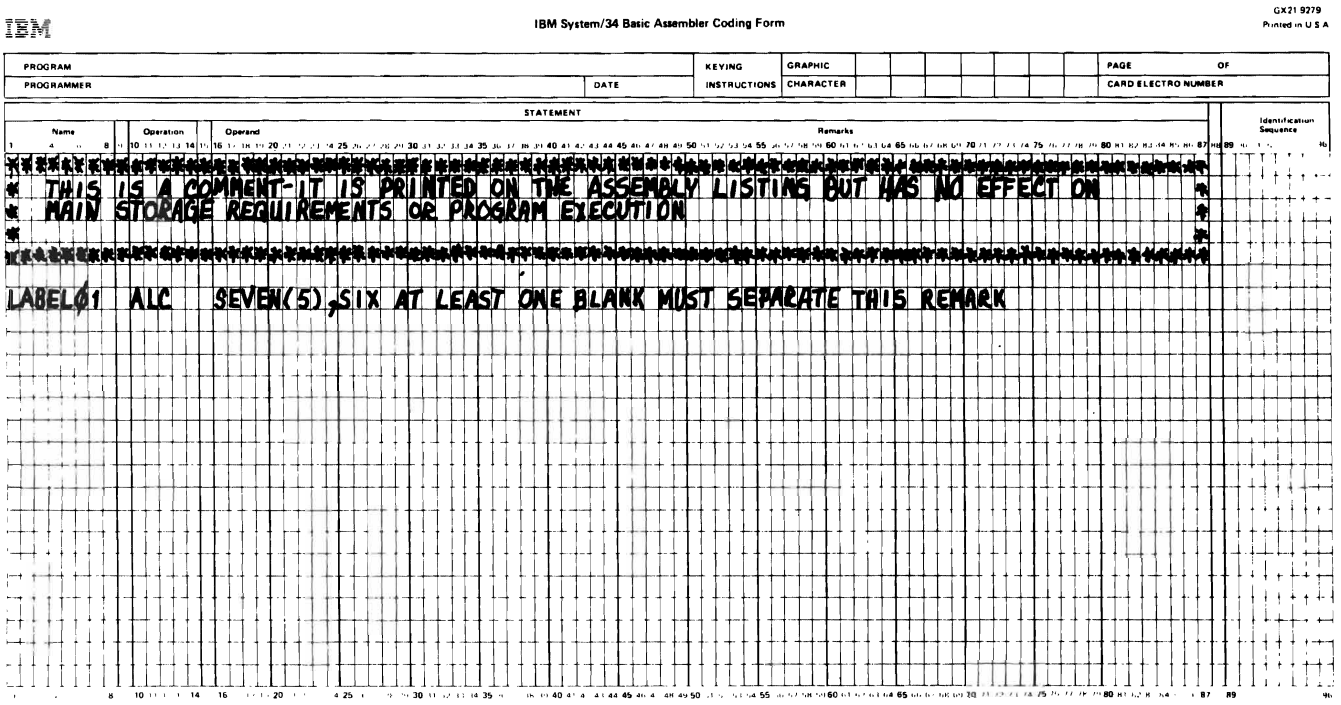


Figure 2-2. Entry Examples of Assembler Language Statements

ASSEMBLER PROGRAM CONVENTIONS

A term is a single symbol, a self-defining value, or a location counter reference. A term is used only in the operand field of an assembler language statement. The three types of terms are described under *Terms* in this section.

An expression consists of one or more terms. The operand fields of assembler language instructions consist of one or more expressions.

Terms and expressions are classed as either absolute or relocatable. A term or expression is absolute if its value is not changed when the assembled program in which it is used is relocated in main storage. A term or expression is relocatable if its value is changed when the program in which it is used is relocated.

Program relocation is the loading of an assembled program (object program) into a different area of main storage from that which was originally assigned by the assembler. The difference in bytes between the originally assigned address of the object program and the address of the relocated object program is the amount of relocation. The addresses assigned to all statements and data in the relocated program are changed by the amount of relocation.

Programs are assembled to begin at address 0000, unless the START statement specifies a different address. If a program is not started on a 2K boundary, however, a dump of the program at execution time shows the program as beginning at the next lowest 2K boundary.

Expressions

The rules for coding an expression are:

1. Two terms or two operators must not be used consecutively in an expression.

2. Parentheses cannot be used in an expression.

3. Only absolute terms can be used in a multiplication operation.

4. Blanks are not allowed in an expression.

5. An expression must be of the form A or A±e when it contains an external symbol. A is the symbol used as the operand of an EXTRN statement, and e is an absolute expression. Any symbol equated to an expression of this form cannot be used in an expression of more than one term.

If there is more than one term in the expression, the terms are reduced to a single value as follows:

1. Each term is evaluated separately.

2. Arithmetic operations are then performed in a left-to-right sequence, except that multiplication is performed before addition or subtraction. Example: A + B*C would be evaluated as A + (B*C), not (A + B)*C. The result would be the value of the expression.

3. The intermediate result of the expression evaluation is a 3-byte, or 24-bit, value. Intermediate results must be in the range of -2^{24} through $2^{24}-1$.

Negative values are carried in the twos complement form. The final value of the expression is the truncated, rightmost 16 bits of the result. In an address constant, the amount of truncation and the length of the result depend on the length of the constant. The value of the expression before truncation must be in the range of -65536 through +65535. A negative result is considered to be a 2-byte positive value.

Absolute Expressions: An absolute expression is one whose value is unaffected by program relocation.

An absolute term may be a nonrelocatable symbol, or any of the self-defining terms. All arithmetic operations are permitted between absolute terms.

An absolute expression can contain relocatable terms or a combination of relocatable and absolute terms under the following conditions:

1. The expression must contain an even number of relocatable terms.

2. The relocatable terms must be paired and each pair must consist of terms with opposite signs. The paired terms need not be adjacent.

3. Relocatable terms cannot be used in a multiplication operation.

Pairing relocatable terms with opposite signs cancels the effect of the relocation, because both terms would be relocated by the same value. Therefore, the value represented by the paired terms would, in effect, remain constant regardless of the program relocation. For example, in the absolute expression A - Y + X, A is an absolute term and X and Y are relocatable terms. If A equals 50, Y equals 25, and X equals 10, the value of the expression would be 35. If X and Y are relocated by a factor of 100, their values would become 110 and 125, respectively. However, the expression would still evaluate as 35 (50 - 125 + 110 = 35).

Relocatable Expressions: A relocatable expression is one whose value changes by the amount of relocation when the program in which it is used is relocated. Every relocatable expression must reduce to a positive value.

A relocatable expression can be a combination of relocatable and absolute terms under the following conditions:

1. There must be an odd number of relocatable terms.
2. All relocatable terms, except one, must be paired and each pair must consist of terms with opposite signs. The paired terms need not be adjacent.
3. The unpaired term must not be immediately preceded by a minus sign.
4. Relocatable terms cannot enter into a multiplication operation.

All terms in a relocatable expression are reduced to a single value, which is the value of the unpaired relocatable term after it has been adjusted (displaced) by the values of the other terms in that expression. For example, in the expression W - X + Y where W, X, and Y are relocatable terms; and W = 10, X = 3, Y = 1 before relocation; the result is the relocatable value of 8.

If the program is relocated by 100 bytes, the resultant value of the expression would be increased by the amount of relocation (100), giving the expression a value of 108.

In the following expression, a combination of absolute and relocatable terms are used: A + F*G - D + B. A, D, and B are relocatable terms; F and G are absolute terms. When given the values A = 3, B = 2, D = 5, F = 1, and G = 4, the result would be a relocatable value of 4. The multiplication occurs first, resulting in 4; then the addition and subtraction of the other terms, including the result of the multiplication, is performed in a left-to-right direction. The result of the arithmetic operations is a relocatable value of 4 for this expression.

Upon relocation, the value of this expression can be determined by adding the amount of relocation to all relocatable terms.

Terms

Every term represents a value. This value may be assigned by the assembler (for symbols and for location counter references) or may be inherent in the term itself (that is, the term may be self-defining). An arithmetic combination of terms, an expression, is reduced to a single value by the assembler.

Symbolic Terms

A symbolic term is a character or combination of characters used to represent a storage location, a register, or an arbitrary value.

Symbols, through their uses as name entries and operand entries, provide the programmer with an easy way to name and reference a program element. The assembler assigns values to symbols appearing as name entries in a source statement. The values assigned to symbols in the name entry of the machine instruction statement are the addresses of the leftmost bytes of the storage records containing the statements. The values assigned to symbols naming storage areas and constants are the addresses of the rightmost bytes of the storage fields containing these items. The symbols naming them are considered relocatable terms because the addresses of these items may change upon relocation. A length attribute is also assigned by the assembler.

A symbol that is a name entry in the equate symbol (EQU) assembler instruction statement is assigned the value designated in the operand entry of the statement. Since the operand entry may represent a relocatable value or an absolute value, the symbol is considered a relocatable term or an absolute term, depending upon the value it is equated to. The length attribute of the symbol on an EQU instruction is obtained from the operand entry.

The value of a symbol may not be negative and may not exceed 2^{16}-1 (65535).

A symbol is said to be defined when it appears as the name entry of a source statement or the operand of an EXTRN. EQU statements require that a symbol appearing in the operand entry be previously defined. In this case, the symbol, before its use in the operand, must have appeared as a name entry in a prior statement.

A symbol may be defined only once in an assembly. That is, each symbol used as the name of a statement or operand of an EXTRN must be unique within that assembly.

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. For example, the decimal self-defining term 15 represents a value of 15.

There are four types of self-defining terms: decimal, hexadecimal, binary, and character. Use of one of these terms is the decimal, hexadecimal, binary, or character representation of the corresponding machine-language binary value or bit configuration. The length attribute of a self-defining term is always one.

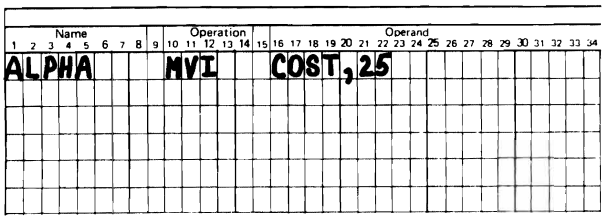
Self-defining terms are the means of specifying machine values or bit configurations without equating the values to symbols and using the symbols.

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

The use of a self-defining term is quite distinct from the use of a data constant. When a self-defining term is used in a machine-instruction statement, its value is assembled into the instruction. When a data constant is referred to in the operand of an instruction, its address is assembled into the instruction.

Self-defining terms are always right-justified; truncation or padding with zeros, if necessary, occurs on the left.

Decimal Self-Defining Terms: A decimal self-defining term is a decimal number written as a sequence of decimal digits. High-order zeros may be used (for example, 007). Limitations on the value of the term depend on its use. For example, a decimal term that represents an address should not exceed the size of storage. In any case, a decimal term may not consist of more than five digits or exceed 2^{16}-1 (65535). A decimal self-defining term is assembled as its binary equivalent. Some examples of decimal self-defining terms are: 8, 147, 4092, and 00021. In the following example, a decimal self-defining term is used in a move immediate (MVI) machine instruction. The 1-byte area referenced by the symbol, COST, would contain the decimal value 25 (binary 00011001) after execution of the instruction.



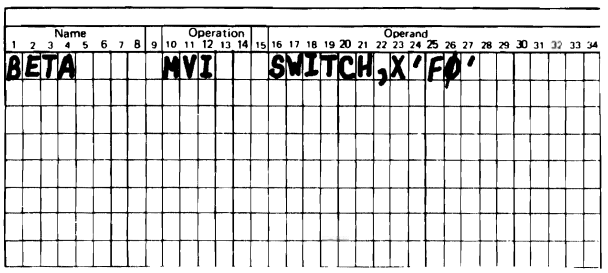
Hexadecimal Self-Defining Term: A hexadecimal self-defining term consists of one to four hexadecimal digits enclosed by apostrophes and preceded by the letter X. An example is X'409'.

Each hexadecimal digit is assembled as its 4-bit binary equivalent. Thus, a hexadecimal term used to represent an 8-bit mask would consist of two hexadecimal digits. The maximum value of a hexadecimal term is X'FFFF'.

The hexadecimal digits and their bit patterns are as follows:

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

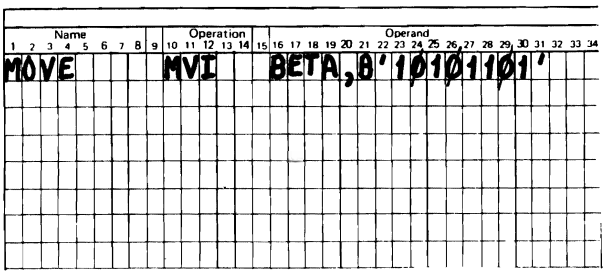
The following is an example of the use of a hexadecimal self-defining term. The 1-byte area referenced by the symbol SWITCH would contain the hexadecimal value F0 (binary 11110000) after execution of the instruction.



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, as follows: B'10001101'. This term would appear in storage as shown, occupying one byte. A binary term may represent up to 16 bits.

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

The following example illustrates a binary term used as immediate information in a move immediate (MVI) machine instruction. The byte of immediate information specified will replace the byte of information referenced by the symbol BETA.



Character Self-Defining Term: A character self-defining term consists of one or two characters 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 remaining valid character (character available on the input device) may be designated. The following are examples of character self-defining terms:

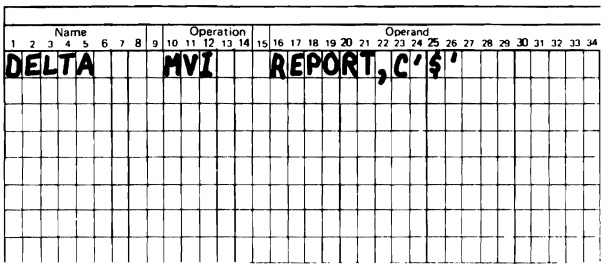
C'/' C'AB' C'13'

Because of the use of apostrophes in the assembler language as syntactic characters, the following rule must be observed when using an apostrophe in a character term:

> For each apostrophe desired in a character self-defining term, two apostrophes must be written. For example, the character value A' would be written as 'A'''.

Each character in the character sequence is assembled as its 8-bit code equivalent. The two apostrophes that must be used to represent an apostrophe are assembled as an apostrophe.

In the following example, a dollar sign ($) would be moved into the 1-byte field at REPORT.



Location Counter Reference

A location counter is used to assign storage addresses. It is the assembler's equivalent of the instruction counter in the computer. As each assembler language statement or data area is assembled, the location counter is incremented the number of bytes used by the assembled item. Thus, it always points to the next available location. If a statement defining an instruction is named by a symbol, the value attribute of the symbol is the value of the location counter before addition of the length. If the statement defines storage or a constant, the value attribute of the symbol is one less than the value of the location counter after addition of the length.

The location counter setting can be controlled by using the START and ORG assembler control statements. The maximum value for the location counter is 2^{16}-1 (65535).

The programmer may refer to the current value of the location counter at any place in a program by using an asterisk (*) as a term in an operand. The asterisk represents the location of the first byte of currently available storage. For example:

	Source		Generated
Location counter	= 1100		
Relocatable	LAB2	DC AL2 (*)	1100
	LAB2	DC AL2 (LAB2)	1101
Nonrelocatable	LAB2	DC AL2 (1100)	1100

Expressions

An expression is an arithmetic combination of terms. Two types of expressions are used, absolute and relocatable. The arithmetic operators are:

+ addition
- subtraction
* multiplication

The following are examples of valid expressions:

AREA+X'2D'	N-25	5*C' '	*+15
AREA's value plus a hexadecimal 2D	N's value minus a decimal 25	Decimal 5 times the hexadecimal 40 (character blank = hexadecimal 40)	Current value of the location counter plus a decimal 15

Addressing

The two methods of addressing available allow the assembler programmer to access any part of storage. These methods are direct addressing and base displacement addressing. The relative addressing technique can be used with both methods.

Direct Addressing

The direct addressing method allows the programmer to represent a 16-bit instruction address by using an expression as an operand entry. The assembler places the value of the expression in the machine instruction which it generates.

Two bytes are always used in the machine instruction for a direct address. A direct address is indicated by the absence of a register in the operand.

Figure 2-3 shows an example of direct addressing.

Base Displacement Addressing

Base displacement addressing involves setting up a base address from which other addresses can be calculated. This base address must be placed in an index register before the index register is used for addressing. One byte is used in the machine instruction for a base displacement address and is indicated by the presence of a register in the operand. Any one value of an index register allows access to 256 storage positions.

You can code the USING statement to make the contents of an index register the basis for base displacement addressing. You can code the DROP statement to terminate base displacement addressing. For information about the USING and DROP statements, see their descriptions in Chapter 3.

Figure 2-4 shows examples of base displacement addressing.

```
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT
    0000                        1 EXAMP    START 0
                                2          PRINT NODATA
                                4 ********************************************************************
                                5 *                                                                  *
                                6 *        AN EXAMPLE OF DIRECT ADDRESSING.                          *
                                7 *                                                                  *
                                8 ********************************************************************
    0000 0C 1D 0064 002F       10          MVC   NAME2,NAME1           MOVE "NAME" OF AREA1 TO "NAME" OF AREA2
    0006 0C 07 006C 0037       11          MVC   PHON2,PHON1           MOVE "PHON" OF AREA1 TO "PHON" OF AREA2
    000C 0C 0E 007B 0046       12          MVC   CITY2,CITY1           MOVE "CITY" OF AREA1 TO "CITY" OF AREA2


                          0012 14 AREA1    EQU   *
    0012 D1D6C8D540D14B40 002F 15 NAME1    DC    CL30'JOHN J. SMITH III' "NAME" OF AREA1
    0030 F2F8F6404F3E3F7F2 0037 16 PHON1   DC    CL08'286-5372'          "PHON" OF AREA1
    0038 D9D6C3C8C5E2E3C5 0046 17 CITY1    DC    CL15'ROCHESTER'         "CITY" OF AREA1


                          0047 19 AREA2    EQU   *
    0047                  0064 20 NAME2    DS    CL30                    "NAME" OF AREA2
    0065                  006C 21 PHON2    DS    CL08                    "PHON" OF AREA2
    006D                  007B 22 CITY2    DS    CL15                    "CITY" OF AREA2
                          0000 23          END   EXAMP
```

Figure 2-3. Example of Direct Addressing

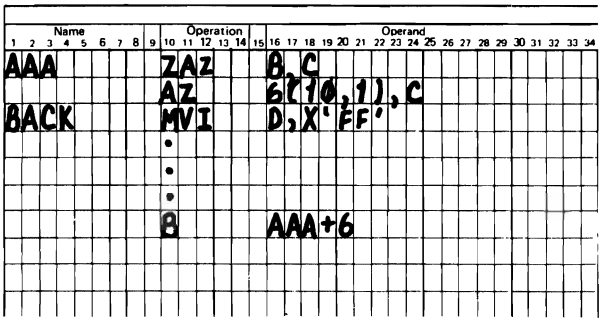
```
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT
    0000                        1 EXAMP1   START 0
                                2          PRINT NODATA
                                4 ***************************************************************
                                5 *                                                             *
                                6 *        AN EXAMPLE OF BASE-DISPLACEMENT ADDRESSING WITH THE  *
                                7 *        "USING" INSTRUCTION.                                 *
                                8 *                                                             *
                                9 ***************************************************************
    0000 C2 01 0014                11          LA    AREA1,R1              POINT TO MOVING "FROM" FIELD
                          0014     12          USING AREA1,R1              SET TO USE LABELS AS DISPLACEMENTS FROM AREA1
    0004 C2 02 0049                14          LA    AREA2,R2              POINT TO MOVING "TO" FIELD
                          0014     15          USING AREA1,R2              SET TO USE LABELS OF AREA1 AS DISPLACEMENTS INTO
                                   16 *                                       AREA2.
    0008 9C 1D 1D 1D               18          MVC   NAME(,R2),NAME(,R1)   MOVE "NAME" OF AREA1 TO "NAME" OF AREA2
    000C 9C 07 25 25               19          MVC   PHON(,R2),PHON(,R1)   MOVE "PHON" OF AREA1 TO "PHON" OF AREA2
    0010 9C 0E 34 34               20          MVC   CITY(,R2),CITY(,R1)   MOVE "CITY" OF AREA1 TO "CITY" OF AREA2

                          0014     22 AREA1   EQU   *
    0014 D1D6C8D540D14B40 0031     23 NAME    DC    CL30'JOHN J. SMITH III' "NAME" OF AREA1
    0032 F2F8F860F5F3F9F2 0039     24 PHON    DC    CL08'288-5392'          "PHON" OF AREA1
    003A D9D6C3C8C5E2E3C5 0048     25 CITY    DC    CL15'ROCHESTER'         "CITY" OF AREA1

                          0049     27 AREA2   EQU   *
    0049                  0066     28         DS    CL30                    "NAME" OF AREA2
    0067                  006E     29         DS    CL08                    "PHON" OF AREA2
    006F                  007D     30         DS    CL15                    "CITY" OF AREA2

                          0001     32 R1      EQU   1                       EQUATE FOR REGISTER 1
                          0002     33 R2      EQU   2                       EQUATE FOR REGISTER 2
                          0000     34         END   EXAMP1
```

```
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT
    0000                        1 EXAMP2   START 0
                                2          PRINT NODATA
                                4 ***************************************************************
                                5 *                                                             *
                                6 *        AN EXAMPLE OF BASE DISPLACEMENT ADDRESSING           *
                                7 *        USING "EQUATES"                                      *
                                8 *                                                             *
                                9 ***************************************************************
    0000 C2 01 0014                11          LA    AREA1,R1              POINT TO MOVING "FROM" FIELD
    0004 C2 02 0049                13          LA    AREA2,R2              POINT TO MOVING "TO" FIELD
    0008 9C 1D 1D 1D               15          MVC   NAME(30,R2),NAME(,R1) MOVE "NAME" OF AREA1 TO "NAME" OF AREA2
    000C 9C 07 25 25               16          MVC   PHON(08,R2),PHON(,R1) MOVE "PHON" OF AREA1 TO "PHON" OF AREA2
    0010 9C 0E 34 34               17          MVC   CITY(15,R2),CITY(,R1) MOVE "CITY" OF AREA1 TO "CITY" OF AREA2

                          0014     19 AREA1   EQU   *
    0014 D1D6C8D540D14B40 0031     20         DC    CL30'JOHN J. SMITH III' "NAME" OF AREA1
    0032 F2F8F860F5F3F9F2 0039     21         DC    CL08'288-5392'          "PHON" OF AREA1
    003A D9D6C3C8C5E2E3C5 0048     22         DC    CL15'ROCHESTER'         "CITY" OF AREA1

                          0049     24 AREA2   EQU   *
    0049                  0066     25         DS    CL30                    "NAME" OF AREA2
    0067                  006E     26         DS    CL08                    "PHON" OF AREA2
    006F                  007D     27         DS    CL15                    "CITY" OF AREA2

                          001D     29 NAME    EQU   29                      "NAME" DISPLACEMENT INTO AREAS 1 & 2
                          0025     30 PHON    EQU   NAME+8                  "PHON" DISPLACEMENT INTO AREAS 1 & 2
                          0034     31 CITY    EQU   PHON+15                 "CITY" DISPLACEMENT INTO AREAS 1 & 2

                          0001     33 R1      EQU   1                       EQUATE FOR REGISTER 1
                          0002     34 R2      EQU   2                       EQUATE FOR REGISTER 2
                          0000     35         END   EXAMP2
```

Figure 2-4. Examples of Base Displacement Addressing

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 or instructions. Thus the expression *+4 specifies an address that is 4 bytes greater than the current value of the location counter. In the sequence of instructions shown in the following example, the instruction with the operation code ZAZ has a length of 6 bytes, the instruction AZ has a length of 5 bytes, and the instruction with MVI has a length of 4 bytes in storage. Using relative addressing, the location of the AZ machine instruction can be expressed in two ways: AAA+6 or BACK-5.



Instruction Addressing

A symbol used as a name entry in a machine-instruction statement addresses the *leftmost* byte of storage occupied by that instruction.

Data Addressing

A symbol used as a name entry in a data definition instruction (see *DC—Define Constant and DS—Define Storage*) addresses the *rightmost* byte of storage occupied by or reserved for that data.

Program Linking References

Symbols may be defined in one program and referred to in another, thus linking independently assembled programs.

The linkages can be made only if the assembler is able to provide information about the linkage symbols to the overlay linkage editor, which resolves these linkage references at link edit time. The assembler places the necessary information in the external symbol list (ESL) on the basis of the linkage symbols identified by the ENTRY and EXTRN instructions. These symbolic linkages are described as linkages between independent assemblies. The name of a START statement (the module name) also has an external attribute and may be used for program linking.

The linkage symbol is identified to the assembler by means of the ENTRY assembler instruction. Once a linkage symbol is identified in a program as a symbol that names an entry point, another program may use that symbol in a branch operation or as a data reference.

Similarly, the program that uses a symbol defined in some other program must identify it by the EXTRN assembler instruction because the symbol is used by the first program to link to the point identified by the symbol in the second program. The formats of the EXTRN and ENTRY assembler control instructions are in Chapter 3.

Chapter 3. Assembler Instruction Statements

Assembler instruction statements are requests to the assembler to perform certain operations during assembly time. Assembler instruction statements, in contrast to machine instruction statements, are not translated into machine language. Some, such as DS and DC, do cause storage areas to be set aside for constants and other data. Others, such as EQU and SPACE, are effective only at assembly time; they generate nothing in the object program and have no effect on the location counter.

There are four types of assembler instruction statements: symbol definition, data definition, listing control, and assembler processor control. This chapter explains each assembler instruction statement in detail. For a complete list of the assembler instruction statements and their operations, see Figure 3-1.

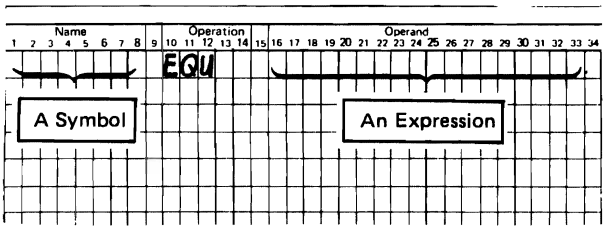
Type	Operation Code	Operation
Symbol definition instruction	EQU	Equate symbol
Data definition instructions	DS DC	Define storage Define constant
Listing control instructions	TITLE	Identify assembly output
	EJECT	Start new page
	SPACE	Space listing
	PRINT	Control program listing
Assembler Processor control instructions	ISEQ	Input sequence checking
	ORG	Set location counter
	START	Start assembly
	USING	Use index register for base-displacement addressing
	DROP	Drop index register for base-displacement addressing
	ENTRY	Identify entry-point symbol
	EXTRN	Identify external symbol
	ICTL	Input format control
	END	End assembly

Figure 3-1. Assembler Instruction Statements

SYMBOL DEFINITION

EQU–Equate Symbol

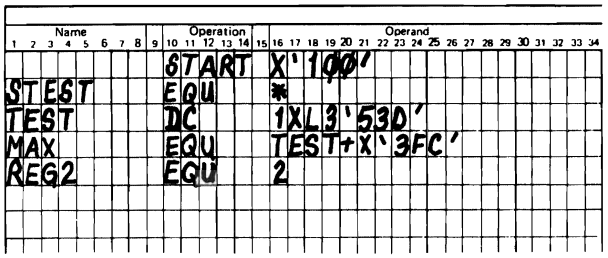
The EQU assembler instruction statement is used to define a symbol by assigning it to the value, length, and relocatability attributes of an expression in the operand field. The format of the EQU control statement is as follows:



The expression in the operand field may be absolute or relocatable. Any symbols appearing in the expression must be previously defined. The name and operand field entries are required.

The symbol in the name field is given the same value, length, and relocatability attributes as the expression in the operand field. The value attribute of the symbol is the value of the expression. The length attribute of the symbol is that of the leftmost or only term of the expression. When an * or a self-defining term is used as an operand, the length attribute is one.

The following example illustrates how this instruction can be used to equate a symbol with the contents of the operand:

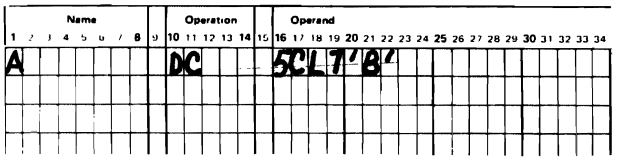


MAX has the value of TEST+X'3FC' (X'102'+X'3FC' or X'4FE') any time it is used in the program. The symbol STEST has the value of the first (leftmost) byte of the data area reserved by the DC instruction. Since the symbol on the DC (TEST) has the value of the rightmost byte, this type of EQU is useful for addressing the leftmost byte. The symbol REG2 in any statement is the same as using the number 2.

EQU is used to equate symbols to register numbers, immediate data, and other arbitrary values. To reduce programming time and improve documentation, the programmer can equate symbols to frequently used expressions and then use the symbols as operands in place of the expressions.

DATA DEFINITION

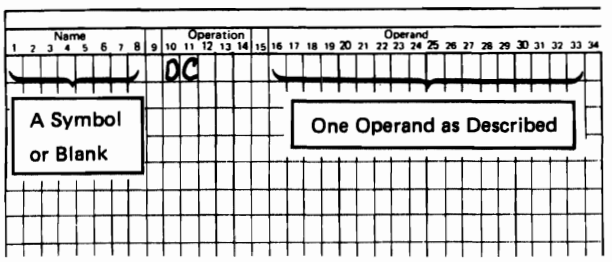
There are two data definition assembler instruction statements: define constant (DC) and define storage (DS). These assembler language statements are used to enter data constants into storage, and to define and reserve areas of storage. Name entries may be used so that other program statements can refer to the generated fields using the same symbols. The length attribute of the symbol is the length of the storage or constant area. In the following example, 35 bytes of storage are allocated and A is pointing to the rightmost byte of the DC.



DC–Define Constant

The DC assembler language instruction is used to reserve areas of storage, assign names to those areas, and then initialize those areas with desired values. This desired value may be one of seven types of constants: storage address, binary, character, decimal, hexadecimal, integer, and floating point.

The format is as follows:



Name

The name entry is optional. 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 rightmost byte of the constant.

Operand

The operand consists of four subfields. The first three describe the constant and the fourth provides the constant. No blanks are permitted within any of the subfields (unless provided as characters in a character constant) or between the subfields. Subfield 1 is optional. Subfields 2, 3, and 4 must be present in the operand field.

The subfields are written in the following sequence:

Sequence	Subfield
1	Duplication factor
2	Type
3	Length
4	Constant

Operand Subfield 1: Duplication Factor: The duplication factor may be omitted. If specified, the constant is generated the number of times indicated by the factor. The factor must be specified by an unsigned, decimal value, 1 through 65535. The duplication factor is applied after the constant is fully assembled; that is, after it has been developed into its proper format.

Operand Subfield 2: Type: The type subfield 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 follows:

I = Integer
X = Hexadecimal
D = Decimal
A = Address
B = Binary
F = Floating Point
C = Character

Operand Subfield 3: Length: The third subfield describes the number of bytes required by the constant.

The entry for this subfield may be written two ways:

1. Ln, where n is an unsigned, decimal value. The value of n is as follows:

   n = 1-256 for I, B, C, X constants

   n = 1-31 for D constants

   n = 1-3 for A constants

   n = 4 or 8 for F constants

2. L (absolute expression), where an absolute expression is enclosed in parentheses. The value limits for the absolute expression are the same as those for n in the previous paragraph. A location counter reference is not allowed in this expression. Refer to *Assembler Program Conventions* in Chapter 2 for information about expressions.

Operand Subfield 4: Constant: This subfield supplies the constant described by the subfields that precede it. A data constant (all types except A) is enclosed in apostrophes. An address constant (type A) is enclosed in parentheses.

The constant types, their identification letters, and an example of each follow. Unless otherwise specified, the maximum length is 256.

Constant Type	ID Letter	Example	Explanation
Integer	I	IL2'15'	Negative numbers are inserted into storage in twos complement notation. If the constant is not the specified length, the constant is padded or truncated on the left–positive constants are padded with zeros, negative constants with ones. The length of the constant is limited to 4 bytes, and the value must be within the range of $-(2^{32})+1$ to $2^{32}-1$. You cannot use the high-order bit as a sign bit if the value is outside the range $-(2^{31})+1$ to $2^{31}-1$.
Decimal	D	DL5'125.66'	This constant is stored in zoned decimal format. The decimal point is used only for documentation; it is ignored by the assembler. If the constant is not the specified length, padding with decimal zeros or truncation occurs on the left. Each decimal digit occupies one byte of storage. The maximum length is 31.
Binary	B	BL1'10110'	If the constant is not the specified length, padding with binary zeros or truncation occurs on the left. Each digit occupies one bit of storage; eight digits occupy one byte of storage.
Character	C	CL14'CHARACTER DATA'	If the constant is not the specified length, padding with blanks or truncation occurs on the right. Each character, including blanks, occupies one byte of storage.
Hexadecimal	X	XL3'ABC55'	If the constant is not the specified length, padding with zeros or truncation occurs on the left. Each two digits occupy one byte of storage.
Floating Point	F	(single) FL4'52.56E-3' (double) FL8'9237.7734E-69'	The only valid lengths for floating point constants are 4 and 8. If the constant is not the specified length, padding with binary zeros or truncation occurs on the right. Floating point numbers have two components: a mantissa and an exponent. The mantissa is a signed or unsigned decimal number. Its decimal point can appear at the beginning, at the end, or within the decimal number. The exponent consists of the letter E, followed by a signed or unsigned decimal integer. Note that there are no assembler floating point instructions. Floating point is supported for scientific macroinstructions.
Address	A	AL2(BETA)	BETA could be an external reference. If the constant is not the specified length, padding with zeros or truncation occurs on the left. The maximum length is 3.

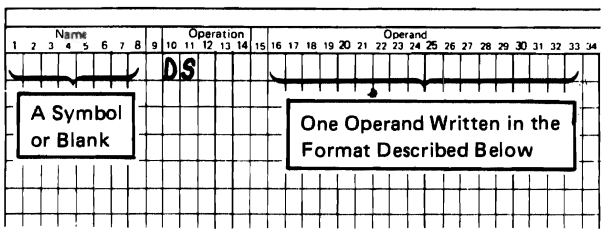
Examples of the DC instructions for each of the constant types are given in Figure 3-2. The object code generated for these constants is also shown.

```
ERR LOC  OBJECT CODE        ADDR STMT     SOURCE STATEMENT

    0000 3E26              0001    3 INT1      DC    IL2'15910'              INTEGER-NORMAL
    0002 26                0002    4 INT2      DC    IL1'15910'              INTEGER-TRUNCATED
    0003 00000F            0005    5 INT3      DC    IL3'+15'                INTEGER-SIGN SPECIFIED & PADDED
    0006 FFFFF1            0008    6 INT4      DC    IL3'-15'                INTEGER-NEGATIVE & PADDED
    0009 F1F2F5            000B    7 DEC1      DC    DL3'1.25'               DECIMAL-NORMAL WITH DECIMAL POINT
    000C F5                000C    8 DEC2      DC    DL1'125'                DECIMAL-TRUNCATED
    000D F0F0F1F2F5        0011    9 DEC3      DC    DL5'125'                DECIMAL-PADDED
    0012 F0F0F1F2D5        0016   10 DEC4      DC    DL5'-125'               DECIMAL-NEGATIVE & PADDED
    0017 89                0017   11 BIN1      DC    BL1'10001001'           BINARY-NORMAL
    0018 000089            001A   12 BIN2      DC    BL3'10001001'           BINARY-PADDED
    001B 1C                001B   13 BIN3      DC    BL1'111100011100'       BINARY-TRUNCATED
    001C C4C1E3C140404040  0023   14 CHR1      DC    CL8'DATA'               CHARACTER-PADDED
    0024 C4C1              0025   15 CHR2      DC    CL2'DATA'               CHARACTER-TRUNCATED
    0026 3F                0026   16 HEX1      DC    XL1'3F'                 HEXADECIMAL-NORMAL
    0027 000F12            0029   17 HEX2      DC    XL3'F12'                HEXADECIMAL-PADDED
    002A 23                002A   18 HEX3      DC    XL1'F123'               HEXADECIMAL-TRUNCATED
    002B 43100000          002E   19 FLT1      DC    FL4'256'                FLOATING POINT-SINGLE PRECISION,NORMAL
    002F 4B2540BE          0032   20 FLT2      DC    FL4'256E+10'            FLOATING POINT-SINGLE PRECISION,TRUNCATED
    0033 3B448B2F          0036   21 FLT3      DC    FL4'25.6E-8'            FLOATING POINT-SINGLE PRECISION,NEG EXPONENT
    0037 C3100000000000000 003E   22 FLT4      DC    FL8'-256'               FLOATING POINT-DOUBLE PRECISION,NEGATIVE
    003F 4310000000000000  0046   23 FLT5      DC    FL8'256.0'              FLOATING POINT-DOUBLE PRECISION,DECIMAL POINT
    0047 4B2540BE40000000  004E   24 FLT6      DC    FL8'256E10'             FLOATING POINT-DOUBLE PRECISION,EXPONENT
    004F 04D2              0050   25 ADD1      DC    AL2(1234)               ADDRESS-DECIMAL
    0051 34                0051   26 ADD2      DC    AL1(X'1234')            ADDRESS-HEXADECIMAL,TRUNCATED
    0052 0000F1            0054   27 ADD3      DC    AL3(X'F1')              ADDRESS-HEXADECIMAL,PADDED
    0055 FB2E              0056   28 ADD4      DC    AL2(-1234)              ADDRESS-DECIMAL,NEGATIVE
    0057 F0F0              0058   29 ADD5      DC    AL2(X'FFFF'-X'0F0F')    ADDRESS-RESOLVED
```

Figure 3-2. DC Instructions

DS–Define Storage

The DS assembler language instruction is used to reserve areas of storage and to assign names to those areas. The format of the DS instruction is as follows:



The format of the DS operand is similar to that of the DC operand. Subfields 1 to 3 are employed and are written in the same sequence as for the DC operand. Subfield 1 (duplication factor) is optional; subfields 2 and 3 (type and length) are required. The name field entry is optional.

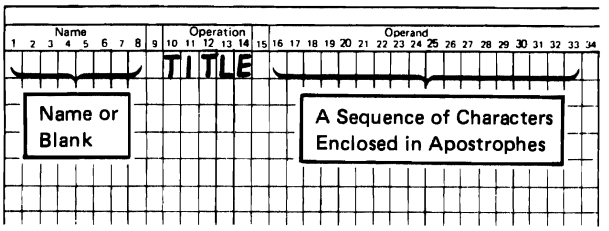
Storage areas of more than 256 bytes may be reserved by use of the duplication factor in a DS instruction. If a duplication factor is included in the operand, the total amount of storage assigned to the constant field is the duplication factor times the length. This product is limited to 65535.

LISTING CONTROL

The listing control instructions help the programmer document the assembler listing so it will be more readable. These instructions are TITLE, EJECT, SPACE, and PRINT.

TITLE–Identify Assembly Output

The TITLE instruction identifies the assembly listing. The format of the TITLE instruction is as follows:



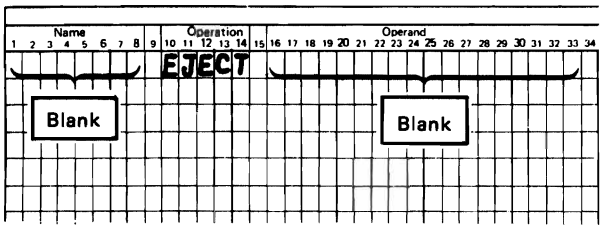
The name field may contain up to eight alphabetic or numeric characters in any combination. The name of the first TITLE instruction is printed on the header line of all listings. The names of all subsequent TITLE instructions are ignored.

The operand field contains a sequence of characters enclosed in apostrophes. Each single apostrophe desired as a character in the constant must be represented by a pair of apostrophes. The contents of the operand field are printed beneath the IBM Assember heading on each page of the assembly listing.

A program may contain more than one TITLE instruction. Each TITLE statement provides the heading for pages in the assembly listing that follow it, until another TITLE statement is encountered. Each TITLE statement advances the listing to a new page before the heading is printed. The TITLE instruction is not printed in the source listing.

EJECT–Start New Page

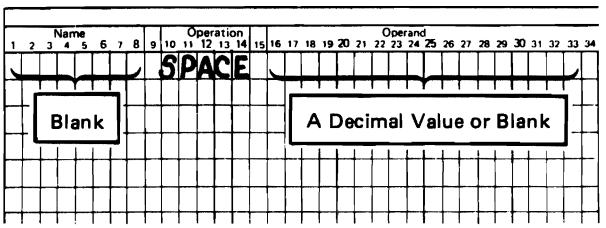
The EJECT operation causes the next line of the listing to appear at the top of a new page. This instruction provides a convenient way to separate routines in the program listing. The format of the EJECT operation statement is as follows:



The EJECT operation statement will not be printed in the listing. The name and operand fields must be blank.

SPACE–Space Listing

The SPACE operation is used to insert one or more blank lines in the listing. The format of the SPACE control statement is as follows:

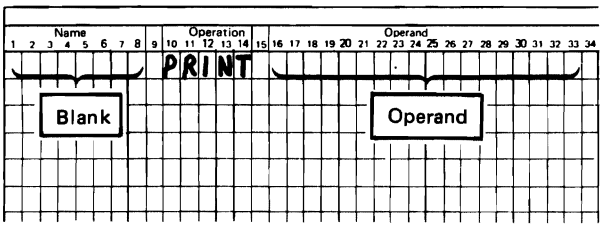


The name field must be blank. An unsigned decimal value is used to specify the number of blank lines that are to be inserted. If the operand contains a blank, a zero, or a one, one blank line will be inserted. If the value of the operand exceeds the number of lines remaining on the current page, the instruction has the same effect on the listing as an EJECT operation. The SPACE operation, like the EJECT and TITLE operations, is not listed on the assembler listing, but does increase the statement counter by one.

Note: The assembler checks the first 87 bytes of the source statement unless you use ICTL to change the source record format. If you have no operand on the SPACE instruction, sequence numbers or comments appearing before byte 87 will cause assembly errors.

PRINT–Control Program Listing

The programmer can control the printing of an assembly listing by using the PRINT operation. A program can have any number of PRINT instructions. Each PRINT instruction controls the listing until the next PRINT instruction is encountered.



The operand field can include one entry from each of the following groups (one, two, or three operands):

1. ON: A listing is printed.
   OFF: No listing is printed.

2. DATA: Constants are printed out in full on the assembler listing.
   NODATA: Only the leftmost 8 bytes of the constants are printed on the assembler listing.

3. GEN: Print operations generated by the macro processor if not overridden by other print control statements (PRINT OFF).
   NOGEN: Suppress printing of statements generated by the macro processor.

Operand entries must be separated by a comma.

The ON, GEN, and DATA conditions are assumed by the assembler unless otherwise specified by a PRINT instruction. If an operand is omitted, it is assumed to be unchanged and continues according to the last specification.

PROGRAM CONTROL STATEMENTS

ISEQ–Input Sequence Checking

The ISEQ instruction is used to check the sequence of source records. Sequence checking begins with the first record after the ISEQ instruction. The sequence entry is read from the position identified by the ISEQ operand. The sequence entry on the next record is then compared to previous sequence value. The ISEQ statement has the following effect:

1. The sequence entries on source statement records are checked for ascending order.

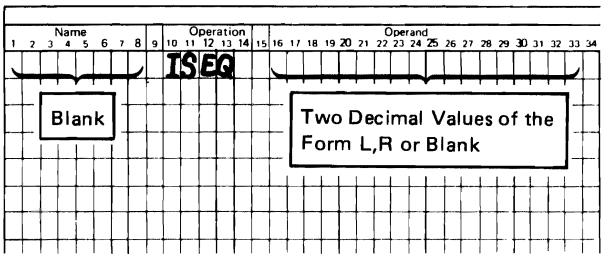
2. Statements that are out of order and statements without sequence entries are flagged in the assembler listing.

3. The total number of flagged statements is noted at the end of the assembler listing.

For example, with sequence values of 13, 27, 31, 6, 8, 45, 47, ƀ, and 48, the record numbered 6 and the record without a sequence value would be out of sequence. These two records are flagged in the error field of the listing, and a statement at the end of the listing shows that two records were out of sequence.

The assembler does not check the sequence unless requested to do so by the ISEQ statement.

The following is the ISEQ instruction format:



The name field entry must be blank. The operands L and R, respectively, specify the leftmost and rightmost columns of the field on the source record to be checked. L,R must be within the range of columns 73 to 96 inclusive. The length of the field (R-L+1) must be 1 to 8. An ISEQ statement with a blank operand terminates the checking. Checking may be resumed with another ISEQ STATEMENT. Columns to be checked must not be before the end column.

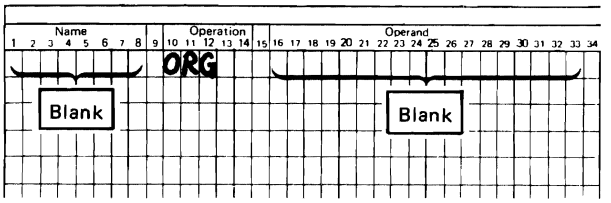
Note: Statements generated by the macro processor are not tested for sequence.

ORG–Set Location Counter

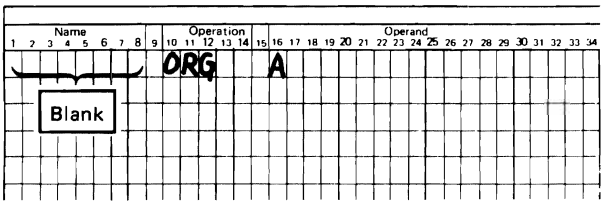
The ORG instruction alters the setting of the location counter. By altering the setting of the location counter, you can specify storage boundaries. For example, you can use the ORG instruction to set the location counter so that an input buffer is aligned on an 8-byte boundary.

The format of the ORG instruction is as follows:

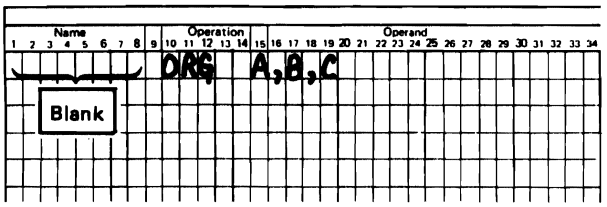
Blank operand:



Expression A as operand:



Expression A as operand optionally followed by two absolute expressions B and C:



The location counter is set to the smallest value that is greater than or equal to A, and is also C more than a multiple of B. The expression A may be either absolute or relocatable; B and C must be decimal values. The default values for B and C are 1 and 0 respectively. For example:

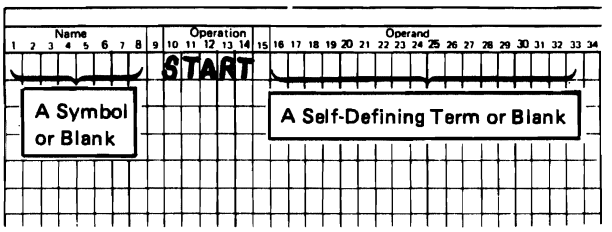
Current Location Counter	A	B	C	New Location Counter
275	*	100	50	350
340	*	100	50	350
350	*	100	50	350
504	*	256	0	512
750	1000	–	–	1000

Any symbols in the expression must have been previously defined. An ORG operation may reduce the location counter for the purpose of redefining the current program, but must not be used to specify a location below the starting location counter value. If the previous ORG statement has reduced the location counter to redefine the current program, an ORG statement with an omitted operand can then be used to restore the location counter to the previous maximum assigned address plus one.

Location Counter	Address	Name	Operation	Operand
0064	0069	SYMBOL	DC	1CL6''
006A	0325	FILLIN	DS	7CL100
00CE	–	–	ORG	FILLIN-599
00CE	01F9	DATA	DC	150CL2'AZ'
0326	–	–	ORG	–
			.	
			.	
			END	

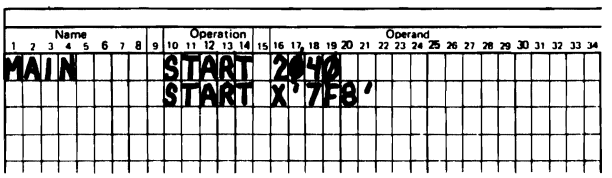
START–Start Assembly

The START instruction specifies an initial location counter value for the program. The format of the START instruction statement is as follows:



The assembler uses the self-defining term specified by the operand as the initial location counter value of the program. If a symbol names the START instruction, the symbol is established as the name of the object program (the name symbol on the START instruction is limited to a maximum of 6 characters). If a symbol name is not specified, the object program is assigned the default name ASMOBJ and a diagnostic message is issued.

For example, either of the following statements could be used to indicate an initial assembly location of 2040. In addition, the first statement establishes MAIN as the object module name.



If the operand is omitted, the assembler sets the initial location counter value of the program at zero.

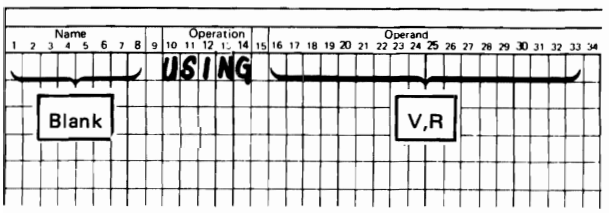
Note: The START instruction may not be preceded by any type of assembler language statement that may either affect or depend upon the setting of the location counter. If no START instruction appears in the program, the initial location counter value will begin at zero.

USING–Use Index Register for Base Displacement Addressing

The USING operation indicates that an index register is to be used for base displacement addressing. This instruction also specifies the relocatable value that the assembler uses to compute base displacements for base displacement addressing.

Notes:

1. A USING instruction does not load the register specified. It is the programmer's responsibility to see that the specified base address value is placed in the register.
2. The USING statement is not required if you code only absolute displacements.

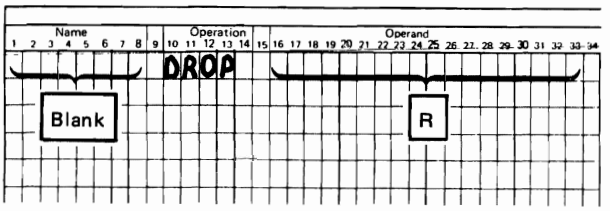


R must be an absolute expression with the value 1 or 2. V is a relocatable expression whose value must be in the range 0 to 65535. The operand R specifies the index register that can be assumed to contain the base address represented by the operand V. If the programmer changes the value in an index register currently used as a base register and wishes the assembler to compute displacement from this new value, the assembler must be told the new value by means of another USING statement. Two USING instructions may be used to have the two index registers as base registers to two different portions of main storage.

An example of how to use the USING instruction in base displacement addressing is given in Chapter 2 under *Addressing*.

DROP–Drop Index Register as Base Register

The DROP operation specifies a previously available index register that may no longer be used as a base register.

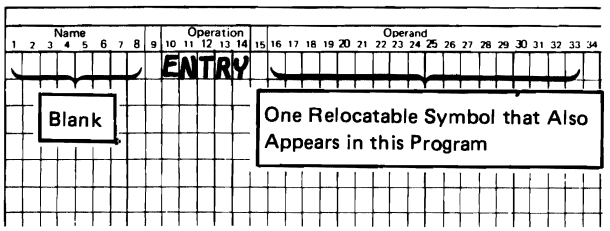


R must be an absolute expression with the value 1 or 2. The expression value indicates which index register, previously referenced in a USING statement, is now unavailable for base register use.

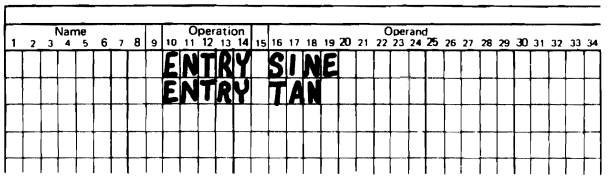
It is not necessary to use a DROP operation when the base address being used is changed by a USING statement, nor are DROP statements needed at the end of the source program.

ENTRY–Identify Entry-Point Symbol

The ENTRY operation identifies linkage symbols that are defined in this program and can be referenced from other programs.

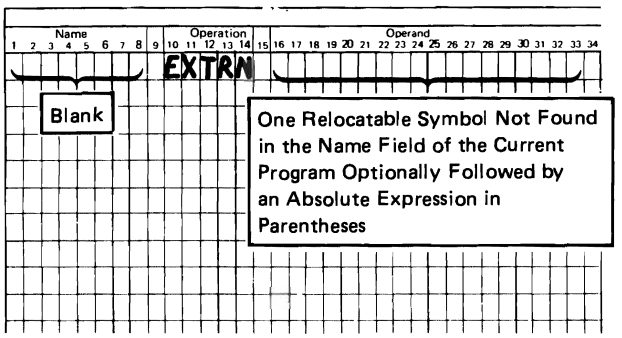


The symbol in an ENTRY operand field can be referenced by another program provided that program uses the same symbol in the operand of an EXTRN statement. The symbol used in the operand field, for both EXTRN and ENTRY instructions, has a maximum limit of 6 characters. See *EXTRN Statement* in this chapter. The following example identifies the statements named SINE and TAN as entry points to the program.



EXTRN–Identify External Symbols

This operation identifies symbols used in the current program that are defined in another program. Each symbol in the operand of an EXTRN control statement must be identified by an ENTRY control statement or be the module name in some other program. The symbol used in the operand field, for both EXTRN and ENTRY instructions, has a maximum of 6 characters.



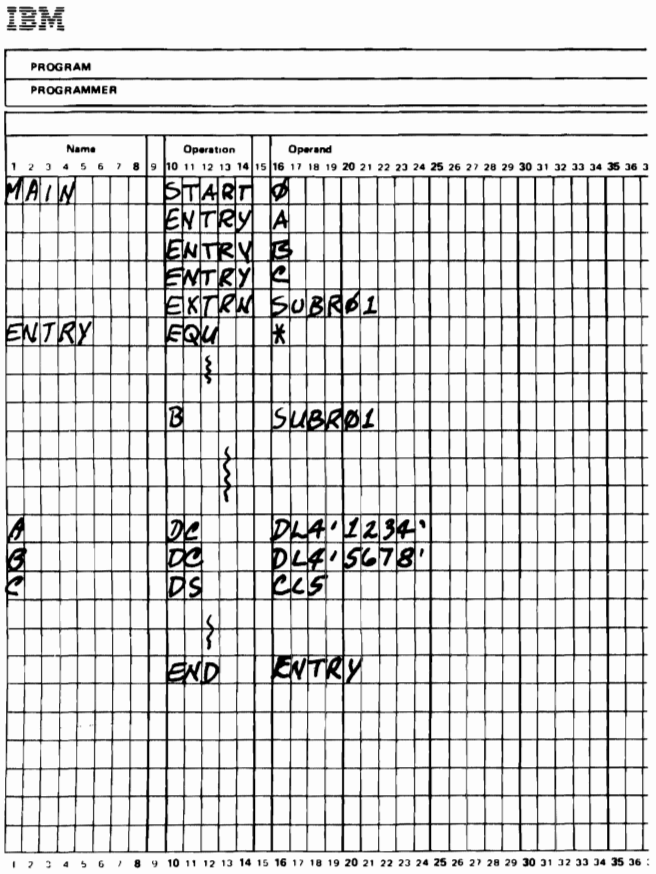
The external symbol cannot be used in a name field in the same program that describes that symbol as an EXTRN. The name field entry of the EXTRN statement must be blank.

An EXTRN subtype can be specified for the EXTRN symbol by following the symbol with an absolute expression enclosed in parentheses. The value of the absolute expression cannot be less than zero nor more than 255. Any symbol in the expression must have been previously defined. For an explanation of the subtype values and their meanings, see the *Overlay Linkage Editor Reference Manual.*

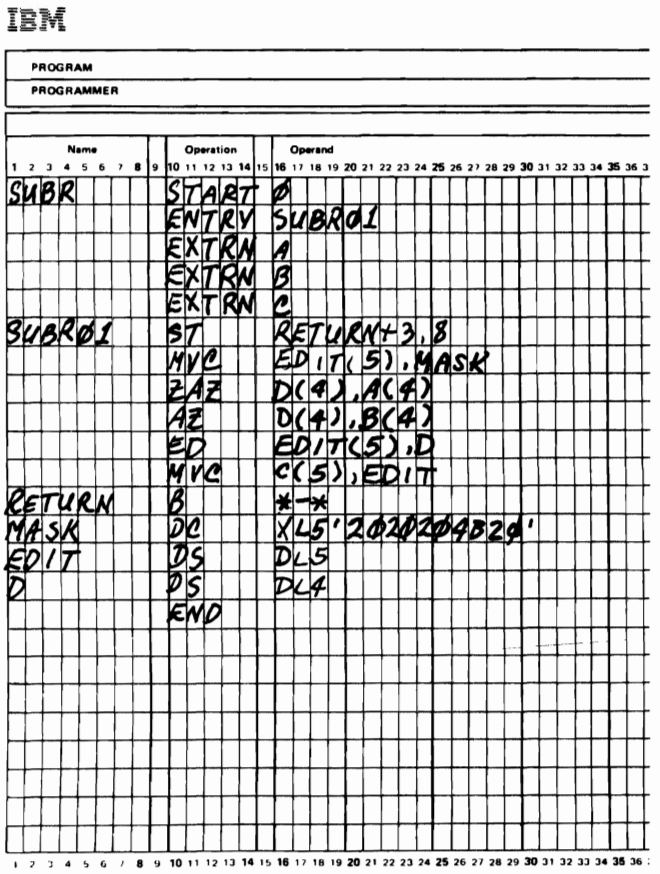
Figure 3-3 shows how ENTRY and EXTRN can be used to make two or more programs act as one program through sharing data and control. The main program defines symbols A, B, and C and identifies them as entry points. These same symbols are identified as EXTRNs (external symbols) in the subroutine. This allows the subroutine to use these symbols just as it would if the symbols had been defined in the subroutine. SUBR01, on the other hand, is defined and identified as an entry point by the subroutine and as an EXTRN, external symbol, by the main routine. These four symbols–A, B, C, and SUBR01– can now be used interchangeably by both the main routine and the subroutine.

The main routine has control first. It executes instructions and then branches to SUBR01, which is defined as an entry point in the subroutine. Instructions in the subroutine are executed. Notice that the subroutine uses symbols A, B, and C, which were defined in the main routine. Control is then passed back to the main routine.

Note: The actual resolution of symbols between programs is performed by the overlay linkage editor and not by the assembler.



Main Routine

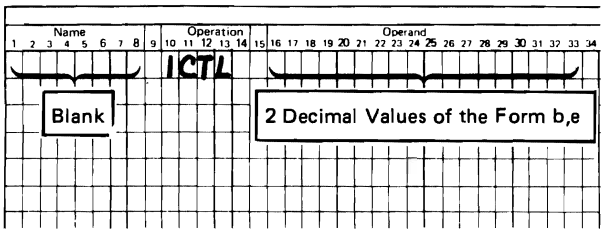


Subroutine

Figure 3-3. Example ENTRY and EXTRN Statements

ICTL–Input Format Control

The ICTL instruction allows the programmer to alter the normal format of his source program statements. The ICTL statement must precede all other source statements in the source program and may be used only once. An invalid or illegal ICTL instruction ends the assembly. The format of the ICTL instruction statement is as follows:



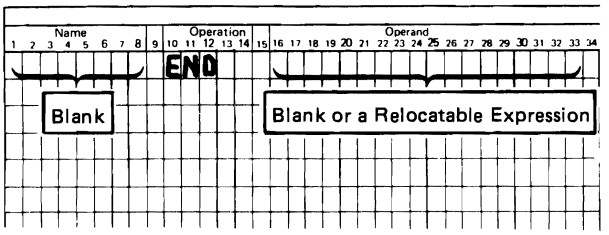
Operand b specifies the begin column for the source statement. It must be from 1 to 48, inclusive. Operand e specifies the end column of the source statement. It must be from 49 to 96, inclusive. The column after the end column must always be blank.

If no ICTL statement is used in the source program, the assembler assumes that the begin column is 1, and the end column is 87.

Note: ICTL must be the first source statement in the source program, including comment statements. The assembler control statements OPTIONS and/or HEADERS, however, must precede the ICTL and all other source statements. The HEADERS and OPTIONS statements are described in Chapter 6.

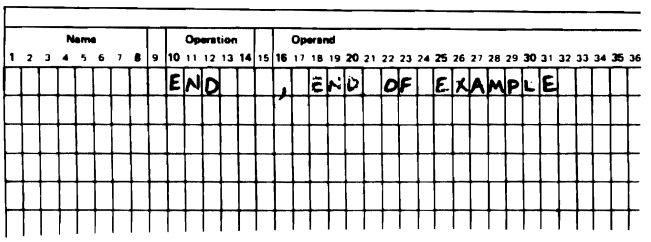
END–End Assembly

The END instruction terminates the assembly of a program. The END instruction must always be the last statement in the source program. The format of the END instruction statement is as follows:



The operand of this instruction can contain: 1) a blank, or 2) an expression (usually a name field entry) which specifies the address to which control is transferred after the program is loaded. This is usually the name given on the START instruction. If the operand is blank, control is transferred to the address identified by the START instruction.

If the operand is blank and you want to put a comment on the instruction, code a comma as the operand. For example:



Note: The assembler checks the first 87 bytes of the source statement unless you use ICTL to change the source record format. If you have no operand on the END instruction, sequence numbers or comments appearing before byte 87 will cause assembly errors.

Chapter 4. Machine Instruction Statements

Machine instruction statements represent machine instructions on a one-for-one basis. The assembler translates these symbolic representations into machine language usable by the computer. Machine instruction statements differ from assembler instruction statements in that the machine instruction statements are executable parts of the program logic (such as MVI, ST, LA), while assembler instruction statements are simply orders to the assembler, each statement directing a specific operation (such as DC, START, SPACE).

The format of a machine instruction statement is closely related to the format of a machine language instruction that results from the assembly process.

A mnemonic operation code is used in place of the actual machine language operation code, and one or more operands provide the information required by the machine instruction. A comment and a sequence entry may be included in the machine instruction statements, but they do not become part of the object code.

Name Entry

Any machine instruction statement may be named by a symbol, which other assembler statements can use as an operand. The value attribute of the symbol is the address of the leftmost byte assigned to the assembled instruction. The length attribute is the length of the instruction.

Mnemonic Operation Entry

The mnemonic operation codes are designed to be easily-remembered codes that remind the programmer of the functions performed by the instructions. The IBM System/34 Basic Assembler and Macro Processor Program Product provides mnemonic and extended mnemonic operation codes. The complete set of mnemonic codes is listed in Figures 4-1 and 4-2. For the operand formats see Figure 4-3. For the formats of the instructions when they are in main storage, see Figure 4-4.

Extended mnemonic codes are provided for the convenience of the programmer. They are unlike other mnemonic codes in that part of the information usually provided in the operand is in the extended mnemonic code itself. Extended mnemonic codes allow the following:

1. Conditional branches (BC) and jumps (JC) can be specified mnemonically, requiring only a branch address as an operand.

2. Half-byte moves (MVX) can be specified mnemonically, requiring only addresses as operands.

Extended mnemonic codes are not part of the set of machine instructions, but are translated by the assembler into the corresponding operation and operand combinations.

Instruction	Mnemonic Operation Code	Operand Formats[1]
Zero and add zoned decimal	ZAZ	Type 1
Add zoned decimal	AZ	Type 1
Subtract zoned decimal	SZ	Type 1
Move hex character	MVX	Type 2
Move characters	MVC	Type 2
Compare logical characters	CLC	Type 2
Add logical characters	ALC	Type 2
Subtract logic characters	SLC	Type 2
Insert and test characters	ITC	Type 2
Edit	ED	Type 2
Move immediate	MVI	Type 3
Compare logical immediate	CLI	Type 3
Set bits on masked	SBN	Type 3
Set bits off masked	SBF	Type 3
Test bits on masked	TBN	Type 3
Test bits off masked	TBF	Type 3
Store register	ST	Type 3
Load register	L	Type 3
Add to register	A	Type 3
Branch on condition	BC	Type 3
Load address	LA	Type 3
Load program mode register	LPMR[2]	Type 4
Supervisor call	SVC	Type 4
Transfer control	XFER	Type 4
Jump on condition	JC	Type 5

[1] Machine-instruction operands are divided into five types. The characteristics of each type are as follows:

Type 1: Two-operand format in which a length is explicit or implied in both operands.

Type 2: Two-operand format in which a length can be explicit in either operand, but not in both. If length is not explicit in either operand, the assembler uses the implied length of operand one.

Type 3: Two-operand format in which a length cannot be specified.

Type 4: Two-operand format in which data operands are immediate data.

Type 5: Two-operand format in which operand one is used by the assembler to calculate a positive displacement and operand two is immediate data.

[2] Privileged instruction

Figure 4-1. Mnemonic Operation Codes

Instruction	Mnemonic Operation Code	Hexadecimal Q-Code[1]
Move hexadecimal character (MVX)		
Move to zone from zone	MZZ	X'00'
Move to numeric from zone	MNZ	X'02'
Move to zone from numeric	MZN	X'01'
Move to numeric from numeric	MNN	X'03'
Branch on condition (BC)		
Branch	B	X'87'
Branch high	BH	X'84'
Branch low	BL	X'82'
Branch equal	BE	X'81'
Branch not high	BNH	X'04'
Branch not low	BNL	X'02'
Branch not equal	BNE	X'01'
Branch overflow zoned	BOZ	X'88'
Branch overflow logical	BOL	X'A0'
Branch no overflow zoned	BNOZ	X'08'
Branch no overflow logical	BNOL	X'20'
Branch true	BT	X'10'
Branch false	BF	X'90'
Branch plus	BP	X'84'
Branch minus	BM	X'82'
Branch zero	BZ	X'81'
Branch not plus	BNP	X'04'
Branch not minus	BNM	X'02'
Branch not zero	BNZ	X'01'
Jump on condition (JC)		
Jump	J	X'87'
Jump high	JH	X'84'
Jump low	JL	X'82'
Jump equal	JE	X'81'
Jump not high	JNH	X'04'
Jump not low	JNL	X'02'
Jump not equal	JNE	X'01'
Jump overflow zoned	JOZ	X'88'
Jump overflow logical	JOL	X'A0'
Jump no overflow zoned	JNOZ	X'08'
Jump no overflow logical	JNOL	X'20'
Jump true	JT	X'10'
Jump false	JF	X'90'
Jump plus	JP	X'84'
Jump minus	JM	X'82'
Jump zero	JZ	X'81'
Jump not plus	JNP	X'04'
Jump not minus	JNM	X'02'
Jump not zero	JNZ	X'01'

[1]The hexadecimal Q-codes for the extended mnemonic operation codes are the contents of the Q-byte in related System/34 machine instructions. For a description of the System/34 machine instructions, see the *Functions Reference Manual*.

Figure 4-2. Extended Mnemonic Operation Codes

Type	Instructions	Possible Operand Formats			
1	ZAZ,AZ,SZ	A,A	A(L),A	D(,R),A	D(L,R),A
		A,A(L)	A(L),A(L)	D(,R),A(L)	D(L,R),A(L)
		A,D(,R)	A(L),D(,R)	D(,R),D(,R)	D(L,R),D(,R)
		A,D(L,R)	A(L),D(L,R)	D(,R),D(L,R)	D(L,R),D(L,R)
2	MVC,CLC,ALC	A,A	A(L),A	D(,R),A	D(L,R),A
	SLC,ITC,ED	A,A(L)	A(L),D(,R)	D(,R),A(L)	D(L,R),D(,R)
		A,D(,R)	A,D(L,R)	D(,R),D(,R)	D(,R),D(L,R)
	MVX	A,A(I)	A(I),A	D(,R),A(I)	D(I,R),A
		A,D(I,R)	A(I),D(,R)	D(,R),D(I,R)	D(I,R),D(,R)
3	MVI,CLI,SBN	A,I		D(,R),I	
	SBF,TBN,TBF,BC				
	L,ST,A,LA	A,R		D(,R),R	
4	LPMR[1]	I,I			
	SVC, XFER				
5	JC	A,I			

[1]Privileged instruction

Code	Meaning	Acceptable For
A	Address	Relocatable expression, absolute expression, or self-defining value
D	Displacement	Relocatable expression, absolute expression, or self-defining value
L	Length	Absolute expression or self-defining value
R	Register	Absolute expression or self-defining value
I	Immediate data (bit masks, condition bit masks, or control bits to be used in the instruction)	Absolute expression or self-defining value

Figure 4-3. Operand Formats

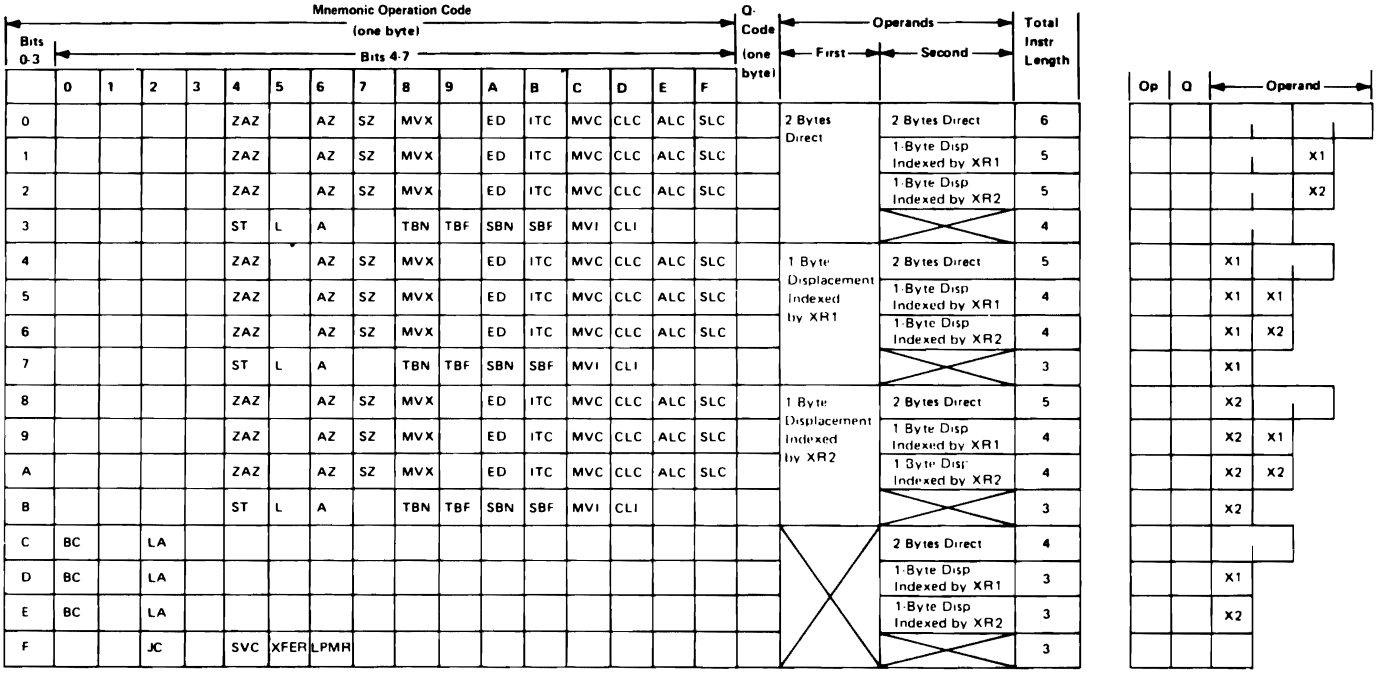


Figure 4-4. Main Storage Instruction Formats

Operand Entry

Some operands are written as single fields, and other operands are written as single fields followed by one or two subfields. For example, addresses may consist of the contents of an index register and a displacement. An operand that specifies an index and displacement is written as a displacement field followed by an index register subfield, as follows:

40(,2)

A comma must separate operands. Parentheses must enclose a subfield or subfields, and a comma must separate two subfields within parentheses. The following rules apply for subfields:

- If both subfields are omitted, the separating comma and the parentheses must also be omitted.
- 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.
- 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.

Fields and subfields in an operand may be represented either by absolute or by relocatable expressions, depending on what the field requires.

Blanks may not appear in an operand unless provided by a character self-defining term.

For base displacement addressing, the base must be specified in the second subfield.

When a length specification is not included in an operand requiring a length, the assembler uses the implied length. The implied length is the length attribute of the first term of the affected operand, as follows:

Term or Expression	Length Attribute
Name on a machine instruction	Length, in bytes, of the instruction.
Location counter reference (*)	Length, in bytes, of the instruction in which it appears, except for EQU where the length attribute is one.
Expression	Length attribute of the leftmost term in the expression.
Self-defining term	Length attribute is one.
START name entry	Length attribute is one.

The operand formats for the machine instructions are given in Figure 4-3.

Notes:

1. For the extended mnemonics of the MVX instruction, the Q code is inherent in the mnemonic and the I field is not used.
2. For the extended mnemonics of the BC and JC instructions, the Q code is inherent in the mnemonic and the second operand is not used.
3. When a relocatable symbol is used as the first operand on a JC instruction, the assembler computes the displacement from the current value of the location counter.
4. When a relocatable symbol is used in the displacement field in a base-displacement specification D(L,R) or D(,R), then the assembler computes the displacement from the base register value specified in a current USING instruction.
5. When the length is not specified in the operands of a Type 1 or Type 2 (excluding MVX) instruction, the assembler uses the implied length of the operands. The implied length of both operands is used in Type 1. The implied length of operand 1 is used in Type 2.
6. Following an EDIT instruction, a conditional branch that tests the second operand for positive, negative, or zero might not work correctly unless the equal or zero condition is forced on before the edit.

Chapter 5. Macroinstruction Definitions

Macroinstruction statements represent predetermined sequences of machine and/or assembler instruction statements. Before a macroinstruction statement can be coded, a macroinstruction definition must have been defined and must reside in the assembler library (#ASMLIB). This macroinstruction definition is either coded by the user or supplied by IBM. For the IBM-supplied macroinstructions, see Chapter 6 in this manual.

The macroinstruction definition is composed of definition control statements. These control statements specify values for the symbolic parameters appearing in the prototype statement of the associated macroinstruction definition. Each of the definition control statements, details about symbols, terms, expressions and mnemonics, and how to define a macroinstruction are discussed in this chapter.

MACROINSTRUCTION CODING CONVENTIONS

Sequence Symbol

Sequence symbols provide labels that can be branched to and therefore determine the sequence in which macro definition statements are processed.

A sequence symbol is written as a period followed by an alphabetic character, $, #, or @, followed by as many as five alphabetic or numeric characters.

Self-Defining Terms

There are four types of self-defining terms: decimal, hexadecimal, binary, and character. These terms represent machine language values, bit configurations, or immediate data in arithmetic expressions. Self-defining terms are always right-justified. Padding with zeros or truncating, if necessary, occurs on the left. Self-defining terms are always positive and may not exceed 65,535.

Decimal Self-Defining Term: An unsigned integer written as a string of decimal digits. High-order zeros may be used. The decimal term is converted to its binary equivalent.

Hexadecimal Self-Defining Term: One to four hexadecimal digits enclosed in apostrophes and preceded by the letter X. Each digit is converted to its binary equivalent.

Binary Self-Defining Term: An unsigned sequence of ones and zeros enclosed in apostrophes and preceded by the letter B. The rightmost 16 digits specified are used to generate the 2-byte binary value.

Character Self-Defining Term: One or two characters enclosed in apostrophes and preceded by the letter C. Any of the 256 hexadecimal combinations may be used, including all letters, digits, and special characters. To represent an apostrophe in a term, two apostrophes must be entered. Each character is converted to its binary equivalent, except as noted for apostrophes.

Character String

A character string can include special characters and blanks and is enclosed by single apostrophes. When a character string is decoded, enclosing apostrophes are removed. Half of the number of apostrophes appearing within the string are removed; so for every apostrophe that is to appear in a decoded character string, two apostrophes must be coded in succession. A decoded character string may be from 1 to 50 bytes long.

Note: Special characters refers to the special characters available in the System/34 character set.

Character Expression

A character expression is a term, null term, or combination of terms that is enclosed in single apostrophes and that may be reduced to a character string from zero to 50 bytes in length. Terms are either literal strings of any of the 256 hexadecimal combinations possible for each byte, except an ampersand (&), or are variable symbols. A null term is indicated by two consecutive single apostrophes. If an apostrophe is required as a data character, it must be entered as two consecutive apostrophes inside the delimiting apostrophes. In multiterm expressions, all the rules of concatenation apply. (See *Concatenation* in this chapter.)

Substring

A substring is a method of selecting specific characters from a character string defined in a character expression. A substring is specified as (m,n) where m and n are each a valid arithmetic expression. The start character of the substring is m; the length of the substring is n. The following rules apply to specifying substrings:

1. The value of m may not be less than or equal to zero.

2. The value of n may not be less than zero.

3. If the value of n is zero or if the value of m is greater than the length of the character string, the substring has no value.

4. If the value of n is greater than the remaining length of the character string, the substring is all the remaining characters of the character string.

In a substring notation, there must be no blanks between the closing single apostrophe of the character string and the left parenthesis of the substring.

Example of substring:

The original character string &CHAR is ABCDEFGHIJKL.

The desired substring is *DEFGH* (five characters beginning with position 4).

The substring is coded as '&CHAR' (4,5).

Alphameric Value

An alphameric value is a continuous string of alphameric characters (not enclosed by apostrophes). When an alphameric value is processed (decoded), commas, blanks, dashes, and equal signs become delimiters. A decoded alphameric value may be from 1 to 50 bytes long.

Variable Symbol

A variable symbol is written as an ampersand (&) followed by an alphabetic character, $, #, or @, and followed by as many as five characters. The characters can be any combination of alphabetic, numeric, or $, #, @ (other special characters and blanks cannot be used). There are two types of variable symbols: symbolic parameters and set symbols. The relationship between these types is:

Symbolic parameters

1. Positional parameters

2. Keyword parameters

Set symbols

1. Global
   a. Arithmetic
   b. Binary
   c. Character

2. Local
   a. Arithmetic
   b. Binary
   c. Character

Symbolic Parameter

Positional or keyword symbolic parameters are assigned values by the macroinstruction statements, prototype statements, and table records. The values assigned to symbolic parameters cannot be changed by the macro processor.

Positional Parameters: Positional parameters appear prior to keyword parameters in the prototype record. Each positional parameter is written as an & followed by an alphabetic character, $, #, or @, followed by as many as five alphabetic or numeric characters. Positional parameters appear on user macroinstructions as parameter values positioned prior to keywords and in the same sequence that they had in the prototypes.

Keyword Parameters: Keyword parameters appear after positional parameters in the prototype record. Each keyword parameter is written as an & followed by an alphabetic character, $, #, or @, followed by five alphabetic or numeric characters.

Keyword parameters appear on user macroinstruction statements with the label of the prototype definition statement excluding the lead &, followed by a dash, followed by the parameter value.

The difference between keyword parameters and positional parameters is that the keyword in a keyword parameter must always be followed by a dash (-). An example of a macroinstruction that contains only keyword parameters is:

```
$EXP1 &PLIST-2,&NOTE-
```

An example of a macroinstruction that contains only positional parameters is:

```
$EXP2 &A,&B
```

An example of a macroinstruction that contains both positional and keyword parameters is:

```
$EXP3 &C,&D,&PLIST-3
```

Set Symbol

A set symbol is a storage area defined by global or local records. The values assigned to these symbols may be changed by the macro processor by use of set records.

Three different kinds of set symbols can be used:

1. Arithmetic set symbols are defined by GBLA (arithmetic global) and LCLA (arithmetic local) records and are assigned values by SETA (set arithmetic) records.

2. Binary set symbols are defined by GBLB (binary global) and LCLB (binary local) records and are assigned values by SETB (set binary) records.

3. Character set symbols are defined by GBLC (character global) and LCLC (character local) records and are assigned values by SETC (set character) records.

Global: A global is a set symbol defined with a global statement. This symbol will have a storage area assigned to it only once for each program assembled. The same set symbol may be defined in other macroinstruction definitions called out in the program, but the storage area will remain as that of the original. The use of globals is a primary means of passing information to macroinstruction definitions called later in the program.

Note: Be careful when using globals, because they retain their values and spaces in the symbol table even when not being used. The use of globals when not needed may cause needless symbol table overflow.

Local: A local is a set symbol (storage area) that retains its value only during the expansion of a single macroinstruction definition. Each time a symbol appears on a local record, it is treated as though it is the first definition of the symbol in the program. These symbols are used to retain values which may be used later in the same macroinstruction definition.

&SYSNDX

&SYSNDX must not be used as a variable set symbol. &SYSNDX is a system variable that may be concatenated with other characters to create unique names for macroinstruction definition statements and generated assembler source instructions. The three-digit number 001 is the value assigned to &SYSNDX when the first macroinstruction statement is processed. The value is increased by one for each subsequent macroinstruction processed in the program.

&SYSNDX can have a maximum value of 999. Therefore, the number of macroinstructions in one job must not exceed 999 when &SYSNDX is used. (No diagnostic messages exist for the incorrect use of &SYSNDX.)

Note: &SYSNDX cannot be used as a keyword or positional parameter.

Attribute

Attribute refers to the kind of value assigned a variable symbol in the variable symbol table (VST). Variable symbols can be assigned the following kinds of values:

Numeric value
Character string value
Null value
Binary value

Count Function

The count function determines the length, in bytes, of the value assigned to a symbolic parameter. This length is obtained by: K' label of symbolic parameter.

Example: If &LIST equals ABCDEFG, then K'&LIST equals 7.

The user may refer to the count function only in the operand of a macro processor control statement (for example, AIF or SETA).

Arithmetic Expression

An arithmetic expression is a term or series of terms separated by operators. The valid terms for an arithmetic expression are variable symbols, self-defining terms, or count functions. The valid operators in an arithmetic expression are addition (+), subtraction (-), multiplication (*), and division (/). Parenthesized expressions are supported up to three nested levels.

The following rules apply to arithmetic expressions:

1. Two or more terms must be separated by operators.
2. Two or more operators must be separated by terms.
3. No more than three nested levels of parentheses are allowed.
4. Parentheses must be balanced; that is, for each left parenthesis there must be a right parenthesis.
5. Unless a left parenthesis is the first element in the expression, it must be immediately preceded by an operator or another left parenthesis.
6. A left parenthesis must be immediately followed by a term or another left parenthesis.
7. A right parenthesis must be immediately preceded by a term or another right parenthesis.
8. A right parenthesis must be immediately followed by an operator or another right parenthesis unless it is the end of the expression.

Arithmetic expressions are evaluated using 24-bit signed arithmetic (a 3-byte field ranging from -8,388,608 to 8,388,607). An expression is reduced to a single value as follows:

1. Parenthesized expressions are evaluated from the innermost set of parentheses outward.
2. All multiplication and division is performed before addition and subtraction. All operations are performed from left to right.

Continuation

Continuation is supported for prototype records only. A nonblank character in position 72 and a comma after the last operand indicate that a continuation of the prototype record follows. At least one operand beginning in position 16 must appear on every continuation of a prototype record. Columns 1 through 15 must be blank. Up to five continuation lines are allowed for any prototype record, which allows a maximum of six lines to be entered for each prototype record.

Concatenation

Separate values physically combined so that they appear as one value are said to be concatenated. Concatenation occurs under any of the following conditions:

1. A symbolic parameter or set symbol immediately precedes or follows another symbolic parameter or set symbol with no intermediate delimiter.
2. Characters immediately precede a symbolic parameter or set symbol with no intermediate delimiter.
3. Characters are joined to a preceding symbolic parameter or set symbol by an intermediate period.

AIF records permit concatenation of symbolic parameters or set symbols and character strings only. Model records and assembler source instructions permit concatenation of symbolic parameters or set symbols and alphameric values only.

Defining Macroinstructions

Definition control statements are used to code macroinstruction definitions. The values established in the definition control statements are used by the macro processor to generate assembler and/or machine instruction statements. Figure 5-1 lists the definition control statements in the order that they must appear in a macroinstruction definition. For the complete list of macroinstruction definition mnemonics available for use in defining macroinstructions, see Figure 5-2.

MACRO (required)
Prototype (required)
Global declares
Local declares
Table
Table definitions
TEXT (required)
  macro logic
MEXIT
MEND (required)

Figure 5-1. Sequence of Definition Control Statements in a Macroinstruction Definition

Mnemonic	Record Type
MACRO	Header
None (macro title used)	Prototype
GBLA	Global arithmetic
GBLB	Global binary
GBLC	Global character
LCLA	Local arithmetic
LCLB	Local binary
LCLC	Local character
TABLE	Table
TABDF	Table definition
TEXT	Text
* or .*	Comment
AIF	Conditional branch
AGO	Unconditional branch
SETA	Set arithmetic
SETB	Set binary
SETC	Set character
ANOP	No-op
MNOTE	Message
MEXIT	Trailer (logical end)
MEND	Trailer (physical end)

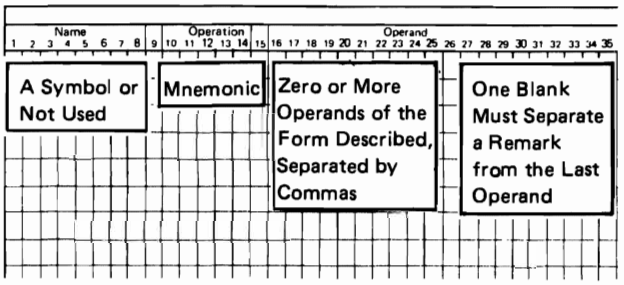
Figure 5-2. Definition Control Mnemonics

Definition Control Statement Format

A definition control statement may contain up to four entries: name, operation, operands, and remarks. The first three entries (name, operation, and operands) are position-dependent and must begin in positions 1, 10, and 16 respectively. The remarks entry may occur in any position following the operands if at least one blank is provided for separation.

Macroinstruction Format

The format of a macroinstruction is:



Name: If the name entry on the macroinstruction contains a symbol and a symbolic parameter appears in the name entry of the associated prototype record, the symbolic parameter is assigned the value of the symbol in the macroinstruction. (See *Prototype* in this chapter.)

If the name entry on the macroinstruction contains a symbol and the name entry of the associated prototype record does not contain a symbolic parameter, the symbol in the name is ignored.

If the name entry on the macroinstruction is not used and a symbolic parameter appears in the name entry of the associated prototype record, the symbolic parameter is assigned a null value. The length of the name entry is 8 bytes with blanks padded on the right.

Operation: The mnemonic operation code must be identical to the mnemonic operation code of the associated prototype record.

Operands: The operand may contain keyword and/or positional parameter operands.

The value assigned a keyword or positional parameter in a macroinstruction is assigned to the corresponding symbolic parameter defined in the associated prototype record.

A symbolic parameter defined without a value in a prototype record is assigned a null value with an undefined attribute, unless an operand referring to the corresponding keyword or positional parameter appears in the associated macroinstruction.

A keyword parameter defined with a value in a prototype record retains the assigned value, unless an operand containing the corresponding keyword appears in the associated macroinstruction.

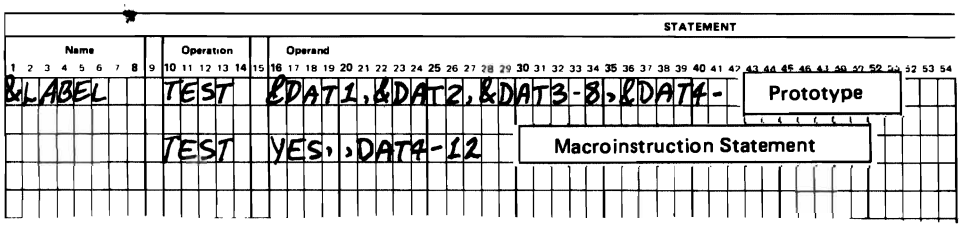
The keyword parameters may be written in any order; however, positional parameters must be in the sequence specified on the prototype statement and must occur before any keyword parameters.

Keyword Parameter Operands: Each keyword operand must consist of a keyword immediately followed by a dash, immediately followed by the value assigned to the keyword.

Each keyword appearing in the operand must correspond to one of the symbolic parameters appearing in the operand of the associated prototype record. (Each symbolic parameter in the associated prototype record does not require a corresponding keyword in the macroinstruction.) A keyword corresponds to a symbolic parameter when the characters in the keyword are identical to the characters following the ampersand in the symbolic parameter.

Positional Parameter Operands: A positional parameter operand corresponds to a keyword value; that is, just the value is given and not the keyword. Commas in succession are used to indicate the omission of positional parameters and null value assignment.

An example of a macroinstruction statement and its relationship to the prototype definition control statement is:

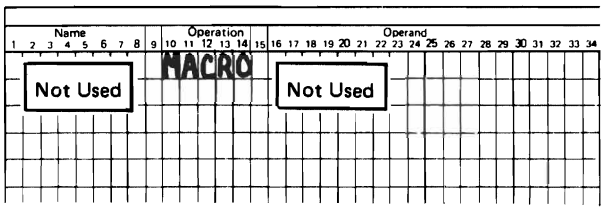


&DAT1 is assigned 'YES' by the macroinstruction.
&DAT2 is assigned null value by omission.
&DAT3 is assigned '8' by prototype default.
&DAT4 is assigned '12' by the macroinstruction.

MACROINSTRUCTION DEFINITION CONTROL STATEMENTS

Header (MACRO)

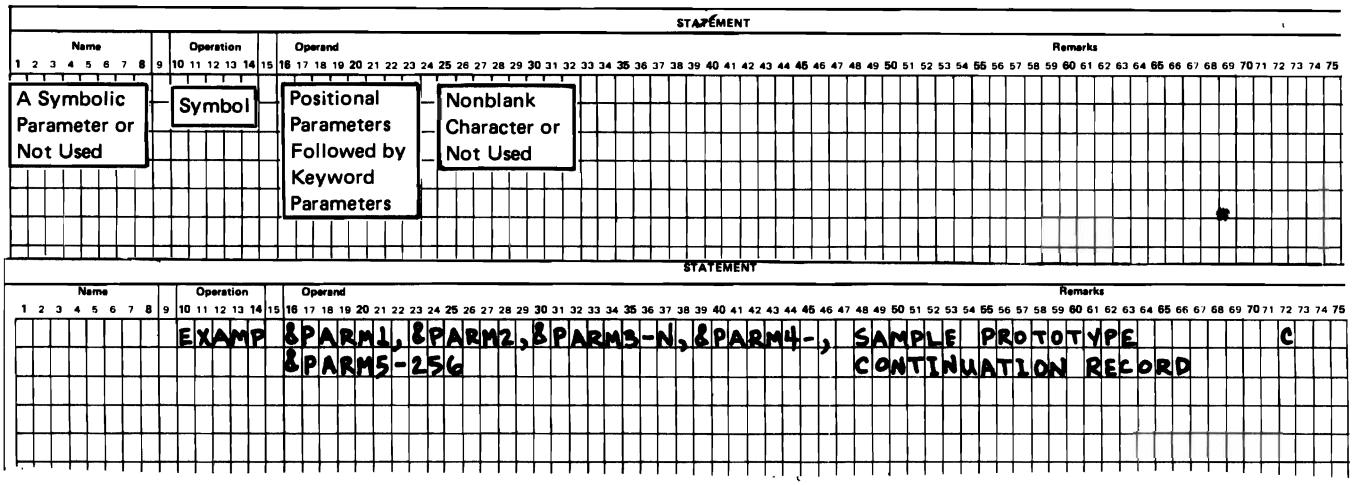
The header statement denotes the beginning of a macro definition and must be the first control record in the definition. A maximum of one comment record (asterisk in position 1) can precede the header record. A comment record preceding a header record is not generated as source output. The format of the header record is:



Prototype

The prototype statement defines the mnemonic operation code that must appear and the parameters that may appear on the corresponding macroinstruction statements. The mnemonic operation code in the definition prototype statement is the same one used to code a macroinstruction statement in the assembler source program. By varying the values assigned to parameters, the user can vary the sequence of assembler source instructions generated for each user macroinstruction.

The prototype record must be the second control record in a macro definition. The format of the prototype record is:



Name: The symbolic name entry of the prototype statement is optional. If the keyword prototype record is continued, the name and operation entries must not be repeated on the continuation records.

Operation: The symbol in the operation entry is the mnemonic operation code that must appear in all user macroinstruction statements that refer to this macroinstruction definition. The operation mnemonic must not be more than five characters long. If the keyword prototype record is continued, the operation entry must not be repeated on the continuation records.

Operand: The mnemonic operand consists of positional and/or keyword symbolic parameters separated by commas. A blank indicates the end of the operands.

Positional parameters are represented by variable symbol names. Positional parameters must precede any keyword parameters in the prototype statement. Each positional parameter is followed by a comma.

Keyword parameters are variable symbol names followed by a dash and immediately following the dash: a parameter value, a comma, or, if keyword parameter is the last parameter in a macroinstruction, a blank. If a parameter value is included, the value is used as the default value. If a parameter value is not included, a null value is used as the default value.

A comment may be entered following the operands as long as at least one blank exists as a delimiter between it and the operands.

If the prototype statement is continued, at least one operand beginning in position 16 must appear on every continuation record. The preceding example shows a continued prototype statement.

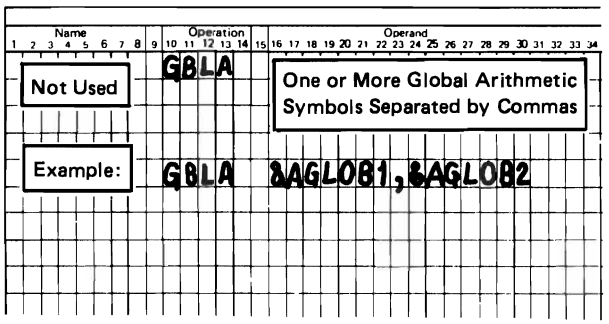
Global

Three types of global statements can be used in macroinstruction definitions to define global set symbols. These types of global statements are arithmetic, binary and character. A global set symbol is a set symbol whose value is available to all macroinstructions in an assembler source program. If used, a global statement must be the first definition control statement following the prototype statement. Global statements can be specified in any order and more than one of each type can be used.

A global set symbol is established when the first specification of a symbol name is given in a global record. Subsequent global records may specify the same symbol name, but the global is not reestablished. Subsequent declares of the symbol must specify it as the same type, either arithmetic, binary, or character.

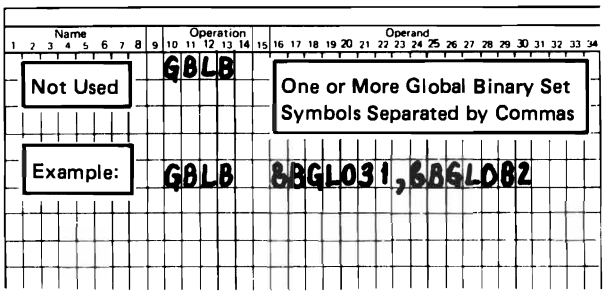
Arithmetic Global (GBLA)

The arithmetic global specifies an arithmetic set symbol. Arithmetic set symbols are initialized to 3 bytes of hexadecimal zeros. The 3-byte field remains through all value assignments. The format of the arithmetic global record is:



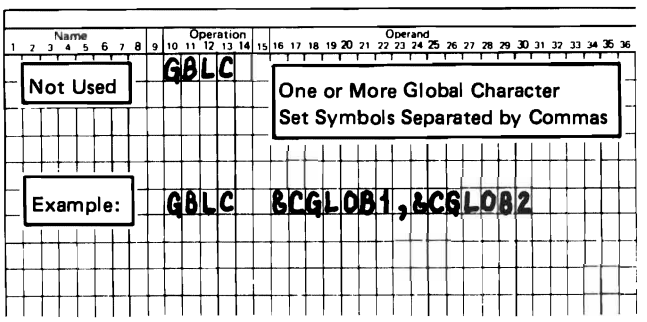
Binary Global (GBLB)

The binary global specifies a binary set symbol. When binary set symbols are defined, they are initialized to zero. The variable can be set to either zero or one by SETB records. The format of the binary global record is:



Character Global (GBLC)

The character global specifies a character set symbol. When a character set symbol is defined, it is given a zero length. A zero- to 8-byte character field can be assigned by the SETC record. The assigned characters may be any of the 256 hexadecimal combinations possible for one byte. The format of the character global record is:



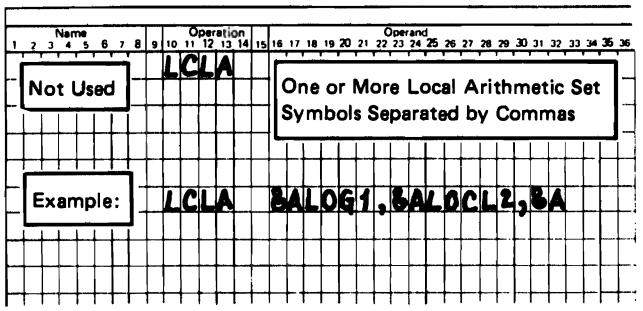
Local

Three types of local records can be used in macroinstruction definitions to generate local set symbols: arithmetic, binary, and character. If used, they must be the first control statements following the global mnemonics, if globals are used, or the first control records following the prototype record, if globals are not used. Local mnemonics can be specified in any order and more than one of each type can be used.

Local set symbols are established and initialized in each macroinstruction definition in which they appear.

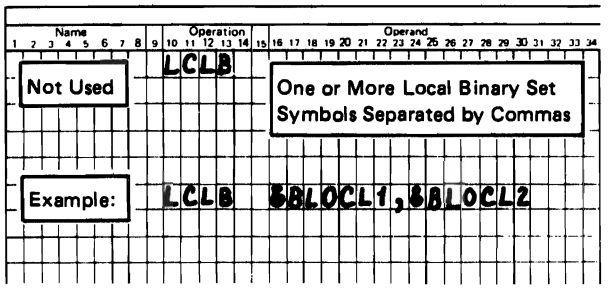
Arithmetic Local (LCLA)

The arithmetic local mnemonic specifies an arithmetic set symbol. Each arithmetic set symbol specified is initialized to 3 bytes of hexadecimal zeros and remains as a 3-byte field. The format of the arithmetic local record is:



Binary Local (LCLB)

The binary local specifies a binary set symbol. The binary set symbol is initialized to zero. The format of the binary local record is:



Character Local (LCLC)

The character local specifies a character set symbol. Each character set symbol is initialized to a null value and zero length. It may then be changed to a character value of from zero to 8 characters. The format of the character local record is:

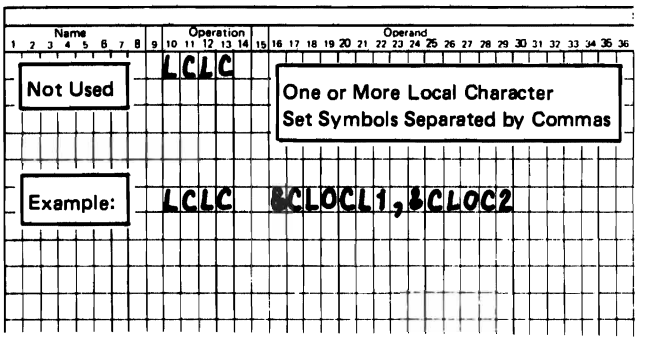


Table (TABLE)

A table statement is used to assign a value to a positional or keyword symbolic parameter. A table statement must be followed by at least one table-definition statement. The format of the table record is:

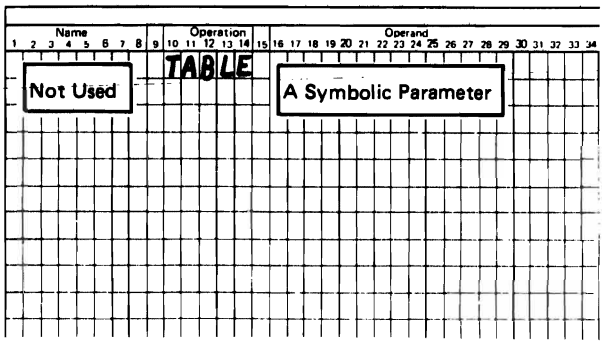


Table-Definition (TABDF)

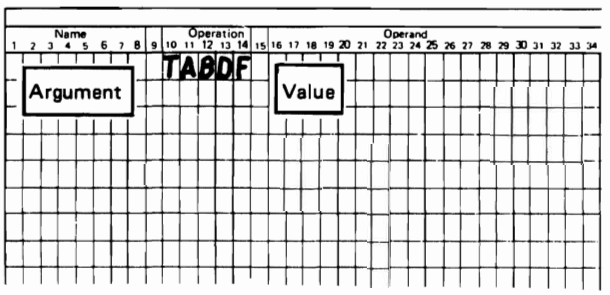
Table-definition statements assign values to symbolic parameters given on table records. The value given in a table-definition statement is assigned to the symbolic parameter given in the previous table statement if one of the following conditions is satisfied:

1. The argument of the table-definition statement matches the value previously assigned to the symbolic parameter by the macroinstruction or prototype record.

2. Positions 1 and 2 of the argument of the table-definition record are occupied by apostrophes and no value has been previously assigned to a symbolic parameter by the macroinstruction or prototype record.

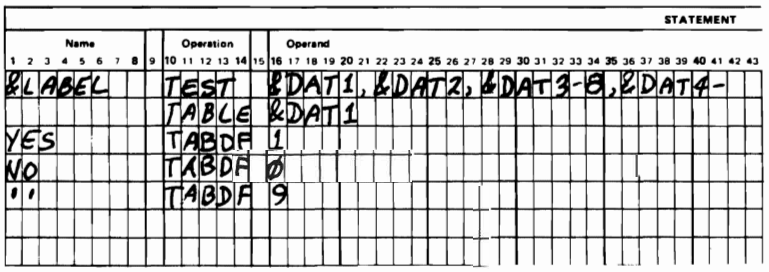
3. The argument of the table-definition statement is blank. A blank argument assigns the specified value to a parameter if the parameter entered does not match an argument specified in a preceding TABDF statement.

At least one table-definition statement must follow each table record. The format of the table definition record is:



Argument: A string of characters with no embedded blanks. The argument may be taken from the prototype record or a user macroinstruction.

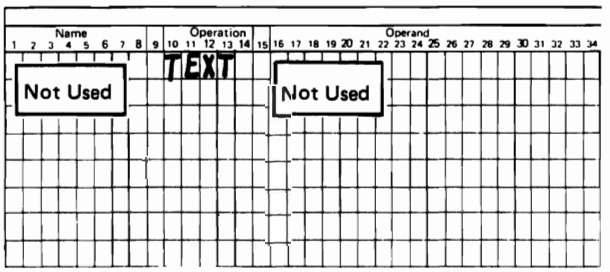
Value: A character string or an alphameric constant. Following is an example of lines from a macro definition that define a table or table record:



In this example if the user enters a yes for the first positional parameter (&DAT1), then &DAT1 is assigned the value 1. If the user makes no data entry for the first positional parameter, then &DAT1 is assigned the value 9.

Text (TEXT)

A text statement must be present in every macroinstruction definition. The text statement denotes the beginning of conditional processing instructions. The definition control records that can appear before the text record in the jobstream are: header, prototype, global, local, table, and table-definition records. Any of these records appearing after the text statement are considered invalid, and errors result. The format of the text statement is:



Comment

Source output comments: These comments can appear after the TEXT record and before the first trailer record (MEND). These comments are written out as part of the macroinstruction expansion. The format of a source output comment is:

```
1 2.....71
* Desired comment
```

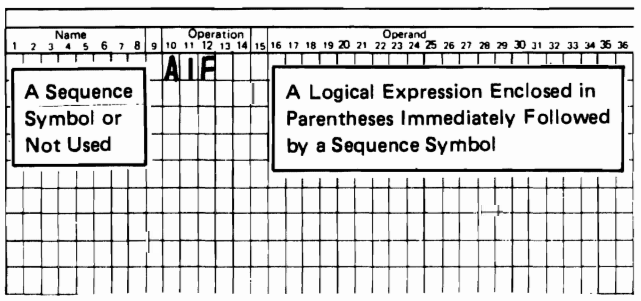
One comment of this format can appear before the header record, but it is not generated as source output.

Comments internal to the macro definition: These comments can appear after the header record and before the first trailer record (MEND). These comments are not included in the macro expansion. The format of an internal macro comment is:

```
1 2 3.....71
  * Desired comment
```

Conditional Branch (AIF)

AIF conditionally alters (forward or backward) the sequence in which macro definition records are processed. The AIF record may appear any place after the TEXT statement. The format of the AIF statement is:



Operand: The logical expression is evaluated to determine whether it is true or false. If the expression is true, the record named by the sequence symbol in the operand is the next record processed by the macro processor. If the logical expression is false, the next sequential instruction of the macro definition is processed.

Whenever AIF operands of unequal length are compared (after assigned values have been substituted for symbolic parameters), the lengths, and not the content, of the operands are compared. Otherwise, three kinds of comparisons of content are possible:

1. Type attribute (T') checking
2. Binary condition checking
3. Value checking

Type Attribute (T') Checking: The user may refer to the type attribute only in the operand of the AIF record. Attribute checking cannot be performed with set symbols.

Condition			Meaning
AIF (T'&name	{NE / EQ} 'N')	.sequence symbol	Test &name for a numeric value.
AIF (T'&name	{NE / EQ} 'U')	.sequence symbol	Test &name for a character string value.
AIF (T'&name	{NE / EQ} 'O')	.sequence symbol	Test &name for a null value (no value assigned). This null test is recommended.
AIF (T'&name	{NE / EQ} ' ')	.sequence symbol	Test &name for a null value (no value assigned). This null test is not recommended.
AIF (T'&name	{NE / EQ} T'&name 1)	.sequence symbol	This test determines whether or not &name and &name 1 have the same attribute.

Note: No concatenation of symbols in an AIF operand is supported in T' processing. If concatenation is specified, an error results.

Binary Condition Checking: The format for binary condition checking is:

```
AIF (&symbol).sequence symbol
```

This format is valid only if &symbol is a binary set symbol. See *Set Binary Record (SETB)* in this chapter. If &symbol has a value of 1, the AIF condition is assumed true, and a branch forward or backward to the sequence symbol is taken. Otherwise, processing continues with the next sequential instruction.

Value Checking

AIF [{count function, &symbol, or 'character expression'} (blank) {GT, GE, EQ, NE, LT, LE} (blank) {count function, &symbol, or 'character expression'}] .sequence symbol

Note: &symbol = any symbolic parameter or set symbol:

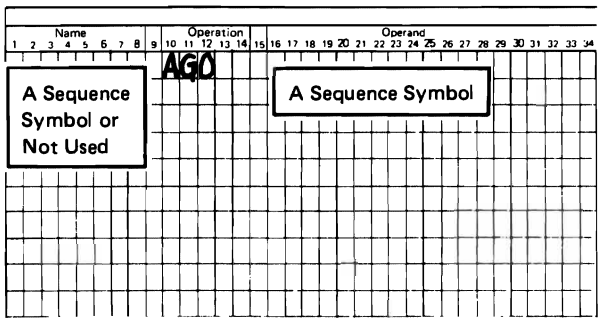
GT = greater than
GE = greater than or equal
EQ = equal
NE = not equal
LT = less than
LE = less than or equal

Concatenation of symbolic parameters, set symbols, and character strings is supported for an AIF record.

Unconditional Branch Record (AGO)

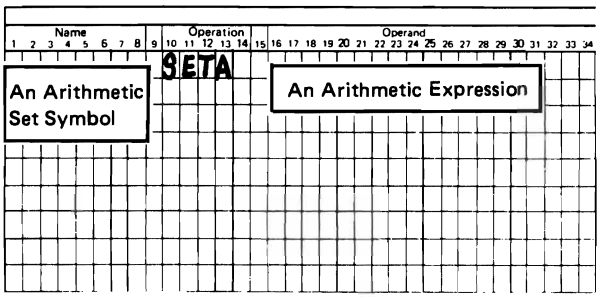
The AGO record unconditionally alters (forward or backward) the sequence in which macro definition records are processed. The AGO record causes a branch forward or backward to the record whose name matches the sequence symbol given in the operand of the AGO record.

The AGO record may appear any place after the TEXT record and before the MEND record. The format of the AGO record is:



Set Arithmetic (SETA)

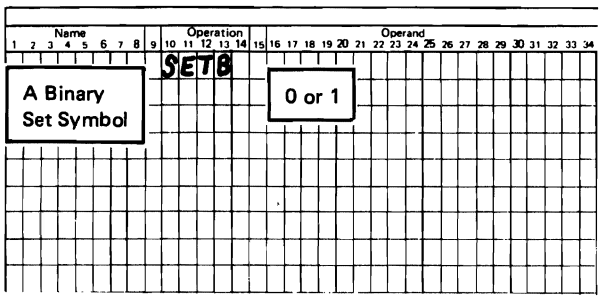
The SETA record assigns a value to the arithmetic set symbol referenced in the name field. The 3-byte hexadecimal value assigned is derived from an evaluation of the operand field. The SETA record may appear any place after the TEXT record. The format of the SETA record is:



Operand: The arithmetic expression may contain arithmetic, character, and/or binary set symbols. Any character set symbols used must have a value of from one to eight decimal digits. Binary set symbols are either zero or 1, and &SYSNDX is given a hexadecimal representation of its current value. The values assigned by the SETA records must be in the range of -8,388,608 to 8,388,607. If you use the count function as an operand, it must appear alone.

Set Binary (SETB)

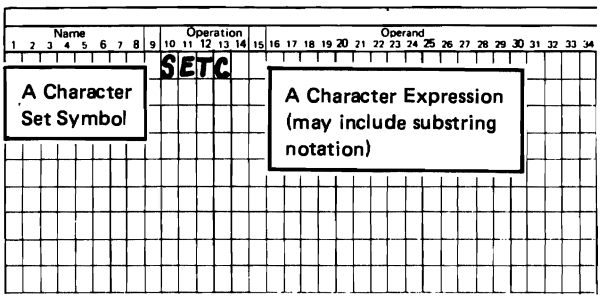
The SETB record assigns a value of zero or 1 to the binary set symbol referenced in the name field. The SETB record may appear any place after the TEXT record. The format of the SETB record is:



Set Character (SETC)

The SETC statement assigns a zero- through 8-character value to the character set symbol referenced in the name field. The character value assigned is derived from an evaluation of the operand field. If the derived value contains more than eight characters, only the first eight characters are used.

The SETC record may appear any place after the TEXT record. The assigned characters may be any of the 256 hexadecimal combinations possible for one byte. The format of the SETC record is:

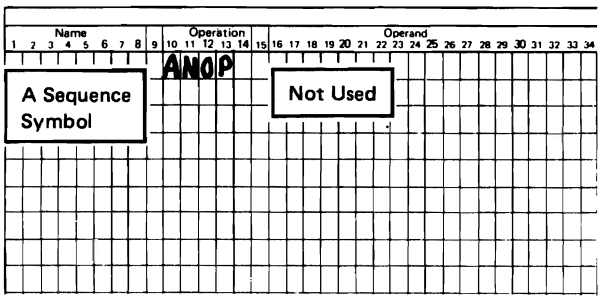


Operand: The character expression, which may include substring notation, may contain character, arithmetic, and binary set symbols. Null values can be assigned in the character expression by specifying two consecutive single apostrophes or by specifying only variable symbols that already have null values.

Arithmetic set symbols used in the character expression are converted to only their significant decimal digits in the string. All leading zeros are dropped, and, if the value of the arithmetic set symbol is zero, a single decimal zero is used. Binary set symbols appear as either zero or 1, and &SYSNDX is given its current value in three decimal digits.

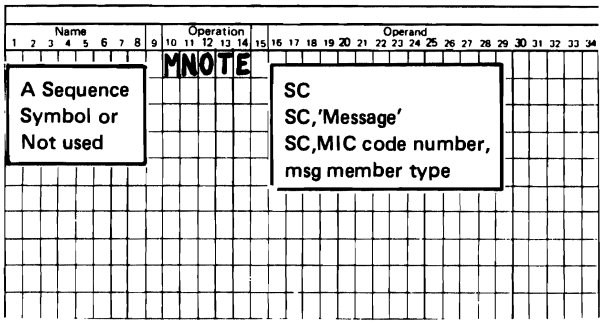
Assembly No Operation (ANOP)

The ANOP statement may be used to provide a name (sequence symbol) to which AIF and AGO statements may branch. ANOP may appear any place after the TEXT statement and before the MEND statement. The format of the ANOP statement is:



Message (MNOTE)

The MNOTE statement may be used by the programmer to generate a message and to indicate the error severity, if any, to be associated with the message. The MNOTE statement may appear any place after the TEXT statement. The format of the MNOTE statement is:



Operand:

SC = Severity code (two digits 00-99)

Severity codes are divided into the following classifications:

SC < 08 The macro processor generates the message as an assembler comment (* in position 1), and *no* error condition occurs.

SC = 08 The macro processor generates a special assembler statement that will cause the message to be printed on the assembler source listing with a warning (W-error).

SC > 08 The macro processor generates the message as an assembler comment without an * in column 1. This will cause the assembler to flag that statement as a hard error (M-error).

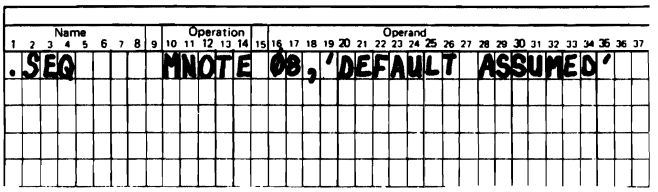
'Message' One to 50 characters enclosed in apostrophes with no embedded apostrophes. This message will eventually occur as coded on the assembler source listing.

MIC code number A four-digit code which identifies the message member within the message member type.

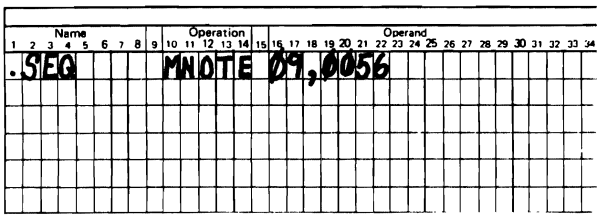
Note: Message member type for the MNOTES for IBM-supplied macroinstructions is ASM.

Examples:

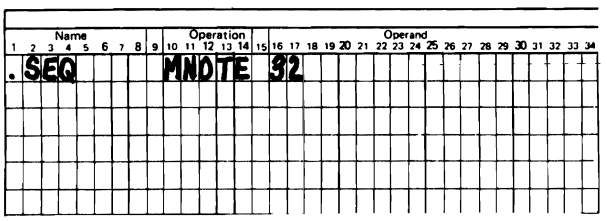
1. An MNOTE instruction which causes a W error and a comment on the source listing:



2. An MNOTE instruction which causes an M error and generates message #56 as obtained from the user message member set:

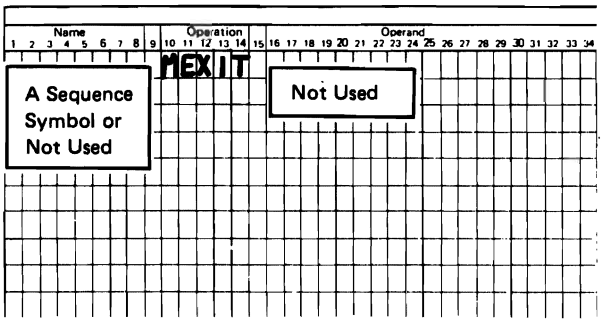


3. An MNOTE instruction which causes an M error but no message:



Logical End (MEXIT)

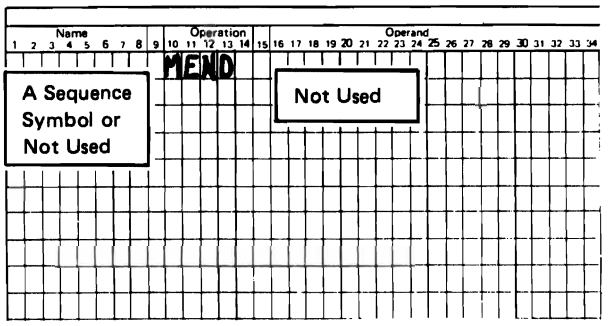
The MEXIT statement denotes that macro definition processing ends with this record. The format of the MEXIT record is:



Physical End (MEND)

The MEND statement denotes the end of, and must be the last control record in, the macro definition. Processing of the macro definition ends when this record is encountered.

The format of the MEND record is:



SAMPLE DEFINITION OF A USER MACROINSTRUCTION

Figure 5-1 shows the definition of a user-defined macroinstruction that generates instructions to move more than 256 bytes of data. Figure 5-2 shows an assembled program in which the user-defined macroinstruction is issued. The macroinstruction is issued several times in the program to demonstrate how parameters specified in the macroinstruction determine which lines of code are generated from the macroinstruction definition.

For a description of how to code macroinstruction statements, see Chapter 6. For an example of IBM-supplied macroinstruction definitions, see *Sample Macroinstructions* in Appendix A.

```
         MACRO
         @MOVL &TO,                        MOVE 'TO' LABEL (LEFT BYTE)     C
               &FROM,                      MOVE 'FROM' LABEL (LEFT BYTE)   C
               &LENGTH,                    LENGTH OF FROM AND TO FIELDS    C
               &ADDR-                      ADDRESS TO BE IN REGISTER ONE
         LCLA  &WRKLNG                     LENGTH--REMAINING BYTES TO MOVE
         LCLA  &WRKLM1                     LENGTH MINUS ONE
         LCLB  &SW                         EDIT SWITCH, IF ON GEN NO CODE
         LCLC  &WRKAD                      SUBSTRING OF ADDR- PARM
         TEXT
         SPACE
&SW      SETB  0                           SET EDIT SWITCH OFF
.*                                         ..IF THERE IS AN EDIT ERROR
.*                                         ..IT IS SET TO ONE AND NO
.*                                         ..INSTRUCTIONS WILL BE GENERATED
.*
.*************************************************************************
.*                                                                       *
.*       CHECK PARAMETER ONE, TO ADDRESS LABEL.                          *
.*                                                                       *
.*************************************************************************
         AIF   (T'&TO NE 'O').MV$001       IF FIRST PARM ENTERED,
.*                                         ..GO CHECK ITS LENGTH, ELSE
.*                                         ..WRITE OUT AN ERROR MESSAGE
.*                                         ..AND SET ON THE EDIT SWITCH SO
.*                                         ..THAT NO CODE IS GENERATED.
.*
         MNOTE 08,'PARM 1 (TO ADDR) MAY NOT BE OMITTED.          '
&SW      SETB  1                           SET EDIT ERROR SWITCH ON
         AGO   .MV$002                     GO CHECK THE 'FROM' ADDRESS
.*
.MV$001  ANOP
         AIF   (K'&TO LT '7').MV$002       IF THE NUMBER OF CHARACTERS IN
.*                                         ..PARM 1 IS 6 CHARACTERS OR LESS
.*                                         ..IT IS OK, ELSE SET ON THE
.*                                         ..ERROR SWITCH.
.*
         MNOTE 08,'PARM 1 (TO ADDR) MUST BE 6 CHARACTERS OR LESS.  '
&SW      SETB  1
.*
.*************************************************************************
.*                                                                       *
.*       CHECK PARAMETER TWO, FROM ADDRESS LABEL                         *
.*                                                                       *
.*************************************************************************
.MV$002  ANOP
         AIF   (T'&FROM NE 'O').MV$003     IF SECOND PARM ENTERED,
.*                                         ..GO CHECK ITS LENGTH, ELSE
.*                                         ..WRITE OUT AN ERROR MESSAGE
.*                                         ..AND SET ON THE EDIT SWITCH SO
.*                                         ..THAT NO CODE IS GENERATED.
.*
         MNOTE 08,'PARM 2 (FROM ADDR) MAY NOT BE OMITTED.        '
&SW      SETB  1                           SET EDIT ERROR SWITCH ON
         AGO   .MV$004                     GO CHECK THE LENGTH PARAMETER
.*
.MV$003  ANOP
         AIF   (K'&FROM LT '7').MV$004     IF THE NUMBER OF CHARACTERS IN
.*                                         ..PARM 2 IS 6 CHARACTERS OR LESS
.*                                         ..IT IS OK, ELSE SET ON THE
.*                                         ..ERROR SWITCH.
.*
         MNOTE 08,'PARM 2 (FROM ADDR) MUST BE 6 CHARACTERS OR LESS.'
&SW      SETB  1
.*
.*************************************************************************
.*                                                                       *
.*       CHECK PARAMETER THREE, LENGTH OF MOVE.                          *
.*                                                                       *
.*************************************************************************
.MV$004  ANOP
         AIF   (T'&LENGTH NE 'O').MV$005   IF LENGTH PARM ENTERED
.*                                         ..GO SEE IF IT IS NUMERIC, ELSE
.*                                         ..WRITE OUT AN ERROR MESSAGE
.*                                         ..AND SET ON THE EDIT SWITCH SO
.*                                         ..THAT NO CODE IS GENERATED.
.*
         MNOTE 08,'PARM 3 (LENGTH ) MAY NOT BE OMITTED.          '
&SW      SETB  1                           SET EDIT ERROR SWITCH ON
         AGO   .MV$006                     GO SEE IF ALL EDITS PASSED.
.*
.MV$005  ANOP
         AIF   (T'&LENGTH EQ 'N').MV$006   IF THE LENGTH PARM IS NUMERIC
.*                                         ..IT IS OK, OTHERWISE
.*                                         ..SET ON THE ERROR SWITCH
.*
         MNOTE 08,'PARM 3 (LENGTH ) MUST BE NUMERIC.             '
&SW      SETB  1
 *
```

Figure 5-1 (Part 1 of 2). Sample Macroinstruction Definition

```
.*********************************************************************
.*                                                                   *
.*       CHECK THE EDIT SWITCH.                                      *
.*                                                                   *
.*********************************************************************
.MV$006  ANOP
         AIF   (&SW).MV$EXIT           IF THE EDIT SWITCH IS ON, EXIT
.*                                     ..THE MACRO AND DO NOT GENERATE
.*                                     ..ANY CODE.
.*
.*********************************************************************
.*                                                                   *
.*       GENERATE THE NECESSARY MOVE INSTRUCTIONS.                   *
.*                                                                   *
.*********************************************************************
&WRKLNG  SETA  &LENGTH                 SET TO TOTAL NUMBER OF BYTES
&WRKLM1  SETA  &LENGTH-1               SET TO NUMBER TO MOVE MINUS ONE
.MV$LOOP ANOP
         AIF   (&WRKLNG LT '257').MV$END  IF THERE ARE LESS THAN 257
.*                                     ..BYTES REMAINING TO BE MOVED
.*                                     ..THEY CAN BE MOVED IN ONE
.*                                     ..INSTRUCTION, OTHERWISE
.*                                     ..MOVE ONLY 256 BYTES AND
.*                                     ..DECREASE THE NUMBER REMAINING
.*                                     ..BY 256.
.*
         MVC   &TO+&WRKLM1(256),&FROM+&WRKLM1
&WRKLNG  SETA  &WRKLNG-256
&WRKLM1  SETA  &WRKLNG-1
         AGO   .MV$LOOP
.MV$END  ANOP
         MVC   &TO+&WRKLM1(&WRKLNG),&FROM+&WRKLM1
.*
.*********************************************************************
.*                                                                   *
.*       CHECK PARAMETER FOUR, ADDRESS TO BE LOADED IN REGISTER ONE  *
.*                                                                   *
.*********************************************************************
         AIF   (T'&ADDR EQ 'O').MV$EXIT IF PARM 4 WAS OMITTED, THIS IS
.*                                     ..OK AS IT IS AN OPTIONAL PARM.
.*
         AIF   ('&ADDR'(1,1) NE '@').MV$LDAD  IF THE FIRST CHARACTER
.*                                     ..OF PARM 4 IS NOT AN '@', GO
.*                                     ..GENERATE A LOAD ADDRESS
.*                                     ..INSTRUCTION, OTHERWISE
.*                                     ..GENERATE A LOAD INSTRUCTION
.*                                     ..USING CHARACTERS TWO THRU
.*                                     ..SEVEN.
.*
&WRKAD   SETC  '&ADDR'(2,7)            SET STRING TO IGNORE THE '@'
         L     &WRKAD,1
         AGO   .MV$EXIT                 MACRO IS DONE, EXIT MACRO
.MV$LDAD ANOP
         LA    &ADDR,1
.MV$EXIT ANOP
         MEXIT
         MEND
```

Figure 5-1 (Part 2 of 2). Sample Macroinstruction Definition

```
                    IBM SYSTEM/34 BASIC ASSEMBLER-MACRO PROCESSOR              RELEASE 04

ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT                  05-09-79   TIME 13.37  PAGE   2
    0000                        1          START X'0000'
                                3 *        @MOVL HERE,THERE,512
    0000 0C FF 0227 0427        5+         MVC   HERE+511(256),THERE+511
    0008 0C FF 0127 0327        6+         MVC   HERE+255(256),THERE+255
                                8 *        @MOVL HERE,THERE,224,ADDR-HERE
    000C 0C DF 0107 0307       10+         MVC   HERE+223(224),THERE+223
    0012 C2 01 0028            11+         LA    HERE,1
                               13 *        @MOVL HERE,THERE,400,ADDR-@HEREADR
    0016 0C FF 01B7 03B7       15+         MVC   HERE+399(256),THERE+399
    001C 0C 8F 00B7 02B7       16+         MVC   HERE+143(144),THERE+143
    0022 35 01 0027            17+         L     HEREADR,1
                               19 *        @MOVL HERE,TOOMANY,375
W                              21 *08  PARM 2 (FROM ADDR) MUST BE 6 CHARACTERS OR LESS.                    *MNOTE
                               23 *        @MOVL HERE,THERE
W                              25 *08  PARM 3 (LENGTH ) MAY NOT BE OMITTED.                                *MNOTE
    0026 0028           0027   27 HEREADR  DC    AL2(HERE)
                        0028   29 HERE     EQU   *
    0028                0027   30          DS    2CL256
                        0228   32 THERE    EQU   *
    0228                0427   33          DS    2CL256
                        0428   35 TOOMANY  EQU   *
    0428                0627   36          DS    2CL256
                        FFFF   38          END
```

Figure 5-2. Use of Sample Macroinstruction

Chapter 6. Macroinstruction Statements

A macroinstruction is a source statement that generates a predetermined set of assembler statements each time the macroinstruction is used. The IBM System/34 Basic Assembler and Macro Processor Program Product provides macroinstructions which perform both system services and input/output device support. By using these macroinstructions, you can perform both system and input/output operations with less coding.

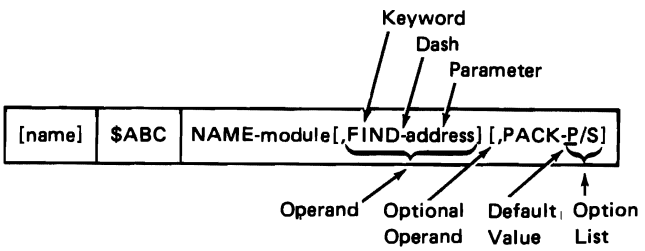
Writing Macroinstructions

You code macroinstructions as follows:

Name	Operation	Operands	Continuation
Symbol or Blank	Macro Name	No Operands or One or More Separated by Commas	Any Nonblank Character if Continuation is Being Used

The name field can contain any valid assembler language symbolic name beginning in column 1. The name is assigned to the first byte of generated code. Since the name is optional, it is shown below in brackets.

The desired mnemonic operation code (macroinstruction name) must appear as specified in that macroinstruction's description. The operation code must start in column 10.



Operands specify the available services and options that you want to use. The operands must start in column 16. Macroinstructions supplied by IBM use only keyword operands. The following conventions apply to the IBM-supplied macroinstructions:

- Each operand consists of a keyword followed by a dash and a parameter.
- Keywords–those shown in capital letters–are coded exactly as shown in this chapter.
- The parameter part of the operand must immediately follow the dash.
- Parameters–those shown in lowercase letters–indicate information you must supply.
- An option list for a keyword parameter is specified as follows:

  KEYWORD-A/B/C

  This list indicates that options A, B, or C are the only valid options for the keyword parameter. When the options Y/N are given in a macroinstruction, Y indicates a yes response, N indicates a no response.
- Commas separate the operands; blanks cannot be left between operands.
- The keyword operands may be written in any order.

Optional operands are indicated in this chapter by enclosing the operand within brackets [KEYWORD-parameter]. If an operand is not specified, the default value is used. A default value is selected for any optional keyword that is omitted. The default value is indicated in this chapter by a line under the default option. For example, [KEY-A/B/C] indicates that option A is the default value.

No operands can be specified beyond column 71. If continuation is required, column 72 must contain a nonblank character and the last operand before continuation must be followed by a comma and a blank. If the comma is in column 71, the blank is not required. An operand cannot be divided and continued on the next line. The operands of the continued field must begin in column 16. For an example of continuation coding, see Figure 6-1.

A comment must be separated from the operand or comma by at least one blank space. A comment cannot be inserted between operands on a one-line macroinstruction. Figure 6-2 shows examples of comments used with macroinstructions. On the assembler listing, all comments on the generated code are aligned by the macro processor to begin in column 40. Any comments too long to be contained in columns 40 through 71 are truncated from the right.

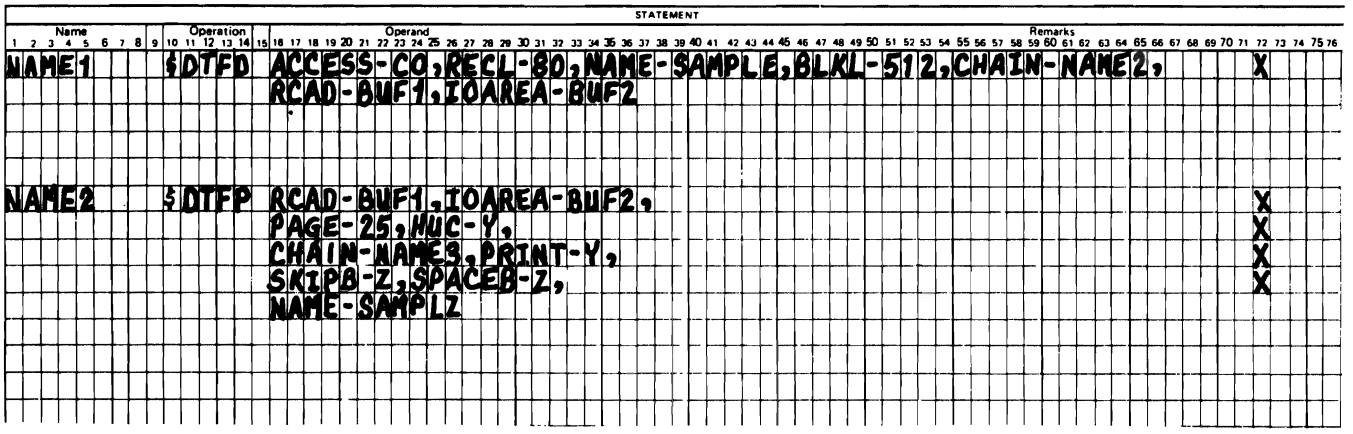


Figure 6-1. Continuation Coding Examples

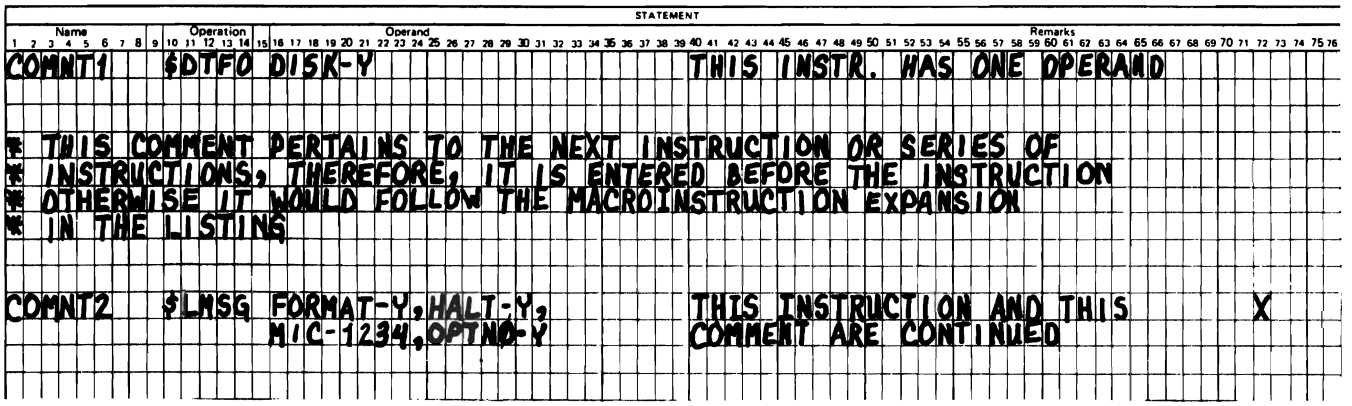


Figure 6-2. Comments on Macroinstructions

Macroinstructions Supplied By IBM

The macroinstructions provided by the IBM System/34 Basic Assembler and Macro Processor Program Product and the functions they perform are shown below.

Note: Communications macroinstructions are described in the *Data Communications Reference Manual* and the *Interactive Communications Feature Reference Manual.* Scientific macroinstructions are described in the *Scientific Macroinstruction Reference Manual.*

Device Type Supported	Macroinstruction Name	Function
System Log	$LMSG	Generate parameter list for message displayed by system log
	$LOGD	Offsets in log parameter list
	$LOG	Linkage to system log
General SSP	$FNDP	Generate find parameter list
	$FIND	Find a directory entry
	$LOAD	Load or fetch a module
	$SNAP	Snap dump of main storage
	$INFO	System information retrieval
	$CKEQ	Generate a parameter list for checkpoint requests
	$CKPT	Establish a checkpoint
	$INV	Inverse data move
	$EOJ	Linkage to end of job
General I/O	$ALOC	Allocate disk space or device
	$OPEN	Prepare a device or file for access
	$CLOS	Prepare a device or file for termination
	$DTFO	Generate DTF offsets for all devices, including data communications
Printer	$DTFP	Define the file for a printer
	$PUTP	Construct a printer PUT interface
Disk	$DTFD	Define the file for disk
	$GETD	Construct a disk GET interface
	$PUTD	Construct a disk PUT interface
Disk Sort	$SRT	Generate a loadable sort parameter list
	$SORT	Construct sort interface
Timer	$TRB	Generate timer request block
	$SIT	Set timer interval
	$RIT	Return/cancel timer interval
	$TOD	Return time and date
Display Station	$DTFW	Define the file for display station
	$WSIO	I/O requests to display station
	$WIND	Generate indicators for PUT and PUT overrides
	$WSEQ	Generate labels and values for display station device-dependent values

Figure 6-3. Macroinstructions Supplied by IBM

System Services Macroinstructions

Use system services macroinstructions when you want to communicate with the System Support Program Product (SSP). These macroinstructions can do the following:

- Log and write error messages
- Obtain object modules from disk and load them into main storage
- Pass control to modules in main storage
- Determine the location of an object module on disk
- Terminate the current job

The system services macroinstructions are divided into these groups:

1. System log macroinstructions provide support for and linkage to system log functions.

   $LMSG
   $LOGD
   $LOG

2. General SSP macroinstructions provide linkage to system functions.

$FNDP	$CKEQ
$FIND	$CKPT
$LOAD	$INV
$SNAP	$EOJ
$INFO	

SYSTEM LOG SUPPORT

Specifying a $LOG macroinstruction in your program generates a call to system log. (System log is a group of system output routines that provide communication between the operator and the system.) You may want to use system log to notify the operator of error conditions, error recovery procedures, and the validity of previous operator responses to halts. If the operator selects an invalid option in response to a halt, the response is not accepted by system log. Instead, the halt is displayed again with another message indicating that the response is invalid.

Note: When an immediate cancel (option 3) is selected, control is passed directly to the end-of-job (EOJ) routine by system log.

Two types of output are available through system log; formatted and unformatted messages. Both are displayed on the system log device.

- A formatted message consists of two lines. The first line, the format line, contains the message ID and available options. The second line contains the actual message text.
- An unformatted message consists of one line. This statement indicates errors or issues instructions to the operator; for example, requesting that a disk file be loaded.

Messages can be issued with or without accompanying halts and can be routed to either the system console or the work station.

To use system log, you must do the following:

1. Build the log parameter list using the $LMSG macroinstruction.
2. Use the $LOGD macroinstruction if you want to establish labels for the log parameter list. You can then use the labels to modify the parameter list during program execution.
3. Issue the $LOG macroinstruction.
4. Process in your program any replies returned by the operator.

Generate a Parameter List for a Message Displayed by System Log ($LMSG)

This macroinstruction generates a system log parameter list for a message to the operator. This parameter list is referenced by a $LOG macroinstruction when $LOG is used to issue a message. See the chart following the parameter descriptions for a summary of which parameters to use with each message type.

The format of the $LMSG macroinstruction is:

```
[name] $LMSG [TYPE-code] [,COMID-code]
             [,SUBID-code] [,FORMAT-Y/N]
             [,HALT-Y/N] [,MIC-number]
             [,OPTN0-Y/N] [,OPTN1-Y/N]
             [,OPTN2-Y/N] [,OPTN3-Y/N]
             [,SKIP-Y/N] [,SPACE-1/2/3]
             [,MSGLN-number]
             [,MSGAD-address][,WRSTE-Y/N]
             [,DRLEN-number]
             [,DRADD-address] [,HIST-Y/N]
             [,CRT-Y/N] [,VARIN-Y/N]
```

TYPE-code specifies the type of system log parameter list. If this operand is omitted, TYPE-1 is assumed. The valid codes and their meanings follow:

Code	Meaning
1	Output from a message member, without data response
1R	Output from a message member, with data response
2	Output from a user program, without data response
2R	Output from a user program, with data response
3	Output from a user program, with a format line. The format line contains the program ID, the MIC number, options if options are available, and the program name.
4	Type 1 with 8 bytes of user-supplied information added to the front of the message

COMID-code specifies a 2-byte field used to identify the module issuing the message. If this operand is omitted, blank is assumed. This field is not displayed, but is logged in the history file if HIST-Y is specified.

SUBID-code specifies a 2-byte field used to further identify the module issuing the message. If this operand is omitted, blank is assumed. This field is not displayed, but is logged in the history file if HIST-Y is specified.

FORMAT-Y/N specifies whether or not to include the format line if the output is from a message member. If this operand is omitted and TYPE-3 is not specified, N (no) is assumed. If TYPE-3 is specified, do not specify FORMAT: FORMAT-Y (yes) is always assumed if TYPE-3 is specified.

HALT-Y/N specifies whether or not response is required (that is, whether the operator is supposed to enter an option number). If this operand is omitted, N (no) is assumed.

MIC-number is a decimal number, within 0001-9999, used to identify a specific message within the message member. If this operand is omitted, 0001 is assumed.

OPTN0-Y/N specifies whether option 0 is allowed. If Y (yes) is entered, option 0 is allowed; if N (no) is entered or if the operand is omitted, option 0 is not allowed.

OPTN1-Y/N specifies whether option 1 is allowed. If Y (yes) is entered, option 1 is allowed; if N (no) is entered or if this operand is omitted, option 1 is not allowed.

OPTN2-Y/N specifies whether option 2 is allowed. If Y (yes) is entered, option 2 is allowed; if N (no) is entered or if this operand is omitted, option 2 is not allowed.

OPTN3-Y/N specifies whether option 3 is allowed. If Y (yes) is specified, option 3 is allowed; if N (no) is specified or if this operand is omitted, option 3 is not allowed. If option 3 is allowed and selected by the user, control will not be returned to your program.

SKIP-Y/N specifies whether or not to skip to line six of the next page before printing. This operand is valid for printed messages only. If this operand is omitted, N (no) is assumed.

SPACE-1/2/3 specifies the number of lines to space after printing a message. This operand is valid for printed messages only. If this operand is omitted, 1 is assumed.

MSGLN-number specifies the text length. The number must be a decimal entry from 1 to 132. Anything over 75 bytes is truncated if the message is routed to a display station or the system console. This parameter specifies the insert data length if VARIN-Y is specified.

MSGAD-address specifies the leftmost byte of the message buffer. This parameter specifies the insert data address if VARIN-Y is specified.

Note: The message buffer should contain only printable characters. For example, the buffer should not contain BSC or SNA control characters.

WRSTE-Y/N specifies whether the message is routed to the work station or the system console. If this operand is omitted, Y (yes) is assumed and the message is routed to the work station. If WRSTE-N is specified, messages are routed to the system console. If the system console is being used as a work station and the job is an SRT, messages are routed to that work station.

Note: The message is displayed only if CRT-Y is specified, regardless of routing.

DRLEN-number is the length of the reply area in decimal. This area must be either 1, 8, 60, or 120 bytes long.

DRADD-address specifies the address of the leftmost byte of the reply area.

HIST-Y/N specifies whether or not the message is to be recorded on the history file. If this operand is omitted, Y (yes) is assumed.

CRT-Y/N specifies whether or not the message is to be displayed on the display screen. If this operand is omitted, Y (yes) is assumed.

VARIN-Y/N specifies a variable length data insert for type 1 messages. The system log function allows you to insert variable length data anywhere into the text of a message that is retrieved from a message member. Substitution occurs wherever the symbol # appears in the message text. If this operand is omitted, N (no) is assumed.

Note: The inserted data should contain only printable characters. For example, the data string should not contain BSC or SNA control characters.

$LMSG Parameter Use Chart

Msg Type / Param	1	1R	2	2R	3	4	Default Values
COMID	R if FORMAT-Y	S			S	R if FORMAT-Y	blanks
SUBID	R if FORMAT-Y	S			S	R if FORMAT-Y	blanks
FORMAT	R 2673	E 2646	E 2646	E 2646	E 2646	R 2673	No
HALT	R 2674	E 2647	E 2647	E 2647	R 2674	R 2674	No
MIC	R 2657	R 2657			R 2657	R 2657	0001
OPTN0	R if HALT-Y 2650				R if HALT-Y 2650	R if HALT-Y 2650	No
OPTN1	R if HALT-Y 2650				R if HALT-Y 2650	R if HALT-Y 2650	No
OPTN2	R if HALT-Y 2650				R if HALT-Y 2650	R if HALT-Y 2650	No
OPTN3	R if HALT-Y 2650				R if HALT-Y 2650	R if HALT-Y 2650	No
SKIP			S	S			No
SPACE			R 2675	R 2675			1
MSGLN	R if VARIN-Y		R 2654	R 2654	R 2654		TYPE-1 and VARIN-Y, 8; else, 75
MSGAD	R if VARIN-Y		R 2649	R 2649	R 2649		FFFF
WRSTE	R 2672	R 2672	R 2672	R 2672	R 2672	R 2672	Yes
DRLEN		R 2653		R 2653			8
DRADD		R 2648		R 2648			FFFF
HIST	S	S	S	S	S	S	Yes
CRT	S	S	S	S	S	S	Yes
VARIN	S						No

If HALT-Y is specified, Y must be specified for at least one OPTN parameter.

Key to chart:

1. No entry = parameter not used with corresponding message type.
2. R = parameter is used with corresponding message type under noted circumstance or diagnostic MIC number issued if not entered.
3. E = parameter invalid with corresponding message type and diagnostic MIC number issued if entered.
4. S = parameter used with corresponding message type and default assumed if not entered.

Generate Displacements for System Log ($LOGD)

This macroinstruction generates the field labels and offsets for the system log parameter lists. To avoid duplicate labels, you should use this macroinstruction only once in a program.

The format of the $LOGD macroinstruction is:

```
[name] $LOGD     no operands
```

Generate the Linkage to the System Log ($LOG)

This macroinstruction generates the linkage required to use the system log function, and checks the response returned.

If you will need the data in register 2 later, you should save the contents of that register before issuing $LOG.

The format of the $LOG macroinstruction is:

```
[name] $LOG   [LIST-address] [,OPTN0-address]
              [,OPTN1-address]
              [,OPTN2-address]
```

LIST-address specifies the address of the leftmost byte of the system log parameter list generated by the $LMSG macroinstruction. If this operand is not specified, the address of the parameter list is assumed to be in register 2.

OPTN0-address specifies the address of the routine that should receive control if option 0 is taken. If this operand is not specified, no check is made for a response of 0. You would use this operand only if OPTN0-Y was specified for the associated system log parameter list.

OPTN1-address specifies the address of the routine that should receive control if option 1 is taken. If this operand is not specified, no check is made for a response of 1. You would use this operand only if OPTN1-Y was specified for the associated system log parameter list.

OPTN2-address specifies the address of the routine that should receive control if option 2 is taken. If this operand is not specified, no check is made for a response of 2. You would use this operand only if OPTN2-Y was specified for the associated system log parameter list.

GENERAL SSP SUPPORT

The general SSP macroinstructions provide linkage to system functions.

Generate Parameter List and Displacements for $FIND ($FNDP)

The $FNDP macroinstruction generates a loader parameter list and generates the labels for the displacements into the parameter list. This parameter list is used as input to the supervisor by $FIND.

The format of the $FNDP macroinstruction is:

```
[name] $FNDP  [NAME-module]
              [,V-DC/EQU/ALL]
              [,TYPE-O/P/R/S] [,SKIP-code]
              [,LOADER-Y/N] [,LOAD-address]
```

NAME-module is the name of the module to be found by $FIND macroinstruction. If this operand is omitted, blanks are assumed.

V-DC/EQU/ALL specifies whether the parameter list, labels, or both are to be generated. If this operand is omitted, EQU is assumed.

DC generates a 12- or 18-byte parameter list used by the $FIND macroinstruction.

EQU generates the displacement labels for the $FIND parameter list. If V-EQU is specified or defaulted, all other operands are ignored.

ALL generates both the parameter list and the corresponding displacement labels.

TYPE-O/P/R/S specifies the library member type. If this parameter is omitted, O is the default.

Code	Meaning
O	Load member
P	Procedure member
R	Subroutine member
S	Source member

SKIP-code specifies the type of library search to perform.

Code	Description
NO	Search the designated user library, then the system library
USER	Skip the user library and search only the system library
SYSTEM	Skip the system library and search only the designated user library

If this operand is omitted, NO is assumed.

LOADER-Y/N specifies whether the parameter list is used by $LOAD. If Y (yes) is specified, a 12-byte parameter list is generated for use by $LOAD. If N (no) is specified, an 18-byte parameter list is generated that cannot be used by $LOAD. If this operand is omitted, N (no) is assumed. LOADER-Y can only be specified with TYPE-O.

Note: When the module is not found: If LOADER-Y is specified in the $FNDP macroinstruction, a cancel-only halt is issued and control is not returned to your program. If LOADER-N is specified in the $FNDP macroinstruction, control is returned to your program for determination of appropriate action.

LOAD-address specifies the address where the module is to be loaded in main storage. This address must be on an 8-byte boundary, due to the I/O buffer boundary restrictions. This operand is processed only if LOADER-Y is specified.

Find a Directory Entry ($FIND)

You can use the $FIND macroinstruction to locate library members that you want to load for use by your program.

The $FIND macroinstruction searches the library directory for the requested module name; if it locates the module name it returns the directory entry data in the parameter list.

$FIND requires the parameter list generated by the $FNDP macroinstruction. $FNDP is described in a preceding paragraph.

You can include more than one $FIND macroinstruction in a program. However, after you issue the first $FIND, you must continue to restore relevant fields in the parameter list generated by $FNDP before you issue successive $FINDs. You can restore fields in the parameter list by moving new values to the fields.

Note: When a module is not found by $FIND: If LOADER-Y is specified in the $FNDP macroinstruction, a cancel-only halt is issued and control is not returned to your program. If LOADER-N is specified in the $FNDP macroinstruction, control is returned to your program for determination of appropriate action.

If you will need the data in register 2 later, you should save the contents of that register before issuing $FIND.

The format of the $FIND macroinstruction is:

```
[name] $FIND      [PLIST-address]
```

PLIST-address specifies the address of the leftmost byte of the 12- or 18-byte parameter list built by $FNDP. After execution, the parameter list contains the directory entry of the module. If this operand is not specified, the address of the parameter list is assumed to be in index register 2.

Load or Fetch a Module ($LOAD)

The $LOAD macroinstruction generates the linkage to load a module into main storage at the address you specify. The address is specified in the $FNDP or $LOAD macroinstruction using the LOAD keyword. LOADER-Y should be specified in the $FNDP macroinstruction so that the parameter list output from $FNDP will be a $LOAD parameter list. You may have control returned after the module is loaded, or you may pass control to the module. If you will need the data in register 2 later, you should save the contents of register 2 before issuing $LOAD.

The format of the $LOAD macroinstruction is:

```
[name] $LOAD [PLIST-address] [,TYPE-code]
             [,LOAD-address]
```

PLIST-address specifies the address of the leftmost byte of the parameter list built by $FNDP, which identifies the directory entry of the module in main storage. If this operand is omitted, the address is assumed to be in register 2.

TYPE-code specifies the type of load to perform. If this operand is omitted, LOAD is assumed.

LOAD loads the module at the specified LOAD address and returns control.

FETCH loads the module at the specified LOAD address and passes control to the module.

LOAD-address specifies the address where the module is to be loaded in main storage. The address must be on an 8-byte boundary. Use this parameter only if the load address is to be filled in or changed. If the PLIST parameter in this macroinstruction uses index register 2, then the LOAD parameter must also use index register 2.

Snap Dump of Main Storage ($SNAP)

This macroinstruction provides an interface with the nonterminating system storage dump routine. You must specify the region or the limits of the area to be dumped. The contents of the specified main storage area are printed on the SYSLIST device. Output from the dump routine consists of:

- The specified dump identifier
- The contents of register 1 (XR1), register 2 (XR2), the instruction address register (IAR), and the address recall register (ARR)
- The contents of the specified main storage area

Control is returned to the next sequential instruction in your program.

The format of the $SNAP macroinstruction is:

```
[name] $SNAP [REGION-Y/N] [,LOW-address]
             [,HIGH-address] [,ID-char]
             [,PLIST-2/address/INLINE]
             [,V-DC/EQU/ALL]
```

REGION-Y/N specifies whether the entire region should be dumped and whether the HIGH and LOW parameters should be ignored. If Y (yes) is specified, the entire region is dumped. If N (no) is specified, the area specified by the HIGH and LOW parameters is dumped. If this operand is omitted, N is assumed[1].

LOW-address specifies the address of the low limit of the storage area to be dumped. The low limit must be lower than the high limit and within the allocated storage area. If this operand is omitted, address X'FFFF' is assumed[1].

HIGH-address specifies the address of the high limit of the storage area to be dumped. If the high limit is not within the allocated storage area, only that storage that is within allocated storage is dumped, and an error message is displayed. If this operand is omitted, address X'0000' is assumed[1].

ID-char specifies any 4 characters, which are used as a dump identifier. If this operand is omitted, blanks are assumed.

[1] If you allow REGION, LOW, and HIGH to default, you will not get a dump (the low address is higher than the high address).

PLIST-2/address/INLINE specifies the address of the $SNAP parameter list. If this operand is omitted, 2 is assumed.

Parameter	Meaning
2	The address is in register 2.
address	Specifies the address of the leftmost byte of the parameter list.
INLINE	The parameter list is generated inline.

Note: The PLIST and V keywords are mutually exclusive. Normally, unless PLIST-INLINE is specified, you use one $SNAP macroinstruction to generate a parameter list (V-DC), and one or more additional $SNAP macroinstructions to dump portions of your program (PLIST-2 or PLIST address).

V-DC/EQU/ALL specifies whether the parameter list labels, DCs, or both, are generated. If this operand is omitted, neither is generated. Do not specify V if you specified PLIST.

Parameter	Meaning
DC	$SNAP initializes the storage area for the parameter list.
EQU	$SNAP generates labels; all other $SNAP operands are ignored.
ALL	$SNAP initializes the storage area for the parameter list and generates labels.

Information Retrieval ($INFO)

$INFO allows access to system information contained in the system communications area or work station local data area which cannot be accessed directly. The macroinstruction performs three separate and distinct functions:

- Generates labels and displacement values for the SVC and the parameter list
- Generates an SVC to retrieve or change specific system information based on the values supplied for $INFO parameters
- Generates a parameter list for the function based on the parameter values supplied for $INFO parameters.

The $INFO macroinstruction must be expanded at least three times to retrieve system information. The first expansion generates the labels supplied in the macroinstruction. This expansion should be placed in the area of your code where you are defining other labels.

To generate the labels, the format is:

```
no label $INFO      no operands
```

The second expansion of the macroinstruction generates the SVC to retrieve or change specific system information. This expansion is placed within your executable code where you want to perform the request.

To generate the SVC, the format is:

```
[name] $INFO      PLIST-2/address
```

PLIST-2/address specifies the address of the leftmost byte of the parameter list. 2 indicates the address is in XR2. The PLIST parameter must be given. If this parameter is omitted, labels are generated again instead of the SVC.

The third expansion of the macroinstruction generates the parameter list, which defines the function desired.

To generate the parameter list, the format is:

```
[name] $INFO  [GET-code] [,PUT-code]
              [,BUFFER-address] [,ID-name]
              [,LEN-number] [,OFFSET-number]
```

GET-code specifies the value to be retrieved from the system and placed in the buffer supplied by the user. If this operand is omitted, UPSI is assumed. A description of each GET function–the number of bytes returned in the buffer and the contents of those bytes–follows:

DATEFRMT returns 1 byte containing the program date format. The character D indicates day-month-year format; M indicates month-day-year format; Y indicates year-month-day format.

PROGDATE returns 3 bytes containing the program date field. This is a six-digit date in year-month-day format.

SDATE returns 3 bytes containing the session date field. This is a six-digit date in year-month-day format.

UPSI returns 1 byte containing the UPSI switch value.

INQUIRY returns 1 byte containing the inquiry switch value. The character Y indicates an inquiry request is pending; N indicates an inquiry request is not pending.

LOCAL returns 1 to 256 bytes of the work station local data area as specified by the LEN and OFFSET operands.

NEP returns 1 byte containing the program attribute byte. The character Y indicates the program is a never-ending program; N indicates the program is not a never-ending program.

MRTMAX returns 1 byte containing the hexadecimal value for the maximum number of requesters allowed.

LINES returns 1 byte containing the hexadecimal value for the number of lines per page.

DATEUNPK returns 6 bytes containing the unpacked program date field in the format defined in the date format field.

PUT-code specifies that the value in the user's buffer is used to update the system communications area or the work station local data area. A description of each PUT function–the number of bytes updated and the contents of those bytes–follows:

UPSI updates the 1-byte UPSI switch with the value in the user's buffer.

PROG1 updates the 8-byte program 1 message member disk address with the value in the user's buffer. The first 3 bytes contain the sector address of the message member; the next 3 bytes are unused; the remaining 2 bytes contain the main storage address of the format 1 for the library in which the message member is located.

PROG2 updates the 8-byte program 2 message member disk address with the value in the user's buffer. The first 3 bytes contain the sector address of the message member; the next 3 bytes are unused; the remaining 2 bytes contain the main storage address of the format 1 for the library in which the message member is located.

USER1 updates the 8-byte user 1 message member disk address with the value in the user's buffer. The first 3 bytes contain the sector address of the message member; the next 3 bytes are unused; the remaining 2 bytes contain the main storage address of the format 1 for the library in which the message member is located.

LOCAL updates 1 to 256 bytes of the work station local data area as specified by the LEN and OFFSET parameters.

BUFFER-address specifies the address of the leftmost byte of the user's buffer where the data is placed for a GET operation or acquired for a PUT operation. If this operand is omitted, address X'FFFF' is assumed.

ID-name specifies the 2-byte logical ID of the terminal used in selecting the job control block. If this operand is omitted, the job control block for the active task is used.

LEN-number specifies a decimal value from 1 to 256, which is used as the length of this local request. Data is counted starting from the offset value specified. If this operand is omitted, 1 is assumed.

OFFSET-number specifies a value from 1 to 256 which is used as the offset for this local request. If this operand is omitted, 1 is assumed.

Generate a Checkpoint Parameter List ($CKEQ)

This macroinstruction generates a parameter list to be used by the $CKPT macroinstruction. $CKPT, which is described in a following paragraph, requests the SSP checkpoint facility to establish a checkpoint in a program. The SSP checkpoint facility records system status and job information at each established checkpoint so that, in the event of a system malfunction, you can restart your program at a checkpoint rather than having to run the entire program again from the beginning.

For a description of considerations and restrictions regarding the SSP checkpoint facility, and for a description of the associated restart facility, see the *Concepts and Design Guide*.

Only one $CKEQ macroinstruction is required in each program that contains one or more $CKPT macroinstructions. The format of the $CKEQ macroinstruction is:

```
[name] $CKEQ [V-DC/EQU/ALL]
             [,LABEL-filelabel]
             [,IMSG-FIRST/ALL]
```

V-DC/EQU/ALL specifies whether the parameter list, labels, or both are to be generated for the checkpoint facility. If this operand is omitted, EQU is assumed.

Parameter	Meaning
DC	Generates the checkpoint parameter list used by the $CKPT macroinstruction.
EQU	Generates the displacement labels for the checkpoint parameter list. If V-EQU is specified or assumed, all other operands for $CKEQ are ignored.
ALL	Generates both the checkpoint parameter list and the corresponding displacement labels.

LABEL-filelabel specifies the label of the file that is to contain the checkpoint records. Checkpoint records contain the information recorded at a checkpoint. Do not use a // FILE statement to define a file for checkpoint records: the file is created dynamically by the checkpoint facility. This operand is required if V-DC or V-ALL is specified.

IMSG-FIRST/ALL specifies whether the checkpoint informational message is to be displayed on the work station display screen only after the first checkpoint is recorded (IMSG-FIRST) or is to be displayed after each checkpoint is recorded (IMSG-ALL). If this operand is omitted, ALL is assumed.

Establish a Checkpoint ($CKPT)

The $CKPT macroinstruction establishes a program checkpoint. Before you issue a $CKPT macroinstruction, you must generate a checkpoint parameter list by issuing the $CKEQ macroinstruction. $CKEQ is described in preceding paragraphs. The $CKPT macroinstruction can be used more than once in a program. However, because only one checkpoint record file is created for a program, the $CKEQ macroinstruction need be issued only once regardless of the number of $CKPTs issued in a program.

For a description of considerations and restrictions regarding the SSP checkpoint facility, see the *Concepts and Design Guide*.

If you will need the data in register 2 later, you should save the contents of register 2 before you issue $CKPT.

The format of the $CKPT macroinstruction is:

```
[name] $CKPT [PLIST-address]
```

PLIST-address specifies the address of the leftmost byte of the checkpoint parameter list that is generated by the $CKEQ macroinstruction. If this operand is omitted, the address is assumed to be in register 2.

Note: Each time you issue $CKPT to establish a checkpoint, you should check the return code provided in the checkpoint parameter list. Check the return code to determine whether or not the system and job information was recorded successfully or the program was restarted successfully. The return code is at displacement $CKCCODE in the checkpoint parameter list. Possible values and their meanings are:

Value	Meaning
$CKCCPNT	Normal checkpoint completion
$CKCCERR	Disk I/O error. If a disk I/O error occurs, retry the checkpoint request, bypass the request, or issue an error message. If a disk I/O error occurs while a checkpoint record is being written to the checkpoint file, alternate requests may be successful because two checkpoint records are maintained in the checkpoint file.
$CKCCIOP	Invalid request. An invalid parameter exists in the parameter list generated by $CKEQ.
$CKCCRES	Normal restart completion. For a description of the restart facility, see the *Concepts and Design Guide*. *Note:* After successful completion of a program restart, the restart facility returns control to the first instruction that follows the last $CKPT executed in the program. Any recovery operations required after a restart, such as restoration of work station displays, should be included in your program so that they are performed upon return from a restart.
$CKCCNOP	No checkpoint record was saved. This return code can be set only when the first checkpoint is requested. If this code is set, one or more of the following may have occurred: • The checkpoint facility detected a condition in which checkpoints cannot be saved (for example, DISP-SHR specified on a // FILE statement). • An explanatory system log device message was displayed. • The operator responded with the 0 option, indicating that the job should be run without saving checkpoints.

Inverse Data Move ($INV)

This macroinstruction generates the code that allows you to do an inverse move on desired data. That is, the bytes of data at the *TO* address are in the opposite order they were in when at the *FROM* address.

The format of the $INV macroinstruction is:

```
[name] $INV    FROM-address,TO-address
               LEN-number
```

FROM-address specifies the rightmost byte of the field where the data is located. This operand can be either a symbolic address or a register displacement address.

TO-address specifies the leftmost byte of the field where the data is to be moved. This operand can be either a symbolic address or a register displacement address.

LEN-number specifies the decimal length (in bytes) of the field to be moved.

Note: If the FROM and TO fields overlap, data will be lost.

End of Job ($EOJ)

The $EOJ macroinstruction generates the linkage required to execute the end-of-job routine.

The format of the $EOJ macroinstruction is:

```
[name] $EOJ    no operands
```

Input/Output Macroinstructions

The input/output support macroinstructions provide access to devices without requiring that you write extensive routines to perform each function. The input/output support macroinstructions are divided into seven groups:

1. General I/O macroinstructions, which are used with all device types:

   $ALOC
   $OPEN
   $CLOS
   $DTFO

2. Printer macroinstructions, which support printer devices:

   $DTFP
   $PUTP

3. Disk macroinstructions, which provide support for and linkage to disk data management:

   $DTFD
   $GETD
   $PUTD

4. Disk sort macroinstructions, which provide an interface to the sort utility (part of the Utilities Program Product, number 5726-UT1) or to the ideographic sort utility (part of the Ideographic Generator/Sort Utilities Program Product, number 5726-IG1):

   $SRT
   $SORT

5. Timer macroinstructions, which provide support for and linkage to the interval timer function:

   $TRB
   $SIT
   $RIT
   $TOD

6. Display station macroinstructions, which support work station devices:

   $DTFW
   $WSIO
   $WIND
   $WSEQ

7. Data communications macroinstructions that support BSC programs. For information about the data communications macroinstructions, see the *Data Communications Reference Manual.*

8. Communications macroinstruction support for the interactive communications feature. For information about the support, see the *Interactive Communications Feature Reference Manual.*

9. Scientific macroinstructions, which provide access to the scientific instruction set. For information about the scientific macroinstructions, see the *Scientific Macroinstructions Reference Manual.*

GENERAL I/O SUPPORT

The general I/O support macroinstructions are used with all devices. $DTFO is used to generate DTF labels, offsets, and fields for each device. The normal execution sequence for the other general I/O support macroinstructions is:

1. $ALOC to allocate the file or device to your program
2. $OPEN to prepare the file or device for use
3. I/O operations and any processing required
4. $CLOS to prepare the file or device for job termination

Allocate Space or Device ($ALOC)

The routines called by the $ALOC macroinstruction allocate all input/output devices and files. These routines check to ensure that:

- The DTF is not open
- The system supports the requested device
- The device requested is either not being used or capable of multiple allocation
- Space is available for a new file
- A FILE statement is given for each disk file

These routines also:

- Match the DTF with the COMM or PRINTER statements given.
- Load the data management task for data communications DTFs.
- Format files allocated with an output access method. Delete-capable direct files are filled with X'FF'; other direct files are filled with blanks; index areas are filled with X'FF'; and data areas of nondirect P or T files are filled with X'00'.
- Sort indexed files requiring keysort unless they are shared or allocated with an access method other than an indexed sequential method.

An allocate request requires that preopen DTFs be supplied as input to the routine. When the allocate request is for a disk, an OCL FILE statement is also required. More than one DTF can be allocated at one time by chaining the DTFs. To chain DTFs, you must enter the address of the next DTF in the DTF you are building. The last DTF in a chain must have X'FFFF' entered in place of the chain address. For a description of the disk, printer, and display station DTFs, see $DTFD, $DTFP, and $DTFW, respectively.

Note that if you will need the data in register 2 later, you should save the contents of that register before issuing $ALOC.

The following output is returned to your program:

- The DTF is prepared as required by $OPEN.
- The contents of register 1 are restored.
- Bit 5 (the sixth bit) of the third attribute byte of the DTF is set on to indicate device allocation.
- The address of the first DTF allocated is returned in register 2.

The format of the $ALOC macroinstruction is:

```
[name] $ALOC [DTF-address]
```

DTF-address specifies the address of the leftmost byte of the first DTF being allocated. If this operand is not entered, the address is assumed to be in register 2.

Prepare a Device or File for Access ($OPEN)

This macroinstruction prepares a file for data transfer. Use the allocate macroinstruction before preparing (opening) the file. Depending on the device, one or more of the following functions are performed for each file opened:

- The DTF is formatted.
- Input/output buffers, index buffers, and IOBs are formatted.
- Buffers are initialized as required.
- Diagnostic tests are performed to ensure that the access method and the file organization are compatible.

Note: A DTF must be closed before it is moved or overlaid; otherwise, unpredictable results via $CLOS will occur. More than one DTF can be opened at one time by chaining the DTFs. To chain DTFs, you must enter the address of the next DTF in the DTF you are building. The last DTF in a chain must have X'FFFF' entered in place of the chain address. See $DTFD, $DTFP, and $DTFW.

Input: The preopen DTF and format-1 label are input to the open routine. Before issuing $OPEN, you must be sure to allocate the device by issuing the $ALOC macroinstruction. Also, if you will need the data in register 2 later, you should save the contents of that register before issuing $OPEN. You must also consider the following when opening a file:

- The record length, key length, and key displacement must be specified correctly.
- For a disk file, the disk access method must be compatible with the organization of the file being opened.
- For a disk file that is also opened by another program level or by an inquiry program, the access methods must be compatible with each other.

Output: The open routine returns control to your program after the requested file is opened. The output consists of:

- The restored contents of register 1
- The updated format-1 labels
- Bit 7 (X '01') in the second attribute byte in the DTF (set on to indicate the file is open)
- The initialized buffers
- The address of the first DTF opened (in register 2)

The format of the $OPEN macroinstruction is:

```
[name] $OPEN [DTF-address]
```

DTF-address specifies the address of the leftmost byte of the first DTF to be opened. If this operand is not entered, the address is assumed to be in register 2.

Prepare a Device or File for Termination ($CLOS)

The $CLOS macroinstruction prepares a device for job termination. $CLOS updates file labels to reflect the current file status. For devices other than disk, only the entries related to the requested functions are restored. If you will need the data in register 2 later, you should save the contents of that register before issuing $CLOS.

Input to $CLOS consists of the opened DTF and the format-1 labels created by the allocate function.

Output from $CLOS is returned to your program when control is returned. The output consists of:

- The restored contents of register 1
- The format-1 label for the disk file (updated to indicate current file status)
- The buffer contents scheduled for disk output and disk update operations (written to disk)
- The data and index (written to disk), and an indicator showing whether key sorting is required at end-of-job for output files and file additions

Notes:

1. If a device or file is to be reused after it is closed, both allocate and open must be issued before I/O operations can be processed.
2. More than one DTF can be closed at one time by chaining the DTFs. To chain DTFs, each DTF to be closed must contain the address of the next DTF in the chain. The last DTF in a chain has X'FFFF' entered in place of the address. See $DTFD, $DTFP, and $DTFW.

The format of the $CLOS macroinstruction is:

```
[name] $CLOS  [DTF-address]
```

DTF-address specifies the address of the leftmost byte of the first DTF to be closed. If this operand is not entered, the address is assumed to be in register 2.

Generate DTF Offsets ($DTFO)

This macroinstruction defines the DTF labels, offsets, field contents, and field lengths for all devices and access methods supported by System/34. To avoid duplicate labels, this macroinstruction should be used only once in each program. For a list of the fields that $DTFO defines, see the DTFs in the *System Data Areas and Diagnostic Aids Handbook*.

The format of the $DTFO macroinstruction is:

```
[name] $DTFO  [DISK-Y/N] [,PRT-Y/N]
              [,BSC-Y/N] [,WS-Y/N]
              [,SNA-Y/N] [,ICRTC-Y/N]
              [,ALL-Y/N] [,FIELD-Y/N]
```

DISK-Y/N specifies whether labels are to be generated for the disk devices. If this operand is omitted, N (no) is assumed.

PRT-Y/N specifies whether labels are to be generated for the printer. If this operand is omitted, N (no) is assumed.

BSC-Y/N specifies whether labels are to be generated for BSC. If this operand is omitted, N (no) is assumed.

WS-Y/N specifies whether labels are to be generated for work station devices. If this operand is omitted, N (no) is assumed.

SNA-Y/N specifies whether labels are to be generated for SNA. If this operand is omitted, N (no) is assumed.

ICRTC-Y/N specifies whether labels are to be generated for SSP-ICF (interactive communications feature) return codes. If this operand is omitted, N (no) is assumed.

ALL-Y/N specifies whether labels are to be generated for all devices supported. If this operand is omitted, N (no) is assumed.

FIELD-Y/N specifies whether to generate the labels which define the contents of the DTF fields. If this operand is omitted, N (no) is assumed.

PRINTER SUPPORT

This section describes the macroinstructions that support the printers. The following functions are provided:

- $DTFP builds a preopen DTF for a printer and assigns its offsets. The DTF provides information to printer data management routines that perform output operations.
- $PUTP builds the interface needed to print data.

Define the File for a Printer ($DTFP)

The DTF provides information needed to allocate, open, and access a printer. This macroinstruction generates the code that builds the printer DTF.

The format of the $DTFP macroinstruction is:

```
[name] $DTFP  RCAD-address,IOAREA-address,
              NAME-filename [,OVFL-number]
              [,PAGE-number] [,UPSI-mask]
              [,HUC-Y/N] [,CHAIN-address]
              [,PRINT-Y/N] [,SKIPB-number]
              [,SPACEB-0/1/2/3]
              [,SKIPA-number]
              [,SPACEA-0/1/2/3]
              [,RECL-number] [,ALIGN-Y/N]
              [,ERROR-Y/N] [,RETURN-Y/N]
```

RCAD-address is a required operand that specifies the address of the leftmost byte of the logical record. The area must be on an 8-byte boundary.

IOAREA-address is a required operand that specifies the address of the leftmost byte of an area in main storage allocated to contain the buffers. The length of this area must be equal to record length (RECL) plus 19.

NAME-filename specifies the name of the print file. This name must be the same as the name specified on the PRINTER OCL statement. This operand is required.

OVFL-number specifies the line on the printer after which the overflow completion code will be returned. If this operand is omitted, it defaults to six lines less than the number specified for the PAGE operand.

PAGE-number specifies the number of lines to print per page. If this operand is omitted, it defaults to the system value for the number of lines per page.

UPSI-mask specifies the settings of the external (// SWITCH statement) indicators used for conditionally opening files. The code must be specified as 8 bits. For example, to set on bits 0, 3, 5, and 7, you would enter UPSI-10010101. When the mask bits that are set to one are also set in the switch, the file is opened. If the DTF is not opened and operations are issued for this DTF, the operations are not performed and you receive a return code of hex'99'.

If this operand is omitted, zeros are assumed.

HUC-Y/N specifies whether to halt if an unprintable character is detected. If N (no) is specified or if this operand is omitted, no halt occurs.

CHAIN-address indicates the address of the next DTF in the chain of DTFs. If there is no DTF chain or if this is the last DTF in a chain, this operand should be omitted (X'FFFF' is then assumed).

PRINT-Y/N specifies whether to perform both a print and a skip or space, or only a skip or space. The default is Y (yes), meaning that a print is performed.

SKIPB-number specifies the line to skip to before the print operation. If this operand is omitted, it defaults to zero, and no skip is performed.

SPACEB-0/1/2/3 specifies the number of lines to space before the print operation. If this operand is not entered or exceeds 3, the default value is zero.

SKIPA-number specifies the line to be skipped to after the print operation. The maximum allowed is 255. If this operand is omitted, it defaults to zero, and no skip is performed.

SPACEA-0/1/2/3 specifies the number of lines to space after the print operation. If this operand is omitted or exceeds 3, it defaults to zero.

Note: If the SKIP or SPACE values exceed the value specified for PAGE, no operation is performed.

RECL-number specifies the length of the line to be printed. If this operand is omitted, the default is 132 positions.

ALIGN-Y/N specifies whether alignment is requested on the first page. If Y (yes) is specified, a halt is issued to the operator after the first data line is printed, allowing the operator to check alignment. If this operand is omitted, N is assumed.

Note: This parameter may be overridden by the ALIGN parameter on the PRINTER OCL statement.

ERROR-Y/N specifies whether an error message should be issued for a permanent error. If N (no) is specified, control is returned to the user program with the completion code set. If this operand is omitted, Y is assumed.

Note: NOT READY conditions on the printer (that is, forms jam, out of forms) are not considered permanent errors.

RETURN-Y/N specifies the options available to the operator after a permanent I/O error message is issued. If Y (yes) is specified, the operator is allowed to take either a 2 option or a 3 option. If the 2 option is taken, control is returned to the user program with the completion code set. If Y (yes) is specified, permanent-error console messages are printed on the system printer. If N (no) is specified, the user is allowed a 3 option only. If this operand is omitted, N is assumed.

Construct a Printer Put Interface ($PUTP)

This macroinstruction generates the interface needed to communicate with printer data management. Before using $PUTP you must provide a DTF for the file (see $DTFP).

If you will need the data in register 2 later, you should save the contents of that register before issuing $PUTP.

The code generated by this macroinstruction gives control to the data management routine. The routine completes execution and returns control to the generated code. If the ERR operand is specified, the generated code checks the completion code for errors and branches to your error routine if errors occurred.

If the OVFL operand is specified, the generated code checks for page overflow and branches to your overflow routine if overflow occurred.

The format of the $PUTP macroinstruction is:

```
[name] $PUTP  [DTF-address] [,PRINT-Y/N]
              [,SKIPB-number]
              [,SPACEB-0/1/2/3]
              [,SKIPA-number]
              [,SPACEA-0/1/2/3]
              [,ERR-address] [,OVFL-address]
```

DTF-address specifies the address of the leftmost byte of the DTF for this file. If this operand is omitted, the address is assumed to be in register 2.

PRINT-Y/N specifies whether to perform both a print and a skip or space, or only a skip or space. If this operand is omitted, the DTF remains unchanged.

SKIPB-number specifies the line to skip to before the print operation. The maximum is 255. If this operand is omitted, the DTF remains unchanged.

SPACEB-0/1/2/3 specifies the number of lines to space before the print operation. If this operand is omitted, the DTF remains unchanged.

SKIPA-number specifies the line to be skipped to after the print operation. The maximum is 255. If this operand is omitted, the DTF remains unchanged.

SPACEA-0/1/2/3 specifies the number of lines to space after the print operation. If this operand is omitted, the DTF remains unchanged.

Note: If the SKIP or SPACE values exceed the value specified for PAGE, no operation is performed.

ERR-address supplies the address in your program that receives control if the controlled cancel option is taken in response to a permanent I/O error. If this operand is omitted, no code is generated to check for the controlled-cancel completion code, and you should check the return code in your program to determine the outcome of the operation.

OVFL-address specifies the address in your program that should receive control if page overflow occurs.

Note: If a PRINT, SKIPB, SPACEB, SKIPA, or SPACEA, operand is specified, the DTF is changed. The DTF is not reset after the operation is complete; the user must reset the DTF if this is required.

DISK DEVICE SUPPORT

This section describes the macroinstructions that support disk devices. The following functions are provided:

- $DTFD builds a preopen DTF for disk GET/PUT operations.
- $GETD builds the interfaces required to get input records from a disk device via a GET or a read.
- $PUTD builds the interfaces required to put output records to a disk device via a PUT or a write.

Define the File for Disk ($DTFD)

This macroinstruction generates the code that builds the disk DTF. The disk DTF provides information needed to allocate, open, and access a file on the disk.

The format of the $DTFD macroinstruction is:

```
[name] $DTFD ACCESS-code,RECL-number
             ,NAME-filename,BLKL-number,
             IOAREA-address [,UPSI-mask]
             [,BUFNO-1/2] [,LIMIT-Y/N]
             [,ORDLD-Y/N] [,CHAIN-address]
             [,RCAD-address] [,KEYL-number]
             [,KDISP-number]
             [,KEYADD-address]
             [,MSTNDX-address]
             [,MSTBYT-number]
             [,CURENT-address]
             [,HIGH-address]
             [,DMADDR-address]
             [,SIAM-Y/N] [,IOBUF-address]
             [,ERROR-Y/N] [,RETURN-Y/N]
```

ACCESS-code specifies the access method used for the file. This operand is required. The codes and their meanings are as follows:

Code	Access Method
CA	Consecutive add
CG	Consecutive input
CO	Consecutive output
CU	Consecutive update
DG	Direct input (decimal RRN)
DO	Direct output (decimal RRN)
DU	Direct update (decimal RRN)
DGA	Direct input addrout (binary RRN)
DOA	Direct output addrout (binary RRN)
DUA	Direct update addrout (binary RRN)
IA	Indexed random add
IO	Indexed output
IS	Indexed sequential input
ISA	Indexed sequential add with input capable
ISU	Indexed sequential update
ISUA	Indexed sequential update and add
IR	Indexed random input
IRA	Indexed random add with input capable
IRU	Indexed random update
IRUA	Indexed random update and add
ISRI	Indexed sequential random input[1]
DUMMY	Dummy open to obtain information about a file
ZPAMA	Sector mode data management add
ZPAMI	Sector mode data management input
ZPAMO	Sector mode data management output

[1] The ISRI (indexed sequential/random input) access method is similar to random-by-key access into an indexed file. A key, which you specify, is retrieved along with the corresponding data record. At this point, you can choose to do one of the following:

a. Request that the next key be read (OP-FGET).
b. Request that the previous key be read (OP-BGET).
c. Provide a new key and request another random read (OP-NGET).
d. Provide a new key and specify reading of the equal key, the next higher key, or the last key, whichever is encountered first (OP-AGET). ISRI then looks for a key (and its corresponding record) that equals the key you provide. If an equal key is not found, ISRI returns the next higher key (and record). If neither an equal nor a higher key is found and you are not accessing a delete-capable file, ISRI returns the last record in the primary portion of the file. The primary portion is the part of the file that reflects the ordered keys in the index. If you are accessing a delete-capable file, the last record in the primary portion may be a deleted record. In this case, ISRI returns the *record not found* completion code. You can specify OP-BGET in response to the *record not found* completion code. OP-BGET causes ISRI to read backward through the file, skipping deleted records until a valid key is found.

Only index entries in the primary index area can be accessed through ISRI, and only input operations are supported by ISRI. Master track index is not supported by ISRI.

The ZPAM access methods (ZPAMA, ZPAMI, ZPAMO) are used to process disk sectors of data rather than records. ZPAM provides an easy way of moving large amounts of data rapidly.

When you use ZPAM, the following operands are required on $DTFD: ACCESS, RECL, NAME, BLKL, IOAREA, DMADDR, and IOBUF.

If you cannot process all of your data with one call to data management ($PUTD), you must use multiple calls. Each $PUTD requires an associated, unique $DTFD. Data management writes one or more disk sectors to main storage depending on the block length (BLKL) you specify. BLKL can vary for each call; however, on all calls except the last one, BLKL must be a multiple of 256 bytes. On the last call, if the number of bytes of data to be passed is less than a multiple of 256 bytes, you must place the exact number of bytes of data to be passed in BLKL. After the last $PUTD, you must issue $CLOS because the last $PUTD does not close the file.

Remember the following when using ZPAM:

- The disk sectors are processed consecutively.
- Input, output, and add functions are supported.
- Only sequential files can be processed.
- The amount of main storage required for IOAREA in $DTFD is 32 bytes.

RECL-number specifies the decimal length of the logical record. The maximum length is 4096. This operand must be specified.

NAME-filename specifies the name of the file. The name must be no more than 8 characters in length, and must be the same as that specified on the FILE OCL statement. This operand must be specified.

BLKL-number specifies the decimal length, in bytes, of the I/O buffer. This operand must be specified.

In general, the length specified must be a multiple of 256 bytes. In particular, for input operations (which are always in locate mode) and the DO and DOA access methods (which involve internal input operations), the following rules apply:

- If the record length is a power of 2, then BLKL must be at least (RECL + 255) rounded down to the next multiple of 256.
- If the record length is not a power of 2, then BLKL must be at least (RECL + 255) rounded up to the next multiple of 256.

IOAREA-address specifies the address of the leftmost byte of an area in main storage allocated to contain all buffers and IOBs for the access method. The area must be on an 8-byte boundary. This operand must be specified.

Disk open divides the I/O area into the required disk input/output blocks (IOBs) and physical buffer areas. If the access to the file is indexed, two IOBs are built; otherwise, one is built. If SIAM-Y is specified, or if a ZPAM access method is used, the address of the physical buffers is provided by the IOBUF parameter.

In the event of very limited user main storage in relation to the size requirements of the physical buffers for disk, it may be advantageous to use the SIAM method to allocate buffers. Through SIAM, the same storage area may be used for a physical buffer for any or all disk files accessed by a program. Data management will then use this area as a physical buffer for every file specified as SIAM. Care should be taken, however, in the use of SIAM since many more I/O operations are likely to occur when SIAM is specified for access to a file. This may hinder the performance of the job.

The amount of main storage required for IOAREA is shown in the following chart:

Sequential and Direct Files

(BLKL * BUFNO) + 39

Indexed Files

BLKL + 335

Sequential and Direct Files (SIAM)

32

Indexed Files (SIAM)

64

Sequential Files (ZPAM)

32

UPSI-mask specifies the settings of the external (// SWITCH statement) indicators used for conditionally opening files. The code must be specified as 8 bits. For example, to test bits 0, 3, 5, and 7, you would enter UPSI-10010101. When the corresponding bits are on in the switch, the file is opened. If the file is not opened and operations are issued for this DTF, the operations are not performed and you receive a return code of hex 99. If this operand is omitted, zeros are assumed.

BUFNO-1/2 allows you to specify either a single or double buffer for the file. You can use a double buffer only with the consecutive access methods. If this operand is omitted, a single buffer is assumed.

LIMIT-Y/N is used only for indexed sequential get and indexed sequential update. It specifies whether the sequential access is within limits. If this operand is omitted, N (no) is assumed.

ORDLD-Y/N specifies whether an ordered load is to be used with the indexed output access method. Use this operand only with the indexed output access method. If this operand is omitted, N (no) is assumed.

CHAIN-address specifies the address of the next DTF in the chain of DTFs. If there is no DTF chain or if this is the last DTF in the chain, this operand should be omitted (X'FFFF' is then assumed).

RCAD-address specifies the address of the leftmost byte of the logical record. When a record is to be written to disk (through output, add, or update), you must provide a logical buffer that contains that record. This allows data management to move the data from the logical buffer to the physical buffer. This type of processing is known as move mode. All put operations are move mode. Note that you must make the DTF field associated with RCAD ($F1WKB field) point to the leftmost byte of the logical record.

After get operations (always locate mode), you must move the obtained record from the physical buffer to a logical buffer if you intend to update the record or if you intend to place the record in another file via another DTF. Note that data management changes the $F1WKB field after a get operation to point to the retrieved record in the physical buffer.

If this operand is omitted, zeros are assumed. You must provide the logical record address before you can request an output operation. This operand is not required for input-only files.

KEYL-number specifies the length of the key field and must be used for all indexed access methods, but no others. If this operand is omitted, a length of 1 is assumed.

KDISP-number is entered for indexed access methods only. It indicates the displacement into the record of the rightmost byte of the key field. The displacement of the first byte in the record is zero, the second byte is one. and so on.

KEYADD-address specifies the following:

- For indexed random access methods, the main storage address of the leftmost byte of the key area. This area must be one key length and cannot contain any X'FF' characters. If the key area is a binary field or generated by a user program, you must ensure that no X'FF' characters appear in the key area.

- For direct access methods, the main storage address of the leftmost byte of the relative record number area. This area must be 10 bytes when using a decimal relative record number, with the relative record number right-adjusted in the rightmost 7 bytes of the area. The leftmost 3 bytes are changed by data management. This area must be 3 bytes when using binary relative record number, with the relative record number right-adjusted in the rightmost 3 bytes of the field. If this operand is omitted, address X'FFFF' is assumed.

MSTNDX-address specifies the address of the leftmost byte of the master track index in main storage. Use this operand for indexed random access and indexed sequential within limits access methods only. If this operand is omitted, address X'FFFF' is assumed. Master track index is not supported when you use the ISRI access method.

To aid the performance of the indexed random and indexed sequential within limits access methods for large files, you may supply data management with space for a master track index. This area will be formatted by open so that the requested key within the primary index area of the indexed file may be located more easily. It is in effect an index into the index area or a second-level index.

MSTBYT-number specifies the number of bytes to be reserved for the master track index. If this operand is omitted, zero is assumed. Use this operand in conjunction with the MSTNDX operand.

There is a minimum and maximum number of bytes that can be reserved. To determine the number of bytes to be reserved, use the following:

Minimum	**Maximum**
MSTBYT = EL*2	MSTBYT = EL*(N+1)

where:

EL = entry length (KEYL (key length) + 3)

N = Number of tracks containing index entries (1 track = 60 sectors)

To determine N, do the following:

1. Use the CATALOG procedure to find the total number of records that the file can contain.

2. Determine the number of keys in each sector by dividing 256 by the entry length. Drop the remainder.

3. Determine the number of sectors in the index by dividing the number of records in the file by the number of keys in each sector (the result of step 2). Round up the result.

4. Determine the number of tracks by dividing the number of sectors (the result of step 3) by 60; if the quotient is not a whole number, round it up to the next whole number.

CURENT-address specifies, for the indexed sequential access method, the address of the leftmost byte of the user's save area for current and last keys. If this operand is omitted, address X'FFFF' is assumed.

HIGH-address specifies the address of the leftmost byte of the user's save area. This save area is two key lengths long, with the low key in the left half and the high key in the right half. Use this in conjunction with indexed sequential processing within limits. If this operand is omitted, address X'FFFF' is assumed.

DMADDR-address specifies where, in the user area of main storage, $OPEN should load data management. The area must be aligned on an 8-byte boundary. This field is used only with the ISRI, ZPAMA, ZPAMI, and ZPAMO access methods.

While much of data management may be used without reserving space for data management programs, you must reserve space if you are accessing disk through ISRI or ZPAM. This area will be initialized at open time. If you are using two DTFs with the same access method (ISRI or ZPAM), you need reserve only one area.

The area reserved for ZPAM data management must be 512 bytes. The area reserved for ISRI data management must be 2048 bytes.

SIAM-Y/N specifies whether SIAM support is used for this DTF. If this operand is omitted, N (no) is assumed.

IOBUF-address specifies the address of the leftmost byte of the user-provided I/O buffer. Use this operand only with SIAM-Y (yes) and ZPAM access methods.

The area required for the buffer is:

ZPAM access methods

Area = BLKL (block length) + 7

SIAM sequential or direct access

Area = BLKL (block length) + 7

SIAM indexed access

Area = BLKL (block length) + 7 or

271, whichever is greater

ERROR-Y/N specifies whether an error message should be issued by IOS for a permanent disk error. If N (no) is specified, control is returned to the user program with the completion code set. If this operand is omitted, N is assumed.

RETURN-Y/N specifies the options allowed to the operator after a permanent disk error message is issued. If Y (yes) is specified, control is returned to the user program with the completion code set and the operator is allowed to take either a 2 option or a 3 option. If N (no) is specified, the operator is allowed a 3 option only. If this operand is omitted, N is assumed.

Construct a Disk Get Interface ($GETD)

The $GETD macroinstruction generates the interface needed to communicate with disk data management when a record is being read from a disk file. Before using $GETD you must provide a DTF for the file (see *$DTFD*). If you will need the data in register 2 later, you should save the contents of that register before issuing $GETD.

Note: Disk data management operates in locate mode for input operations. In locate mode, disk data management locates a record by placing the address of the record in the disk DTF. The address points to the record's location in a physical buffer.

The code generated by this macroinstruction gives control to the data management routine; the routine completes execution and returns control to the generated code. The generated code performs any requested tests on the completion codes returned by data management.

The format of the $GETD macroinstruction is:

```
[name] $GETD [DTF-address] [,INVKEY-address]
             [,OP-code] [,ERR-address]
             [,EOF-address] [,NRF-address]
             [,DIRDRF-address]
```

DTF-address specifies the address of the leftmost byte of the DTF for this file. If this operand is omitted, the address is assumed to be in register 2.

INVKEY-address specifies the address in your program that receives control if an invalid key value is detected. This condition can occur only with indexed random accesses. If the key field is a binary field or generated by a user program, you must ensure that no X'FF' characters appear in the key field.

OP-code may be specified when the access method is ISRI (indexed sequential/random input). The codes are:

NGET	Random get by equal key
AGET	Random get by high/equal/last key
BGET	Get backward (previous) by key
FGET	Get forward (next) by key

ERR-address supplies the address in your program that receives control if the controlled cancel option is taken in response to a permanent I/O error. If this operand is omitted, no code is generated to check for the controlled cancel completion code.

EOF-address specifies the address in your program that receives control when the end of file is detected. If this operand is not supplied, no code is generated to check for the end-of-file condition. You must not use this operand with random or direct access methods.

NRF-address must be used only for random and direct access methods. It specifies the address in your program that receives control if a no-record-found condition occurs.

DIRDRF-address must be used only for the direct access method. It specifies the address in your program that receives control if a deleted record is encountered.

Note: If INVKEY, ERR, EOF, NRF, or DIRDRF is applicable but not specified, you should check the return code in your program to determine the outcome of the operation.

Construct a Disk Put Interface ($PUTD)

The $PUTD macroinstruction generates the interface needed to communicate with disk data management when putting a record to disk or updating or deleting a previously retrieved record. Before using $PUTD you must provide a DTF for the file (see *$DTFD*). If you need the data in register 2 later, you should save the contents of that register before issuing $PUTD.

Note: Disk data management operates in move mode for output operations. In move mode, disk data management moves a record from the logical buffer identified in the disk DTF to a physical buffer.

The code generated by this macroinstruction gives control to the data management routine; the routine completes execution and returns control to the generated code. Completion codes are tested if requested and control is returned to your program.

The format of the $PUTD macroinstruction is:

```
[name] $PUTD [DTF-address] [,INVKEY-address]
             [,ERR-address] [,EOX-address]
             [,DUPREC-address]
             [,SEQERR-address]
             [,KEYERR-address]
             [,INVDRP-address]
             [,DIRNDR-address]
             [,UPDATE-Y/N] [,DELETE-Y/N]
```

DTF-address specifies the address of the leftmost byte of the DTF associated with this file. If this operand is omitted, the address is assumed to be in register 2.

INVKEY-address specifies the address in your program that receives control if an invalid key value is detected. This condition can occur only with indexed random accesses. This field must be one key length and cannot contain any X'FF' characters. If the key field is a binary field or generated by a user program, you must ensure that no X'FF' characters appear in the key field.

ERR-address supplies the address in your program that receives control if the controlled cancel option is taken in response to a permanent I/O error. If this operand is omitted, no code is generated to check for the controlled cancel completion code.

EOX-address supplies the address in your program that receives control when an end of extent is reached during the operation.

DUPREC-address provides the address in your program that receives control when a duplicate record is encountered. Use this operand only with the indexed add access method.

SEQERR-address is the address in your program that receives control in the event of a sequence error during an ordered load of an indexed sequential file.

KEYERR-address specifies the address of your program that receives control when an attempt is made to update a record in an indexed file and the attempt would destroy the record key.

INVDRP-address specifies the address in your program that receives control if an invalid put to a delete-capable file is detected. This condition can occur with all access methods except ZPAM and ISRI. An invalid put is signaled if the record to be added to or updated in the file contains X'FF' in the first byte.

DIRNDR-address must be used only for the direct access method. It specifies the address in your program that receives control if you are doing direct output to a delete-capable file and the current record in the file is not a deleted record (it does not contain X'FF' in the first byte).

UPDATE-Y/N indicates whether an update is to be performed. If this operand is omitted, N is assumed.

DELETE-Y/N indicates whether a delete is to be performed. If this operand is omitted, N (no) is assumed.

Note: If ERR, EOX, DUPREC, SEQERR, INVKEY, or KEYERR is applicable but not specified, you should check the return code in your program to determine the outcome of the requested operation. For a complete list of currently defined return conditions, see *Return Conditions* under *Accessing Records in the File* in Chapter 7.

DISK SORT SUPPORT

Generate a Loadable Sort Parameter List ($SRT)

The $SRT macroinstruction generates the loadable sort parameter list used by the $SORT macroinstruction, which is described in following paragraphs. $SORT requires a parameter list in order to load the sort utility or the ideographic sort utility. The sort utility is part of the Utilities Program Product, number 5726-UT1. The sort utility and the loadable sort parameter list are described in the *Sort Reference Manual*. The ideographic sort utility is part of the Ideographic Generator/Sort Utilities Program Product, number 5726-IG1. The ideographic sort utility and the loadable sort parameter list are described in the *Ideographic Sort Reference Manual*. The maximum size of the parameter list is 2048 bytes, including 125 bytes reserved as a work area for the sort utility or for the ideographic sort utility.

The format of the $SRT macroinstruction is:

```
[name] $SRT   [V-DC/EQU/ALL]
              [,OUTPUT-filename]
              [,SOURCE-source member name]
              [,USERLB-library name]
              [,INPUT1-filename]
              [,INPUT2-filename]
              [,INPUT3-filename]
              [,INPUT4-filename]
              [,INPUT5-filename]
              [,INPUT6-filename]
              [,INPUT7-filename]
              [,INPUT8-filename]
              [,ALTSEQ-Y/N]
              [,KASRT-Y/N]
```

V-DC/EQU/ALL specifies whether the parameter list, labels, or both are to be generated. If this operand is omitted, EQU is assumed.

Parameter	Meaning
DC	Generates the loadable sort parameter list used by the $SORT macroinstruction.
EQU	Generates the displacement labels for the loadable sort parameter list. If V-EQU is specified or assumed, all other operands for $SRT are ignored.
ALL	Generates both the loadable sort parameter list and the corresponding displacement labels.

OUTPUT-filename specifies the name of the file that is to contain the sorted data. If this operand is omitted, blanks are assumed.[1, 2]

SOURCE-source member name specifies the name of the source member that contains the sort sequence specifications. If this operand is omitted, no entry is created for it in the generated parameter list, and the 34-byte sequence specifications must be placed immediately after the generated portion of the loadable sort parameter list. Omit this operand if you want to supply the sequence specifications in the loadable sort parameter list.[1]

USERLB-library name specifies the name of the user library that contains the source member specified in the SOURCE parameter, if any. If this operand is omitted, no entry is created for it in the generated parameter list. Omit this operand if you want to supply the sequence specifications in the loadable sort parameter list.[1]

INPUT1-filename specifies the name of the first, or only, input file to sort. If this operand is omitted, blanks are assumed.[1, 2]

INPUT2-filename specifies the name of the second input file to sort. If this operand is omitted, no entry is created for it in the generated parameter list.[2]

INPUT3-filename specifies the name of the third input file to sort. If this operand is omitted, no entry is created for it in the generated parameter list. Before INPUT3 can be specified, INPUT2 must be specified.[2]

INPUT4-filename specifies the name of the fourth input file to sort. If this operand is omitted, no entry is created for it in the generated parameter list. Before INPUT4 can be specified, INPUT2 and INPUT3 must be specified.[2]

INPUT5-filename specifies the name of the fifth input file to sort. If this operand is omitted, no entry is created for it in the generated parameter list. Before INPUT5 can be specified, INPUT2 through INPUT4 must be specified.[2]

INPUT6-filename specifies the name of the sixth input file to sort. If this operand is omitted, no entry is created for it in the generated parameter list. Before INPUT6 can be specified, INPUT2 through INPUT5 must be specified.[2]

INPUT7-filename specifies the name of the seventh input file to sort. If this operand is omitted, no entry is created for it in the generated parameter list. Before INPUT7 can be specified, INPUT2 through INPUT6 must be specified.[2]

INPUT8-filename specifies the name of the eighth input file to sort. If this operand is omitted, no entry is created for it in the generated parameter list. Before INPUT8 can be specified, INPUT2 through INPUT7 must be specified.[2]

ALTSEQ-Y/N specifies whether an alternate collating sequence table is contained in bytes 1793 through 2048 (the last 256 bytes) of the loadable sort parameter list: Y if yes, N if no. If this operand is omitted, N is assumed. If Y is specified, you must place the alternate collating sequence table in bytes 1793 through 2048 of the loadable sort parameter list.

KASRT-Y/N specifies whether the ideographic sort utility (#KASRT) should be loaded: Y if yes, N if no. If this operand is omitted, N is assumed. If N is specified or assumed, the sort utility (#GSORT) is loaded. The sort utility and the ideographic sort utility each require 14 K-bytes. However, the ideographic sort utility requires 16 K-bytes whenever:

- Ideographic control fields are specified and
- Either SOURCE- is not specified or ALTSEQ-Y is specified in $SRT.

It is your responsibility to ensure that this space is available. For information on how to increase this space, see the *System Support Reference Manual*.

---

[1] Space is always reserved in the generated parameter list for an OUTPUT filename and an INPUT1 filename. If you want to reserve space in the parameter list for other operands, specify names in $SRT for the operands (actual names can then be inserted in the parameter list by your program).

[2] All files named in $SRT must be defined by FILE statements before the $SORT macroinstruction is used. The files must also be closed before $SORT is used. If one or more of the input files named in $SRT are offline multivolume files, a sort work file must be supplied before $SORT is used. The name of the file must be WORK. (For a description of how to code FILE statements, see the *System Support Reference Manual*.)

The following example shows how to build a parameter list to be passed to the loadable sort transient. In this example:

- The input file is named IN and has 100-byte records.
- The output file is named OUT and contains input file records sorted on columns 1 through 10.
- The sort sequence specifications are included in the parameter list, not stored in a source member.
- The alternate collating sequence specified sorts all characters except blanks, uppercase alphabetic characters, and numeric characters to the end of the file.

```
SORTPARM $SRT  V-ALL.INPUT1-IN,OUTPUT-OUT,ALTSEQ-Y
         DC    CL34'HSORTR   ØØ1ØA        XØ1ØØ'   SORT HEADER SPEC
         DC    CL34'FNCØØØ1ØØ1Ø'                   SORT COLS. 1 THRU 1Ø
         DC    CL34'FDCØØØ1Ø1ØØ'                   DATA IS WHOLE RECORD
         DC    CL34'// END'                        END SORT SPEC
         ORG   SORTPARM+X'Ø1ØØ'                    LOCATE ALTSEQ AREA
         DC    64XL1'FF'                           CHARS. BEFORE BLANK
         DC    CL1' '                              BLANK
         DC    128XL1'FF'                          CHARACTERS THRU A
         DC    CL9'ABCDEFGHI'                      A THRU I
         DC    7XL1'FF'                            CHARACTERS THRU J
         DC    CL9'JKLMNOPQR'                      J THRU R
         DC    8XL1'FF'                            CHARACTERS THRU S
         DC    CL8'STUVWXYZ'                       S THRU Z
         DC    6XL1'FF'                            CHARACTERS THRU Ø
         DC    CL1Ø'Ø123456789'                    Ø THRU 9
         DC    6XL1'FF'                            REMAINING CHARACTERS
```

The following operation calls the loadable sort:

```
         $SORT PLIST-SORTPARM
```

Construct a Loadable Sort Interface ($SORT)

The $SORT macroinstruction generates an interface to the sort utility or to the ideographic sort utility. The sort utility is part of the Utilities Program Product, number 5726-UT1. The sort utility is described in the *Sort Reference Manual*. The ideographic sort utility is part of the Ideographic Generator/Sort Utilities Program Product, number 5726-IG1. The ideographic sort utility is described in the *Ideographic Sort Reference Manual*. The interface generated by $SORT permits you to load the sort utility or the ideographic sort utility as though it were part of your program. Before you issue $SORT, you must generate a loadable sort parameter list by issuing the $SRT macroinstruction. $SRT is described in preceding paragraphs.

If you will need the data in register 2 later, you should save the contents of register 2 before you issue $SORT.

The code generated by $SORT gives control to the sort utility or to the ideographic sort utility. After completing the sort, the utility returns control to the instruction that follows the code generated by $SORT. You should check the sort completion indicator to determine whether the sort was successful. The indicator ($SRTCOMP) is at displacement $SRTINDB in the loadable sort parameter list: if $SRTCOMP is off, the sort was successful; if $SRTCOMP is on, the sort was unsuccessful.

$SORT can be issued more than once to perform multiple sorts in a single program. Before you issue $SORT, all files named in $SRT must be defined by FILE statements and the files must be closed. If one or more of the input files named in $SRT are offline multivolume files, a sort work file must be supplied before $SORT is used. The name of the work file must be WORK. (For a description of how to code FILE statements, see the *System Support Reference Manual*.)

The sort utility (#GSORT) and the ideographic sort utility (#KASRT) each require 14 K-bytes. However, the ideographic sort utility requires 16 K-bytes whenever:

- Ideographic control fields are specified and
- Either SOURCE- is not specified or ALTSEQ-Y is specified in SRT.

It is your responsibility to ensure that this space is available. For information on how to increase this space, see the *System Support Reference Manual*.

The format of the $SORT macroinstruction is:

```
[name] $SORT [PLIST-address]
```

PLIST-address specifies the address of the leftmost byte of the loadable sort parameter list that is generated by the $SRT macroinstruction. If this operand is omitted, the address of the loadable sort parameter list is assumed to be in register 2.

TIMER SUPPORT

Generate Timer Request Block ($TRB)

This macroinstruction generates a timer request block (TRB). You must use $TRB if you use $SIT, $RIT, or $TOD in your program.

The format of the $TRB macroinstruction is:

```
[name] $TRB   [V-DC/EQU/ALL]
```

V-DC/EQU/ALL specifies whether the parameter list, labels, or both are generated. If this operand is omitted, DC is assumed.

Parameter	Meaning
DC	Generates the timer request block parameter list used by $RIT, $SIT, and $TOD
EQU	Generates the displacement labels for the timer request block
ALL	Generates the timer request block and the corresponding displacement labels

Set Interval Timer ($SIT)

This macroinstruction sets the interval timer, which causes an interrupt after the specified amount of time. Before issuing $SIT you must place the desired interval in the time field of the timer request block.

The format of the $SIT macroinstruction is:

```
[name] $SIT   [TRB-address]
              [,TYPE-DEC/BIN/TU/TOD]
              [,ITYPE-REAL/WAIT]
```

TRB-address specifies the address of the leftmost byte of the timer request block. If this operand is omitted, the address of the timer request block is assumed to be in register 2.

TYPE-DEC/BIN/TU/TOD specifies the format of the time interval in the timer request block. (You must place the time interval in the time field of the timer request block before issuing $SIT. The time field is at displacement $TRBTIME in the timer request block generated by $TRB.) If this operand is omitted, DEC is assumed. The valid time interval formats are:

DEC: A 6-byte decimal number specifying the hours, minutes, and seconds (HHMMSS) that are to elapse before the timer interrupt.

BIN: A 32-bit binary number specifying the number of seconds that are to elapse before the timer interrupt. The binary value must be right-adjusted in bytes 4-7 of the timer request block time field.

TU: A 32-bit binary number specifying the number of timer units that are to elapse before the timer interrupt. One timer unit is 8.192 milliseconds. The binary value must be right-adjusted in bytes 4-7 of the timer request block time field.

TOD: The actual time of day when the timer interrupt is to occur. The time is a 6-byte decimal number specifying the hour, minute, and second (HHMMSS).

ITYPE-REAL/WAIT specifies the type of interval to be timed. If this operand is omitted, REAL is assumed. The types of time intervals are:

REAL: The timer decreases the time interval continuously for all types of processing.

WAIT: The program issuing the $SIT macroinstruction is placed in a wait state for the specified time interval. When the time expires, control returns to the instruction following the $SIT macroinstruction.

Return Interval Time ($RIT)

This macroinstruction returns the remaining amount of a time interval or cancels an unexpired time interval. The remaining time is returned in the time field, displacement $TRBTIME, of the TRB established by the $TRB macroinstruction. The time interval is set by $SIT and is returned in the format specified in that macroinstruction.

The format of the $RIT macroinstruction is:

```
[name] $RIT    [TRB-address] [,CANCEL-Y/N]
               [,WAIT-Y/N]
```

TRB-address specifies the address of the leftmost byte of the timer request block. If this operand is omitted, the address of the timer request block is assumed to be in register 2.

CANCEL-Y/N specifies whether the remaining time in the interval is to be canceled. If this operand is omitted, N is assumed.

WAIT-Y/N specifies whether the task issuing the $RIT macroinstruction is in a wait state until the time interval expires. If this operand is omitted, N is assumed. This operand is ignored if CANCEL-Y is specified.

Return Time and Date ($TOD)

This macroinstruction returns the time of day and the system date to the program. The time of day is returned in the time field of the timer request block, the system date in the date field. The time and date fields are at displacements $TRBTIME and $TRBDATE, respectively, in the timer request block generated by $TRB. Time and date are returned in the format specified during system configuration.

The format of the $TOD macroinstruction is:

```
[name] $TOD    [TRB-address]
               [,TYPE-DEC/BIN/TU]
```

TRB-address specifies the address of the leftmost byte of the timer request block. If this operand is omitted, the address of the timer request block is assumed to be in register 2.

TYPE-DEC/BIN/TU specifies how the time is to be returned in the timer request block. The valid formats are:

DEC: A 6-byte decimal number indicating the time in hours, minutes and seconds (HHMMSS).

BIN: A 32-bit binary number indicating the time in seconds. The number is right-adjusted in bytes 4-7 of the time field of the timer request block.

TU: A 32-bit binary number indicating the time in timer units. One timer unit is 8.192 milliseconds. The number is right-adjusted in bytes 4-7 of the time field of the timer request block.

If this operand is omitted, DEC is assumed.

DISPLAY STATION SUPPORT

All communication with the display stations or system console is done via work station management (WSM). Work station management consists of two parts: a generator routine and a data management routine. The screen format generator routine (SFGR) builds the library load member that is required when a display station is used as a formatted input/output device. For further information about the screen format generator routine (SFGR), see the *System Support Reference Manual.*

Work station data management (WSDM) provides the interface between the system and the display stations. This section describes the macroinstructions that support display station devices. You build your DTF using the $DTFW macroinstruction. You then use the $WSIO macroinstruction to modify the DTF fields for each operation.

Note: A guide to the concepts of work station data management is provided in the *Concepts and Design Guide.*

Define the File for Display Station ($DTFW)

This macroinstruction generates the code that builds the display station DTF. The display station DTF provides information needed to allocate, open, and access a display station file.

Note: For a description of how to code $DTFW for the interactive communications feature, see the *Interactive Communications Feature Reference Manual.*

The format of the $DTFW macroinstruction is:

```
[name] $DTFW [DEV-code]
             [,UPSI-mask] [,CHAIN-address]
             [,OUTLEN-number]
             [,INDEXA-address] [,RESET-Y/N]
             [,NUMFMT-number]
             [,ROLINE-number]
             [,STRTLN-number]
             [,ENDLN-number]
             [,VARLIN-number]
             [,INDA-address]
             [,MEMBER-name]
             [,INLEN-number]
             [,TERMID-name] [,PRNT-Y/N]
             [,ROLL-Y/N] [,CLEAR-Y/N]
             [,RECBKS-Y/N] [,HELP-Y/N]
             [,FKDATA-Y/N]
             [,TIDTAB-address]
             [,ENTLEN-number]
             [,TNUM-number]
             [,F1ADDR-address]
             [,RPGEXT-address]
             [,HALTS-Y/N]
```

DEV-code specifies the file type for which this DTF is to be used. If this operand is omitted, display station (WSTN) is assumed. The codes and their meanings are as follows:

Code	File Type
CONS	RPG console
KBD	RPG keyboard
CRT	RPG display screen
WSTN	Display station

UPSI-mask specifies the setting of the external (// SWITCH statement) indicators used for conditionally opening files. The code must be specified as 8 bits. For example, to test bits 0, 3, 5, and 7, you would enter UPSI-10010101. When the corresponding bits are on in the switch, the file is opened. If the file is not opened and operations are issued for this DTF, the operations are not performed and you receive a return code of hex 99. If this operand is omitted, zeros are assumed, and the file will be unconditionally opened.

CHAIN-address specifies the address of the next DTF in the chain of DTFs. If there is no DTF chain or if this is the last DTF in the chain, this operand should be omitted (X'FFFF' is then assumed).

OUTLEN-number is only required for OPMODs of ERROR and UNF, or OPCs of PUT, PTG, PNW, and PTI. If the operation is ERROR, the OUTLEN value must be between 1 and 78, and it represents the amount of data written from the logical record area to the error line at the display station. If the operation is UNF, the OUTLEN value must be between 2 and 4096, and it represents the amount of data sent from the user's logical record to the display station. If the operation is a PUT, PTG, PNW, or PTI, then OUTLEN represents the maximum amount of data that can be written from the logical record area to the output fields in the display screen format. The OUTLEN value must be at least as large as the sum of the lengths of all program output fields. A program output field is as field where either constant data was not supplied in locations 57 through 79 of the $SFGR field definition specification, or where an indicator was specified in locations 23 and 24 of the $SFGR field definition specification. If this operand is omitted, an OUTLEN value of X'0000' is assumed. After a successful input operation, the actual length of data returned is in this field; therefore, OUTLEN should be respecified after every input operation.

Note: If the operation is an unformatted PUT to a display station that has ideographic support and if GAIJI-ON is specified on the WORKSTN OCL statement, OUTLEN should not be greater than the display station buffer size (the minimum display station buffer size is 2048 bytes). If the execution time output data from the user's logical record area also contains MIC data, the user must reserve 6 bytes to contain the 4-character digit and the 2-character message member identifier. This 6-byte length must be included in the total OUTLEN value.

INDEXA-address specifies the symbolic address of the leftmost byte of the area in which display station OPEN will build the display screen format index. This area must be 16 bytes for each format in the load member to be opened. All open display station DTFs must use the same display screen format index area. If this operand is omitted, address X'0000' is assumed.

Note: To open more than one format load member for a display station at the same time, you can chain multiple DTFs for the display station.

RESET-Y/N specifies whether to reset the active format index address. If Y is specified, a new format index is built at the address specified by INDEXA. If N is specified and there is an active format index, INDEXA must equal the address of the active format index–formats can be added to the index during open and duplicate entries result in a halt. If N is specified and there is no active format index, a format index is built at the address specified by INDEXA. If this parameter is omitted, N is assumed.

NUMFMT-number specifies in decimal the maximum number of display screen formats for which an index is to be built. The maximum allowed is 255. If this operand is omitted, up to 32 new format index entries can be built.

ROLINE-number specifies in decimal the number of lines to roll the displayed data on a roll operation. The maximum number is equal to the display screen size. For example: If the display screen size is 24 lines, the maximum number that can be entered here is 24. If this operand is omitted, 01 is assumed.

STRTLN-number specifies in decimal the first line of the roll area on a roll operation. The maximum number is equal to the display screen size minus 1. For example: If the display screen size is 24 lines, the maximum number that can be entered here is 23. If this operand is omitted, 01 is assumed.

ENDLN-number specifies in decimal the last line of the roll area on a roll operation. The minimum number is 02. The maximum number is equal to the display screen size. For example: If the display screen size is 24 lines, the maximum number that can be entered here is 24. If this operand is omitted, 24 is assumed.

VARLIN-number specifies in decimal the actual start line number if a variable start line number was specified to SFGR for the format for this request. The maximum number is equal to the display screen size. For example: If the display screen size is 24 lines, the maximum number that can be entered here is 24. If this operand is omitted, 01 is assumed.

INDA-address specifies the symbolic address of the leftmost byte of the override indicator area, if override indicators were specified at SFGR time for this format. The indicator area must not start at location X'0000' because WSDM ignores all indicators at address X'0000' and they are assumed off. If this operand is omitted, address X'0000' is assumed, and work station data management ignores all indicators.

MEMBER-name specifies the name of the SFGR load member containing all the formats to be opened. If this operand is omitted, blanks are assumed, and no formats are opened.

INLEN-number specifies in decimal the size of the user's input buffer; that is, the maximum amount of input data that the application program is prepared to receive. This number must not be greater than 65535. If this operand is omitted, zero is assumed, and no data is transferred.

Note: If the operation being performed is an unformatted PUT, this value must equal the total length of all input fields defined on the screen.

TERMID-name specifies the symbolic name of the display station. This is the two-character ID, which the user assigned via system configuration or the SYMID parameter on the // WORKSTN statement, that represents the display station to which the request is directed. If this operand is omitted, blanks are assumed, and for an SRT program the requesting display station is assumed; for MRT programs a halt is issued unless the operation is an ACI (accept), INQ (status inquiry) or GTA (get attributes) operation.

The parameters PRNT, ROLL, CLEAR, RECBKS, and HELP are the function-control-key mask specifications. The function-control-key mask supplied to WSDM via the DTF is used in conjunction with the function-control-key mask specified in the SFGR source specifications. If a specific function control key is disabled in either the work station DTF or the SFGR specifications, the key becomes disabled on the keyboard. A function control key must be enabled in both cases to be enabled to the application program.

PRNT-Y/N specifies whether your program will process the Print key. If Y (yes) is specified, the print key indicator is placed in the AID byte field of your program DTF when the operator presses the Print key. If N (no) is specified, the system attempts to print the work station ID, the user sign-on ID, the system date, the time, and the current display on the display station's associated printer. If this operand is omitted, N (no) is assumed.

ROLL-Y/N specifies whether your program will process the Roll↑ (Roll Up) and Roll↓ (Roll Down) keys. If Y (yes) is specified, the roll key indicator is placed in the AID byte of your program DTF when the operator presses a Roll key, and data is returned as if the Enter/Rec Adv key was pressed. If N (no) is specified, an error message is displayed to the operator when the operator presses the Roll key. If this operand is omitted, N (no) is assumed.

CLEAR-Y/N specifies whether your program is able to process the Clear key. If Y (yes) is specified, the clear key indicator is placed in the AID byte field of your program DTF when the operator presses the Clear key. If N (no) is specified, an error message is displayed to the operator when the operator presses the Clear key. If this operand is omitted, N (no) is assumed.

RECBKS-Y/N specifies whether your program is able to process record backspace (that is, the Home key when the cursor is in the home position). If Y (yes) is specified, the record backspace key indicator is placed in the AID byte field of your program DTF when the operator presses the Home key. If N (no) is specified, an error message is displayed to the operator when the operator presses the Home key. If this operand is omitted, N (no) is assumed.

HELP-Y/N specifies whether your program is able to process the Help key. If Y (yes) is specified, the help indicator is placed in the AID byte field of your program DTF when the operator presses the Help key. If N (no) is specified, the command is not issued and an error message is displayed to the operator when the operator presses the Help key. If this operand is omitted, N (no) is assumed.

FKDATA-Y/N specifies whether input data is to be returned along with a function control key indicator for all enabled function control keys. If Y (yes) is specified, the appropriate function control key indicator is placed in the AID byte field of your program DTF when the operator presses an enabled function control key, and input data is returned regardless of whether the operator has modified any of the fields. This does not apply to remote work stations; see Note 2. If N (no) is specified, the appropriate function control key indicator is placed in the AID byte field of your program DTF when the operator presses an enabled function control key, but no input data is returned. If this operand is omitted, N (no) is assumed.

Notes:

1. The FKDATA parameter has no effect on the operation of the Roll↑ (Roll Up) and Roll↓ (Roll Down) function control keys. These keys always operate as specified by the ROLL parameter.
2. You must use the FKDATA parameter with caution when you are programming for a remote work station. Your job could permanently halt if there are no modified input fields on the screen of the remote work station when a function key is pressed while the FKDATA parameter is active.

TIDTAB-address specifies the address of a work station ID table. Programs that support multiple display stations typically maintain a table of display station IDs and associated status indicators. By specifying the TIDTAB, ENTLEN, and TNUM parameters, you reserve an area for the ID table. Open places the ID of the display station that requests the program in the first 2 bytes of the first entry of the table, and sets the first bit in the third byte on. Open also places the SYMID value from each // WORKSTN statement into other entries in the table. The IDs are placed in the first 2 bytes of the entries. If REQD-YES is specified in a WORKSTN statement, open sets on the first bit of the third byte in the corresponding table entry. The ID table must be large enough to contain an ID for each display station supported by the program plus additional entries up to the program's MRTMAX value (MRTMAX is specified in a // COMPILE statement and can be overriden by a // ATTR statement). The entire table must be initialized to 00s before open is called. After open is complete, the user program must maintain the table. If this operand is omitted, address X'0000' is assumed, and no table is built. (For a description of ATTR, COMPILE, and WORKSTN statements, see the *System Support Reference Manual.*)

ENTLEN-number specifies in decimal the length of each entry in the display station ID table TIDTAB. The maximum allowed is 255. If TIDTAB was specified, the minimum ENTLEN is 3: two bytes for an ID and a third byte for status indicators.

TNUM-number specifies in decimal the total number of TIDTAB table entries. The total space allocated for the table is assumed to be the product of ENTLEN and TNUM. The maximum TNUM allowed is 255. If this operand is omitted, 01 is assumed.

F1ADDR-address specifies the address of the leftmost byte of the library format 1 used to find the load member. If this operand is omitted, work station data management scans the user library (first requester's library if an MRT program), then the system library for the load member.

RPGEXT-address specifies the address of an RPG extension to the work station DTF. If this parameter is omitted, address X'0000' is assumed. This parameter is not used by work station data management.

HALTS-Y/N is valid only if this DTF is to be used to communicate with the interactive communications feature, which is described in the *Interactive Communications Feature Reference Manual.* The parameter specifies whether interactive communications data management should halt for permanent communications errors: Y if yes, N if no. If this operand is omitted, N (no) is assumed.

Construct a Display Station Input/Output Interface ($WSIO)

This macroinstruction builds the executable code to modify a display station DTF according to the parameters specified, then issues a call to work station data management to execute the specified operation. Before using $WSIO you must provide a DTF for the file (see $DTFW) and establish the offsets for the DTF (see $DTFO). If you will need the data in registers 1 and 2 later, you should save the contents of those registers before issuing $WSIO.

After each $WSIO macroinstruction, you should check the return code. The return codes are defined in the $DTFO macroinstruction with WS-Y and FIELD-Y. Return codes from $WSIO are described under *Display Station Data Management Considerations* in Chapter 7.

Note: For a description of how to code $WSIO for the interactive communications feature, see the *Interactive Communications Feature Reference Manual.*

```
[name] $WSIO [DTF-address] [,OPMOD-code]
             [,OPC-code] [,OUTLEN-number]
             [,INLEN-number]
             [,RCAD-address] [,ROLDIR-U/D]
             [,RLCLER-Y/N]
             [,ROLINE-number]
             [,STRTLN-number]
             [,ENDLN-number]
             [,VARLIN-number]
             [,INDA-address]
             [,FORMAT-name]
             [,TERMID-name] [,PRNT-Y/N]
             [,ROLL-Y/N] [,CLEAR-Y/N]
             [,RECBKS-Y/N] [,HELP-Y/N]
             [,FKDATA-Y/N] [,PID-id]
             [,PL@-address]
```

DTF-address specifies the address of the leftmost byte of the display station DTF to be modified. If this operand is omitted, the address is assumed to be in register 2.

OPMOD-code specifies the operation code modifier to be generated if desired. The codes and their meanings are as follows:

Code	Meaning
ERROR	PUT for displaying information on the error line.
OVR	PUT for displaying only override fields and attributes. (If an override indicator was specified on the SFGR S specification, this value is not required.)
ROLL	Roll the display with the specified operation.
UNF	The data does not need formatting by WSDM.
PRINT	Print the displayed data on the printer specified in the PID parameter.
PRUF	PUT for read under format.
FMH	This code is for use with the interactive communications feature, which is described in the *Interactive Communications Feature Reference Manual*. The code indicates that a function management header precedes the data associated with an evoke operation. The code is valid only for evoke operations for the SNUF (SNA upline facility) subsystem.
ZERO	Clear any previous OPMOD specification.

Notes:

1. An OPC of PUT, PTG, PNW, or PTI must also be specified for OPMOD values of OVR, UNF, or PRUF.
2. The OPMOD keyword can be coded as OPM.

OPC-code specifies the operation requested of WSDM. The codes and their meanings are as follows (codes unique to the interactive communications feature are described in the *Interactive Communications Feature Reference Manual*):

ZERO: Sets the operation code field to X'00'. This code is used with operation code modifiers for which you do not want a WSDM operation code. For example, if you wanted to roll or print displayed data without requesting any other work station operation in the call to WSDM, you could use the ZERO operation code with the modifier ROLL or PRINT.

GET: Receive data from the display station specified by the TERMID parameter. Control is returned to your program when the data is available in the user record area. This operation ignores the OPMOD value.

PUT: Send data to the display station specified by the TERMID parameter. Control is returned to your program when data transfer is complete.

PTG: A combination of a put-no-wait (PNW) operation to the display station specified by the TERMID parameter, followed by a GET request to the same display station. Control is returned to your program when the data resulting from the GET operation is available in the user record area.

INV: Enable the display station, specified by the TERMID parameter, to send data to the system. The data entered by the display station operator is presented to your program in response to a subsequent accept input (ACI) operation. Control returns as soon as the invite input (INV) is scheduled.

PNW: Send data to the display station specified by the TERMID parameter. Control is returned to your program when the operation is scheduled, and the program's DTF, record area, and indicators are available for reuse. If a second put-no-wait (PNW) is issued to the same display station, the first PUT must be complete before the second operation is queued. The main difference between a PUT and PNW is the return code. On a PUT, the return code reflects the status of the entire PUT operation, while on a PNW, the return code reflects only the scheduling of the operation.

PTI: A combination of a put-no-wait (PNW) and an invite input (INV) to the same display station. Control is returned to your program when the invite input request is scheduled.

ACI: This operation is not to a specific display station. It requests data from any display station that responded to a previous invite input operation. For example, suppose your program issues three invite input operations to display stations A, B, and C. The program could now issue an accept input request, and be presented with data from any display station (A, B, or C) that responds with a data transmission. The ID of the display station that sent the data is returned at displacement $WSNAME in the DTF. The accept input operation is also used as the first request from a program to receive program data passed from the invoking procedure. This operation ignores the OPMOD value.

ACQ: Allocate the display station specified by the TERMID parameter to this program. This operation ignores the OPMOD value.

REL: Release the display station specified by the TERMID parameter from this program. This operation ignores the OPMOD value.

GTA: Get the attributes of the display station specified by the TERMID parameter, and place them in the program's record area. This operation ignores the OPMOD value.

Following a get attribute operation, the program's record area appears as follows:

Byte 0	Device type.
C'D'	Display type.
C'N'	Non-display type.
	All remaining letters are reserved.
Byte 1	Display size.
C'1'	1920-character display.
C'2'	960-character display.
Byte 2	Type of attachment.
C'L'	Local attachment.
C'R'	Remote attachment.
Byte 3	Online/offline status.
C'0'	Device is online.
C'F'	Device is offline.
Byte 4	Allocation status of device.
C'A'	Device allocated to requester.
C'E'	Device allocated to other user.
C'V'	Not allocated but available.
C'N'	Not allocated, not available.
C'U'	Device unknown to system.
Byte 5	Invite status of device.
C'Y'	Device is invited.
C'N'	Device not invited.
Byte 6	If invited, completion status.
C'Y'	Invite completed.
C'N'	Invite not completed.
Byte 7	Inquiry status.
C'Y'	Device in inquiry.
C'N'	Device not in inquiry.

EGTA: Get the attributes of the ideographic display station specified by the TERMID parameter, and place them in the program's record area. This operation ignores the OPMOD value. Following an extended get attribute operation, the program's record area appears as follows:

Bytes 0-7	Same as for GTA operation.
Byte 8	Display type.
C'A'	Alphanumeric/Katakana type.
C'I'	Ideographic type.
Byte 9	Keyboard type.
C'A'	Alphanumeric/Katakana type.
C'I'	Ideographic type.
Byte 10	Sign-on type.
C'A'	Alphanumeric/Katakana type.
C'I'	Ideographic type.
Bytes 11-15	Reserved. X'00' will be returned.

STI: Cancel a previously issued invite input request to the display station specified by the TERMID parameter. If the stop invite fails (operator already pressed the Enter/Rec Adv key), your program will be informed via a return code, and the data will remain at the display station, available for a subsequent request. However, if an output request is issued to the display station, the input data is destroyed.

Note: A stop invite is not required to override an existing invite input. However, if input is already available, the input data is lost.

RES: Resets the keyboard of the display station specified by the TERMID parameter without requesting a format. This allows an application to ignore keys that are not supported.

RTG: Performs a keyboard reset (RES) followed by a GET.

RTI: Performs a keyboard reset (RES) followed by an invite input (INV).

ERS: Erases all modified input capable fields that are currently defined on the display of the display station specified by the TERMID parameter. This operation locks the keyboard and repositions the cursor to the first input field. For a detailed explanation of how erase input fields works, see the erase input fields entry (columns 31 and 32) under the $SFGR–Screen Format Generator Utility Program in the *System Support Reference Manual*.

ETG: Performs an erase input fields (ERS) followed by a GET.

ETI: Performs an erase input fields (ERS) followed by an invite input (INV).

CLR: Clears the entire display screen of the display station specified by the TERMID parameter, including attribute bytes. This operation also destroys any existing field definitions pertaining to that specific display station.

INQ: Determines the invite status of the display stations associated with this program. This operation returns a 2-byte return code in index register 2. In the high-order byte, X'00' means no invites outstanding; X'10' means at least one invite outstanding; X'30' means at least one outstanding invite, at least one of which is completed. In the low order byte, X'00' means stop system is not in effect; X'02' means stop system is in effect. This operation has no associated DTF. Register 2 need not contain a DTF address. Register 1 contents are preserved. If this operation code is specified, all other specified parameters are ignored.

SIQ: Determines the invite status of the display stations associated with this program. This operation performs a function similar to INQ, except SIQ utilizes the DTF to issue the operation and return the data. Two 1-byte return codes are returned in the DTF as a result of this operation. In the DTF at displacement $WSRSIQ, hex 00 means no invites outstanding; hex 30 means at least one outstanding invite, at least one of which is completed. In the DTF at displacement $WSRTC, hex 00 means stop system is not in effect; hex 02 means stop system is in effect. If this operation code is specified, any specified operation code modifier is ignored, and the operation code modifier field in the DTF is cleared to hex 00.

OUTLEN-number is only required for OPMODs of ERROR and UNF, or OPCs of PUT, PTG, PNW, and PTI. If the operation is ERROR, the OUTLEN value must be between 1 and 79 and it represents the amount of data written from the logical record area to the error line at the display station. If the operation is UNF, the OUTLEN value must be between 2 and 4096 and it represents the amount of data sent from the user's logical record to the display station. If the operation is a PUT, PTG, PNW, or PTI, then OUTLEN represents the maximum amount of data that can be written from the logical record area to the output fields in the display screen format. The OUTLEN value must be at least as large as the sum of the lengths of all program output fields. A program output field is a field where constant data was not supplied in locations 57 through 79 of the $SFGR field definition specification, or where an indicator was specified in locations 23 or 24 of the $SFGR field definition specification. If this operand is omitted, the DTF value remains unchanged. After a successful input operation, the actual length of data returned is in this field; therefore, OUTLEN should be respecified after every input operation.

Note: If the operation is an unformatted PUT to a display station that has ideographic support and if GAIJI-ON is specified on the WORKSTN OCL statement, OUTLEN should not be greater than the display station buffer size (the minimum display station buffer size is 2048 bytes). If the execution time output data from the user's logical record area also contains MIC data, the user must reserve 6 bytes to contain the 4-character digit and the 2-character message member identifier. This 6-byte length must be included in the total OUTLEN value.

INLEN-number specifies in decimal the size of the user's input buffer, that is, the maximum amount of input data that your program is prepared to receive. This number must not be greater than 65535. If this operand is omitted, the DTF remains unchanged. The INLEN and PID parameters use the same field in the DTF; therefore, INLEN must be specified after each operation that specified a PID.

Note: If the operation being performed is an unformatted PUT, this value must equal the total length of all input fields defined on the screen.

RCAD-address specifies the symbolic address of the leftmost byte of the logical record area. This operand must be specified in the first $WSIO you issue in your program to establish the record address. Then, if this operand is subsequently omitted, the DTF remains unchanged.

Note: If the operation being performed involves GET or ACI or UNF, the record area must be on an 8-byte boundary.

ROLDIR-U/D specifies the direction to roll the display when requested. This operand must be specified in the first $WSIO you issue with a roll operation. Then, if this operand is subsequently omitted, the DTF remains unchanged.

RLCLER-Y/N specifies whether the lines vacated by a roll operation should be cleared. This operation must be specified in the first $WSIO you issue with a roll operation. Then, if this operand is subsequently omitted, the DTF remains unchanged.

ROLINE-number specifies in decimal the number of lines to roll the data being displayed on a roll operation. The maximum number is equal to the display screen size. For example: If the display screen size is 24 lines, the maximum number that can be entered here is 24. If this operand is omitted, the DTF remains unchanged.

STRTLN-number specifies in decimal the first line of the roll area on a roll operation. The maximum number is equal to the display screen size minus 1. For example: If the display screen size is 24 lines, the maximum number that can be entered here is 23. If this operand is omitted, the DTF remains unchanged.

ENDLN-number specifies in decimal the last line of the roll area on a roll operation. The minimum number is 02. The maximum number is equal to the display screen size. For example: If the display screen size is 24 lines, the maximum number that can be entered here is 24. If this operand is omitted, the DTF remains unchanged.

VARLIN-number specifies in decimal the actual start line number if a variable start line number was specified to SFGR for the format for this request. The maximum number is equal to the display screen size. For example: If the display screen size is 24 lines, the maximum number that can be entered here is 24. If this operand is omitted, the DTF remains unchanged.

INDA-address specifies the symbolic address of the leftmost byte of the override indicator area, if override indicators were specified at SFGR time for this format. The indicator area must not start at location X'0000' because WSDM ignores all indicators at address X'0000', and they are assumed off. If this operand is omitted, the DTF remains unchanged.

FORMAT-name specifies the name of the display screen format to be used for this operation. This operand is required only for formatted PUT operations. If this operand is omitted, the DTF remains unchanged.

TERMID-name specifies the symbolic name of the display station. This is the 2-character ID, which the user assigned via system configuration or the SYMID parameter on the // WORKSTN statement, that represents the display station to which the request is directed. If this operand is omitted, the DTF remains unchanged.

PRNT-Y/N specifies whether your program is able to process the Print key. If Y (yes) is specified, the print key indicator is placed in the AID byte field of your program DTF when the operator presses the Print key. If N (no) is specified, the system attempts to print the following on the display station's associated printer[1]: the work station ID, the user sign-on ID, the system date, and the time (all on one line, enclosed within a box of asterisks), followed by four blank lines, followed by the current display.

ROLL-Y/N specifies whether your program is able to process the Roll↑ (Roll Up) and Roll↓ (Roll Down) keys. If Y (yes) is specified, the roll key indicator is placed in the AID byte field of your program DTF when the operator presses a roll key and data is returned as if the Enter/Rec Adv key was pressed. If N (no) is specified, an error message is displayed to the operator when the operator presses the roll key.[1]

CLEAR-Y/N specifies whether your program is able to process the Clear key. If Y (yes) is specified, the clear key indicator is placed in the AID byte field of your program DTF when the operator presses the Clear key. If N (no) is specified, an error message is displayed to the operator when the operator presses the Clear key.[1]

RECBKS-Y/N specifies whether your program is able to process the record backspace (that is, the Home key when the cursor is in the home position). If Y (yes) is specified, the record backspace indicator is placed in the AID byte field of your program DTF when the operator presses the Home key. If N (no) is specified, an error message is displayed to the operator when the operator presses the Home key.[1]

HELP-Y/N specifies whether your program is able to process the Help key. If Y (yes) is specified, the help key indicator is placed in the AID byte of your program DTF when the operator presses the Help key. If N (no) is specified, an error message is displayed to the operator when the operator presses the Help key.[1]

FKDATA-Y/N specifies whether input data is to be returned along with a function control key indicator for all enabled function control keys. If Y (yes) is specified, the appropriate function control key indicator is placed in the AID byte field of your program DTF when the operator presses an enabled function control key, and input data is returned regardless of whether the operator has modified any of the fields. This does not apply to remote work stations; see Note 2. If N (no) is specified, the appropriate function control key indicator is placed in the AID byte field of your program DTF when the operator presses an enabled function control key, but no input data is returned.[1]

Notes:

1. The FKDATA parameter has no effect on the operation of the Roll↑ (Roll Up) and Roll↓ (Roll Down) function control keys. These keys always operate as specified by the ROLL parameter.
2. You must use the FKDATA parameter with caution when you are programming for a remote work station. Your job could permanently halt if there are no modified input fields on the screen of the remote work station when a function key is pressed while the FKDATA parameter is active.

PID specifies the ID of the desired printer on a print request. Allowable values are:

SYSTEM–the system printer
WSTN–the ID of the printer associated within the display station specified by the TERMID parameter
XX–the 2-character ID of the desired printer

If this operand is omitted, the DTF remains unchanged. The INLEN and PID parameters use the same field in the DTF; therefore, PID must be specified after each input operation.

---

[1] *Use of PRNT, ROLL, CLEAR, RECBKS, HELP, and FKDATA* is discussed here. The parameters PRNT, ROLL, CLEAR, RECBKS, and HELP are the function-control-key mask specifications. The function-control-key mask supplied to WSDM via the DTF is used in conjunction with the function-control-key mask specified in the SFGR source specifications. If a specific function control key is disabled in either the work station DTF or the SFGR specifications, the key becomes disabled on the keyboard. A function control key must be enabled in both cases to be enabled to the application program. If any of these parameters or the FKDATA parameter is specified, N is assumed for the unspecified parameters. If none are specified, the DTF remains unchanged.

PL@-address is for use with the interactive communications feature. The parameter specifies the address of an associated evoke parameter list, which is generated by the $EVOK macroinstruction. $EVOK is described in the *Interactive Communications Feature Reference Manual*. This operand must be specified for the first evoke operation and remains unchanged if not specified thereafter.

Generate Override Indicators for Display Station ($WIND)

This macroinstruction generates a table of override indicators and offsets for PUT and PUT overrides used by work station data management.

The format of the $WIND macroinstruction is:

[name] $WIND [MAXIND-number]

MAXIND-number specifies in decimal the highest number used by SFGR as an override indicator for your program. If this operand is omitted, 99 is assumed.

Generate Labels for Display Station ($WSEQ)

This macroinstruction generates labels and offsets to reference certain work station device dependent values, such as attention identification (AID) bytes, and bit representations for the display screen attribute bytes and write control characters.

The format of the $WSEQ macroinstruction is:

[name] $WSEQ no operands.

Chapter 7. Programming Considerations

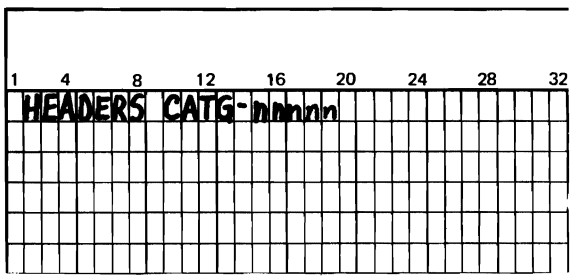
ASSEMBLER CONTROL STATEMENTS

Two control statements are used: the HEADERS statement and the OPTIONS statement. A total of 45 of these control statements may be used, in any order. Each statement is limited to six operands. All control statements must appear before any assembler source statements.

HEADERS Statement

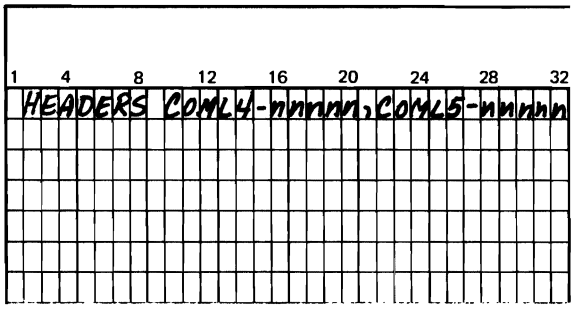
The HEADERS control statement specifies control information other than output control information to the assembler. The programmer may specify a category level for the object module through the CATG operand, or the length of the control section for any subtype 4 or 5 EXTRNs in the assembler through the COML4 and COML5 operands. For an explanation of category levels and subtype 4 and 5 EXTRNs, see the *System/34 Overlay Linkage Editor Reference Manual.*

The format of the HEADERS statement with the CATG operand is:



nnnnn is a 1- to 5-character decimal string whose value must be less than 256. If more than one CATG operand appears in the assembler control statements, the value of the last valid operand is used for the module category level. The module category level is placed in the module ESL record. The HEADERS keyword must start in column 2 or greater; the preceding column must be blank; and there must be one or more blanks between keywords. Blanks are not allowed between selected options.

The format of the HEADERS statement with the COML4 and COML5 operands is:

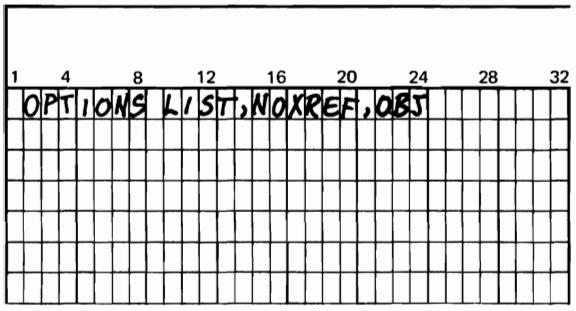


nnnnn is a 1- to 5-character decimal string whose value must be less than 65536. If more than one COML4 or COML5 operand is present in the assembler control statements, the length in the last valid operand is used for the appropriate subtype control section length. The lengths specified are placed in the ESL records for the subtype 4 or 5 EXTRNs. The HEADERS keyword must start in column 2 or greater; the preceding column must be blank; and there must be one or more blanks between keywords. Blanks are not allowed between selected options.

OPTIONS Statement

An OPTIONS statement is for assembler control options. All OPTIONS statements must precede the source file. The user may specify the following assembler options on OPTIONS statements: LIST, NOLIST, XREF, NOXREF, OBJ, NOOBJ. Several options may appear on one statement in any order, but must be separated by commas. If the programmer prefers, separate statements may be used for each option. The OPTIONS keyword must start in column 2 or greater; the preceding column must be blank; and there must be one or more blanks between the keyword and the selected options. Blanks are not allowed between the selected options.

The following example shows options appearing on one statement:



The following list provides a brief description of all the options available:

Option	Explanation
LIST	The following sections of the assembler listing are printed: • Options information • External symbol list • Source and object program listing • Diagnostic listing • Error summary statements
NOLIST	Only the following listings are printed: • Options information • Any statements in error and the associated diagnostics • Error summary statements The NOLIST option overrides all assembler PRINT statements.
XREF	A cross-reference listing is generated.
NOXREF	A cross-reference listing is not generated.
OBJ	The object program is placed in the library as a subroutine member.
NOOBJ	The object program is not placed in the library.

If OBJ is entered on the OPTIONS statement and there are errors in the assembly, a halt is issued giving the choice to terminate or place the object program in the library as a subroutine member.

If no OPTIONS statement is used, the assembly is processed as though LIST, XREF, and OBJ had been specified.

EXECUTION INFORMATION

Procedures for Assembler

The loading and running of the assembler and macro processor can be done through the use of System/34 procedures. These procedures, and the procedure commands that request them, are described here. (For a complete description of System/34 procedures and procedure commands, see the *System Support Reference Manual*.)

ASM Procedure

The ASM procedure invokes the assembler and can invoke the macro processor. The ASM procedure is requested by way of the ASM procedure command.

ASM Procedure Command

The ASM procedure command requests execution of the ASM procedure, which invokes the assembler and, optionally, the macro processor. If you just enter ASM, a display appears prompting you for command parameters.

```
ASM  source name  , [ #LIBRARY ]
                    [ source library ] ,

   [ parameter 2 ]
   [ object module library ] .      [ MAC ]
                                    [ NOMAC ]

   , [ srcblksz ]    , [ asmblksz ]
     [ 30       ]      [ 45       ]

   , [ wrkblksz ]    , [ wrk2blksz ]  , [ yes ]
     [ 10       ]      [ 36        ]    [ no  ]
```

source name — Required source program name parameter.

#LIBRARY
source library — Specifies the name of the library in which the source, named in the first parameter, is located. If omitted, the system library (#LIBRARY) is assumed.

parameter 2
object module library — Specifies the name of the library in which the object module will be placed. If omitted, the library specified in the second parameter is assumed. If the second parameter is omitted, the system library (#LIBRARY) is assumed.

MAC
NOMAC — Macro processor parameter. NOMAC bypasses the macro processor; MAC invokes the macro processor. MAC is the default if the parameter is omitted.

srcblksz
30 — $SOURCE file size parameter. $SOURCE provides source input to the macro processor. If the macro processor is not invoked, $SOURCE provides source input to the assembler.

srcblksz: three-digit decimal number indicating blocks required by $SOURCE. 30 is the default.

asmblksz
45 — $ASMINPT file size parameter. $ASMINPT provides source input to the assembler if the macro processor is invoked. $ASMINPT contains the source program and macro processor generated code. If the macro processor is not invoked, this file is not used. However, the parameter is still used.

asmblksz: three-digit decimal number indicating blocks required by $ASMINPT. 45 is the default.

wrkblksz
10 — $WORK file size parameter. $WORK contains the object code produced by the assembler.

wrkblksz: three-digit decimal number indicating blocks required by $WORK. 10 is the default.

wrk2blksz
36 — $WORK2 file size parameter. $WORK2 is used as a work file by the assembler.

wrk2blksz: three-digit decimal number indicating blocks required by $WORK2. 36 is the default.

yes
no — Place job on the input queue. Do not place job on the input job queue. No is the default.

OLINK Procedure

This procedure invokes the overlay linkage editor to create a load module. The OLINK procedure is described in the *Overlay Linkage Editor Reference Manual.*

Data Files Used by the Assembler

Disk files are used for the following:

- Intermediate text ($WORK2 file)
- Cross-reference file ($WORK2 file)
- Overflow symbol table(s) ($WORK2 file)
- Object program records ($WORK file)
- Source program records ($SOURCE file and $ASMINPT). $SOURCE provides source program records for the macro processor. If the macro processor isn't called, $SOURCE also provides source program records for the assembler. If the macro processor is called, $ASMINPT provides source program records to the assembler.

If the source records are 80 (rather than 96) columns in length, they are padded on the right with 16 blanks before being placed in the input file. In this case, the user should provide an ICTL statement to prevent the assembler from processing the sequence field of the 80-column record.

$WORK2, $ASMINPT, $SOURCE, and $WORK are automatically allocated but their default sizes may be overridden by specifying the respective parameters on the ASM procedure.

$WORK2 requires approximately 40 sectors per 100 source statements:

Source Program Size (number of statements)	**Number of Required Blocks (One block equals 10 sectors.)**
100	4
200	8
300	12
400	16
500	20
600	24
700	28
800	32
900	36
1000	40

The $WORK file contains the object records. One sector contains four 64-byte object records. The default is 10 blocks.

$SOURCE size requirements are as follows:

Source Program Size (number of statements)	**Number of Required Blocks**
100	4
200	8
300	12
400	15
500	19
600	23
700	27
800	30
900	34
1000	38

$ASMINPT uses the same chart as $SOURCE. Note, however, that the number of generated statements should be counted in the program size.

ASSEMBLER LISTING

The printed output of the assembler includes the control statements, external symbol list, object code and source program listing, the cross-reference listing, and the error message listing. These listings are described in detail in this section.

Control Statements

Any OPTIONS or HEADERS statements specified by the user are printed and any specification errors are noted. A list of OPTIONS in effect during the assembly is then printed.

External Symbol List (ESL)

The object program name, EXTRNs and ENTRYs are printed in the following format:

Symbol	Type
Object program name	MODULE
ENTRY symbol	ENTRY
EXTRN symbol	EXTRN

Object Code and Source Program Listing

The following items are printed for each entry in the source program. See Appendix A for examples of an object code and source program listing.

(ERR) Error Field: This field contains an E, I, W, or M for those statements in error.

E	For assembler and macro processor errors
W	MNOTE warnings with a severity of 8
I	Information or image messages from the macro processor
M	MNOTE errors with severity greater than 8

(LOC) Location Counter: A four-digit hexadecimal number that is left-padded with zeros. This number represents the leftmost byte of any object code printed on this line.

Object Code: Translated code. All code in this field is left-justified.

- Instructions: Maximum of 6 bytes (12 hexadecimal characters). The operation, Q code, operand 1, and operand 2 fields are separated by one blank.
- Data Constants: Maximum of 8 bytes (16 hexadecimal characters) per line. No blanks are inserted among the data.
- (ADDR) Address Field: Blank except for the following:
  - For the DC and DS instructions, it contains the address of the reference byte, that is, the rightmost byte of the field.
  - For the END instruction, it contains the address to which control will be passed to start execution of the program.
  - For the USING instruction, it contains the address referenced in the first operand field.
  - For the EQU instruction, it contains the value of the operand field.
  - For the ENTRY instruction, it contains the address of the entry point.

(STMT) Statement Number Field: This field contains the number of the source statement starting from one. All source statements, including comments, are numbered. Valid SPACE, EJECT, and TITLE statements are always assigned statement numbers but are never printed. The statement number field is 4 characters long and therefore the program listing is accurate for only 9,999 statements.

Source Statement: A reproduction of the entire source record. All source records are printed except for the listing control statements: SPACE, EJECT, and TITLE.

Items printed include:

Column	Item
1	Error flag
5-8	Location
9	Blank
10-25	Object code
27-30	Address
32-35	Statement number
37-132	Source statement

Statements generated by the macro processor are preceded by plus (+) signs.

Page Headings

The following information is printed for each page in the listing:

- A header stating that the object code listing was produced by the IBM System/34 Basic Assembler and Macro Processor Program Product and identifying the release level.
- The content of the user's current TITLE card.
- A descriptive header, which gives a short description of the contents of the various fields of the source-object listing, the current date and time, and the page number.

Diagnostics

The source and object program listing includes error codes for improperly coded statements. These codes are documented at the end of the source and object program listing under the heading *Diagnostics*. The diagnostics list provides the following information:

- Statement: The statement number in decimal, as assigned by the assembler, of the statement in error.
- Error code: A four-digit code. See Chapter 8 for a complete list of these codes and the corresponding messages.
- Message: A translation of the error code, indicating the type of error made.

Also included under the heading *Diagnostics* are these error summary statements:

- A count of the total statements in error in the assembly. Total does not include missing module name and missing end statement errors.
- A count of total sequence errors in the assembly, if a sequence check is requested.

Cross-Reference List

If XREF is specified on the OPTIONS statement, this list includes all symbol names referred to in the source program. This list includes the following columns:

- SYMBOL: The symbol name.
- LEN: The length attribute of the symbol, in decimal.
- VALUE: The hexadecimal value of the symbol.
- DEFN: The statement number, in decimal, where the symbol is defined.
- REFERENCES: The statement numbers, in decimal, where the symbol is referenced. Each symbolic reference to a data area or machine register whose contents may be altered by the execution of a machine instruction is flagged with an asterisk.

At the end of the cross-reference list, the error summary statements are printed again.

OBJECT PROGRAM

The assembler program converts the source program into control information, machine language instructions, and data, all of which collectively are called an object program. There is one object program produced per assembly. Each object record is produced as a 64-byte field.

Each object program generated by the assembler contains three types of records.

- ESL (external symbol list) record
- TEXT-RLD (text-relocation directory) records
- END record

Record Formats

The following describes the format of each record type.

ESL Record: The object program name, module name, and all EXTRN and ENTRY symbols are placed in the ESL record. The ESL record format is:

- Byte 1: Record type identifier S.
- Byte 2: Length-1 of the ESL record.
- Bytes 3-62: ESL record.
- Bytes 63-64: Filled with zeros.

TEXT-RLD Records: Text records and RLD pointers are combined in this type of input record. The text portion of each record contains the object code for the program, while the RLD pointers indicate where the address constants and relocatable operands of the text are located. The format for the TEXT-RLD record is:

- Byte 1: Record type identifier T.
- Byte 2: Length-1 (of text only).
- Bytes 3-4: Assembled address of the low-order (rightmost) text byte in the record.
- Bytes 5-64: Text starts at byte 5 and goes right. RLD starts at byte 64 and goes left. The leftmost end of the RLD section is marked by the hexadecimal zeros that fill the space between the text and RLD sections. The end of text is always followed by at least one byte of X'00'.

END Record: The last record in each object program is an END record. The END record contains the entry address of the object program. If the user did not include an operand in his source program END statement, the object program END record generated by the assembler contains the address X'FFFF'. The format for the END record is:

- Byte 1: Record type identifier E.
- Bytes 2-3: Entry address of the object program.
- Bytes 4-64: Unused

MACROINSTRUCTION CODING RESTRICTIONS

The generated code for some macroinstructions uses register 1 and/or register 2. The contents of the register used by the generated code must be saved before issuing the macroinstruction; otherwise, the contents are destroyed. The $WSIO macroinstruction uses registers 1 and 2. These macroinstructions use register 2:

$ALOC	$OPEN
$CKPT	$PUTD
$CLOS	$PUTP
$FIND	$RIT
$GETD	$SIT
$INFO	$SNAP
$LOAD	$SORT
$LOG	$TOD

The code generated by the macroinstructions is assigned labels; these labels begin with the dollar sign ($). To avoid duplicate-label errors, do not use the dollar sign as the first character of a label.

MACROINSTRUCTION DEFINITION RESTRICTIONS

The macro processor assumes that any ampersand starts a variable symbol. An ampersand used elsewhere, including within a comment, results in error ASM-5402.

DISK DATA MANAGEMENT CONSIDERATIONS

Access Methods

Figure 7-1 indicates which access methods may be used with which file types. Note that four different situations are covered on the chart.

- The combination of the file type and the access method is not allowed by allocate or open. For these situations, the number of the corresponding message is given.
- The combination of the file type and the access method is allowed. These situations are indicated by a blank entry.
- In several situations, the actual file type of the file will change to that of the access method. These situations are indicated by *File Change*.
- The combination of the file type and the access method cannot occur. These situations are indicated by X.

File Type

Access Method	Consecutive					Direct					Indexed				
	DISP-OLD	DISP-NEW	DISP-SHR	DISP-NULL Existing File	DISP-NULL New File	DISP-OLD	DISP-NEW	DISP-SHR	DISP-NULL Existing File	DISP-NULL New File	DISP-OLD	DISP-NEW	DISP-SHR	DISP-NULL Existing File	DISP-NULL New File
CG					ALOC 1356		X			X		X			X
CU					ALOC 1356		X			X	OPEN 2204	X	OPEN 2201 2204	OPEN 2204	X
CA						OPEN 2202	X	OPEN 2201 2202	OPEN 2202	X	OPEN 2204	X	OPEN 2201 2204	OPEN 2204	X
CO			ALOC 1360	ALOC 1359 1360 1361		File Change	X	ALOC 1360	ALOC 1359 1360 1361	X	File Change	X	ALOC 1360	ALOC 1359 1360 1361	X
DG DGA		X			X					ALOC 1356		X			X
DU DUA		X			X					ALOC 1356		X			X
DO DOA	File Change	X	ALOC 1360	ALOC 1359 1360 1361	X			ALOC 1360	ALOC 1359 1360 1361		File Change	X	ALOC 1360	ALOC 1359 1360 1361	X
IR	OPEN 2203	X	OPEN 2203	OPEN 2203	X	OPEN 2203	X	OPEN 2203	OPEN 2203	X					ALOC 1356
IRU	OPEN 2203	X	OPEN 2203	OPEN 2203	X	OPEN 2203	X	OPEN 2203	OPEN 2203	X					ALOC 1356
IA IRA IRUA	OPEN 2203	X	OPEN 2203	OPEN 2203	X	OPEN 2203	X	OPEN 2203	OPEN 2203	X					ALOC 1356
IO	File Change	X	ALOC 1360	ALOC 1359 1360 1361	X	File Change	X	ALOC 1360	ALOC 1359 1360 1361	X			ALOC 1360	ALOC 1359 1360 1361	

Figure 7-1 (Part 1 of 2). Access Methods

File Type

Access Method	Consecutive					Direct					Indexed				
	DISP-OLD	DISP-NEW	DISP-SHR	DISP-NULL Existing File	DISP-NULL New File	DISP-OLD	DISP-NEW	DISP-SHR	DISP-NULL Existing File	DISP-NULL New File	DISP-OLD	DISP-NEW	DISP-SHR	DISP-NULL Existing File	DISP-NULL New File
IS	OPEN 2203	X	OPEN 2203	OPEN 2203	X	OPEN 2203	X	OPEN 2203	OPEN 2203	X					ALOC 1356
ISU	OPEN 2203	X	OPEN 2203	OPEN 2203	X	OPEN 2203	X	OPEN 2203	OPEN 2203	X					ALOC 1356
ISA ISUA	OPEN 2203	X	OPEN 2201 2203	OPEN 2203	X	OPEN 2203	X	OPEN 2201 2203	OPEN 2203	X			OPEN 2201		ALOC 1356
ISRI	OPEN 2203	X	OPEN 2203	OPEN 2203	X	OPEN 2203	X	OPEN 2203	OPEN 2203	X					ALOC 1356
ZPAMI					ALOC 1356		X			X		X			X
ZPAMA			OPEN 2201			OPEN 2202	X	OPEN 2201	OPEN 2202	X	OPEN 2204	X	OPEN 2201	OPEN 2204	X
ZPAMO			ALOC 1360	ALOC 1359 1360 1361		File Change	X	ALOC 1360	ALOC 1359 1360 1361	X	File Change	X	ALOC 1360	ALOC 1359 1360 1361	X

Figure 7-1 (Part 2 of 2). Access Methods

Of the access methods listed in Figure 7-1, the ISRI and ZPAM (ZPAMI, ZPAMA, ZPAMO) methods require the user program to provide space for the access method code.

The ISRI (indexed sequential/random input) access method is similar to random-by-key access into an indexed file. A key, which you specify, is retrieved along with the corresponding data record. At this point, you can choose to do one of the following:

- Request that the next key be read (OP-FGET).
- Request that the previous key be read (OP-BGET).
- Provide a new key and request another random read (OP-NGET).
- Provide a new key and specify reading of the equal key, the next higher key, or the last key, whichever is encountered first (OP-AGET). ISRI then looks for a key (and its corresponding record) that equals the key you provide. If an equal key is not found, ISRI returns the next higher key (and record). If neither an equal nor a higher key is found and you are not accessing a delete-capable file, ISRI returns the last record in the primary portion of the file. The primary portion is the part of the file that reflects the ordered keys in the index. If you are accessing a delete-capable file, the last record in the primary portion may be a deleted record. In this case, ISRI returns the *record not found* completion code. You can specify OP-BGET in response to the *record not found* completion code. OP-BGET causes ISRI to read backward through the file, skipping deleted records until a valid key is found.

Only index entries in the primary index area can be accessed through ISRI, and only input operations are supported by ISRI. Master track index is not supported when you use the ISRI access method.

The ZPAM access methods are used to process disk sectors of data rather than records. The amount of data transferred must be a multiple of 256 bytes. This provides an easy way of moving large amounts of data rapidly. During processing, the record length of the file is not used by data management; rather, data management processes a number of disk sectors of data based upon the block length you specify. Processing of the disk sectors through ZPAM is consecutive. Input, output, and add functions are supported through ZPAM.

Data Management Control Blocks and Buffers

To interface with disk data management, you are required to provide storage space for interface information. These areas must be available to the system from the time the file is allocated until it is closed.

DTF

The DTF is the major control block for communication between data management and you. It provides the information needed to allocate, open, access a file on the disk device, and close the file. It also contains pointers to the other control blocks and the buffer areas. The DTF can vary in length from 72 bytes for a consecutive access to 138 bytes for an indexed sequential access. For information on generating a disk DTF, see *Define the File for Disk ($DTFD)* in Chapter 6.

I/O Buffer Area

This area is divided into the required disk input/output blocks (IOBs) and physical buffer areas by disk open. If the access to the file is indexed, two IOBs are built; otherwise, one is built.

In the event of very limited user main storage in relation to the size requirements of the physical buffers for disk, it may be advantageous to use the SIAM method to allocate buffers. Through SIAM, the same storage area may be used for a physical buffer for any or all disk files. Data management will then use this area as a physical buffer for every file specified as SIAM. Care should be taken, however, in the use of SIAM since many more I/O operations are required when SIAM is specified for a file. This may hinder the performance of the job.

Logical Buffer Area

Whenever data is being written to disk (through output, add, or the output portion of an update), you must provide a logical buffer. This allows data management to move the data from the logical buffer to the physical buffer. This type of processing is known as move mode.

Master Track Index Area

To aid the performance of the indexed random and indexed sequential within limits access methods for large files, you may supply data management with main storage space for a master track index. This area will be formatted by open so that the requested key within the index area of the indexed file may be located more easily. It is in effect an index into the index area or a second-level index. Master track index is not supported when you use the ISRI access method.

Address of Data Management Routines

While much of data management may be used without reserving space, you must reserve space if you are accessing disk through ISRI or ZPAM. This area will be initialized at open time.

If you are using two DTFs with the same access method (ISRI or ZPAM), you need reserve only one area.

Requested Record Number or Key Area

While processing under a direct access method, the user must specify a relative record number of the requested record. While processing under an indexed random access method, the user must specify the key of the relative record number. This area correlates to the KEYADD parameter of the $DTFD macroinstruction.

Key Hold Areas

While processing under index sequential access method, the user must provide a space two key lengths long for the use of data management. This area correlates to the CURENT parameter of the $DTFD macroinstructions.

Key Limits Area

When the use of key limits is requested, the user must provide an area for containing the low and high limits. This area correlates to the HIGH parameter of the $DTFD macroinstruction.

Allocating and Opening the File

Before processing data from any disk file, the file must be allocated ($ALOC) and opened ($OPEN). $ALOC and $OPEN perform the following operations:

- If the file is new, space on the disk is reserved for the data and the space is initialized.
- Diagnostics are performed to ensure that the access method and file organization are compatible and that all necessary information about the file was provided.
- The input/output blocks (IOBs) and buffer areas are formatted.
- The DTF is formatted to a post-open state.

For more information on the $ALOC and $OPEN macros, see *Allocate Space or Device ($ALOC)* and *Prepare a Device or File for Access ($OPEN)* in Chapter 6.

Accessing Records in the File

After the file has been allocated and opened, you may begin accessing records of that file. The communications vehicle between your calling program and the disk data management program is the same DTF that was used for allocating and opening the file. Certain fields in the DTF are for communication from the calling program to data management, some are for communication from data management to the calling program, some are bidirectional communication fields, and still other DTF fields are for internal data management use only.

Several DTF communication fields are pointers to main storage areas. (These main storage areas will be referred to as DTF areas in order to differentiate them from the DTF fields.) Each field in the DTF has a name as defined in the $DTFO macro expansion. Those field names (excluding the prefix, $F1) will be used to identify specific fields and areas.

Figure 7-2 describes the DTF fields that comprise the external interface. All DTF fields not described on this chart are reserved for internal data management use and may not be altered or otherwise depended upon by any calling program.

DTF Field	How Specified Macro/Keyword	Direction of Communication C/P (Vector) D/M	Access Methods Applicable	Reqd	Set By	Can Be Altered After Open	Notes
–DEV	$DTFD/*	→	All	Yes	C/P	No	*No keyword applies to this field.
–AT1*	$DTFD/ACCESS	→	All	Yes	C/P	No	*Also certain bits in other ATTR bytes may be set by $DTFD.
–NAM	$DTFD/NAME	→	All	Yes	C/P	No	
–OPC	$GETD & $PUTD/ UPDATE or DELETE	→	All	Yes	C/P	Yes	For ISRI, BGET and FGET may be specified .
–WKB	$DTFD/RCAD	↔	All	Yes*	C/P D/M	Yes	*Required of C/P for update and delete add/output.
–CMP	----------	←	All	–	D/M	Yes	*See section on completion code.
–RCL	$DTFD/RECL	→	All	Yes	C/P	No	
–BKL	$DTFD/BLKL	→	All	Yes	C/P	No*	*Can be altered after open for ZPAM or direct access methods.
–IOB	$DTFD/IOAREA	→	All	Yes	C/P OPEN	No	
–KAD	$DTFD/KEYADD	→	I/R D	Yes*	C/P OPEN**	Yes***	*Not required for I/R output. **Alter by open for I/R accesses. ***User beware basis– alterable.
–KL	$DTFD/KEYL	→	I/R I/S	Yes	C/P	No	
–KD	$DTFD/KDISP	→	I/R I/S	Yes	C/P	No	
–AT2*	$DTFD/ORDLD	→	I/R I/S	No*	C/P	No	*If indexed access part of AT2 affected.

Figure 7-2 (Part 1 of 2). DTF Fields

DTF Field	How Specified Macro/Keyword	Direction of Communication C/P (Vector) D/M	Access Methods Applicable	Reqd	Set By	Can Be Altered After Open	Notes
–CHN	$DTFD/CHAIN	→*	All	No		No	*Communication to ALLOC and OPEN.
–CUR	$DTFD/CURENT	→	I/S	Yes	C/P	No	
–LST	$DTFD/CURENT	→	I/S	Yes	OPEN*	No	*Set by OPEN based on CURENT.
–HI	$DTFD/HIGH*	→	I/S	No	C/P	No	*Used when limit specified.
–LO	$DTFD/HIGH*	→	I/S	No	OPEN*	No	*Set by OPEN based on HIGH.
–AT*	$DTFD/SIAM	→	ALL	No	C/P		*Part of field.
–PBF	$DTFD/IOBUF	→	ALL*	No*	OPEN C/P	No	*Need only be specified with SIAM or ZPAM.
–AT*	$DTFD/ERROR	→	ALL	No	C/P	No	*Part of field.
–AT*	$DTFD/RETURN	→	ALL	No	C/P	No	*Part of field.
–AT*	$DTFD/BUFNO	→	CONS	No	C/P	No	*Part of field.
–AT*	$DTFD/LIMIT	→	I/S	No	C/P	No	*Part of field.
–NDX	$DTFD/MSTNDX	→	I/R I/SLL*	No	C/P	No	*Indexed sequential within limits.
–BYT	$DTFD/MSTBYT	→	I/R I/SLL*	No	C/P	No	*Indexed sequential within limits.
–DMA	$DTFD/DMADDR	→	ISRI ZPAM	Yes*			*For the special access methods specified.

C/P = calling program
D/M = data management

Figure 7-2 (Part 2 of 2). DTF Fields

DTF Fields Common To All Access Methods

As noted in the previous chart, several DTF fields are used for communication between the calling program and data management. The following are used for all access methods:

- DEV Field Is initialized by the $DTFD macro to specify a disk DTF.
- ATI Field Reflects the general type of access method specified. Other attribute bytes further qualify the access method.
- NAM Field Specifies the file name. The NAM field must correspond to the name specified in the FILE statement.
- OPC Field Specifies either input, output, update, or delete. If output is specified here, the attributes are checked to distinguish add from normal output.
- WKB Field Contains a logical record pointer. This is a bidirectional field. For output type operations (output, update, or add), the calling program must point to the beginning of the logical record for output. For successful input operations, data management points to the beginning of the logical record retrieved (always within the physical buffer).
- WKB Area Contains a logical record with the length specified in RCL field.
- CMP Field Contains completion code upon return from data management. This field tells whether the operation was successful; and if not, why not. See *Return Conditions* later in this chapter.
- RCL Field Contains logical record length set by calling program.
- BKL Field Contains block length set by user. In general, this length must be a multiple of 256 bytes. In particular, for input operations (which are always in locate mode) and the DO and DOA access methods (which involve internal input operations), the following rules apply:
  - If the record length is a power of 2, then BKL must be at least (RCL + 255) rounded down to the next multiple of 256.
  - If the record length is not a power of 2, then BKL must be at least (RCL + 255) rounded up to the next multiple of 256.
- IOB Field and Area At DTF creation, contains a pointer to the left byte of the I/O area provided by the calling program. This area must be large enough to hold all necessary IOBs and physical I/O buffers. Furthermore, the area must be large enough or aligned such that OPEN can begin each buffer on an 8-byte boundary. Unless you are using SIAM, the following formulas can be used:

  Consecutive or Direct Accesses:
  32 + 7 + (BKL * BUFNO)
  Indexed Accesses:
  2(32) + 7 + 264 + BKL

  32 is the IOB length.

  OPEN, in addition to dividing this area as indicated, fills the DTF field PBF with the beginning address of the physical I/O buffer (unless SIAM or ZPAM is specified).
- CHN Field Contains pointer to next DTF on chain if calling program chooses to allocate and/or open several DTFs with one call. The last DTF on a chain should not specify the CHAIN parameter.
- AT2-3-4 Fields Contain information which qualifies the access method, tells whether SIAM was specified, and tells what to do in case of I/O error.

Consecutive Processing Fields

Consecutive processing is used when you want to process each record in order of physical location within the file.

In addition to those fields and areas described under *DTF Fields Common to All Access Methods*, there is one other field you can specify for consecutive processing:

- BUFNO Attribute Field Specifies whether or not you want a double size buffer.

Direct Processing Fields

Direct processing is used when you want to process a file randomly by relative physical location of the data record.

In addition to those fields and areas described under *DTF Fields Common to All Access Methods*, you must specify one other field and one other area for direct processing:

- KAD Field Contains a pointer to left byte of the area that contains the relative record number of the record to be processed.

- KAD Area For direct processing, this is either a 3-byte or a 10-byte area depending on whether the relative record number is binary (3 bytes) or decimal (10 bytes). In either case, the number is right justified.

  If binary relative record numbers are used, the first record position in the file is 0 and the binary number of the highest possible position is equivalent to decimal 16711407 and hexadecimal FEFEEF. If decimal relative record numbers are used, the first record position in the file is 1 and the number of the highest possible position is 9999999. The number specified is used as an absolute value.

Indexed Random Processing Fields

Indexed random processing is used when you want to process an indexed file and want the capability to process randomly by key value.

In addition to those fields and areas described under *DTF Fields Common to All Access Methods*, the following are used for indexed random processing:

- KAD Field For indexed random processing at DTF creation, this field must point to the left byte of an area the length of one key. Open adjusts this pointer to make it a right-byte pointer to the area. When interspersing input operations with update, delete, or add operations, this field must be reinitialized as a right-byte pointer before each input operation. Note that this field is also used in direct processing.

- KAD Area Contains the key value of the record to be processed at each entry to data management. The calling program should ensure that no byte contains a value of X'FF'.

  For update-, delete-, or add-capable processing (accesses IRU or IRUA), this area must not be within the logical record area.

- KL Field Contains the key length (a binary number).

- KD Field Contains the key displacement (a binary origin 0 number) to the rightmost byte of the key in the record. For example, if the rightmost byte of the key is in the eighth byte position of the record, then KD must contain the value 7.

- MIX Field Optional master index pointer field, which points to the leftmost byte of an area where the master index is built by open.

- BYT Field Used in conjunction with the MIX field, specifies length of field. Must be a multiple of (KL + 3).

- MIX Area Must not be altered by the calling program after open.

Indexed Sequential Processing Fields

Indexed sequential processing is used when you want to process an indexed file in ascending order by key. With this access method, you have access only to the primary portion of the file, which is the part reflected by the ordered keys in the index. With the IFILE characteristic, you have access to both the primary portion of the file and the overflow portion of the file. (For information about ISRI, see *Access Methods* in this chapter.)

In addition to those fields and areas described under *DTF Fields Common to All Access Methods*, the following are used for indexed sequential processing:

- KL Field Key length (see *Indexed Random Processing Fields*).

- KD Field Key displacement (see *Indexed Random Processing Fields*).

- CUR Field Contains pointer to a 2-key-length area. Prior to open, it is a left-byte pointer. After open, the CUR area is divided into two subareas, current and last.

- CUR Area This 2-key-length area is reserved by data management after open.

- HI Field When using limits processing, this is a required field that, before open, points to the leftmost byte of a 2-key-length area. After open, the HI area is divided into two subareas, high and low.

- HI Area The above 2-key-length area is reserved by data management after open.

- AT2 Field Part of this field tells whether LIMITS is specified.

The following fields can also be used if you specified limits processing (LIMIT-Y):

- MIX Field Optional master index pointer field, which points to the leftmost byte of an area where the master index is built by open.

- BYT Field Used in conjunction with the MIX field, specifies length of field. Must be a multiple of (KL + 3).

- MIX Area Must not be altered by the calling program after open.

Updating Records

Update is used when one or more fields of an existing record are to be changed. In general, prior to issuing an update operation to data management, you must have just issued an input operation for that record. If you issue an add operation between an input operation and an update operation, the update will not be successful.

Although you may succeed at times in the practice of doing an input followed by several updates to that record, that practice is discouraged for two reasons. One is performance; the other is that the sequence will not be successful when either SIAM or file sharing is designated.

You must generally avoid updating records within the physical buffer. In the event of SIAM or file sharing, data management primes the I/O buffers as part of the update process. Hence, updates would be lost. After receiving the input, move the record to the logical record area (specified by the RCAD parameter in $DTFD) outside the I/O buffer, make any desired field updates, then issue an update operation.

Finally, you cannot change the KAD area value between an input, update sequence.

Deleting Records

Delete is used when you want to delete a record from a file. After a record is deleted, it is no longer accessible. The record is not physically removed from the file, but the data is erased.

The rules for deleting records are the same as those for updating records. Updating records is described in preceding paragraphs.

Adding Records

Add is used when more records are to be included in an existing file.

Direct access method add is not a supported function.

Index sequential add requires that you first issue get operations (beginning with the lowest key in the file) until you encounter the first key higher than the key you want to add or until end of file is reached. At that point you can issue an add operation.

Return Conditions

The following list describes all currently defined return conditions. These are conveyed in the completion code field in the DTF. For the actual labels and hex values of the return codes, see the values generated by the $DTFO macroinstruction.

- Normal return
- Permanent disk error
- End of file
- Invalid operation code
- Record not found–indexed random
- Out of extent–direct
- Update–previous operation not input
- Invalid key–indexed random
- Invalid block length
- Direct–record not found
- Invalid update, add, or output
- Update key error
- Override–deleted record not found
- Direct–put to nondeleted record
- Duplicate key add attempted
- Out of sequence
- End of extent
- Undefined access type
- DTF not opened

Terminating The File

When all desired records have been processed, you should close ($CLOS) the file . Once the DTF has been through close, no additional record processing is allowed via that DTF unless and until the process of allocate and open is repeated. For more information on the $CLOS macroinstruction, see *Prepare a Device or File for Termination ($CLOS)* in Chapter 6.

DISPLAY STATION DATA MANAGEMENT CONSIDERATIONS

Following each DTF operation issued via $WSIO, a 2-byte return code is passed back in the DTF at displacements $WSRTC-1 and $WSRTC. The return codes possible after the various $WSIO operations are described here, except for operations issued to the interactive communications feature. Return codes from the interactive communications feature are described in the *Interactive Communications Feature Reference Manual*. All the return codes listed for an operation are mutually exclusive.

Note: For a guide to work station data management concepts and considerations, see the *Concepts and Design Guide*.

GET and ACI Return Codes

After a GET or ACI operation, the following return codes are possible at $WSRTC:

Label	Value	Explanation
$WSROK	X'00'	Operation successful
$WSRACC	X'01'	New requester
$WSRSTP	X'02'	Stop system requested by system operator (see the *System Data Areas and Diagnostic Aids Handbook* for the contents of $WSRTC-1)
$WSRACR	X'11'	ACI rejected. No invites outstanding.
$WSRKBD	X'14'	Input rejected, keyboard disabled.
$WSRNAV	X'24'	Display station released by display station operator
$WSRREL	X'28'	GET rejected. Display station previously released by program.
$WSRIRJ	X'34'	Input rejected. Input buffer (INLEN parameter) too small
$WSRPE	X'80'	Permanent I/O error occurred at the display station. In response to the error, the system operator selected a 2 option.

ACQ Return Codes

After an ACQ operation, the following return codes are possible at $WSRTC:

Label	Value	Explanation
$WSROK	X'00'	ACQ successful
$WSRAQO	X'08'	ACQ successful. Display station already allocated to the task.
$WSRAFW	X'18'	ACQ failed. Display station allocated to a non-NEP.
$WSRAFS	X'32'	ACQ failed. Unauthorized user.
$WSRAFN	X'38'	ACQ failed: – Display station is not in standby mode. – Display station is in command reject mode. – A permanent I/O error occurred at the display station. – The display station is allocated to an NEP.

STI Return Codes

After an STI operation, the following return codes are possible at $WSRTC:

Label	Value	Explanation
$WSROK	X'00'	STI successful
$WSRNAV	X'24'	Display station released by display station operator
$WSRREL	X'28'	STI ignored. Display station previously released by program.
$WSRSPF	X'44'	STI failed. Display station operator entered data, which should be read by a GET or ACI operation.
$WSRPE	X'80'	Permanent I/O error occurred at the display station. In response to the error, the system operator selected a 2 option.

Return Codes for All Operations Except GET, ACI, ACQ, and STI

After any operation except GET, ACI, ACQ, and STI, the following return codes are possible at $WSRTC:

Label	Value	Explanation
$WSROK	X'00'	Operation successful
$WSRNAV	X'24'	Display station released by display station operator
$WSRREL	X'28'	Operation ignored. Display station previously released by program.
$WSRIRJ	X'34'	Input rejected. Input buffer (INLEN parameter) too small.
$WSRDFL	X'40'	Printer specified by print operation is offline.
$WSPOGE	X'45'	Invalid ideographic character during print operation.
$WSRGRF	X'50'	On an output operation, a display station ideographic character table full of ideographic characters was detected. The user selected a 2 option.
$WSRGI	X'51'	On an output operation, an invalid ideographic character was found. The user selected a 2 option.
$WSRGU	X'52'	On an output operation, one of the following errors was detected: • An undefined ideographic character was found. • The extended file of ideographic characters has not been allocated. • The extended file of ideographic characters has not been restored. The user selected a 2 option.
$WSRPE	X'80'	Permanent I/O error occurred at the display station. In response to the error, the system operator selected a 2 option.

Chapter 8. Printed Messages

MACROINSTRUCTION STATEMENT ERRORS

Any errors made in coding macroinstructions are flagged in the $ASMINPT file by placing an error code and an error message immediately after the macroinstruction. The error code and message are then printed on the assembly listing when the source program is assembled.

The following listing shows the error codes that may be caused by errors in macroinstructions. Other error codes may be generated by the macro processor and are caused by errors in the macroinstruction definitions.

ASM-2600 INVALID V PARAM GIVEN. NO MACRO CODE GENERATED.

Explanation: Something other than DC, EQU or ALL was coded for V parameter.

ASM-2601 INVALID TYPE PARAM SPECIFIED. TYPE-DEC ASSUMED.

ASM-2602 INVALID ITYPE PARAM SPECIFIED. ITYPE-REAL ASSUMED.

ASM-2603 INVALID CANCEL PARAM SPECIFIED. CANCEL-N ASSUMED.

ASM-2604 INVALID WAIT PARAMETER SPECIFIED. WAIT-N ASSUMED.

ASM-2605 UPDATE-Y AND DELETE-Y BOTH SPECIFIED. DELETE-Y ASSUMED.

ASM-2606 INVALID NREF PARAMETER SPECIFIED. NREF-N ASSUMED.

ASM-2607 INVALID XLOFF PARAM SPECIFIED. XLOFF-N ASSUMED.

ASM-2609 RCAD PARAMETER NOT SPECIFIED. ZEROS ASSUMED.

ASM-2610 NAME PARAM NOT SPECIFIED. NAME-FILENAME ASSUMED.

ASM-2611 IOAREA PARAMETER NOT SPECIFIED. ZEROS ASSUMED.

ASM-2612 ACCESS PARAMETER NOT SPECIFIED. ACCESS-CG ASSUMED.

ASM-2614 DMADDR PARAM GIVEN BUT NOT NEEDED. PARAM IGNORED.

ASM-2615 DMADDR PARAMETER NOT SPECIFIED. ZEROS ASSUMED.

ASM-2617 RECL PARAMETER NOT SPECIFIED. RECL-32 ASSUMED.

ASM-2618 RECL PARAMETER GREATER THAN 4096. RECL-32 ASSUMED.

ASM-2619 BLKL PARAMETER NOT SPECIFIED. BLKL-256 ASSUMED.

ASM-2620 IOBUF PARAMETER USED INVALIDLY. PARAMETER IGNORED.

Explanation: This parameter is valid only with SIAM-Y and ZPAM access methods.

ASM-2621 KEYADD PARAMETER NOT SPECIFIED. HEX FFFF ASSUMED.

ASM-2622 KEYL PARAM NOT SPECIFIED, INDEXED FILE. 1 ASSUMED.

ASM-2623 KDISP PARAM NOT GIVEN, INDEXED FILE. 0 ASSUMED.

ASM–2624 CURENT PARAMETER NOT SPECIFIED. HEX FFFF ASSUMED.

ASM–2625 HIGH PARAMETER NOT SPECIFIED. HEX FFFF ASSUMED.

ASM–2626 KEYADD INVALID FOR CONSECUTIVE ACCESS. IGNORED.

ASM–2627 KEYL INVALID FOR NON-INDEXED ACCESS. IGNORED.

ASM–2628 KDISP INVALID FOR NON-INDEXED ACCESS. IGNORED.

ASM–2629 MSTNDX INVALID FOR NON-INDEXED ACCESS. IGNORED.

ASM–2630 CURENT INVALID FOR NON-INDEXED ACCESS. IGNORED.

ASM–2631 HIGH INVALID FOR NON-INDEXED ACCESS. IGNORED.

ASM–2632 ORDLD-Y INVALID FOR NON-INDEXED OUTPUT. IGNORED.

ASM–2633 LIMIT-Y INVALID FOR NON-INDEXED ACCESS. IGNORED.

ASM–2634 BUFNO-2 INVALID, NON-CONSECUTIVE ACCESS. IGNORED.

ASM–2635 SIAM-Y REQUIRES IOBUF PARAM. HEX FFFF ASSUMED.

ASM–2636 GET AND PUT BOTH GIVEN. NO MACRO CODE GENERATED.

ASM–2637 LENGTH OR OFFSET INVALID. NO MACRO CODE GENERATED.

Explanation: Something other than a decimal value from 1 to 256 was coded.

ASM–2638 FROM MISSING LEFT PAREN. NO MACRO CODE GENERATED.

ASM–2639 FROM PARAM MISSING REG. NO MACRO CODE GENERATED.

ASM–2640 FROM PARAM MISSING DISP. NO MACRO CODE GENERATED.

ASM–2641 TO MISSING LEFT PAREN. MACRO GENERATION STOPPED.

ASM–2642 TO PARAM MISSING REG. MACRO GENERATION STOPPED.

ASM–2643 TO PARAM MISSING DISP. MACRO GENERATION STOPPED.

ASM–2644 PLIST-2 WITH LOAD PARAM. MACRO GENERATION STOPPED.

ASM–2645 INVALID TYPE PARAMETER. MACRO GENERATION STOPPED.

ASM–2646 FORMAT INVALID WITH TYPE GIVEN. FORMAT IGNORED.

ASM–2647 HALT INVALID WITH TYPE GIVEN. HALT IGNORED.

ASM–2648 TYPE GIVEN REQUIRES DRADD PARAM. HEX FFFF ASSUMED.

ASM–2649 TYPE GIVEN REQUIRES MSGAD PARAM. HEX FFFF ASSUMED.

ASM–2650 HALT-Y REQUIRES OPTN0, OPTN1, OPTN2, OR OPTN3.

ASM–2651 INVALID DRLEN PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than 8 or 60 was coded.

ASM–2652 HIST-N, CRT-N BOTH GIVEN. NO MACRO CODE GENERATED.

ASM–2653 TYPE GIVEN REQUIRES DRLEN PARAM. DRLEN-8 ASSUMED.

ASM–2654 TYPE GIVEN REQUIRES MSGLN PARAM. MSGLN-75 ASSUMED.

ASM–2655 TYPE PARAMETER NOT SPECIFIED. TYPE-1 ASSUMED.

ASM–2656 INVALID TYPE PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than 1, 1R, 2, 2R, 3 or 4 was coded.

ASM–2657 TYPE GIVEN REQUIRES MIC PARAM. HEX 0001 ASSUMED.

ASM–2658 INVALID WRSTE PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2659 INVALID HALT PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2660 INVALID FORMAT PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2661 INVALID HIST PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2662 INVALID CRT PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2663 INVALID OPTN0 PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2664 INVALID OPTN1 PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2665 INVALID OPTN2 PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2666 INVALID OPTN3 PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2667 INVALID SPACE PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than 1, 2, or 3 was coded.

ASM–2668 INVALID SKIP PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2669 INVALID VARIN PARAMETER. NO MACRO CODE GENERATED.

Explanation: Something other than Y or N was coded.

ASM–2671 FORMAT-N, HALT-Y GIVEN. NO MACRO CODE GENERATED.

ASM–2672 WRSTE PARAMETER NOT SPECIFIED. WRSTE-Y ASSUMED.

ASM–2673 FORMAT PARAMETER NOT SPECIFIED. FORMAT-N ASSUMED.

ASM–2674 HALT PARAMETER NOT SPECIFIED. HALT-N ASSUMED.

ASM–2675 SPACE PARAMETER NOT SPECIFIED. SPACE-1 ASSUMED.

ASM–2678 LOADER-Y INVALID WITH TYPE GIVEN. PARAM IGNORED.

ASM–2679 INVALID CODE PARAMETER. MACRO GENERATION STOPPED.

ASM–2680 BLKL PARAMETER AND RECL PARAMETER CONFLICT.

Explanation: BLKL must be equal to or greater than RECL.

ASM–2681 STATION IDS RECOMMENDED ON SWITCHED LINES.

ASM–2682 INVALID TRANSP PARAMETER.

Explanation: Something other than Y or N was coded.

ASM–2683 INVALID ITB PARAMETER.

Explanation: Something other than Y or N was coded.

ASM–2684 INVALID UPSI PARAMETER.

ASM–2685 INVALID CODE PARAMETER.

Explanation: Something other than E or A was coded.

ASM–2686 RCAD PARAMETER REQUIRED.

ASM–2687 ITB PARAM, TRANSP PARAM AND FTYP PARAM CONFLICT.

ASM–2688 TRANSP PARAMETER AND CODE PARAMETER CONFLICT.

Explanation: TRANSP is valid only with CODE-E.

ASM–2689 FTYP PARAMETER REQUIRED.

ASM–2690 INVALID TYPE PARAMETER.

ASM–2691 INVALID BUFNO PARAMETER.

ASM–2692 RECL PARAMETER REQUIRED.

ASM–2693 TERMAD PARAMETER AND TYPE PARAMETER CONFLICT.

Explanation: TERMAD is valid only with TYPE-MP.

ASM–2694 INVALID TERMAD PARAMETER.

ASM–2695 TERMAD PARAMETER REQUIRED.

ASM–2696 INVALID DLYCT PARAMETER.

ASM–2697 BLKL PARAMETER REQUIRED.

ASM–2698 RVIMSK PARAMETER AND RVIADR PARAMETER CONFLICT.

ASM–2699 INVALID ERRCT PARAMETER.

ASM–2700 RCVID PARAMETER AND TYPE PARAMETER CONFLICT.

Explanation: This parameter is valid only with switched lines.

ASM–2701 RCVCT PARAMETER AND TYPE PARAMETER CONFLICT.

Explanation: This parameter is valid only with switched lines.

ASM–2702 RCVCT PARAMETER REQUIRED.

ASM–2703 RCVID PARAMETER REQUIRED.

ASM–2704 INVALID RCVCT PARAMETER.

ASM–2705 SNDID PARAMETER AND TYPE PARAMETER CONFLICT.

Explanation: This parameter is valid only with switched lines.

ASM–2706 SNDCT PARAMETER AND TYPE PARAMETER CONFLICT.

Explanation: This parameter is valid only with switched lines.

ASM–2707 SNDCT PARAMETER REQUIRED.

ASM–2708 SNDID PARAMETER REQUIRED.

ASM–2709 INVALID SNDCT PARAMETER.

ASM–2710 RECSEP PARAMETER AND ITB PARAMETER CONFLICT.

ASM–2711 RECSEP PARAMETER AND TRANSP PARAMETER CONFLICT.

ASM–2712 INVALID RECSEP PARAMETER. RECSEP PARAM IGNORED.

ASM–2713 INVALID SKIP PARAMETER. SKIP-N ASSUMED.

ASM–2714 INVALID LOADER PARAMETER. LOADER-N ASSUMED.

ASM–2715 LOADER-N GIVEN OR ASSUMED. LOAD PARAMETER IGNORED.

ASM–2716 INVALID TYPE PARAMETER. TYPE-O ASSUMED.

ASM–2717 INVALID OPC PARAMETER SPECIFIED. OPC-N ASSUMED.

ASM–2718 NAME PARAMETER REQUIRED. BLANKS ASSUMED.

ASM–2768 V PARAM NOT ALLOWED WHEN PLIST-INLINE SPECIFIED.

ASM–2769 LABEL PARAMETER MISSING. NO MACRO CODE GENERATED.

ASM–2770 INVALID IMSG PARAMETER. IMSG-ALL ASSUMED.

ASM–2771 TYPE PARAMETER REQUIRED.

ASM–2772 SYMID PARAMETER LENGTH INVALID.

Explanation: The SYMID parameter must be 2 characters long.

ASM–2773 CTYPE PARAMETER REQUIRED.

ASM–2774 INVALID CTYPE PARAMETER.

Explanation: See the *Interactive Communications Feature Reference Manual* for a description of the CTYPE parameter.

ASM–2775 LUNUM PARAMETER LENGTH INVALID.

Explanation: The LUNUM parameter must be 3 characters long.

ASM–2776 RECL PARAM INVALID WITH RECLAD PARAM.

ASM–2777 SSENSE PARAMETER LENGTH INVALID.

Explanation: The SSENSE parameter must be 4 characters long.

ASM–2778 USENSE PARAMETER LENGTH INVALID.

Explanation: The USENSE parameter must be 4 characters long.

ASM–2779 INVALID DR1 PARAMETER. DR1-Y ASSUMED.

Explanation: See the *Interactive Communications Feature Reference Manual* for a description of the DR1 parameter.

ASM–2780 INVALID DR2 PARAMETER. DR2-N ASSUMED.

Explanation: See the *Interactive Communications Feature Reference Manual* for a description of the DR2 parameter.

ASM–2781 INVALID ERI PARAMETER. ERI-Y ASSUMED.

Explanation: See the *Interactive Communications Feature Reference Manual* for a description of the ERI parameter.

ASM–2782 USERLB IGNORED WHEN SOURCE NOT SPECIFIED.

ASM–2783 INPUT2 THROUGH INPUT8 MUST BE GIVEN SUCCESSIVELY.

ASM–2784 INVALID ALTSEQ PARAM SPECIFIED. ALTSEQ-N ASSUMED.

ASM–2785 INVALID KASRT PARAMETER GIVEN. KASRT-N ASSUMED.

ASM–2786 INVALID USE OF RECFMT PARAMETER.

ASM–3500 REQUIRED STATEMENT LABEL MISSING.[1]

ASM–3501 PARAMETER 1 MISSING OR INVALID.[1]

ASM–3502 PARAMETER 2 MISSING OR INVALID.[1]

ASM–3503 PARAMETER 3 MISSING OR INVALID.[1]

ASM–3504 PARAMETER 4 MISSING OR INVALID.[1]

ASM–3505 NO CASE KEYWORDS SPECIFIED.[1]

ASM–3506 EDIT PARAMETER INVALID.[1].

ASM–3507 DELIMS PARAMETER MISSING OR INVALID.[1]

ASM–3508 INCLDL PARAMETER INVALID.[1].

ASM–3509 $DE STATEMENT ALREADY ISSUED.

Explanation: Only one $DE macroinstruction is allowed in a single program. For a description of $DE, see the *1255 Magnetic Character Reader Reference Manual.*

ASM–3510 EXCLUD PARAMETER INVALID.[1]

ASM–3511 DLSEQ PARAMETER MISSING OR INVALID.[1]

ASM–3512 ALTCLS PARAMETER INVALID.[1]

ASM–3513 NUM PARAMETER MISSING OR INVALID OR DUPLICATE.

Explanation: The NUM parameter is missing, is invalid, or duplicates the NUM parameter on a previous $DF macroinstruction. The $DF macroinstruction is described in the *1255 Magnetic Character Reader Reference Manual.*

[1]For a description of the required value, see the *1255 Magnetic Character Reader Reference Manual.*

ASM–3514 MAXL PARAMETER MISSING OR INVALID.[1]

ASM–3515 MINL PARAMETER INVALID[1].

ASM–3516 $DF STATEMENT MISPLACED.

Explanation: All $DF macroinstructions must precede the $DE macroinstruction. $DE and $DF are described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3517 MOD PARAMETER MISSING OR INVALID.[1]

ASM–3518 REM PARAMETER MISSING OR INVALID.[1]

ASM–3519 WF PARAMETER MISSING OR INVALID.[1]

ASM–3520 SUM PARAMETER MISSING OR INVALID.[1]

ASM–3521 TABLE PARAMETER INVALID.[1]

ASM–3522 LEN PARAMETER MISSING OR INVALID.[1]

ASM–3523 NUM PARAMETER MISSING OR INVALID.[1]

ASM–3524 PAD PARAMETER INVALID.[1]

ASM–3525 PREVIOUS TABLE NOT CLOSED.

Explanation: A previous table which was opened by way of a $DT macroinstruction was not closed by way of a $DTD LAST macroinstruction. $DT and $DTD are described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3526 NO TABLE DEFINITION OPEN.

Explanation: A $DTD macroinstruction was issued to define table data, but no $DT macroinstruction was issued to define the table. $DT and $DTD are described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3527 HEX STRING IS NOT EVEN LENGTH.

Explanation: Excluding the first X, a string of hex characters specified in a $DTD macroinstruction must contain an even–not odd–number of characters. $DTD is described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3528 DATA PARAMETER TOO SHORT OR TOO LONG.

Explanation: Excluding the first C or X, a string of data characters specified in a $DTD macroinstruction must contain 1 through 32 characters. For a description of $DTD, see the *1255 Magnetic Character Reader Reference Manual.*

ASM–3529 PARAMETER 9 INCORRECTLY SPECIFIED.

Explanation: If it is specified, the ninth (positional) parameter in a $DTD macroinstruction must be LAST. $DTD is described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3530 ACTUAL TABLE LENGTH GREATER THAN SPECIFIED.

Explanation: The actual length of data entered in a table by way of $DTD macroinstructions is greater than the length specified for the table in the LEN and NUM parameters of the $DT macroinstruction. $DT and $DTD are described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3531 FIRST CHARACTER OF DATA PARAMETER IS NOT C OR X.[1]

---

[1]For a description of the required value, see the *1255 Magnetic Character Reader Reference Manual.*

ASM–3532 TABLE PARAMETER MISSING OR INVALID.[1]

ASM–3533 TYPE PARAMETER MISSING OR INVALID.[1]

ASM–3534 ALEN PARAMETER MISSING OR INVALID.[1]

ASM–3535 COMP PARAMETER MISSING OR INVALID.[1]

ASM–3536 ELEN PARAMETER MISSING OR INVALID.[1]

ASM–3537 NUM PARAMETER MISSING OR INVALID.[1]

ASM–3538 POS PARAMETER INVALID.[1]

ASM–3539 WORKAREA LENGTH EXCEEDED.

Explanation: The total length of the work area(s) defined by the $DW macroinstruction(s) exceeds 256 bytes. $DW is described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3540 NO $STRT STATEMENT ISSUED.

Explanation: The program must begin with a $STRT macroinstruction. For a description of $STRT, see the *1255 Magnetic Character Reader Reference Manual.*

ASM–3541 NO $DE STATEMENT ISSUED IN A MAIN PROGRAM.

Explanation: One $DE macroinstruction is required in each main program (TYPE-MAIN on $STRT). $DE (and $STRT) is described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3542 NO $DF STATEMENT ISSUED IN A MAIN PROGRAM.

Explanation: At least one $DF macroinstruction is required in each main program (TYPE-MAIN on $SRTR). $DF (and $STRT) is described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3543 PARAMETERS 2 AND 3 MISSING OR THE SAME.[1]

ASM–3544 TYPE PARAMETER INVALID.[1]

ASM–3545 LRSIZE PARAMETER MISSING OR INVALID.[1]

ASM–3546 TTSIZE PARAMETER INVALID, 16 ASSUMED.[1]

ASM–3547 $STRT STATEMENT ALREADY ISSUED.

Explanation: Only one $STRT macroinstruction is allowed in a single program. $STRT is described in the *1255 Magnetic Character Reader Reference Manual.*

ASM–3548 HOZCF AND/OR HOZCL PARAMETER INVALID.[1]

ASM–3549 RESBUF PARAMETER INVALID.[1]

ASM–3550 BUFNUM PARAMETER INVALID.

[1]For a description of the required value, see the *1255 Magnetic Character Reader Reference Manual.*

MACRO PROCESSOR

Any errors made in coding macroinstructions are flagged in the $ASMINPT file by placing an error code and an error message immediately after the macroinstruction. The error code and message are then printed on the assembly listing when the source program is assembled. An error condition diagnosed by the macro processor is reported on the source listing in the following format:

```
      *        Macroinstruction
E     MIC#     Diagnostic Error Message
I     Image of last macro definition record read in
```

Example:

```
      *        $GETD
E     5428     Invalid AIF Record
I              AIF (&AB EQ '1' .A)
```

However, there are some cases when the last macro definition record read in is of no value for debugging. Under these conditions the image of the last macro definition record read in will not be displayed.

ASM–5400 INVALID CONTINUATION ON MACRO CALL

Explanation: Positions 1-15 of a macro call statement contain a nonblank entry.

ASM–5401 INVALID OPERATION CODE

Explanation: The mnemonic operation code of the record being processed is not a valid System/34 assembler operation code.

ASM–5402 INVALID VARIABLE SYMBOL

Explanation: An invalid variable symbol was found. This error could be caused by an ampersand (&) in a comment.

ASM–5403 VARIABLE SYMBOL TABLE IS FULL

Explanation: The variable symbol table is full. (The user should split the job into smaller requests.)

ASM–5404 VARIABLE SYMBOL NAME NOT FOUND

Explanation: A reference has been made to an undefined variable symbol.

ASM–5405 GLOBAL VARIABLE REFERENCE INVALID

Explanation: A set symbol identified on a global or local record is also identified on a prototype or TABLE record within the same macro definition.

ASM–5406 INVALID CHARACTER STRING

Explanation: An invalid value exists on the record being processed:

- Null value when not permitted
- Value exceeds 50 bytes when decoded
- Value exceeds the limits of the record on which it appears

ASM–5407 SEQUENCE SYMBOL NOT FOUND

Explanation: A sequence symbol is missing or misspelled.

ASM–5408 MACRO DEFINITION NOT FOUND

Explanation: The macro definition was not found in the source library.

ASM–5409 INCOMPATIBLE ATTRIBUTES ENCOUNTERED

Explanation: A set symbol identified on a global record has been identified as another type of set symbol within a previous macro definition statement.

The attribute of a set symbol referenced in the name field of an SETA, SETB, or SETC record does not match its assigned attribute.

ASM–5410 INVALID GLOBAL OR LOCAL RECORD

Explanation: A format error occurred in an operand of a GBLA, GBLB, GBLC, LCLA, LCLB, or LCLC record.

ASM–5411 HEADER STATEMENT IS INVALID (MACRO)

Explanation: Misplaced control records following the text record within a macro definition.

ASM–5412 PROTOTYPE STATEMENT IS INVALID

Explanation: A prototype record has one of the following:

- Format error in an operand field
- Invalid entry in a name field
- Operation field name incorrect
- More than five prototype continuations

ASM–5413 INVALID KEYWORD ON MACRO CALL

Explanation: An invalid keyword was found on a macroinstruction.

ASM–5414 INVALID INPUT DATA ON MACRO CALL

Explanation: An invalid response to a keyword parameter was found on a macroinstruction.

ASM–5415 INVALID DELIMITER ON PROTOTYPE

Explanation: No dash follows the keyword in a keyword parameter on a prototype statement.

ASM–5416 INVALID CONTINUATION ON MACRO CALL

Explanation: The format of a macroinstruction is for a continuation record to follow but continuation is not indicated.

ASM–5417 TABLE RECORD WITHOUT TABDF RECORD

Explanation: A TABDF record does not follow a TABLE record.

ASM–5418 MEND STATEMENT OUT OF SEQUENCE

Explanation: A MEND record was found immediately following a TABLE record.

ASM–5419 INVALID RECORD BEFORE TEXT RECORD

Explanation: An error has been encountered in the placement of control records prior to the TEXT record within a macro definition. Invalid table record encountered.

ASM–5420 INVALID TABLE DEFINITION RECORD

Explanation: A table-definition record is invalid:

- The value does not start in position 16
- The argument is not left-justified starting in position 1
- The argument exceeds the limits defined for the record
- The mnemonic operation code (TABDF) is missing

ASM–5421 INVALID AGO RECORD

Explanation: An AGO record has an invalid sequence symbol.

ASM–5422 DEFINITION STATEMENTS OUT OF ORDER

Explanation: The macro definition records are not in the expected sequence.

ASM–5423 INVALID SEQ SYMBOL ON AGO STATEMENT

Explanation: The length of the sequence symbol is invalid.

ASM–5424 INVALID SETB RECORD

Explanation: An error exists in the format of a variable symbol required in the name field of an SETB record, or the operand is not 0 or 1.

ASM–5425 INVALID FORMAT ON MNOTE STATEMENT

Explanation: Invalid format on an MNOTE record.

ASM–5426 MODEL RECORD IS IN ERROR

Explanation: One of the fixed format fields of a model record has exceeded its defined limits. An entry in field 1 must begin in position 1.

ASM–5427 MODEL RECORD FIELD BUFFER EXCEEDED

Explanation: A value compared in the operand of an AIF record is more than 50 bytes long or has an invalid format. (Only symbolic parameters, set symbols, character strings, count functions, and type attributes are valid for comparison.)

A model record is more than 71 bytes long.

ASM–5428 INVALID AIF RECORD

Explanation: An error has been detected in the format of an AIF record.

ASM–5429 INVALID USE OF COUNT FUNCTION

Explanation: The count function is being used with other than symbolic parameters.

ASM–5430 ERROR IN SETA STATEMENT SYNTAX

Explanation: An error exists in the format of a variable symbol required in the name field of an SETA record, or the operand is blank.

ASM–5431 ERROR IN SETC STATEMENT SYNTAX

Explanation: An error exists in the format of a variable symbol required in the name field of an SETC record, or the operand is not enclosed with quotes and delimited by a blank.

ASM–5432 DECIMAL NUMBER IS INVALID

Explanation: Arithmetic term exceeds bounds of -8,388,608 to +8,388,607.

The value of a symbolic parameter or a decimal self-defining term exceeds maximum value of 65,535.

ASM–5433 BINARY TERM INVALID

Explanation: A position in a binary self-defining term is other than 0 or 1.

ASM–5434 EXPRESSION TERM INVALID

Explanation: An invalid operand or operator is used in an arithmetic expression. Valid operands are binary, character decimal, and hexadecimal self-defining terms; variable symbols; and count functions. Valid operators are addition (+), subtraction (-), multiplication (*), and division (/).

ASM–5435 CONSECUTIVE OPERATORS ENCOUNTERED

Explanation: Consecutive operators have been detected within an arithmetic expression.

ASM–5436 EXPRESSION ENDS WITH AN OPERATOR

Explanation: An arithmetic expression has been ended with an operator.

ASM–5437 INVALID HEX TERM

Explanation: A hexadecimal self-defining term contains an invalid hexadecimal digit.

ASM–5438 INVALID USE OF LEFT PARENTHESIS

Explanation: Improper placement of left parenthesis or more than 3 levels of nested parentheses within an arithmetic expression.

ASM–5439 NO OPERATOR FOR OPERAND

ASM–5440 CONSECUTIVE OPERANDS ENCOUNTERED

Explanation: Consecutive operands have been detected within an arithmetic expression.

ASM–5441 INVALID COMBINATION OF OPERATORS

ASM–5442 INVALID RIGHT PARENTHESIS

Explanation: A parenthesis is not paired properly.

ASM–5443 NULL VALUE IN ARITHMETIC EXPRESSION

Explanation: Arithmetic expressions cannot contain null values.

ASM–5444 CHARACTER EXPRESSION IS TOO LARGE

Explanation: There are more than the maximum number of bytes for a character expression.

ASM–5445 INVALID SUBSTRING TERM

Explanation: When evaluating a character expression, the value of either a term of a substring is negative or the substring term is 0.

ASM–5446 SUBSTRING SYNTAX ERROR

Explanation: Syntax error in use of substring character expression exceeds the limits of the input record.

ASM–5447 INVALID LABEL ON MACRO INSTRUCTION

Explanation: A macroinstruction label contains an invalid character.

ASM–5448 MNOTE MESSAGE NOT FOUND

ASM–5449 MACRO IS A BAD MEMBER

Explanation: The definition for a macroinstruction cannot be found.

ASM–5450 MISPLACED POSITIONAL PARAMETER

Explanation: All positional parameters must precede any keyword parameters.

ASSEMBLER

A flag of E precedes each error code in the error field on the assembly listing. After the assembly is complete, a table of statement numbers, MIC codes, and error messages is listed.

ASM–5500 INVALID NAME LENGTH

Explanation: The name field entry is greater than the maximum length allowed.

ASM–5501 INVALID CHARACTER IN NAME

Explanation: The first position of a name field entry starts with a nonalphabetic character or contains an invalid character.

ASM–5502 NAME NOT ALLOWED IN INSTRUCTION

Explanation: A name field entry was found on an instruction where one is not allowed.

ASM–5503 REFERENCE TO UNDEFINED SYMBOL

Explanation: The referenced symbol is not defined in this program.

ASM–5504 NAME REQUIRED ON THIS INSTRUCTION

Explanation: An EQU instruction does not have the required name field entry.

ASM–5505 PREVIOUSLY DEFINED SYMBOL

Explanation: This symbol has been previously defined in this program.

ASM–5506 MODULE NAME MISSING

Explanation: Either the START instruction is missing, or the START instruction is present but the name field entry (module name) is missing. The assembler program assigns the default module name ASMOBJ.

ASM–5508 INVALID OPERATION CODE

Explanation: Undefined operation field entry.

ASM–5509 INVALID ORIGIN

Explanation: There has been an attempt to change the value of the location counter to a value less than the initial value of the location counter using the ORG instruction.

ASM–5510 INVALID OR ILLEGAL ICTL

Explanation: There is an operand error on an ICTL instruction, or the ICTL instruction is not the first statement in the program. (The ICTL is treated as the last source statement in the program.)

ASM–5511 INVALID START INSTRUCTION

Explanation: The START instruction was encountered after the location counter was initialized.

ASM–5512 LOCATION COUNTER ERROR

Explanation: There is a location counter overflow (greater than 65535) or there has been an attempt to reference the location counter at 65536.

ASM–5513 MISSING END STATEMENT

Explanation: The END statement is missing from the program.

ASM–5516 INVALID OPERAND DELIMITER

Explanation: An operand field syntactical delimiter is either misplaced or missing.

ASM–5517 INVALID OPERAND FORMAT

Explanation: The operand field format is not correct for this instruction.

ASM–5518 MISSING OPERAND

Explanation: An operand field entry is missing from an instruction requiring one.

ASM–5519 INVALID SYNTAX IN EXPRESSION

Explanation: There has been a violation of one or more of the expression syntax rules.

ASM–5520 EXPRESSION VALUE TOO LARGE

Explanation: The final expression value is not in the range -2^{16} to $2^{16}-1$.

ASM–5521 INVALID OPERAND

Explanation: One or more operand entries do not meet the specifications for this instruction.

ASM–5522 ARITHMETIC OVERFLOW

Explanation: An intermediate expression value is not in the range -2^{24} to $2^{24}-1$.

ASM–5523 ADDRESSABILITY ERROR

Explanation: A relocatable displacement is outside the range of the USING instruction.

ASM–5524 REGISTER SPECIFICATION ERROR

Explanation: The index register specification is not 1 or 2.

ASM–5525 INVALID CONSTANT

Explanation: There is an error in a constant specification on a DC instruction.

ASM–5526 INVALID CONSTANT TYPE

Explanation: The data type specified in a DC or DS is not valid.

ASM–5527 INVALID DUPLICATION FACTOR

Explanation: There is an error in the duplication factor specification on a DC or DS.

ASM–5528 INVALID LENGTH SPECIFICATION

Explanation: There is an error in the length specification.

ASM–5529 INVALID STATEMENT DELIMITER

Explanation: The column following the statement field is not blank.

ASM–5530 RELOCATABLE MULTIPLICATION

Explanation: A relocatable term was used in a multiply operation.

ASM–5531 RELOCATABILITY ERROR

Explanation: A relocatable expression is used where an absolute expression is required; or an absolute expression is used where a relocatable expression is required.

ASM–5532 INVALID SYMBOL

Explanation: There is an invalid character in or invalid length of a symbol in the operand field.

ASM–5533 INVALID SELF-DEFINING TERM

Explanation: There is an error in the format of a self-defining term.

ASM–5534 SELF-DEFINING VALUE TOO LARGE

Explanation: The value of self-defining term is outside of the range of -2^{16} to $2^{16}-1$.

ASM–5535 INVALID IMMEDIATE FIELD

Explanation: The value of the immediate field is not in the range of X'00' to X'FF'.

ASM–5536 INVALID DISPLACEMENT

Explanation: The value of the absolute displacement is not in the range of 0 to 255.

ASM–5537 INVALID EXTRN

Explanation: The symbol is invalid, already defined in the program, or the subfield is invalid.

ASM–5538 TOO MANY ESL RECORDS

Explanation: More EXTRN and ENTRY statements were found in the program than are permitted. This count includes multiple EXTRNs and ENTRYs, ENTRYs with valid symbols which are not defined, and EXTRNs with valid symbols which are defined in the program.

The region size determines the number of permissible ESL records as given in the following table:

Region Size	Maximum Statements
14 - 18	85
20	125
22 - 26	170
28 - 34	210
36 and up	255

Appendix A. Samples

This appendix contains:

- A sample assembler program
- Sample macroinstruction definitions and related macroinstruction expansions

SAMPLE ASSEMBLER PROGRAM

```
THE LIST OF OPTIONS USED DURING THIS ASSEMBLY IS--   LIST,XREF,OBJ



                        IBM SYSTEM/34 BASIC ASSEMBLER-MACRO PROCESSOR            RELEASE 04
ASSMPL                         EXTERNAL SYMBOL LIST
                                                                  05-09-79   TIME 09.45  PAGE   1
SYMBOL TYPE
ASSMPL   MODULE



                        IBM SYSTEM/34 BASIC ASSEMBLER-MACRO PROCESSOR            RELEASE 04
ASSMPL
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT                         05-09-79   TIME 09.45  PAGE   2
                                 1        ICTL  1,71                                                      00020000
                                 2        ISEQ  73,80                                                     00030000
                                 3        PRINT NOGEN,NODATA                                              00040000


                        IBM SYSTEM/34 BASIC ASSEMBLER-MACRO PROCESSOR            RELEASE 04
ASSMPL DISK FILE TO PRINTER (80/80 LIST PROGRAM)
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT                         05-09-79   TIME 09.45  PAGE   3
                                 5 ********************************************************************************  00060000
                                 6 * THIS PROGRAM READS A FILE FROM THE DISK AND LISTS IT                       *  00070000
                                 7 * ON THE PRINTER.                                                            *  00080000
                                 8 *                                                                            *  00090000
                                 9 *   THERE ARE THREE POSSIBLE MESSAGES ISSUED BY THIS PROGRAM:                *  00100000
                                10 *     MESSAGE                    MEANING                                    *  00110000
                                11 *   'EOF ON SYSIN'          END OF FILE ENCOUNTERED FROM DISK READ.         *  00120000
                                12 *                            THE PROGRAM ISSUES THE MESSAGE                  *  00130000
                                13 *                            AND GOES TO EOJ.                                *  00140000
                                14 *   'PRINTER ERROR'         THERE HAS BEEN A PERMANENT PRINTER              *  00150000
                                15 *                            ERROR.  THE PROGRAM ISSUES THE                  *  00160000
                                16 *                            MESSAGE AND GOES TO END OF JOB.                 *  00170000
                                17 *   'SYSIN ERROR'           THERE HAS BEEN A PERMANENT READ                 *  00180000
                                18 *                            ERROR.  THE PROGRAM ISSUES THE                  *  00190000
                                19 *                            MESSAGE AND GOES TO END OF JOB.                 *  00200000
                                20 ********************************************************************************  00210000

    0800                        22 ASSMPL START X'0800'                                                         00230000

                                24 * PREPARE THE FILES FOR USE (DTFS ARE CHAINED)                               00250000

                                26 *        $ALOC DTF-DSKDTF              ALLOCATE ALL FILES                    00270000

                                33 *        $OPEN DTF-DSKDTF              OPEN ALL FILES                        00290000

                                40 * READ FROM SYSTEM SOURCE LIBRARY AND PRINT RECORDS UNTIL END OF FILE        00310000
                         0810   41 REDAGN EQU   *                                                               00320000
                                42 *        $GETD DTF-DSKDTF,ERR-SYSER,EOF-EOF,OP-NGET                          00330000

    0829 B5 02 09               54        L     $F1WKB(,XR2),XR2          POINT TO RETRIEVED RECORD           00350000
    082C 2C 4F 0847 4F          55        MVC   DSKREC(80),79(,XR2)       MOVE DATA TO PRINT BUFFER           00360000
                                56 *        $PUTP DTF-PRTDTF,ERR-PRNERR,SPACEA-1,PRINT-Y                        00370000

    0846 C0 87 0810             67        B     REDAGN                    BRANCH BACK AND READ AGAIN          00390000
```

```
                        IBM SYSTEM/34 BASIC ASSEMBLER-MACRO PROCESSOR              RELEASE 04
ASSMPL DISK FILE TO PRINTER (80/80 LIST PROGRAM)
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT                    05-09-79   TIME 09.45  PAGE   4
                                 69 * END OF FILE ON SYSIN                                                  00410000
    084A C2 02 0BD4              70 EOF    LA    EOFMSG,LOG                                                 00420000
                                 71 *        $LOG                  EOF MESSAGE                              00430000
    0852 C0 87 0869              76        B     EOJ                 INVALID REPLY, TRY AGAIN                 00440000

                                 78 * ERROR ON DISK READ                                                    00460000
    0856 C2 02 0BDC              79 SYSER  LA    SERMSG,LOG                                                 00470000
                                 80 *        $LOG                  DISK READ ERROR MESSAGE                  00480000
    085E F2 87 08                85        J     EOJ                 GO TO EOJ                                00490000

                                 87 * ERROR ON PRINTER                                                      00510000
    0861 C2 02 0BE4              88 PRNERR LA    PERMSG,LOG                                                 00520000
                                 89 *        $LOG                  PRINTER ERROR MESSAGE                    00530000

                                 95 * END OF JOB ROUTINE                                                    00550000
                        0869     96 EOJ    EQU   *                                                          00560000
                                 97 *        $CLOS DTF-DSKDTF      CLOSE ALL FILES                          00570000
                                103 *        $EOJ                  END JOB                                  00580000

                                109 * CONSTANTS AND DATA AREAS                                              00600000

                                111 * DISK FILE TABLES ETC.                                                 00620000
                                112 *SKDTF   $DTFD ACCESS-CG,RECL-80,NAME-INPUT,BLKL-512,IOAREA-INBUF,     *00630000
                                113 *              CHAIN-PRTDTF,RCAD-INRCRD                                 00640000

                                158 * BUFFER AND WORK AREAS FOR DISK INPUT INTERFACE                        00660000
                        0BD1    159 INBUF  EQU   *                                                          00670000
    0801                0AF7    160 IOB    DS    CL39                                                       00680000
    08F8                0AF7    161 INAREA DS    2CL256                                                     00690000
                        0AF8    162 INRCRD EQU   *                                                          00700000
    0AF8                0B47    163 DSKREC DS    CL80                                                       00710000

                                165 * PRINT FILE TABLES ETC.                                                00730000
                                166 *RTDTF   $DTFP RCAD-INRCRD,IOAREA-OUTPUT,RECL-80,NAME-FILENAME          00740000

                                196 * BUFFER AND WORK AREAS FOR PRINTER INTERFACE                           00760000
                        0B71    197 OUTPUT EQU   *                                                          00770000
    0B71                0BD3    198 IOAREA DS    CL99                                                       00780000

                                200 * SYSTEM LOG TABLES                                                     00800000

                                202 *OFMSG   $LMSG TYPE-2,SPACE-2,MSGLN-15,MSGAD-EOFMGC,WRSTE-N            X00820000
```

```
                        IBM SYSTEM/34 BASIC ASSEMBLER-MACRO PROCESSOR              RELEASE 04
ASSMPL DISK FILE TO PRINTER (80/80 LIST PROGRAM)
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT                    05-09-79   TIME 09.45  PAGE   5
                                219 *ERMSG   $LMSG TYPE-2,SPACE-2,MSGLN-15,MSGAD-SERMGC,WRSTE-N            X00840000

                                236 *ERMSG   $LMSG TYPE-2,SPACE-2,MSGLN-15,MSGAD-PERMGC,WRSTE-N            X00860000

                        0BEC    253 EOFMGC EQU   *                                                          00880000
    0BEC C5D6C640D6D540E2 0BFA  254        DC    CL15'EOF ON SYSIN   '                                      00890000

                        0BFB    256 SERMGC EQU   *                                                          00910000
    0BFB E2E8E2C9D540C5D9 0C09  257        DC    CL15'SYSIN ERROR    '                                      00920000

                        0C0A    259 PERMGC EQU   *                                                          00940000
    0C0A D7D9C9D5E3C5D940 0C18  260        DC    CL15'PRINTER ERROR  '                                      00950000

                                262 * OFFSETS FOR ALL DTFS DEFINED IN THIS PROGRAM                          00970000

                                264 *        $DTFD DISK-Y,PRT-Y,FIELD-Y                                     00990000

                                611 * REGISTER LABELS                                                       01010000
                        0002    612 $DTF   EQU   2                                                          01020000
                        0002    613 SYS    EQU   2                 SYSIN PARAMETER LIST POINTER             01030000
                        0002    614 LOG    EQU   2                 SYSLOG PARAMETER LIST POINTER            01040000
                        0002    615 XR2    EQU   2                 INDEX REGISTER 2 REFERENCE               01050000

                        0800    617        END   ASSMPL                                                      01070000
TOTAL STATEMENTS IN ERROR IN THIS ASSEMBLY--      0
TOTAL SEQUENCE ERRORS IN THIS ASSEMBLY--          0
```

SAMPLE MACROINSTRUCTIONS

The definitions of the IBM-supplied $PUTP (Construct a Printer Put Interface) and $LOG (Generate the Linkage to the System Log) macroinstructions are given here. The definitions are followed by a partial assembler program in which $PUTP and $LOG are issued. The macroinstruction expansions listed in the program show how the expansion generated for a macroinstruction is related to the definition of the macroinstruction.

Definition of $PUTP

```
         MACRO
&LABEL   $PUTP &DTF-,&PRINT-,&SKIPB-,&SPACEB-,&SKIPA-,                X
               &SPACEA-,&ERR-,&OVFL-
         TABLE &PRINT
Y        TABDF X'40'
N        TABDF X'00'
YES      TABDF X'40'
NO       TABDF X'00'
         TABLE &SPACEB
0        TABDF 00
1        TABDF 01
2        TABDF 02
3        TABDF 03
         TABLE &SPACEA
0        TABDF 00
1        TABDF 01
2        TABDF 02
3        TABDF 03
         TEXT
         AGO   .ARCR
*     COPYRIGHT=5726-SS1 COPYRIGHT IBM CORP 1977 LICENSED MATERIAL - *
*     PROGRAM PROPERTY OF IBM. REFER TO COPYRIGHT INSTRUCTIONS       *
*     FORM NUMBER G-120-2083.                                        *
.ARCR    ANOP
* LINKAGE TO PRINTER DATA MANAGEMENT
         AIF   (&LABEL EQ '').Z
&LABEL   EQU   *
.Z       ANOP
         AIF   (&DTF EQ '').A
         LA    &DTF,2                    XR2 ----> DTF
.A       ANOP
         AIF   (&PRINT EQ '').B
         MVI   11(,2),&PRINT             SET OP CODE IN DTF
.B       ANOP
         AIF   (&SKIPB EQ '').C
         MVI   26(,2),&SKIPB             SET SKIP BEFORE
.C       ANOP
         AIF   (&SPACEB EQ '').D
         MVI   27(,2),&SPACEB            SET SPACE BEFORE
.D       ANOP
         AIF   (&SKIPA EQ '').E
         MVI   28(,2),&SKIPA             SET SKIP AFTER
.E       ANOP
         AIF   (&SPACEA EQ '').F
         MVI   29(,2),&SPACEA            SET SPACE AFTER
.F       ANOP
         SVC   04,01                     TRANSFER CONTROL TO
         DC    XL1'13'                   DATA MANAGEMENT
         AIF   (&ERR EQ '').H
         CLI   10(,2),X'41'              PERMANENT ERROR ?
         BE    &ERR                      YES, GO TO ERROR ROUTINE
.H       ANOP
         AIF   (&OVFL EQ '').ENDP
         CLI   10(,2),X'48'              PAGE OVERFLOW ?
         BE    &OVFL                     YES, GO TO OVERFLOW ROUTINE
.ENDP    ANOP
* END OF EXPANSION
         MEND
```

Definition of $LOG

```
         MACRO
&LABEL   $LOG  &LIST-,&OPTN0-,&OPTN1-,&OPTN2-
         TEXT
         AGO   .LOG00                    SKIP COPYRIGHT GENERATION
* COPYRIGHT-5726-AS1 COPYRIGHT IBM CORP 1977 LICENSED MATERIAL
* - PROGRAM PROPERTY OF IBM. REFER TO COPYRIGHT INSTRUCTIONS
* FORM NUMBER G-120-2083.
.LOG00   ANOP
* LINKAGE TO SYSLOG ROUTINES
         AIF   (T'&LABEL EQ 'O').LOG01 LABEL NOT SPECIFIED?
&LABEL   EQU   *
.LOG01   AIF   (T'&LIST EQ 'O').LOG02  WAS LIST SPECIFIED
         LA    &LIST,2                 REGISTER 2 --> PARAMETER LIST
.LOG02   SVC   X'04',X'01'             CALL SYSLOG WITH REFRESH
         DC    XL1'05'                 SYSLOG RIB
         AIF   (T'&OPTN0 EQ 'O').LOG03
         TBN   10(,2),X'80'            WAS OPTION ZERO TAKEN
         BT    &OPTN0                  YES,GO TO OPTN0 ADDRESS
.LOG03   ANOP
         AIF   (T'&OPTN1 EQ 'O').LOG04
         TBN   10(,2),X'40'            WAS OPTION ONE TAKEN
         BT    &OPTN1                  YES,GO TO OPTN1 ADDRESS
.LOG04   ANOP
         AIF   (T'&OPTN2 EQ 'O').LOG05
         TBN   10(,2),X'20'            WAS OPTION TWO TAKEN
         BT    &OPTN2                  YES,GO TO OPTN2 ADDRESS
.LOG05   ANOP
* END OF EXPANSION
         MEND
```

Expansions of $PUTP and $LOG

```
                IBM SYSTEM/34 BASIC ASSEMBLER-MACRO PROCESSOR            RELEASE 04

ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT                  04-10-79   TIME 16.13  PAGE    N

                            2 *
                            3 *
                            4 *       THIS IS AN EXAMPLE OF A PARTIAL ASSEMBLER PROGRAM
                            5 *       SHOWING EXPANSIONS OF THE MACROS $PUTP AND $LOG
                            6 *       TO DEMONSTRATE HOW MACRO EXPANSIONS WORK
                            7 *
                            8 *
                            9 *        $PUTP DTF-PRTDTF,ERR-PRNERR,SPACEA-1,PRINT-Y
                           10+* LINKAGE TO PRINTER DATA MANAGEMENT
 0000 00 00 0000            11+         LA    PRTDTF,2                XR2 ----> DTF
 0004 BC 40 0B              12+         MVI   11(,2),X'40'            SET OP CODE IN DTF
 0007 BC 01 1D              13+         MVI   29(,2),01               SET SPACE AFTER
 000A F4 01 04              14+         SVC   04,01                   TRANSFER CONTROL TO
 000D 13                000D 15+         DC    XL1'13'                 DATA MANAGEMENT
 000E BD 41 0A              16+         CLI   10(,2),X'41'            PERMANENT ERROR ?
 0011 00 00 0000            17+         BE    PRNERR                  YES, GO TO ERROR ROUTINE
                           18+* END OF EXPANSION
                           19 *
                           20 *
                           21 *
                           22 *       $LOG
                           23+* LINKAGE TO SYSLOG ROUTINES
 0015 F4 01 04              24+         SVC   X'04',X'01'             CALL SYSLOG WITH REFRESH
 0018 05                0018 25+         DC    XL1'05'                 SYSLOG RIB
                           25+* END OF EXPANSION
                           27 *
```

Appendix B. EBCDIC

The coded character set for EBCDIC (extended binary coded decimal interchange code) is shown in the following table.

EBCDIC

		Main Storage Bit Positions 0, 1, 2, 3															
Main Storage Bit Positions		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
4, 5, 6, 7	Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	NUL	DLE	DS		¢	&	-						{	}	\	0
0001	1	SOH	DC1	SOS				/		a	j	~		A	J		1
0010	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
0011	3	ETX	DC3	TM / DC3						c	l	t		C	L	T	3
0100	4	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101	5	HT	NL	LF	RS					e	n	v		E	N	V	5
0110	6	LC	BS	EOB / ETB	UC					f	o	w		F	O	W	6
0111	7	DEL	IL	PRE / ESC	EOT					g	p	x		G	P	X	7
1000	8		CAN							h	q	y		H	Q	Y	8
1001	9	RLF	EM						`	i	r	z		I	R	Z	9
1010	A	SMM	CC	SM		¢	!										LVM
1011	B	VT	CU1	CU2	CU3		$	,	=								
1100	C	FF	IFS		DC4	<	*	%	@								
1101	D	CR	IGS	ENQ	NAK	(	)	_	'								
1110	E	SO	IRS	ACK		+	;	>									
1111	F	SI	IUS	BEL	SUB	|	¬	?	"								E0

Duplicate Assignment

Glossary

arithmetic expression: A conditional assembler expression that is a combination of arithmetic terms, arithmetic operators, and paired parentheses.

assembler: A computer program that prepares an object program from a source program written in a symbolic source language in which there is a one-to-one correspondence between the instruction formats and data formats coded and those used by the computer.

assembler instruction statement: A statement that controls the functions of the assembler.

base displacement addressing: An addressing method that involves setting up a base address from which other addresses can be calculated.

character string: A string consisting solely of characters.

checkpoint: A reference point in a program at which system and job status is recorded so that, if necessary, the program can be restarted at that point.

delete-capable file: A file that can contain records that are logically deleted, though no physical compression occurred when the records were deleted.

direct addressing: An addressing method that allows the programmer to represent a 16-bit instruction address by using an expression as an operand entry.

expression: An arithmetic combination of terms.

global: Available to all macroinstructions in an assembler source program.

keyword parameter: A parameter that consists of a keyword, followed by one or more values.

literal: See *self-defining term.*

locate mode: A way of providing data to the user by pointing to its location rather than by moving it.

location counter: A counter used to assign storage addresses.

machine instruction statement: A statement that represents a machine language instruction on a one-for-one basis.

macroinstruction statement: A statement that represents a sequence of machine and/or assembler instruction statements.

MIC: Message identification code.

move mode: A way of transferring data by identifying its location to data management.

operand: An entry that follows an operation code and further defines the operation to be performed.

positional parameter: A parameter that must appear in a specified location, relative to other positional parameters.

processor: A computer program that includes the compiling, assembling, translating, and related functions for a specific programming language.

relative addressing: A means of addressing instructions and data areas by designating their location in relation to the location counter or to some symbolic symbol.

self-defining term: A term whose value is inherent in the term.

sequence symbols: Labels used in coding macroinstructions that determine the sequence of macroinstruction definition statement processing.

set symbols: In assembler programming, a variable symbol used to communicate values during conditional assembly processing.

term: A single symbol, self-defining value, or location counter reference used only in the operand field of an assembler language statement.

Index

READER'S COMMENT FORM

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

Error in publication (typographical, illustration, and so on). **No reply.**

Page Number *Error*

Inaccurate or misleading information in this publication. Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

Page Number *Comment*

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Name _____

Address _____

_____

● No postage necessary if mailed in the U.S.A.

IBM System/34
Basic Assembler and
Macro Processor
Reference Manual

SC21-7705-3

Cut Along Line

IBM System/34 Basic Assembler and Macro Processor Reference Manual (File No. S34-21) Printed in U.S.A. SC21-7705-3

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N. Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM CORPORATION
General Systems Division
Development Laboratory
Publications, Dept. 532
Rochester, Minnesota 55901

Fold and tape | Please do not staple | Fold and tape

IBM

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30055
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30055
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM System/34 Basic Assembler and Macro Processor Reference Manual (File No. S34-21) Printed in U.S.A. SC21-7705-3