IBM System/36

Programming with Assembler

When You Are:	You Can Find Information In:
Planning to Install Your Computer	What to Do Before Your Computer Arrives or Converting from System/34 to System/36
Getting Your Computer Ready to Use	Setting Up Your Computer Performing the First System Configuration For Your System System Security Guide
Operating Your Computer	Learning About Your Computer Operating Your Computer
Operating and Using the Utilities	Source Entry Utility Guide Data File Utility Guide Creating Displays Work Station Utility Guide Utilities Messages
Programming Your Computer	Concepts and Programmer's Guide System Reference Sort Guide Work Station Utility Guide **Programming with Assembler** **Assembler Messages**
Communicating with Another Computer or Remote Device	(communication manuals) (communication message manuals)
Determining the Cause of a Problem	System Messages (message manuals) System Problem Determination

What Is Your Opinion of This Manual?

Your comments can help us produce better manuals. Please take a few minutes to evaluate this manual as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

FINDING INFORMATION

Y N Is the table of contents helpful?
What would make it more helpful?

Y N Is the index complete?
List specific terms that are missing.

Y N Are the chapter titles and other headings meaningful?
What would make them more meaningful?

Y N Is information organized appropriately?
What would improve the organization?

Y N Does the manual refer you to the appropriate places for more information?
List specific references that are wrong or missing.

UNDERSTANDING INFORMATION

Y N Is the purpose of this manual clear?
What would make it clearer?

Y N Is the information explained clearly?
Which topics are unclear?

Y N Are the examples clear?
Which examples are unclear?

Y N Are examples provided where they are needed?
Where should examples be added or deleted?

Y N Are terms defined clearly?
Which terms are unclear?

Y N Are terms used consistently?
Which terms are inconsistent?

Y N Are too many abbreviations and acronyms used?
Which ones are not understandable?

Y N Are the illustrations clear?
Which ones are unclear?

USING INFORMATION

Y N Does the information apply to your situation?
Which topics do not apply?

Y N Is the information accurate?
What information is inaccurate?

Y N Is the information complete?
What information is missing?

Y N Is only necessary information included?
What information is unnecessary?

Y N Are the examples useful models?
What would make them more useful?

Y N Is the format of the manual (shape, size, color) effective?
What would make the format more effective?

OTHER COMMENTS

Use the space below for any other opinions about this manual or about the entire set of manuals for this system.

YOUR BACKGROUND

What is your job title?

What is your primary job responsibility?

How many years have you used computers?

Which programming languages do you use?

How many times per month do you use this manual?

Your name
Company name
Street address
City, State, ZIP

No postage necessary if mailed in the U.S.A.

Cut Along Line

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
Information Development
Department 532
Rochester, Minnesota, U.S.A. 55901

Fold and tape | Please do not staple | Fold and tape

IBM

IBM System/38: Programming with Assembler (File No. S38-21) Printed in U.S.A. SC21-7908-3

IBM System/36

Programming with Assembler

Program Number 5727-AS1

Program Number 5727-AS6

File Number
S36-21

Order Number
SC21-7908-3

Fourth Edition (January 1986)

This major revision obsoletes SC21-7908-2.

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition. See *About this Manual* for a summary of major changes to this edition.

This edition applies to Release 4, Modification Level 0, of the IBM System/36 Assembler Program Product (Program 5727-AS1 and Program 5727-AS6), and to all subsequent releases and modifications until otherwise indicated in new editions.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Canada Ltd. Information Development, Department 849, 895 Don Mills Road, Don Mills, Ontario, Canada, M3C 1W3. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

About This Manual

Who should use this manual . . .

This *System/36 Programming With Assembler* manual is intended for the experienced programmer who will be using the System/36 Assembler and Macro Processor licensed program. This manual contains the following:

- The relationship of the assembler language (source code) to the machine language (object code)
- How to code, assemble, follow, and debug assembler programs
- How to create, store, and call macroinstructions.

How this manual is arranged . . .

This manual is arranged as follows:

- Chapter 1 explains the assembler language and the relationship between an assembler language and machine language. Some of the characteristics of the System/36 assembler language are presented.
- Chapter 2 presents the assembler language components, coding conventions and programming conventions.
- Chapter 3 describes the assembler instruction statements.
- Chapter 4 describes the macro processor and the coding of macroinstruction definitions. Some macroinstruction examples are given.
- Chapter 5 describes macroinstruction statements and IBM-supplied macroinstructions.
- Chapter 6 describes procedures for identifying and correcting assembler problems.
- Appendix A gives programming examples.
- Appendix B contains the EBCDIC and ASCII character sets and a list of the valid display screen symbols.
- Appendix C shows the assembler coding form, GX21-9279.
- Appendix D shows the assembler machine instruction formats and operation codes.
- Appendix E provides detailed information on disk file access methods.
- Appendix F describes the various return codes that are used with display station operations.

Note: Some terms will appear earlier in the manual than any discussion explaining them. If you do not understand a term, please refer to the index or to the glossary.

What you should know . . .

An understanding of the IBM System/36 architecture can be gained through the following manuals:

- *IBM System/36 Functions Reference Manual*, SA21-9436
- *IBM System/36 System Reference (SSP)*, SC21-9020
- *IBM System/36 Concepts and Programmer's Guide*, SC21-9019
- *IBM System/36 Learning About your Computer*, SC21-9018.

Users of the assembler and macro processor should have the following manuals available while coding programs, entering data, or clearing errors (debugging a program):

- *IBM System/36 Guide to Publications*, SC21-9015
- *IBM System/36 Assembler Messages*, SC21-7942
- *IBM System/36 Utilities Messages*, SC21-7939

Other manuals you might expect to use are listed under *If You Need More Information* in this section.

If you need more information . . .

You will find further information in these related publications

- *IBM System/36 Overlay Linkage Editor Guide*, SC21-9041
- *IBM System/36 Interactive Communications Feature Guide and Examples*, SC21-7911
- *IBM System/36 Interactive Communications Feature Reference*, SC21-7910
- *IBM System/34 IBM System/32 Scientific Macroinstruction Functions Reference Manual*, SA21-9275
- *IBM System/36 System Data Areas* , LY21-0592
- *IBM System/36 Source Entry Utility Guide*, SC21-7901
- *IBM System/36 System Problem Determination*, SC21-7919
- *IBM System/36 System Problem Determination*, SC21-9063
- *IBM System/36 Operating Your Computer - 5364*, SC21-9085
- *IBM System/36 Operating Your Computer - 5360, 5362*, SC21-9026
- *IBM System/36 Distributed Data Management Guide*, SC21-8011
- *IBM System/36 Getting Started with Interactive Data Definition Utility*, GC21-8003
- *Using System/36 Communications*, SC21-9082.

Information about linking assembler subroutines to programs written in higher-level languages is contained in the following publications:

- *IBM System/36 Programming with RPG II*, SC21-9006
- *IBM System/36 Programming with COBOL*, SC21-9007
- *IBM System/36 Programming with FORTRAN IV*, SC21-9005.

Assembler Coding Material

- *IBM System/34 System/36 Assembler Coding Form*, GX21-9279.

Summary of changes . . .

The following changes have been made for release 4, modification 0:

- A Problem Determination chapter has been added to assist in assembly-time, linkage-time, and execution-time problems. Details are provided in Chapter 6.

- Various technical and editorial changes have been made to improve the quality and usability of this manual.

Notes:

Chapter 1. Introduction to the IBM System/36 Assembler Language

This chapter introduces the assembler language, and explains the relationship between machine language and assembler language.

An assembler language is a set of labels that are used to represent the various machine language instructions available in a system. Most labels in the assembler language are simple and easy to remember. Each instruction will have values, addresses, and other parameters, which the assembler program uses to create all of the machine language code necessary to perform the desired task.

A machine language is the set of binary instructions that the system hardware can interpret and use to manipulate data. For instance, the following series of binary data is an instruction and its parameters:

```
0011      This

1100                instruction moves

1010      this byte
1111                      of data

0010      in
1111            to
1100                  this
1011                          address
```

The preceding instruction is easier to write, and much easier to understand, when it is written using IBM System/36 assembler language as:

```
Operation (Move Immediate)

          MVI  X'2FCB',X'AF'

              Address  Data
```

This instruction moves specified data to a selected location.

System/36 Assembler Language

The IBM System/36 assembler language gives you a convenient method to represent the machine instructions and related data needed to create a program. Before you run it, your assembler language source program is assembled into System/36 machine language (an object program) by the assembler program.

When using the assembler language you can refer to instructions, data areas, and other program elements by symbolic names you assign or by machine addresses. You have the EBCDIC bit pattern, and binary arithmetic capabilities available, and you have access to SSP control blocks such as the DTFs and the IOBs. Because programming with assembler is done at the most elementary level, it is possible for an assembler language programmer to write programs that will run more efficiently than some routine procedures (with their inherent compromises) generated by COBOL, RPG, or other high-level languages.

You do not have to write routines to handle IOBs and DTFs. You can use the IBM-supplied macroinstructions to perform system services and to support input/output devices. Macroinstructions usually represent a sequence of instructions. The macroinstruction processor scans for any macroinstructions you have used before an assembler language program is assembled. When a macroinstruction is encountered, the associated complete set of instructions (a definition) is combined with any parameters you used with the macroinstruction statement. This combination creates a series of assembler language statements that are inserted into the *source program* in place of your macroinstruction statement. The macroinstruction statement is changed into a comment and is printed in the listing of the program.

The Source Program

The statements that make up an assembler source program are entered through the source entry utility (SEU). The SEU assembler display resembles the assembler coding form with the primary fields defined.

After you have signed onto the system, sign on to SEU by entering:

```
SEU
```

After you enter the name of the member that contains the assembler program, the member type (S for source), and the appropriate library, SEU presents the Z-display. You can enter assembler source statements on the Z-display, but you will find it is easier to use the assembler display because the entry fields are defined for you. You will need to select the assembler display from the Select display. Press command key 3 to see the Select display.

When you have the Select display, select the ASSEM display. When the ASSEM display appears, you can enter the following types of assembler program statements:

- Assembler language statements
- Assembler instruction statements
- Macroinstruction statements
- Macroinstruction definition statements including prototype statements and definition control statements.

The requirements and formats of the various kinds of statements recognized by the assembler and the macro processor are described in later chapters of this manual.

For the additional information you need about SEU, see the *Source Entry Utility Guide*.

System Procedures and Considerations

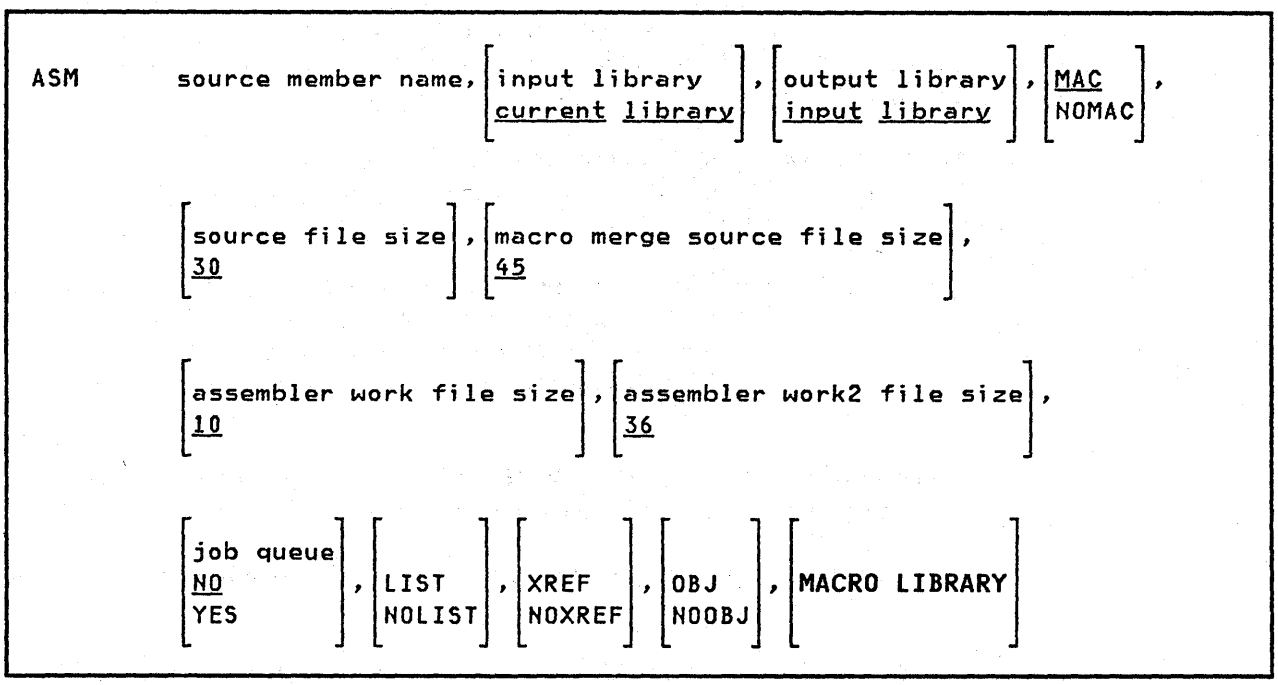
System/36 procedures are used to load and run the assembler and macro processor. These procedures, and the procedure commands that request them, are described here. (For a complete description of System/36 procedures and procedure commands, see the manual *System Support Reference*.)

ASM Procedure

The ASM procedure calls the assembler program and can call the macro processor.

ASM Procedure Command

The ASM procedure command requests the ASM procedure, which in turn calls the assembler and, optionally, the macro processor. If you enter ASM, or HELP ASM, or you type ASM and press the Help key; a display prompts you for parameters.



source member name: Specifies the source program name.

input library: Specifies the name of the library in which the source program is located. The current library is the default.

output library: Specifies the name of the library in which the object module will be placed. If omitted, the source library specified in the second parameter is assumed. If the second parameter is also omitted, the current library is the default.

MAC, NOMAC: Specifies the use or bypass of the macro processor.

MAC: Calls the macro processor.

NOMAC: Bypasses the macro processor.

The default is MAC.

source file size: Specifies $SOURCE file size, a 3-digit decimal number indicating the number of blocks required by $SOURCE. The default is 030.

$SOURCE provides source input to the macro processor. If the macro processor is not called, $SOURCE provides source input to the assembler. See *Files Used by the Assembler.*

macro merge source file size: Specifies $ASMINPT file size, a 3-digit decimal number indicating the number of blocks required by $ASMINPT. The default is 045.

$ASMINPT provides source input to the assembler if the macro processor is called. $ASMINPT contains the merged source program and macro processor-generated code. If the macro is not called, this file is not allocated and $SOURCE provides source input. See *Files Used by the Assembler.*

assembler work file size: Specifies $WORK file size, a 3-digit decimal number indicating the number of blocks required by $WORK. The default is 010.

$WORK contains the object code produced by the assembler. See *File Used by the Assembler.*

assembler work2 file size: Specifies $WORK2 file size, a 3-digit decimal number indicating the number of blocks required by $WORK2. The default is 036.

$WORK2 is used as a work file by the assembler. See *Files Used by the Assembler.*

job queue: Specifies placement of the job on the job queue.

NO: Does not place job on the job queue.

YES: Places the job on the job queue.

The default is NO.

LIST, NOLIST: Specifies the listing option to be used. If not specified, the LIST option specified on the assembler OPTIONS statement is used.

LIST: Specifies that the assembler is to produce a complete compiler listing including control statements, statements in error and associated diagnostics, and error summary statements.

NOLIST: Specifies that the assembler is not to produce the compiler listing. Only the prologue, the control statements, statements in error and associated diagnostics, and the error summary statements are printed.

XREF, NOXREF: Specifies the cross-reference option to be used. If not specified, the XREF option specified on the assembler OPTIONS statement is used.

XREF: Specifies that the assembler is to produce a cross-reference listing of the program.

NOXREF: Specifies that the assembler is not to produce a cross-reference listing.

OBJ, NOOBJ: Specifies whether the assembler should place the compiled program in the specified library. If not specified, the OBJ option specified on the assembler OPTIONS statement is used.

OBJ: Specifies that the assembler is to place the object (compiled) program in the library as a subroutine member.

NOOBJ: Specifies that the assembler is not to place the object (compiled) program in the library.

MACRO LIBRARY: Specifies the name of the library in which user macros are located. The order of library search will vary as follows:

- If MACLIB is blank, search #ASMLIB then #LIBRARY
- If MACLIB is #ASMLIB, search #ASMLIB then #LIBRARY
- If MACLIB is #LIBRARY, search #LIBRARY then #ASMLIB
- Otherwise, search USER-LIBRARY, then #ASMLIB, then #LIBRARY.

Note: If you specify a library that is not found, error message is issued.

OLINK Procedure

The OLINK procedure calls the Overlay Linkage Editor to create a load module. The OLINK procedure is described in the *Overlay Linkage Editor Guide*.

Files Used by the Assembler Program

The assembler program uses the following disk files:

- Source program records ($SOURCE and $ASMINPT files). $SOURCE contains source program records for the macro processor or the assembler. If the macro processor is not called, $SOURCE contains source program records for the assembler. If the macro processor is called, $ASMINPT contains source program records for the assembler.
- Intermediate text ($WORK2 file).
- Cross-reference file ($WORK2 file).
- Overflow symbol table(s) ($WORK2 file).
- Object program records ($WORK file).

If source records are 80 (rather than 96) positions long, they are padded on the right with 16 blanks before they are placed in the input file. In this case, you should provide an ICTL statement to prevent the assembler from processing the sequence field of the 80-column record.

$SOURCE, $ASMINPT, $WORK2, and $WORK are automatically allocated but their default sizes can be overridden by parameters in the ASM procedure command. These files are specified as extendable files; if the specified or default file sizes are not large enough, these files are made larger by an extent value when they become filled. The extent values in blocks are:

Work File	Extend Value
$SOURCE	20
$ASMINPT	30
$WORK	10
$WORK2	25

You can save extension overhead by specifying adequate file sizes. The number of blocks required for the $SOURCE and $ASMINPT files are:

Source Program Size (number of statements)	**Number of Required Blocks (one block = 2560 bytes)**
100	4
200	8
300	12
400	15
500	19
600	23
700	27
800	30
900	34
1000	38

Note: The number of generated statements should be included in the program size when calculating the size of $ASMINPT.

$WORK2 requires approximately 4 blocks (40 sectors) per 100 source statements:

Source Program Size (number of statements)	**Number of Required Blocks (one block = 2560 bytes)**
100	4
200	8
300	12
400	16
500	20
600	24
700	28
800	32
900	36
1000	40

$WORK contains the object records. One sector contains four 64-byte object records. The default is 10 blocks.

Assembler Listing

Printed output from the assembler includes the prologue, control statements, external symbol list, object code and source program, page heading, error messages (diagnostics), and cross-reference list.

Note: A printer is required to print the assembler prologue and error messages.

Prologue

The prologue contains procedure parameters, modification information, and a list of options in effect during an assembly.

Control Statements

Any OPTIONS or HEADERS control statements you specify are printed and any specification errors are noted.

External Symbol List (ESL)

The ESL contains the object program name, EXTRNs and ENTRYs, which are printed in the following format:

Symbol	Type
Object program name	MODULE
EXTRN symbol	EXTRN
ENTRY symbol	ENTRY

Object Code and Source Program Listing

The following items are printed for each entry in the source and object programs:

(ERR) Error Field: Which contains an E, I, W, or M for those statements in error. (Severity codes are described under MNOTE in Chapter 4.)

E Assembler and macro processor errors

W MNOTE warnings with a severity of 8

I Informational messages from the macro processor

M MNOTE errors with severity greater than 8

(LOC) Location Counter: Which is a 4-digit hexadecimal number representing the leftmost byte of any object code printed on this line.

Object Code: Which is translated code. All code in this field is left-justified. The parts of the object code are:

- 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 between the data constants.
- (ADDR) Address Field: Blank except for the following:
  - For the DC and DS instructions: The address of the reference byte, that is, the rightmost byte of the field.
  - For the END instruction: The address to which control is passed to start the program.
  - For the USING instruction: The address referenced in the first operand field.
  - For the DROP instruction: The register dropped (0001 or 0002).
  - For the EQU instruction: The value of operand 1.
  - For the ENTRY instruction: The entry point address.

(STMT) Statement Number Field: Which contains the sequential source statement number. All source statements, including comments, are numbered. Valid SPACE, EJECT, and TITLE statements are always assigned statement numbers but are not printed. The statement number field is a 4-character field; therefore, the program listing is accurate for only 9999 statements.

Source Statement: Which is a reproduction of the source record. All source records, except for the listing control statements (SPACE, EJECT, and TITLE) are printed as follows:

Column	**Item**
1	Error flag
5 through 8	location counter
10 through 25	Object code
27 through 30	Address
32 through 35	Statement number
36	A plus sign (+) indicating that a source statement generated by the macro processor follows.
37 through 132	Source statement

Page Heading

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/36 Assembler and Macro Processor Program Product, with an identifier of the release level.
- The content of the current TITLE statement.
- A short description of the contents of the various fields of the source program and object code listing, the current date and time, and the page number.

Diagnostics

The printed list of the source program and object code includes error codes for improperly coded statements. These codes are documented at the end of the source program and object code listing under the heading *Diagnostics:* The diagnostic list provides the following information:

- *Statement:* A decimal number assigned by the assembler to the statement in error.
- *Error code:* A 4-digit code. See *Assembler Messages*, SC21-7942, for a complete list of these codes and the corresponding messages.
- *Message:* A description of the error and the type of error.
- The number of sequence errors in the assembled program if a sequence check was requested.

The number of statements in error in the assembly does not include a missing module name and missing end statement errors.

Cross-Reference List

If XREF is specified for an assembly, a list of all symbol names referred to in the source program is generated. This list contains the following information:

- SYMBOL: The symbol name.
- LEN: The decimal length of the symbol.
- VALUE: The hexadecimal value of the symbol.
- DEFN: The decimal number of the statement that defines the symbol.
- REFERENCES: The decimal numbers of the statements that reference the symbol. Each reference by symbol to a data area or machine register that can be altered by a machine instruction is flagged with an asterisk.

At the end of the cross-reference listing, the error summary statements are printed again.

Statements in the Assembler Source Program

An assembler language source program is a set of assembler language statements that perform some task for you. Each statement has an identification-sequence number associated with it. When you use the coding sheet, the assembler language statement is entered in positions 1 through 87, position 88 is always left blank, and the identification-sequence number is entered in positions 89 through 96. You can change these position numbers with an ICTL statement.

There are three types of assembler language statements:

- Machine instruction statements that represent machine language instructions on a one-for-one basis.
- Assembler instruction statements that cause the assembler to perform various operations while your source program is being assembled. These instruction statements are not translated into machine language.
- Macroinstruction statements that represent a sequence of machine instruction statements, assembler instruction statements, or both.

Assembler Coding Form

The assembler coding form is shown in Appendix C. The following material describes the use of the coding form.

Valid Characters

Assembler statements can be written using these characters:

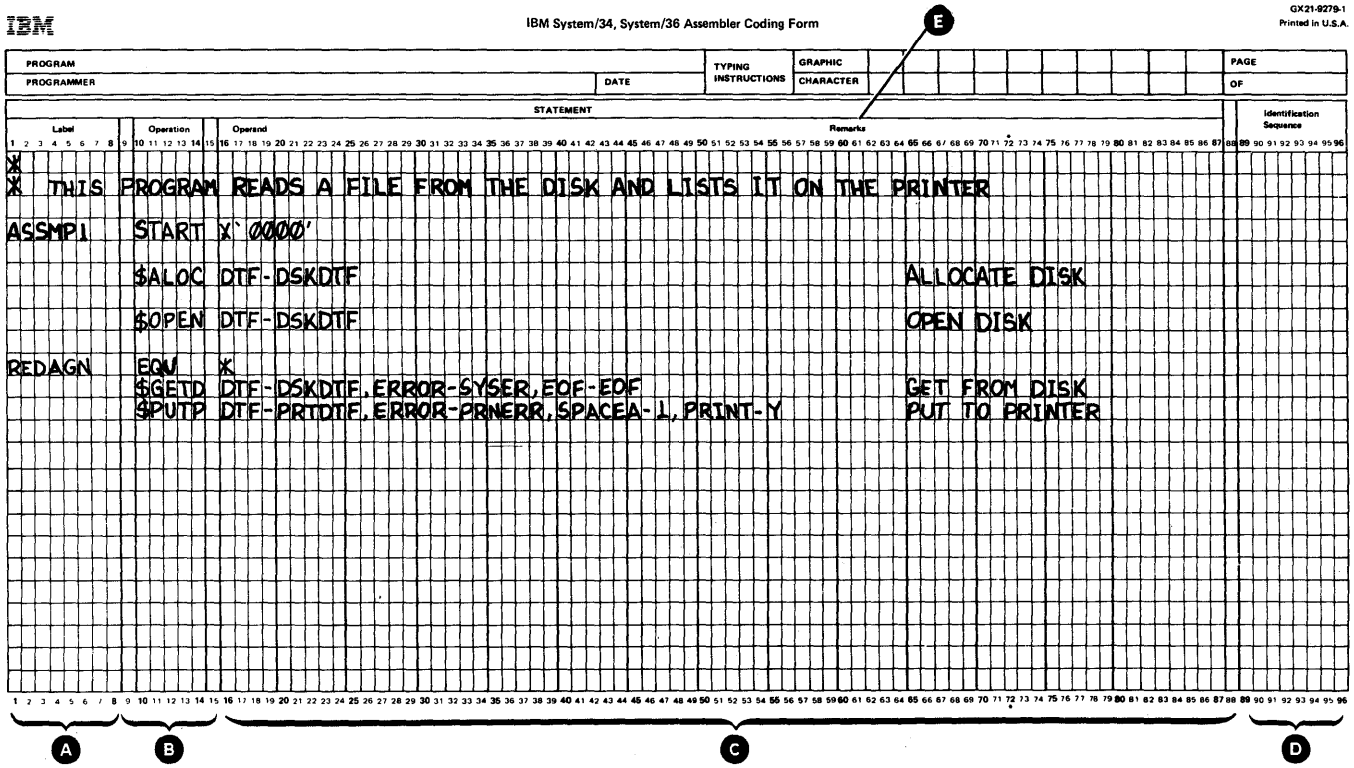
Alphabetic characters	A through Z, and $ # @ _
Digits	0 through 9
Special characters	+ - , . *) (' blank

In addition to these characters, any valid character that you can enter with your input device, with the exception of the ampersand (&) in a macroinstruction definition, can be placed between single quotes, or in the remarks and comments. Note that not all printers are able to print all characters, even though the assembler accepts them as input. There might be some display stations at which you can enter characters that other stations cannot display.

Coding Form Parts

The coding form has four fields: LABEL, OPERATION, OPERAND, and IDENTIFICATION SEQUENCE. The REMARKS heading is provided only as a reminder.

A blank is used to separate the parts of the assembler statement. The following illustrates the parts of a statement on a coding form.



Ⓐ *Label*

A label is any symbol that a programmer uses to identify either an assembler language statement, a storage area, or a value used by the program. A label is allowed in most instructions.

The first character of a label must be alphabetic. Enter it in the first position (position 1 on a coding form) of the language statement. If the first position is blank, the assembler program does not treat the rest of the entry as a label. The remaining characters can be letters or numbers, but not special characters. Follow the label by at least one blank.

Ⓑ *Operation*

An operation is the mnemonic term used to identify a particular machine instruction, assembler instruction, or macroinstruction. You must enter an operation for each assembler language statement. This entry can start in any position except position 1 and must be followed by at least one blank.

Ⓒ *Operand*

An operand identifies and describes the data to be acted upon by the operation. The operand indicates storage locations, registers, masks, number of bytes of storage affected, or types of data. Operands are required in all machine instructions. An operand is defined as a term, or an arithmetic combination of terms.

Separate the operand entries with commas, with no blanks between operands. The last operand of a language statement must be followed by at least one blank.

Ⓓ *Identification Sequence*

You can use positions 89 through 96 to enter program identification or statement identification sequence numbers. These numbers are used to ensure that the statements are in order. During assembly, you can have the assembler verify the sequence of the source statements by using the input sequence instruction (ISEQ).

Ⓔ *Remarks*

You can provide descriptive information about the program in a remark. Use any valid character available on your input device, except the ampersand within a macroinstruction definition. Separate the remark from the operand with at least one blank, and do not extend a remark past position 87. If there is no operand, separate the operation from the remark with a comma (,). Remember that every character you can enter might not be available on your printer.

Comment Statements

The entire statement field, to position 87, can be used as a comment if you place an asterisk (*) in the first position (position 1) of a statement. You can make a comment several lines long by placing an asterisk at the beginning of each line. Your comments can be placed anywhere in the source program except before HDR OPTIONS and ICTL. They do affect the storage requirements of the source program, and the assembly time. However, they do not affect the running

or storage requirements of the assembled program. The comments are printed in the assembler listing and can be an aid in following the listing.

Records in the Assembler Object Program

The assembler program converts the source program into control information, machine language instructions, and data, all of which make up the object program. There is one object program produced per assembly.

An object program contains three types of records:

- ESL (external symbol list) record
- TEXT-RLD (text-relocation directory) records
- END record.

Each object record is produced as a 64-byte field.

ESL Record: Contains the object program name, module name and all EXTRN and ENTRY symbols. The ESL record format is:

- Byte 1: Record type identifier S
- Byte 2: Length minus 1 of the ESL record
- Bytes 3 through 62: ESL record
- Bytes 63 and 64: 0's.

TEXT-RLD Records: Combination of text records and RLD pointers. The text portion of each record contains the object code for the program; 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 minus 1 (of text only).
- Bytes 3 and 4: Assembled address of the low-order (rightmost) text byte in the record.
- Bytes 5 through 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 hex 0's that fill the space between the text and RLD sections. The end of text is always followed by at least 1 byte of hex 0's.

END Record: Contains the entry address of the object program. If there is not an operand in the source program END statement, the object program END record generated by the assembler contains the hexadecimal address FFFF. The format for the END record is:

- Byte 1: Record type identifier E
- Bytes 2 and 3: Entry address of the object program
- Bytes 4 through 64: Reserved.

Communications Programs

Data Communications Programming with SSP-ICF

You can use assembler to program applications for SSP-ICP. Refer to the *SSP-ICF Guide and Examples* and to the SSP-ICF reference manuals for detailed information. If you are programming for binary synchronous communications for the IBM 3270, refer to the *3270 Device Emulation Guide*, SC21-7912.

Data Communications Programming with BSC

The IBM System/36 assembler provides binary synchronous communications (BSC) macroinstructions for batch BSC support. BSC macroinstructions let you write programs that send and receive data over communications lines. The BSC support performs all functions necessary to connect exchange identification sequences, send and receive data, and use the correct termination or disconnect procedures.

System/36 BSC support runs as a separate task from the assembler program, allowing the assembler program to be swapped into and out of main storage. The BSC task requires 4K bytes of main storage that will not be swapped and up to 8K per line for mapping to your buffers.

The BSC data management program that runs under control of your task is required. This program can be swapped, and requires 10K of user area.

Other Systems with BSC

You can have binary synchronous data transfers between System/36 and the following:

- Another System/36 with assembler, RPG II, or BSCEL subsystem
- System/34 with basic assembler, RPG II, or BSCEL subsystem
- System/32 with either basic assembler or RPG II
- System/3 with RPG II, MLMP, or CCP
- System/7 with MSP/7
- Operating System or Disk Operating System Basic Telecommunications Access Method (OS, OS/VS, DOS/VS, or DOS BTAM)
- System/360 Model 20 Input/Output Control System for the Binary Synchronous Communications Adapter
- Customer Information Control System (CICS/DOS/VS or CICS/VS)
- Information Management System (IMS/VS)
- IBM 3741 Model 2 Data Station or Model 4 Programmable Work Station
- IBM 3747 Data Converter
- IBM 5231 Data Collection Controller Model 2 (as a 3741 in transmit mode only)
- IBM 3750 Switching System (World Trade only)
- IBM 5110 (in 3741 mode)
- IBM 5120 (in 3741 mode)
- IBM Series/1 (in System/3 mode)
- IBM 5260 Point of Sale Terminal (in 3740 mode)
- IBM 5280 Distributed Data System (in 3740 mode)
- IBM 6640 Document Printer
- IBM Office System/6 Information Processor
- IBM Magnetic Card II Typewriter – Communicating
- IBM 6670 Information Distributor
- IBM 6240 Magnetic Card Typewriter – Communicating

- IBM Displaywriter System
- IBM System/38 with RPG III.

Note: System/36 data communications operation procedures are explained in the *System/36 Operator's Guide.*

Magnetic Character Reader

There are two subroutines that provide you with a methods for processing document information read by the 1255 Magnetic Character Reader. This two subroutines are part of the System Support Program (SSP) and are described as follows:

1. SUBR08

   System and stacker specifications describe the job to be done by the 1255. These system and stacker specifications are specified by the programmer as compile-time arrays for RPGII and COBOL. They are hard coded in the Assembler Source Program.

2. SUBR25

   A device control language (DCL) program describes the job to be done by the 1255. The SUBR25 parameter list is the data management interface between SUBR25 and the DCL program. The parameter list replaces the system and stacker specifications in the RPGII, COBOL or Assembler program. The DCL program is a separate program that runs in the attachment I/O controller for the 1255. The DCL source program consists of assembler-like statements that are actually MACROS that need to be expanded by the System/36 Assembler Macro processor.

Refer to the *Using and Programming the 1255 Magnetic Character Reader* manual for a detailed explanation of how to use the 1255 MCR. The manual contains in-depth explanations of the following:

- SUBR08 and SUBR25
- System and Stacker Specifications
- SURBR25 Parameter List and Device Control Language program
- Input record format.

Notes:

Chapter 2. Using IBM System/36 Assembler Programming Language

Before You Write an Assembler Language Program

Before you write an assembler program, you must be familiar with the meanings of certain terms, coding conventions, and other features of the assembler language. It is also necessary to understand the rules and conventions of the language.

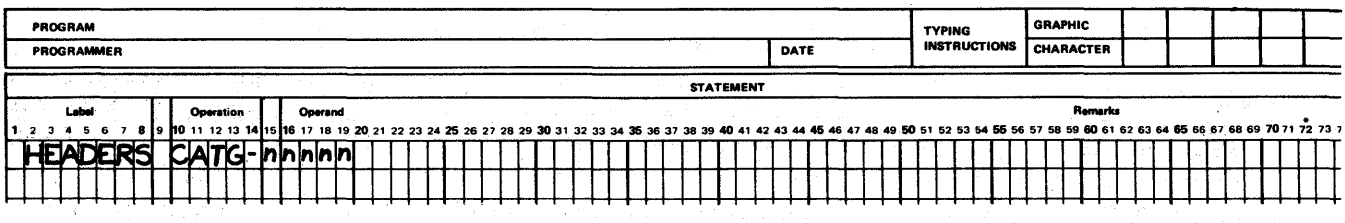
Assembler Control Statements

You can use two types of control statements: HEADERS and OPTIONS. A total of 45 control statements can be used, in any order. Each statement is limited to six operands. All control statements must be placed ahead of assembler source statements.

HEADERS Statement

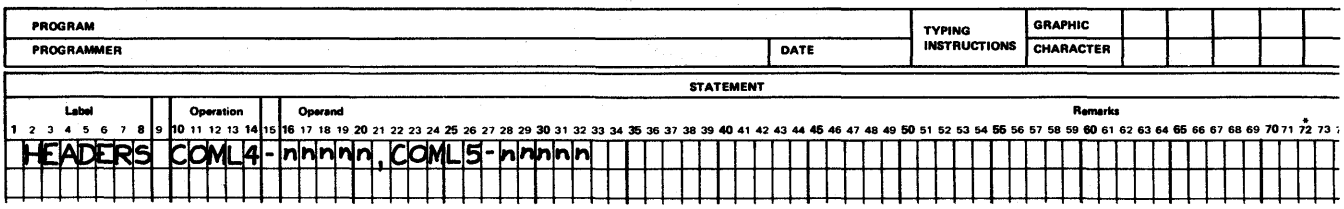
The HEADERS control statement specifies control information, other than output control, to the assembler. You can specify a category level for the object module through the CATG operand, or the length of the control section for any subtype 4 or 5 EXTRNs in the assembly through the COML4 and COML5 operands. For an explanation of category levels and subtype 4 and 5 EXTRNs, see the *System/36 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 of a value less than 256. If more than one CATG operand appears in the assembler control statements, the value of the last valid operand is used to designate the module category level. The module category level is placed in the module external symbol list (ESL) record. The HEADERS keyword can start in any column (except column 1), and must be preceded by a blank. One or more blanks are required between the keyword and the selected options. Blanks are not allowed between options.

The format of the HEADERS statement with the COML4 and COML5 operands is:

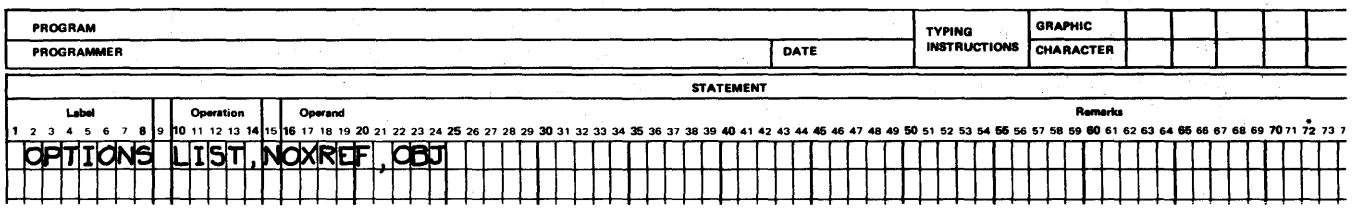


nnnnn is a 1- to 5-character decimal string whose value must be less than 65536. If more than one COML4 or COML5 operand is present in the assembler control statements, the length in the last valid operand is used for the appropriate subtype control section length. The lengths specified are placed in the ESL records for the subtype 4 or 5 EXTRNs. The HEADERS keyword can start in any column (except column 1), and must be preceded by a blank. One or more blanks are required between the keywords and the selected options. Blanks are not allowed between options.

OPTIONS Statement

An OPTIONS statement supplies assembler control options. All OPTIONS statements must be placed before the source statement with the HEADERS statement(s). You can specify the following assembler options on OPTIONS statements: LIST, NOLIST, XREF, NOXREF, OBJ, NOOBJ. Several options can be placed on one statement in any order, separated by commas. If you prefer, separate statements can be used for each option. The OPTIONS keyword can start in any column (except column 1), and must be preceded by a blank. One or more blanks are required between the keyword and the selected options. Blanks are not allowed between the selected options.

The following example shows options placed in one statement:



The following list briefly describes 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: • Prologue and control statements • 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 printed.
NOXREF	A cross-reference listing is not printed.
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 you the choice of either ending the assembly or placing the object program in the library as a subroutine member. The defaults are: LIST XREF OBJ

You can override (replace) the OPTION statement for an assembly by specifying the options in the ASM procedure (see the *ASM Procedure Command* in Chapter 1).

Assembler Program Conventions

You must follow certain conventions and rules for using terms and expressions, for addressing, and for linking references.

Programs are assembled to begin at address 0000, unless you use the START statement to specify a different address. If you do not specify a 2K boundary when you specify the beginning address, the program will begin at the next higher 2K boundary.

When a data constant is used in the operand of an instruction, the constant's address is assembled into the instruction.

You should understand the meanings of *expression* and *term* to follow the rest of this chapter. A *term* is a single symbol, a self-defining value, or a location counter reference in the operand 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 and makes up the operand field of an assembler language instruction.

Terms and expressions are classed as either absolute or relocatable. (see table on page 1-14)A term or expression is absolute if its value is not changed when the assembled program in which it is used is loaded into main storage and is relocatable if its value depends on addresses within the program.

Terms

Every term represents a value, a constant. This value can be assigned by the assembler program (for symbols and for location counter references) or can be part of the term itself (self-defining).

An absolute term is a nonrelocatable symbol, or any of the self-defining terms. Arithmetic operations are permitted between absolute terms.

Symbols (Symbolic Terms)

A symbol is a character or combination of characters used to represent a storage location, a register, or a value.

Symbols can be used as label entries and operand entries. This is an easy way to name and refer to a field or an instruction. Symbols used as label entries in a source statement are assigned values and a length attribute by the assembler.

The value assigned to a symbol in the label entry of a machine-instruction statement is the address of the *leftmost* byte of the storage location containing the statement. 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 storage areas and constants are considered relocatable terms because the address of an area or constant might change.

A symbol that is a label entry in an equate symbol assembler instruction statement (EQU) is assigned the length and the value designated in the operand of the statement. You can have the operand represent a relocatable value or an absolute value. The length attribute of the symbol on an EQU instruction is the length of the operand entry (or operand 2, if specified).

The value of a symbol cannot be a negative number and cannot exceed 65535. The length attribute, if specified, must be from 1 to 256.

EQU statements require that a symbol in the operand be previously defined. A symbol is defined when it appears as the label of a source statement or the operand of an EXTRN instruction.

A symbol used as a label in a statement or operand of an EXTRN instruction can be defined only once in an assembly.

Constants (Self-Defining Terms)

A constant is a term whose value is shown by the term. When a self-defining term is used in a machine-instruction statement, its value is assembled into the instruction. It is not assigned a value by the assembler. For example, the decimal term 15 represents a value of 15.

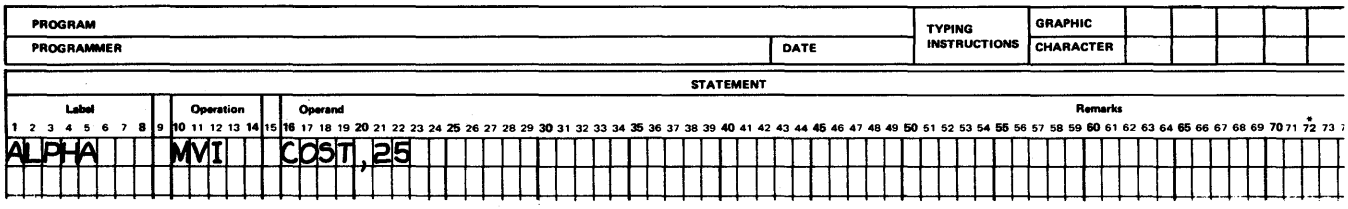
Constants have the following characteristics:

- The length attribute of a constant term is always 1.
- Constants specify machine values or bit configurations without equating the values to symbols and then using the symbols.
- Constants are always right-justified. Truncation and padding with 0's occurs from the left.

The four types of constants are: decimal, hexadecimal, binary, and character. Each term is a representation of a corresponding machine-language bit configuration. Constant terms are used to specify program elements such as immediate data, masks, registers, displacements, lengths, addresses, and address increments. The use of decimal, hexadecimal, binary, or character constants depends on what you are specifying.

Decimal Constants: Consist of decimal numbers written as a set of decimal digits. High-order 0's can be used (for example, 007). A decimal constant is assembled as its binary equivalent. A decimal constant cannot consist of more than 5 digits or exceed 65535. A decimal constant that represents an address should not exceed the size of storage. Some examples of decimal constants are: 8, 147, 4092, and 00,021.

In the following example, a decimal constant is used in a move immediate (MVI) machine instruction. The 1-byte area referred to by the symbol, COST, will contain the decimal value 25 (binary 0001 1001) after the instruction in an assembled program is performed.

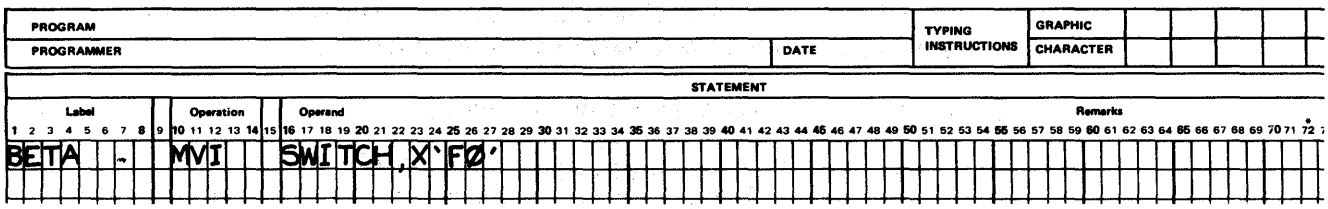


Hexadecimal Constants: Consist of the letter X (coding character for hexadecimal), followed by one to four hexadecimal digits (0 through 9, A through F) enclosed in apostrophes. An example is X'4A9'. Each hexadecimal digit is assembled as its 4-bit binary equivalent, in this case, as 0100 1010 1001. The largest hexadecimal constant is hex FFFF.

The hexadecimal digits and their bit patterns are as follows:

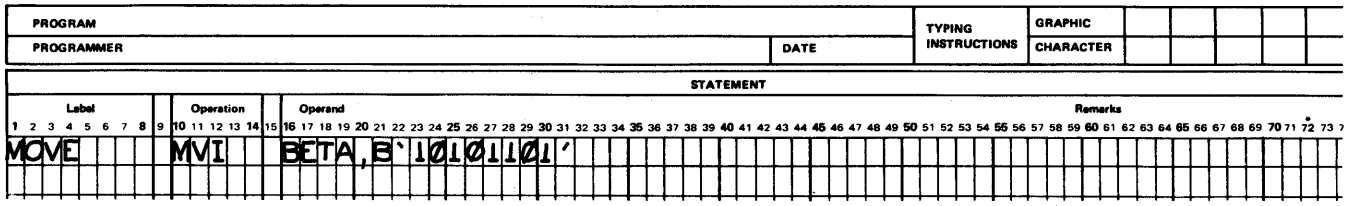
Digit	Bit Pattern	Digit	Bit Pattern
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

The following is an example of a hexadecimal constant term. The 1-byte area referred to by the symbol SWITCH would contain the hexadecimal value F0 (binary 1111 0000) after the instruction is performed.



Binary Constants: Consist of the letter B (coding character for binary), followed by an unsigned sequence of 1's and 0's enclosed in apostrophes, as follows: B'10101101'. This term would occupy 1 byte of storage. A binary constant can have up to 16 bits. Binary terms are used in logical operations or to designate bit patterns of masks.

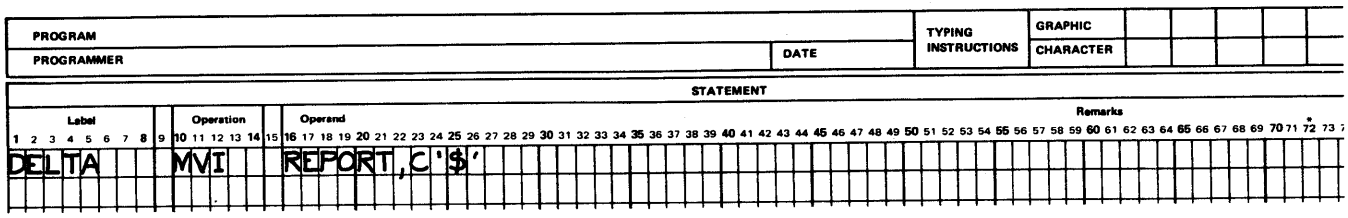
The following example illustrates a binary term used as immediate information in a move immediate (MVI) machine instruction. The byte of information replaces the byte of information referred to by the symbol BETA.



Character Constants: Consist of the letter C, followed by characters enclosed in apostrophes. Letters, decimal digits, and special characters can be used in a character constant. In addition, any character available on an input device can be entered. The following are examples of character constants:

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

Each character in a sequence is assembled as its EBCDIC equivalent. Because apostrophes are used as characters in the syntax of the assembler language, two apostrophes must be written as input for each apostrophe desired in a character constant as output. For example, you would write the character value A' as C'A''. In the following example, a dollar sign ($) would be moved into the 1-byte field at REPORT.



Expressions

An expression is an arithmetic combination of terms. The arithmetic operators used to combine terms into expressions are:

```
+  Addition

-  Subtraction

*  Multiplication
```

The following are examples of valid expressions:

AREA+X'2D'	**N-25**	**5*X'C1'**	***+15**
AREA value plus a hexadecimal 2D	N value minus a decimal 25	Decimal 5 times the hexadecimal C1 (result is decimal 965)	Current value of the location counter plus decimal 15

The rules for coding an expression are:

- Blanks are not allowed in an expression.
- Parentheses cannot be used in an expression.
- In a multiplication operation, only absolute terms can be used.
- Two terms or two operators must not be used consecutively in an expression.
- When an expression contains an external symbol, the symbol must have the form A or A $\pm$ e, where A is the symbol used as the operand of an EXTRN statement and e is an absolute expression. Any symbol equated to an expression like this cannot be used in an expression that has more than one term.

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

- Each term is evaluated separately.
- Arithmetic operations are performed in a left-to-right sequence, with multiplication operations performed before addition or subtraction operations. For example: A + B*C would be evaluated as A + (B*C), not (A+B)*C.

The intermediate result of an 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 result of an expression is a 2-byte value, 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 (-2^{16} through

2^{16}-1). The result will not be a negative number. A negative result is considered to be a 2-byte positive value.

Absolute Expressions: Contain relocatable terms or a combination of relocatable and absolute terms under the following conditions:

- The expression must contain an even number of relocatable terms.
- The relocatable terms must be paired, and each pair must consist of terms with opposite signs. Paired terms need not be next to each other.
- Relocatable terms cannot be used in a multiplication operation.

Because both terms would be relocated by the same value, pairing relocatable terms with opposite signs cancels the effect of the relocation. The value represented by the paired terms remains 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: Contain a combination of relocatable and absolute terms under the following conditions:

- There must be an odd number of relocatable terms.
- All relocatable terms, except one, must be paired, and each pair must consist of terms with opposite signs. The paired terms need not be next to each other.
- The unpaired term must not immediately follow a minus sign.
- Relocatable terms cannot be used in a multiplication operation.
- Every relocatable expression must reduce to a positive value.

All of the terms in a relocatable expression are combined and reduced to a single value. This is the adjusted value of the unpaired relocatable term after it is 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 this program is relocated by 100 bytes, the resulting 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. Given the values A = 3, B = 2, D = 5, F = 1, and G = 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.

The value of this expression after relocation can be determined by adding the amount of relocation to the relocatable result.

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 increased by the number of bytes required for the assembled item. The counter always points to the next available location. If an instruction statement is labeled by a symbol, the label is assigned the value of the location counter before addition of the assembled length. If a statement defines storage or a constant, the symbol is assigned a value one less than the value of the location counter after the addition of the assembled 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 65535.

You can refer to the current value of the location counter 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:

Location counter = 1300

	Source	**Generated**
Relocatable	LAB2 DC AL2(*) LAB3 DC AL2(LAB2)	1300 1301
Absolute (nonrelocatable)	LAB2 DC AL2(X'1300')	1300

Addressing

The two methods of addressing any part of storage are direct addressing and base displacement (relative) addressing.

Direct Addressing

Direct addressing allows you to represent a 16-bit instruction address by using an expression as an operand. The assembler places the value of the expression in the machine instruction as 2 bytes. A direct address never refers to a register in the operand. See the following figure for an example of direct addressing.

```
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT

    0000                         1 EXAMP    START 0
                                 2          PRINT NODATA,NOGEN

                                 4 ************************************************************************
                                 5 *                                                                      *
                                 6 *        AN EXAMPLE OF DIRECT ADDRESSING.                              *
                                 7 *                                                                      *
                                 8 ************************************************************************

    0000 0C 1D 0068 0033         10         MVC   NAME2(30),NAME1        MOVE "NAME" OF AREA1 TO "NAME" OF A
    0006 0C 07 0070 003B         11         MVC   PHON2(08),PHON1        MOVE "PHON" OF AREA1 TO "PHON" OF A
    000C 0C 0E 007F 004A         12         MVC   CITY2(15),CITY1        MOVE "CITY" OF AREA1 TO "CITY" OF A
                                 13 *       $EOJ                         END OF JOB


                          0016   19 AREA1   EQU   *
    0016 D1D6C8D540D14B40 0033   20 NAME1   DC    CL30'JOHN J. SMITH'    "NAME" OF AREA1
    0034 F2F8F860F5F3F9F2 003B   21 PHON1   DC    CL08'288-5392'         "PHON" OF AREA1
    003C D9D6C3C8C5E2E3C5 004A   22 CITY1   DC    CL15'ROCHESTER'        "CITY" OF AREA1


                          004B   24 AREA2   EQU   *
    004B                  0068   25 NAME2   DS    CL30                   "NAME" OF AREA2
    0069                  0070   26 PHON2   DS    CL08                   "PHON" OF AREA2
    0071                  007F   27 CITY2   DS    CL15                   "CITY" OF AREA2
                          0000   28         END   EXAMP
```

Base Displacement Addressing

Base displacement addressing involves the programmer having assembler placing a base address in a register. Other addresses can then be calculated from this base address. This base displacement is one byte in the machine instruction. Any value in an index register allows access to 256 storage positions.

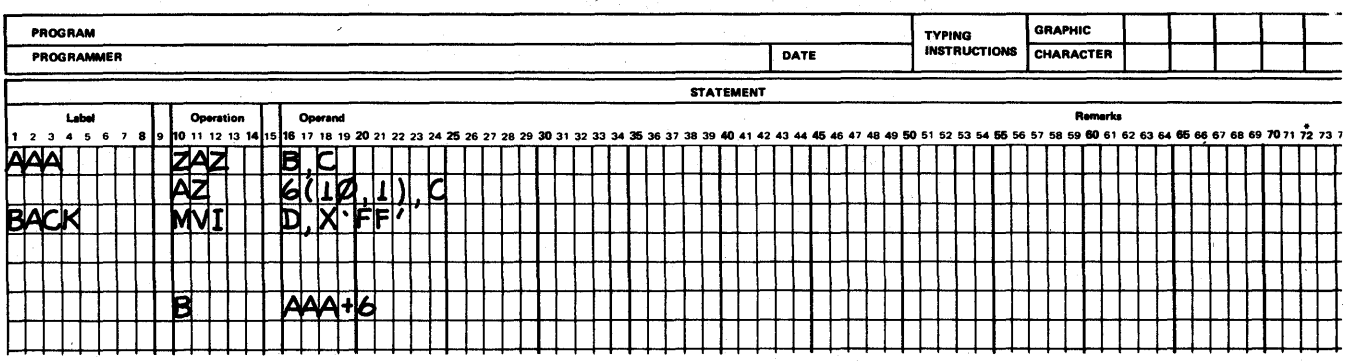
The USING statement makes the contents of an index register the basis for base displacement addressing. The DROP statement ends base displacement addressing. For information about the USING and DROP statements, see their descriptions in Chapter 3. The following figure shows examples of base displacement addressing:

```
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT

    0000                         1 EXAMP1   START 0
                                 2          PRINT NODATA,NOGEN

                                 4 ********************************************************************
                                 5 *                                                                  *
                                 6 *        AN EXAMPLE OF BASE-DISPLACEMENT ADDRESSING WITH THE       *
                                 7 *        "USING" INSTRUCTION.                                      *
                                 8 *                                                                  *
                                 9 ********************************************************************

    0000 C2 01 0018              11         LA    AREA1,R1              POINT TO THE MOVING "FROM" FIELD
                          0018   12         USING AREA1,R1              SET TO USE LABELS AS DISPLACEMENTS FROM AREA1

    0004 C2 02 004D              14         LA    AREA2,R2              POINT TO THE MOVING "TO" FIELD
                          0018   15         USING AREA1,R2              SET TO USE LABELS OF AREA1 AS DISPLACEMENTS
                                 16 *                                  INTO AREA2.

    0008 9C 1D 1D 1D             18         MVC   NAME(30,R2),NAME(,R1)  MOVE "NAME" OF AREA1 TO "NAME" OF AREA2
    000C 9C 07 25 25             19         MVC   PHON(08,R2),PHON(,R1)  MOVE "PHON" OF AREA1 TO "PHON" OF AREA2
    0010 9C 0E 34 34             20         MVC   CITY(15,R2),CITY(,R1)  MOVE "CITY" OF AREA1 TO "CITY" OF AREA2
                                 21 *       $EOJ                        END OF JOB


                          0018   27 AREA1   EQU   *
    0018 D1D6C8D540D14B40 0035   28 NAME    DC    CL30'JOHN J. SMITH'   "NAME" OF AREA1
    0036 F2F8F860F5F3F9F2 003D   29 PHON    DC    CL08'288-5392'        "PHON" OF AREA1
    003E D9D6C3C8C5E2E3C5 004C   30 CITY    DC    CL15'ROCHESTER'       "CITY" OF AREA1


                          004D   32 AREA2   EQU   *
    004D                  006A   33         DS    CL30                  "NAME" OF AREA2
    006B                  0072   34         DS    CL08                  "PHON" OF AREA2
    0073                  0081   35         DS    CL15                  "CITY" OF AREA2


                          0001   37 R1      EQU   1                     EQUATE FOR REGISTER 1
                          0002   38 R2      EQU   2                     EQUATE FOR REGISTER 2
                          0000   39         END   EXAMP1
```

```
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT

    0000                         1 EXAMP2   START 0
                                 2          PRINT NODATA,NOGEN

                                 4 ********************************************************************
                                 5 *                                                                  *
                                 6 *        AN EXAMPLE OF BASE DISPLACEMENT ADDRESSING                *
                                 7 *        USING "EQUATES"                                           *
                                 8 *                                                                  *
                                 9 ********************************************************************

    0000 C2 01 0018              11         LA    AREA1,R1              POINT TO THE MOVING "FROM" FIELD

    0004 C2 02 004D              13         LA    AREA2,R2              POINT TO THE 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
                                 18 *       $EOJ                        END OF JOB


                          0018   24 AREA1   EQU   *
    0018 D1D6C8D540D14B40 0035   25         DC    CL30'JOHN J. SMITH'   "NAME" OF AREA1
    0036 F2F8F860F5F3F9F2 003D   26         DC    CL08'288-5392'        "PHON" OF AREA1
    003E D9D6C3C8C5E2E3C5 004C   27         DC    CL15'ROCHESTER'       "CITY" OF AREA1


                          004D   29 AREA2   EQU   *
    004D                  006A   30         DS    CL30                  "NAME" OF AREA2
    006B                  0072   31         DS    CL08                  "PHON" OF AREA2
    0073                  0081   32         DS    CL15                  "CITY" OF AREA2


                          001D   34 NAME    EQU   29,30                 "NAME" DISPLACEMENT INTO AREAS 1 & 2
                          0025   35 PHON    EQU   NAME+8,8              "PHON" DISPLACEMENT INTO AREAS 1 & 2
                          0034   36 CITY    EQU   PHON+15,15            "CITY" DISPLACEMENT INTO AREAS 1 & 2


                          0001   38 R1      EQU   1                     EQUATE FOR REGISTER 1
                          0002   39 R2      EQU   2                     EQUATE FOR REGISTER 2
                          0000   40         END   EXAMP2
```

Relative Addressing

Relative addressing addresses instructions and data areas by giving their location (in bytes) in relation to the location counter or to some symbolic location. This type of addressing is only entered in bytes. 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.



When you use 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 label entry in a machine-instruction statement addresses the *leftmost* (low-address) byte of storage occupied by that instruction.

Data Addressing

A symbol used as a label entry in a data definition instruction (see *DC-Define Constant and DS-Define Storage* in Chapter 3) addresses the *rightmost* (high-address) byte of storage occupied by or reserved for that data.

Program Linking References

You can link independently assembled programs by defining symbols in one program and referring to them in another. The assembler must provide information about the linkage symbols to the overlay linkage editor. Using the linkage symbols provided by the ENTRY and EXTRN instructions, the assembler places the necessary information in the external symbol lists.

You will use the ENTRY assembler instruction to identify the linkage symbol in program A. Once the symbol is identified as naming an entry point, you can use that symbol in another program (B), in a branch operation, or as a data reference. With an EXTRN assembler instruction in program B, you must also identify the symbol used in program A. You can also use the label of a START statement for program linking. The formats of the EXTRN and ENTRY assembler control instructions are included in Chapter 3.

Machine Instructions

Machine instructions are the most elementary instructions you can use with System/36. A summary chart of all instructions is in Appendix D.

General Programming Notes

These programming notes apply to all machine instructions. Notes that apply to specific instructions are explained with those instructions.

- Operand 2 is not changed unless the fields overlap. However, overlapping operands can destroy part of operand 2 before it is used in the operation.
- A length value is not required. If you omit the length, implied lengths are used.
- Operations other than AZ, SZ, ZAZ, and SRC, where you can specify length, allow a maximum length of 256 bytes.
- For zoned operations, AZ, SZ, and ZAZ, the maximum value of length1 is 31 bytes and the maximum value of length2 is 16 bytes. Also, length1 must be greater than or equal to length2 and length1 minus length2 must be 16 or less.
- For Shift Right Characters (SRC), the maximum length is 16 bytes.
- Save areas for registers occupy 2 bytes.
- Labels on all data in examples point to the rightmost byte shown.
- Information about the setting of the Program Status Register after each instruction is given in the *System/36 Functions Reference* manual.

A (Add to Register)

The A instruction adds the binary number in the location indicated by the operand to the contents of the 2-byte register indicated by the hexadecimal values assigned as follows:

Register	Hex Value
Index Register 1	01 or 03
Index Register 2	02
Program Status Register	04
Address Recall Register	08
Instruction Address Register	10 or 20
Work Register 4	44
Work Register 5	45
Work Register 6	46
Work Register 7	47

For example:

```
A BIGE,08
```

Label	Before (hex)	After (hex)
BIGE	01 32	01 32
ARR	09 5E	0A 90

This instruction adds the contents of BIGE-1 and BIGE to the contents of the address recall register (ARR).

This instruction can be used in the following formats:

Operation	Operand	Hex Value (XX)
A	address	,XX
A	displacement(,reg1)	,XX
A	displacement(,reg2)	,XX

Programming Notes

- This instruction changes the contents of only one register.
- Constants must be at least 2 bytes long.
- The operand value is not changed.
- Adding to the Program Status Register causes an unpredictable result. Hex 04 is forced into the high-address byte of the program status register.
- Adding to the IAR causes an unconditional branch without changing the ARR.

ALC (Add Logical Characters)

The ALC instruction adds the binary number at the locations indicated by operand 2 to the binary number at the locations indicated by operand 1. For example:

```
ALC FIL1(4),FIL2
```

Label	Before (hex)	After (hex)
FIL1	FE ED FF FF	FF 0F 00 00
FIL2	00 21 00 01	00 21 00 01

This instruction adds the 4-byte hexadecimal number at FIL2 to the 4-byte hexadecimal number at FIL1.

Operation	Operand1	Operand2
ALC	address1(length1)	,address2
ALC	address1(length1)	,displacement2(,reg1)
ALC	address1(length1)	,displacement2(,reg2)
ALC	displacement1(length1,reg1)	,address2
ALC	displacement1(length1,reg1)	,displacement2(,reg1)
ALC	displacement1(length1,reg1)	,displacement2(,reg2)
ALC	displacement1(length1,reg2)	,address2
ALC	displacement1(length1,reg2)	,displacement2(,reg1)
ALC	displacement1(length1,reg2)	,displacement2(,reg2)

Programming Notes

- A length can be given in either operand, but not in both.
- Both operands must be the same length, up to a maximum of 256 bytes.
- If you do not specify a length, the implied length of operand 1 is used.
- The system sets the binary overflow bit to 0 if a carry does not occur and to 1 if a carry does occur.

ALI (Add Logical Immediate)

The ALI instruction adds the binary number in the supplied byte to the binary number in the location specified by the operand. For example:

```
ALI DEPT,X'2E'
```

This instruction adds the value 2E (8 bits) to the contents of the location indicated by DEPT.

Label	Before (hex)	After (hex)
Dept	15	43

Operation	Operand	Hex Value (XX)
ALI	address	,XX
ALI	displacement1(,reg1)	,XX
ALI	displacement1(,reg2)	,XX

Programming Notes

- This instruction affects only a single location.
- The ALI instruction uses the SLI operation code and changes the value to twos complement when assembled.
- The program status register is set according to the twos complement value of the operand's Q-code.

AZ (Add Zoned Decimal)

The AZ instruction adds the decimal value of the numeric portion of the locations indicated by operand 2 to the decimal value of the numeric portion of the locations indicated by operand 1. For example:

Label	Before (hex)	After (hex)
FIL1	F3 F6 F9	F3 F9 F4
FIL2	F2 F5	F2 F5

This instruction adds the two numeric portions of FIL2 to the three numeric portions of FIL1.

Operation	Operand1	Operand2
AZ	address1(length1)	,address2(length2)
AZ	address1(length1)	,displacement2(length2),reg1)
AZ	address1(length1)	,displacement2(length2,reg2)
AZ	displacement1(length1,reg1)	,address2(length2)
AZ	displacement1(length1,reg1)	,displacement2(length2,reg1)
AZ	displacement1(length1,reg1)	,displacement2(length2,reg2)
AZ	displacement2(length1,reg2)	,address2(length2)
AZ	displacement1(length1,reg2)	,displacement2(length2,reg1)
AZ	displacement1(length1,reg2)	,displacement2(length2,reg2)

Programming Notes

- The address of operand 1 remains in the address recall register until another AZ, BC, ITC, SZ, or ZAZ instruction is performed, or until the register is the target of an A, L, LA, or S instruction.
- The system does not check for valid decimal digits in either operand.
- The zone bits of all but the rightmost byte in operand 1 are always set to hex F. The zone bits of the rightmost byte are set to hex F if the result of the operation is positive, or to hex D if the result is negative.
- If the zone bits of the rightmost byte are hex D or B, the operand is negative. If the zone bits are anything else, the operand is positive.

BC (Branch on Condition)

The branch instructions cause a transfer of control to the specified instruction if bits 2 through 7 of the high-address byte of the program status register meet the conditions represented by the data byte.

These are the conditions tested for, and the data byte bits:

Condition	Data Byte Bit
Equal	7
Low	6
High	5
Decimal overflow	4
Test false	3
Binary overflow	2

If bit 0 of the data byte is 1, match of 1 bits between the program status register and the data byte causes a branch. If bit 0 is 0, all of the tested bits of the program status register must be 0 for a branch to occur. If no conditions are met, the system performs the next instruction. To perform a branch, the system places the address of the next sequential machine instruction in the address recall register, then branches to the supplied address. You can test for a combination of conditions with the BC instruction. For example:

```
BC BURT,X'A8'
```

This instruction branches to the location labeled BURT if the value in the program status register indicates either a decimal overflow or a binary overflow condition.

The instruction has the following format:

Operation	Operand	Hex Value (XX)
BC	address	,XX
BC	displacement(,reg1)	,XX
BC	displacement(,reg2)	,XX
BC	displacement(,reg8)	,XX

Branch Mnemonics

Use the branch mnemonics from the following list to branch on a specific condition. Use one of these mnemonics in place of BC and specify only operand 1.

Instruction	Mnemonic	Generated Data Byte (hex)
Branch (unconditional)	B	87
Branch high	BH	84
Branch low	BL	82
Branch equal	BE	81
Branch not high	BNH	04
Branch not low	BNL	02
Branch not equal	BNE	01
Branch overflow zoned	BOZ	88
Branch overflow logical	BOL	A0
Branch no overflow zoned	BNOZ	08
Branch no overflow logical	BNOL	20
Branch true	BT	10
Branch false	BF	90
Branch plus	BP	84
Branch minus	BM	82
Branch zero	BZ	81
Branch not plus	BNP	04
Branch not minus	BNM	02
Branch not zero	BNZ	01

Programming Notes

- An address remains in the address recall register until another AZ, BC, ITC, SZ, or ZAZ instruction is performed, or until the register is the target of an A, L, LA, or S instruction.
- If the ARR is used in the operand, the address being branched to is determined before the ARR is changed to the next sequential instruction.
- Test data of hex 80, 07, 17, 27, 37, 47, 57, 67, 77, 0F, 1F, 2F, 3F, 4F, 5F, 6F, or 7F is a no operation (no-op) condition; that is, no branch occurs.
- Test data of hex 00, 87, 97, A7, B7, C7, D7, E7, F7, 8F, 9F, AF, BF, CF, DF, EF, or FF causes an unconditional branch.
- When the branch instruction is performed, only bit 4 (decimal overflow) and bit 3 (test false) of the program status byte are set off as they are tested.
- The program status byte is never hex 00, and only one of 5, 6, or 7 is always on.

BD (Branch Direct)

The BD instruction causes an unconditional branch to the address represented by the operand, and does not change the ARR. For example:

```
BD BURT
```

This instruction branches to the location labeled BURT, and does not change the address recall register.

Operation	Operand
BD	address
BD	displacement1(,reg1)
BD	displacement2(,reg2)

Programming Notes

This instruction assembles into the same object code as LA of the IAR.

CLC (Compare Logical Characters)

The CLC instruction compares the contents of the location indicated by operand 1 with the contents of operand 2. The setting of the program status register is determined by the results of the comparison. The program status register settings are given in the following table:

Set on Condition	Register Bit	Name
Operand 1 value larger	5	High
Operand 1 value smaller	6	Low
Compare equal	7	Equal

For example:

```
CLC BARB(2),MIKE
```

This instruction compares the 2-byte contents of BARB (16 bits) with the contents of MIKE (16 bits).

Operation	Operand1	Operand2
CLC	address1(length1)	,address2
CLC	address1(length1)	,displacement2(,reg1)
CLC	address1(length1)	,displacement2(,reg2)
CLC	displacement2(,reg1)	,address2
CLC	displacement1(length1,reg1)	,displacement2(,reg1)
CLC	displacement1(length1,reg1)	,displacement2(,reg2)
CLC	displacement1(length1,reg2)	,address2
CLC	displacement1(length1,reg2)	,displacement2(,reg1)
CLC	displacement1(length1,reg2)	,displacement1(,reg2)

Programming Notes

- The length (number of locations), up to a maximum of 256, can be given in either operand of the instruction, but not in both. If you do not specify a length, the implied length of operand 1 is used.
- The contents of neither operand location is changed by this instruction.

CLI (Compare Logical Immediate)

The CLI instruction compares all the bits of the supplied byte with the bits in the location specified by the operand. The setting of the program status register is determined by the results of the comparison. The program status register settings are given in the following table:

Set on Condition	Register Bit	Name
Supplied value smaller	5	High
Operand value smaller	6	Low
Compare equal	7	Equal

For example:

```
CLI DENN,X'3F'
```

This instruction compares the value hex 3F (8 bits) with the contents of the location indicated by DENN.

Operation	Operand	Value
CLI	address	,XX
CLI	displacement1(,reg1)	,XX
CLI	displacement1(,reg2)	,XX

Programming Notes

Neither the supplied value nor the contents of the storage location is changed by this instruction.

ED (Edit)

The ED instruction replaces bytes containing hexadecimal 20 in the locations indicated by operand 1 with bytes from the locations indicated by operand 2, starting at the rightmost position of both operands. The zone bits of the copied bytes are set to hexadecimal F as they are written. For example:

```
ED BERT(6),ERNI
```

Label	Before (hex)	After (hex)
BERT	20 20 20 4B 20 20	F1 F9 F9 4B F5 F0
ERNI	F5 F1 F9 F9 D5 D0	F5 F1 F9 F9 D5 D0

This instruction copies the data from each of the locations indicated by ERNI (changing the zone bits to hex F) into each of the six locations indicated by BERT that contain hex 20. If only the first location at BERT had contained a hex 20, then only the data in the rightmost location at ERNI would have been used.

Operation	Operand1	Operand2
ED	address1(length1)	,address2
ED	address1(length1)	,displacement2(,reg1)
ED	address1(length1)	,displacement2(,reg2)
ED	displacement1(length1,reg1)	,address2
ED	displacement1(length1,reg1)	,displacement2(,reg1)
ED	displacement1(length1,reg1)	,displacement2(,reg2)
ED	displacement1(length1,reg2)	,address2
ED	displacement1(length1,reg2)	,displacement2(,reg1)
ED	displacement1(length1,reg2)	,displacement2(,reg2)

Programming Notes

- Operand 2 must contain at least as many bytes as there are hex 20s in operand 1.
- The length (number of locations) can be supplied by either operand, but not both.
- The first location addressed in either operand is the highest addressed (rightmost) location, with the next lower address being used in each successive cycle.
- Resultant data in operand 1 is unsigned numeric.

ITC (Insert and Test Characters)

The ITC instruction replaces characters in the locations indicated by operand 1 with the character in the location indicated by operand 2. All characters to the left of the first significant digit (decimal 1 through 9) are replaced with the character from the operand 2 location. This replacement continues for the specified number of locations, or until a location indicated by operand 1 is found to contain a number (hex F1 through F9). For example:

```
ITC BOB1(6),RAY2
```

Label	Before (hex)	After (hex)
BOB1	F0 F0 6B F8 F5 F0	5C 5C 5C F8 F5 F0
RAY2	5C	5C

The character found at RAY2 is copied into the locations indicated by BOB1, beginning with location BOB1-5 and continuing until a number (hex F1 through F9) is found.

Operation	Operand1	Operand2
ITC	address1(length1)	,address2
ITC	address1(length1)	,displacement2(,reg1)
ITC	address1(length1)	,displacement2(,reg2)
ITC	displacement1(length1,reg1)	,address2
ITC	displacement1(length1,reg1)	,displacement2(,reg1)
ITC	displacement1(length1,reg1)	,displacement2(,reg2)
ITC	displacement1(length1,reg2)	,address2
ITC	displacement1(length1,reg2)	,displacement2(,reg1)
ITC	displacement1(length1,reg2)	,displacement2(,reg2)

Programming Notes

- At the end of this operation, the ARR contains either the address of the first significant digit or, if none, the address + 1 of the first operand.
- An address remains in the address recall register until another AZ, BC, ITC, SZ, or ZAZ instruction is performed, or until the register is the target of an A, L, LA, or S instruction.
- The length (number of locations) can be supplied by either operand, but not both.
- This operation occurs from low address (leftmost) to high address (rightmost).
- The second operand is a 1-byte field.

JC (Jump on Condition)

The jump instructions cause the program to jump either forward or backward to a new instruction address when bits 2 through 7 in the high-address byte of the program status register satisfy the conditions tested by the supplied data byte.

The conditions represented and the bits tested for are given in the following table:

Condition	Bit On (1)
Equal	7
Low	6
High	5
Decimal overflow	4
Test false	3
Binary overflow	2

If bit 0 of the supplied data is 1, any matching 1 bits between the program status register and the data byte cause a jump. If bit 0 is 0, all of the tested bits of the register must be 0 for a jump to occur. If none of the conditions are met, the system performs the next sequential instruction.

To perform a jump, the program determines a displacement from the operand of the jump instruction. This displacement is added to or subtracted from the address in the instruction address register after the IAR has been increased beyond the jump instruction. The program jumps to that new address at the end of the jump on condition operation. For example:

```
JC MORT,X'81'
```

This instruction causes a jump, using the address (or displacement) found in the location indicated by MORT, if the value in the program status register is odd (condition equal).

This instruction has the following format:

Operation	Operand	Value
JC	address	,XX
JC	displacement	,XX

Jump Mnemonics

Use the jump mnemonics from the following list to jump on a specific condition. Use one of these mnemonics in place of JC and specify only operand 1.

Instruction	Mnemonic	Generated Data Byte (hex)
Jump unconditional	J	87
Jump high	JH	84
Jump low	JL	82
Jump equal	JE	81
Jump not high	JNH	04
Jump not low	JNL	02
Jump not equal	JNE	01
Jump overflow zoned	JOZ	88
Jump overflow logical	JOL	A0
Jump no overflow zoned	JNOZ	08
Jump no overflow logical	JNOL	20
Jump true	JT	10
Jump false	JF	90
Jump plus	JP	84
Jump minus	JM	82
Jump zero	JZ	81
Jump not plus	JNP	04
Jump not minus	JNM	02
Jump not zero	JNZ	01

Programming Notes

- The operand must be an address within 255 bytes (hex FF) of the next sequential instruction.
- The program status byte is never hex 00, and only one of bits 5, 6, or 7 is always on.
- The ARR is not changed by this instruction.
- Test data of hex 80, 07, 17, 27, 37, 47, 57, 67, 77, 0F, 1F, 2F, 3F, 4F, 5F, 6F, or 7F causes a no operation (no-op) condition, which means no jump occurs.
- Test data of hex 00, 87, 97, A7, B7, C7, D7, E7, F7, 8F, 9F, AF, BF, CF, DF, EF, or FF causes an unconditional jump.
- Bit 4 (decimal overflow) and bit 3 (test false) of the program status byte are set off as they are tested. The other bits are not affected.
- An absolute value of 0 through 255 can be used as the operand.

L (Load Register)

The L instruction copies data from the 2-byte field specified by operand 1 into the specified register, shown in the following table:

Register	Hex Value (XX)
Index Register 1	01 or 03
Index Register 2	02
Program Status Register	04
Address Recall Register	08
Instruction Address Register	10 or 20
Work Register 4	44
Work Register 5	45
Work Register 6	46
Work Register 7	47

For example:

```
L FOX1,X'08'
```

This instruction copies the 2-byte contents of the location indicated by FOX1 into the address recall register:

Operation	Operand1	Hex Value (XX)
L	address	,XX
L	displacement1(,reg1)	,XX
L	displacement1(,reg2)	,XX

Programming Notes

- This instruction copies data into only one register.
- The system performs an unconditional branch to any address placed in the instruction address register without changing the ARR.

LA (Load Address)

The LA instruction places the 2-byte address represented by operand 1 into the specified register shown in the following table:

Register	Hex Value (XX)
Index Register 1	01 or 03
Index Register 2	00 or 02
Program Status Register	04
Address Recall Register	08
Instruction Address Register	10 or 20
Work Register 4	44
Work Register 5	45
Work Register 6	46
Work Register 7	47

For example:

```
LA PAT3,X'02'
```

This instruction places the address of PAT3 into index register 2.

Operation	Operand1	Hex Value (XX)
LA	address	,XX
LA	displacement(,reg1)	,XX
LA	displacement(,reg2)	,XX

Programming Notes

- This instruction copies data into only one register.
- This instruction causes an unconditional branch to any address placed in the IAR without changing the ARR.

MVC (Move Characters)

The MVC instruction copies the data at the location (starting with the rightmost byte) indicated by operand 2 into the location indicated by operand 1. For example:

```
MVC  DED1(6),DED2
```

Label	Before (hex)	After (hex)
DED1	D9 96 83 88 85 A2	E3 96 99 96 95 E3
DED2	E3 96 99 96 95 E3	E3 96 99 96 95 E3

This instruction copies the 6 bytes at DED2 into six locations beginning at DED1.

Operation	Operand1	Operand2
MVC	address1(length1)	,address2
MVC	address1(length1)	,displacement2(,reg1)
MVC	address1(length1)	,displacement2(,reg2)
MVC	displacement1(length1,reg1)	,address2
MVC	displacement1(length1,reg1)	,displacement2(,reg1)
MVC	displacement1(length1,reg1)	,displacement2(,reg2)
MVC	displacement1(length1,reg2)	,address2
MVC	displacement1(length1,reg2)	,displacement2(,reg1)
MVC	displacement1(length1,reg2)	,displacement2(,reg2)

Programming Notes

- A length, up to a maximum of 256, can be given in either operand, but not in both. If you do not specify a length, the implied length of operand 1 is used.
- You can propagate a character through a field by setting the operand 2 address one byte higher (to the right) than the operand 1 address.
- The contents of the locations of operand 2 are not changed unless the fields overlap.

MVI (Move Logical Immediate)

The MVI instruction copies a 1-byte value into the location indicated by operand1. For example:

```
MVI FERD,X'00'
```

This instruction sets the contents of location FERD to 0.

Operation	Operand1	Hex Value (XX)
MVI	address	,XX
MVI	displacement1(,reg1)	,XX
MVI	displacement1(,reg2)	,XX

Programming Notes

The first operand is a 1-byte storage location.

MVX (Move Hexadecimal Character)

The MVX instruction copies either the numeric or the zone portion of the location indicated by operand 2 to either the numeric or the zone portion of the location indicated by operand 1, as shown in the following table:

Operation	Mnemonic	Hex
Copy to zone from zone	MZZ	00
Copy to zone from numeric	MZN	01
Copy to numeric from zone	MNZ	02
Copy to numeric from numeric	MNN	03

For example:

```
MVX ADD1(01),ADD2
```

Label	Before (hex)	After (hex)
ADD1	12	52
ADD2	F5	F5

This instruction copies the numeric portion of ADD2 into the zone portion of ADD1. The instruction could also be written as NZ ADD1,ADD2.

Operation	Operand1	Operand2
MVX	address1(length1)	,address2
MVX	address1(length1)	,displacement2(,reg1)
MVX	address1(length1)	,displacement2(,reg2)
MVX	displacement1(hex,reg1)	,address2
MVX	displacement1(hex,reg1)	,displacement2(,reg1)
MVX	displacement1(hex,reg1)	,displacement2(,reg2)
MVX	displacement1(hex,reg2)	,address2
MVX	displacement1(hex,reg2)	,displacement2(,reg1)
MVX	displacement1(hex,reg2)	,displacement2(,reg2)

Programming Notes

- Both operands specify 1-byte storage locations.
- The second operand is changed if both operands specify the same byte.

S (Subtract from Register)

The S instruction subtracts the binary number in the 2-byte location indicated by the operand from the contents of the 2-byte register indicated by the hexadecimal values assigned as follows:

Register	Hex Value (XX)
Index Register 1	01 or 03
Index Register 2	02
Program Status Register	04
Address Recall Register	08
Instruction Address Register	10 or 20
Work Register 4	44
Work Register 5	45
Work Register 6	46
Work Register 7	47

For example:

```
S FIL1,08
```

This instruction subtracts the contents of FIL1 -1 and FIL1 from the contents of the address recall register (ARR).

Operation	Operand1	Hex Value (XX)
S	address	,XX
S	displacement1(,reg1)	,XX
S	displacement1(,reg2)	,XX

Programming Notes

- This instruction changes the contents of only one register.
- The operand value is not changed.
- Subtraction from the program status register causes unpredictable results.
- Subtraction from the instruction address register causes an unconditional branch without changing the ARR.

SBF (Set Bits Off Masked)

The SBF instruction changes a bit of the data at the location specified by operand 1 to binary 0 if a corresponding bit of the 1-byte value is binary 1. If a bit of the value is binary 0, no changes are made to the corresponding bit in operand 1. For example:

```
SBF NODR,X'0F'
```

This instruction sets all bits of the numeric portion of the data in NODR to binary 0's, but will not alter the zone portion.

Operation	Operand1	Hex Value (XX)
SBF	address	,XX
SBF	displacement1(,reg1)	,XX
SBF	displacement1(,reg2)	,XX

Programming Notes

The length of operand 1 is 1 byte.

SBN (Set Bits On Masked)

The SBN instruction changes a bit of the data at the location specified by operand 1 to binary 1 if a corresponding bit of the 1-byte value is binary 1. If a bit of the value is binary 0, no changes are made to the corresponding bit in operand 1. For example:

```
SBN FERN,X'F0'
```

This instruction sets all bits of the zone portion of FERN to binary 1's, but does not alter the numeric portion.

Operation	Operand1	Hex Value (XX)
SBN	address	,XX
SBN	displacement1(,reg1)	,XX
SBN	displacement1(,reg2)	,XX

Programming Notes

The length of operand 1 is 1 byte.

SLC (Subtract Logical Characters)

The SLC instruction subtracts the binary number in the location indicated by operand 2 from the binary number in the location indicated by operand 1. For example:

```
SLC ONE2(2),FEW2
```

Label	Before (hex)	After (hex)
ONE2	DD DD	CB A9
FEW2	12 34	12 34

This instruction subtracts the 2-byte binary number at FEW2, operand 2, from the 2-byte binary number at ONE2, operand 1, leaving the result of the operation in the operand 1 locations.

Operation	Operand1	Operand2
SLC	address1(length1)	,address2
SLC	address1(length1)	,displacement2(,reg1)
SLC	address1(length1)	,displacement2(,reg2)
SLC	displacement1(length1,reg1)	,address2
SLC	displacement1(length1,reg1)	,displacement2(,reg1)
SLC	displacement1(length1,reg1)	,displacement2(,reg2)
SLC	displacement1(length1,reg2)	,address2
SLC	displacement1(length1,reg2)	,displacement2(,reg1)
SLC	displacement1(length1,reg2)	,displacement2(,reg2)

Programming Notes

- A length, up to a maximum of 256, can be given in either operand, but not in both. If you do not specify a length, the implied length of operand 1 is used.
- If the value of operand 2 is greater than operand 1, an additional high-order bit is implied for operand 1. Therefore, the resulting difference will never be a negative number.

SLI (Subtract Logical Immediate)

The SLI instruction subtracts the binary number in the supplied byte from the binary number in the location specified by the operand. For example:

```
SLI FIL1,X'3F'
```

Label	Before (hex)	After (hex)
FIL1 (hex)	EE	AF

This instruction subtracts the value hex 3F (8 bits) from the contents of FIL1.

Operation	Operand1	Hex Value (XX)
SLI	address	,XX
SLI	displacement1(,reg1)	,XX
SLI	displacement1(,reg2)	,XX

Programming Notes

- This instruction affects only a single storage location.
- If the value of operand 2 is greater than operand 1, an additional high-order bit is implied for operand 1.

SRC (Shift Right Character)

The SRC instruction causes the specified storage locations to shift the specified number of bits to the right. The leftmost bits are set to zero and the rightmost bits that were shifted out are lost. The shifted field can be up to 16 bytes long and up to 16 bits can be shifted. For example:

```
SRC NITA(2),4
```

Label	Before (hex)	After (hex)
NITA-1	X'F4'	X'OF'
NITA	X'F5'	X'4F'

This instruction shifts the 16 bits (2 bytes) at the locations NITA and NITA-1 4 bits to the right. The leftmost 4 bits of NITA-1 are set to 0.

The format of this instruction is as follows:

Operation	Operand1	Hex Value (XX)
SRC	address	,XX
SRC	displacement1(,reg1)	,XX
SRC	displacement1(,reg2)	,XX

The resulting program status register settings are:

Condition	Bit	Name
Only zeros remain in the string	7	Equal
String even, not zero	6	Low
String odd	5	High
Any 1s shifted out	2	Binary overflow

The binary overflow bit is set to 0 if no 1-bits are shifted out of the rightmost byte during this operation.

ST (Store Register)

The ST instruction places the contents of the specified register into the 2-byte field specified by operand 1, as shown in the following table:

Register	Hex Value (XX)
Index Register 1	01 or 03
Index Register 2	02
Program Status Register	04
Address Recall Register	08
Instruction Address Register	10 or 20
Work Register 4	44
Work Register 5	45
Work Register 6	46
Work Register 7	47

For example:

```
ST DOG1,X'08'
```

This instruction copies the contents of the address recall register into the 2-byte location indicated by DOG1.

Operation	Operand1	Hex Value (XX)
ST	address	,XX
ST	displacement1(,reg1)	,XX
ST	displacement1(,reg2)	,XX

Programming Notes

This instruction copies only one register.

SZ (Subtract Zoned Decimal)

The SZ instruction subtracts the decimal value of the numeric part of the locations indicated by operand 2 from the decimal value of the numeric part of the locations indicated by operand 1. For example:

```
SZ FIL1(3),FIL2(2)
```

Label	Before (hex)	After (hex)
FIL1	F5 F0 F4	F4 F8 F2
FIL2	F2 F2	F2 F2

This instruction subtracts the contents of FIL2 from the contents of FIL1 and leaves the result at FIL1.

Operation	Operand1	Operand2
SZ	address1(length1)	,address2
SZ	address1(length1)	,displacement2(,reg1)
SZ	address1(length1)	,displacement2(,reg2)
SZ	displacement1(length1,reg1)	,address2
SZ	displacement1(length1,reg1)	,displacement2(,reg1)
SZ	displacement1(length1,reg1)	,displacement2(,reg2)
SZ	displacement1(length1,reg2)	,address2
SZ	displacement1(length1,reg2)	,displacement2(,reg1)
SZ	displacement1(length1,reg2)	,displacement2(,reg2)

Programming Notes

- The address of operand 1 remains in the address recall register until another AZ, BC, ITC, SZ, or ZAZ instruction is performed, or until the register is the target of an A, L, LA, or S instruction.
- The system does not check for valid decimal digits in either operand.
- The zone part of all bytes except the rightmost byte of operand 1 is set to hex F. The zone part of the rightmost byte is set to hex F if the result of the operation is positive; it is set to hex D if the results are negative.
- If the zone bits of the rightmost byte are hex D or B, the operand is negative. If the zone bits are anything else, the operand is positive.

TBF (Test Bits Off Masked)

The TBF instruction tests specified bits of the data in the location specified by the operand. Each bit that is on in the supplied data causes the same bit in the operand location to be tested. If any of the tested bits are on, bit 3 (test false) of the program status register is set on. For example:

```
TBF MEL2,X'01'
```

This instruction tests the contents indicated by MEL2 for an even condition. If the value is not even, bit 3 (test false) of the program status register is set on.

Operation	Operand1	Hex Value (XX)
TBF	address	,XX
TBF	displacement1(,reg1)	,XX
TBF	displacement1(,reg2)	,XX

Programming Notes

- The supplied value and the contents of the storage location are not changed by this instruction.
- Bit 3 (test false) of the program status register is set off by system reset, when you use the bit as a condition for a branch or jump on condition, or by loading a binary 0 into the program status register bit 3. Bit 3 is never set off by TBF or TBN (only set on).

TBN (Test Bits On Masked)

The TBN instruction tests specified bits of the data in the location specified by the operand. Each bit that is on in the supplied data causes the same bit in the operand location to be tested. If any of the tested bits are off, bit 3 (test false) of the program status register is set on. For example:

```
TBN BENT,X'01'
```

This instruction tests the contents indicated by BENT for an odd condition. If the value in BENT is even, bit 3 of the program status register is set on.

Operation	Operand1	Hex Value (XX)
TBN	address	,XX
TBN	displacement1(,reg1)	,XX
TBN	displacement1(,reg2)	,XX

Programming Notes

- The supplied value and the contents of the storage location are not changed by this instruction.
- Bit 3 (test false) of the program status register is set off by system reset, when you use the bit as a condition for a branch or jump on condition, or by loading a binary 0 into the program status register bit 3. Bit 3 is never set off by TBF or TBN (only set on).

XFER (Transfer)

The XFER instruction gives the extended control storage supervisor the control to perform the selected function as shown in the following table:

Function	Value
Begin main program	01
Begin subroutine	02
Reenter program after call	03
Subroutine return to calling module	04
Perform next scientific instruction	05
Perform next scientific instruction after invoke	07
Place scientific interpreter in double mode	0A
Place scientific interpreter in real mode	0B
Do fixed to floating-point conversion	0C
Do real*8 floating-point to fixed conversion	0D
Do real*4 floating-point to fixed conversion	0E

For example:

```
XFER 01,X'05'
```

This instruction causes the next scientific instruction encountered to be performed after control is given to the extended control storage supervisor.

Operation Hex Value (XX)

```
XFER 01,XX
        |
        Fixed Value
```

ZAZ (Zero and Add Zoned)

The ZAZ instruction sets all bytes of operand 1 to zero (hex F0) then adds the numeric part of the locations indicated by operand 2 to the numeric part of the locations indicated by operand 1. The instruction copies the numeric data from the locations indicated by operand 2 into the locations indicated by operand 1 after setting the zone portions of operand 1 to hex F (binary 1111). The zone part of the rightmost byte of the result contains hex F (positive result) or hex D (negative result). For example:

```
ZAZ FIL1(5),FIL2(2)
```

Label	Before (hex)	After (hex)
FIL1	65 84 23 92 A1	FO FO FO F8 F2
FIL2	F8 F2	F8 F2

This instruction copies 2 bytes of data from FIL2 into FIL1 and sets 3 bytes of FIL1 to EBCDIC 0.

Operation	Operand1	Operand2
ZAZ	address1(length1)	,address2
ZAZ	address1(length1)	,displacement2(,reg1)
ZAZ	address1(length1)	,displacement2(,reg2)
ZAZ	displacement1(length1,reg1)	,address2
ZAZ	displacement1(length1,reg1)	,displacement2(,reg1)
ZAZ	displacement1(length1,reg1)	,displacement2(,reg2)
ZAZ	displacement1(length1,reg2)	,address2
ZAZ	displacement1(length1,reg2)	,displacement2(,reg1)
ZAZ	displacement1(length1,reg2)	,displacement2(,reg2)

Programming Notes

- An address remains in the address recall register until another AZ, BC, ITC, SZ, or ZAZ instruction is performed, or until the register is the target of an A, L, LA, or S instruction.
- The zone bits of all bytes except the rightmost byte in operand 1 are set to hex F. The zone bits of the rightmost byte are set to hex F (binary 1111) if the value of operand 2 is positive, or are set to hex D (binary 1101) if the value is negative.
- If the zone bits of the rightmost byte are D or B, the operand is negative. If the zone bits are anything else, the operand is positive.

Supervisor Call Instructions

Supervisor call instructions stop the main storage processor and generate an interrupt to the control storage processor. The control storage processor saves the current status of the main storage processor and uses an operand (a call identifier) to determine which supervisor call instruction to perform. Other operands define the details of the instruction. Most supervisor call instructions are privileged, and cannot be used by the assembler programmer. All supervisor call instructions must be used with caution because of the sensitivity of the system to changes in location of reference data. The supervisor call instructions are usually generated through macroinstruction expansions and are individually documented in the *Functions Reference Manual.*

Notes:

Chapter 3. Using Assembler Instructions

This chapter explains each assembler instruction. Assembler instructions cause the assembler program to perform certain operations during assembly. Assembler instruction statements, unlike machine instruction statements, are not translated into machine language. Some statements, such as DS and DC, cause the assembler to set aside storage areas for constants and other data. Other statements, such as EQU and SPACE, are effective only during assembly; they generate nothing in the object program and have no effect on the location counter.

You use assembler instruction statements to define symbols and data, to control listings, and to control the assembler processor.

Assembler Instruction Statements

The operations that can be performed by each assembler instruction are shown in the following table:

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

DC (DEFINE CONSTANT)

The DC instruction is used to reserve areas of storage, assign names to the reserved areas, and to initialize the reserved areas with one of seven types of constants. The seven types of constants are shown in the following table:

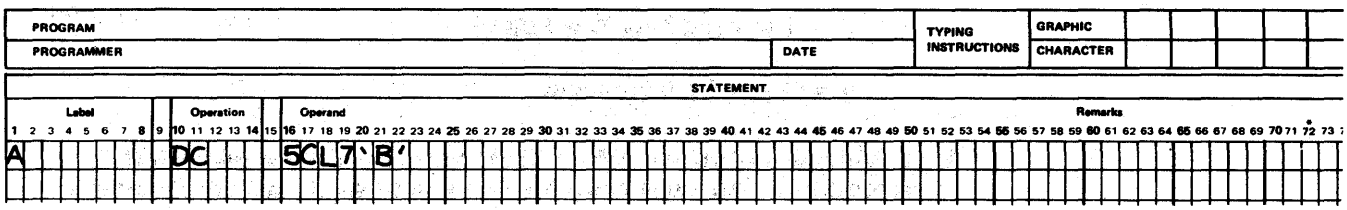
Type	ID	Example	Explanation
Address	A	AL2(BETA)	BETA could be an external reference. If a constant is not the specified length, padding with binary 0's or truncation occurs on the left. The maximum length is 3 bytes.
Binary	B	BL1'10110'	If a constant is not the specified length, padding with binary 0's or truncation occurs on the left. Each digit occupies 1 bit of storage; 8 digits occupy 1 byte of storage. The maximum length is 256 bytes.[1]
Character	C	CL14'CHARACTERS'	If a constant is not the specified length, padding with blanks or truncation occurs on the right. Each character, including blanks, occupies 1 byte of storage. The maximum length is 256 bytes.[1] not the specified length, padding with binary 0's or truncation occurs on the left. The maximum length is 3 bytes.[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 0's or truncation occurs on the left. Each decimal digit occupies 1 byte of storage. The maximum length is 31.

1 The system permits a maximum of 256 bytes. The actual length of the constant (before padding) will be restricted by the positions remaining on the statement line.

Type	ID	Example	Explanation
Floating Point	F	FL4'52' (single precision) FL8'9237.7734E-69' (double precision)	Floating-point numbers have two components: a mantissa and an exponent. The mantissa is a signed or unsigned number. Its decimal point can be 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. The only valid lengths are either 4 or 8. If a constant is not the specified length, passing with binary 0's or truncation occurs on the right. **Note**: There are no assembler floating-point instructions. Floating point is supported only for specific macroinstructions.
Hexa-decimal	X	XL3'ABC55'	If the constant is not the specified length, padding with binary 0's or truncation occurs on the left. Each two digits occupy 1 byte of storage. The maximum length is 256 bytes.[2]
Integer	I	IL2'15'	Negative numbers are inserted into storage in twos complement notation. The constant is padded or truncated on the left if it is not the specified length. Positive constants are padded with o's, and negative constants with hex 1s.The value must be within the range of $-(2^{32})+1$ to $2^{32}-1$ (-4294967295 to 4294967295). The maximum length of the constant is 256 bytes. The rightmost 4 bytes will contain the value.

2 The system permits a maximum of 256 bytes. The actual length of the constant (before padding) will be restricted by the positions remaining on the statement line.

The format of the DC assembler instruction is as follows:



Label

The label is optional. The symbol in the label field of the DC statement is the name of the constant. The value of the symbol naming the DC statement is the address of the rightmost byte of the constant.

Operand

The operand is the constant and its description. The parts of the operand are written in the following sequence.

The first three parts of the operand describe the constant in the fourth. No blanks are permitted within the operand unless provided as characters in a character constant. Part 1 of the operand is optional. Parts 2, 3, and 4 must be present in the operand field.

1 Duplication factor
2 Type
3 Length
4 Constant

Duplication Factor: Optional. If specified, the constant is generated the number of times indicated by the factor. The factor must be specified by an unsigned decimal value of 1 through 65535. The duplication factor is applied after the constant is assembled into its proper format.

Type: Defines the type of constant being specified. From the type specification, the assembler determines how to interpret the constant and translate it into the appropriate machine format. The type is specified by a letter code as follows:

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

Length: The number of bytes of storage occupied by the constant. The length can be written in two ways.

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

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

n = 1 through 31 for D constants

n = 1 through 3 for A constants

n = 4 or 8 for F constants

- 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 as an unsigned, decimal value. A location counter reference is not allowed in this expression. Refer to *Assembler Program Conventions* in Chapter 2 for information about expressions.

Constant: Described by the subfields that come before it. A data constant (any type except A) must be enclosed in apostrophes. An address constant (type A) must be enclosed in parentheses.

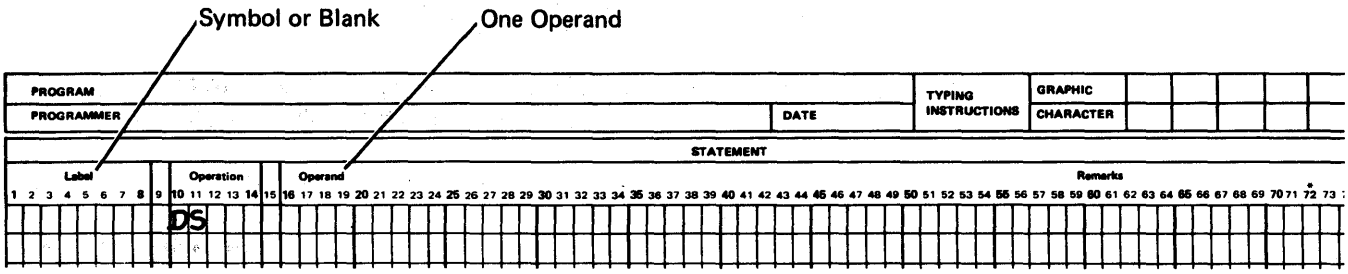
The constant types, their identification letters, and an example of each are shown in the table under *DC (Define Constant)* in this chapter.

Examples of the DC instructions for each of the constant types are given in the following table. The object code generated for these constants is also shown.

```
                    5726AS1AS R00M00 YYMMDD       IBM SYSTEM/36 ASSEMBLER-MACRO PROCESSOR

ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT                    10/14/82   TIME 14:59   PAGE    4

    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 3B44B82F         0036   21 FLT3     DC    FL4'25.6E-8'            FLOATING POINT-SINGLE PRECISION,NEG EXPONENT
    0037 C310000000000000 003E   22 FLT4     DC    FL8'-256'               FLOATING POINT-SINGLE PRECISION,NEGATIVE
    003F 4310000000000000 0046   23 FLT5     DC    FL8'256.0'              FLOATING POINT-SINGLE PRECISION,DECIMAL POINT
    0047 4B2540BE40000000 004E   24 FLT6     DC    FL8'256E10'             FLOATING POINT-SINGLE 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
```

DS (Define Storage)

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



Label

The label is optional. The symbol in the label of the DS instruction is the name of the constant. The value attribute of the symbol naming the instruction is the address of the high-order address (rightmost) byte of the constant.

Operand

The operand is the constant and its description. The duplication factor is optional; type and length are required.

Duplication Factor: Used to reserve storage areas larger than 256 bytes. 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. The total value is limited to 65535.

Type: Requires one of the seven letter codes I, X, D, A, B, F, or C. The use of the type during execution is not tested, but length restrictions are different for each type.

Length: The value up to 256, depending on constant type for the number of bytes of storage to be reserved. The duplication factor is used to reserve larger areas. The length can be written two ways:

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

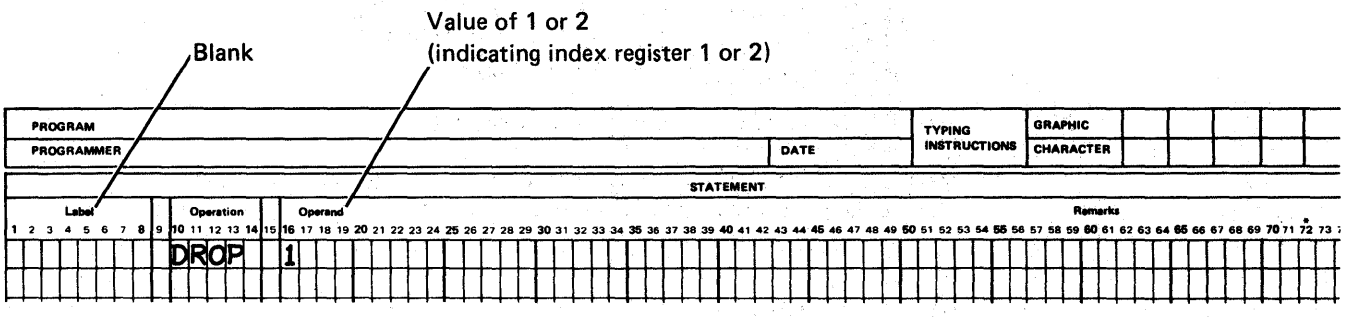
  n = 1 through 256 for I, B, C, X constants
  n = 1 through 31 for D constants
  n = 1 through 3 for A constants
  n = 4 or 8 for F constants

- 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 as an unsigned, decimal value. A location counter reference is not allowed in this expression. Refer to *Assembler Program Conventions* in Chapter 2 for information about expressions.

DROP (Drop Index Register as Base Register)

The DROP instruction specifies that an index register can no longer be used as a base register. It ends a USING instruction.

The format of the DROP instruction is as follows:



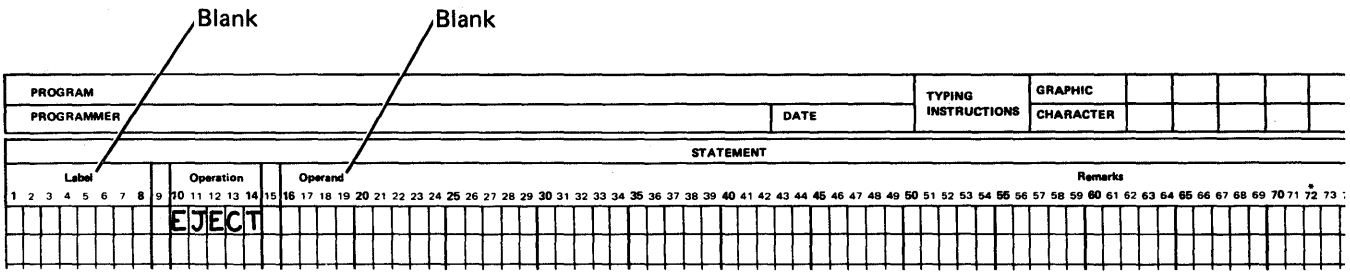
It is not necessary to use a DROP instruction when the base address is changed by a USING instruction, nor are DROP statements needed at the end of the source program.

Operand

The operand is an absolute expression with a value of 1 or 2 that indicates which index register, previously specified in a USING statement, is no longer available for base register use.

EJECT (Start New Page)

The EJECT instruction causes the next line of the listing to start a new page. This instruction gives you a convenient way to separate routines in the assembler listing. The format of the EJECT statement is as follows:



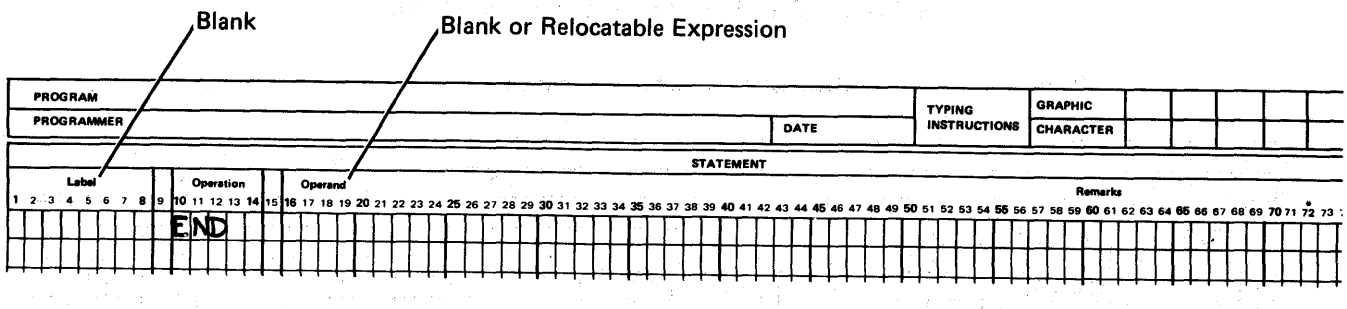
The EJECT statement is not printed in the source listing, but does increase the statement counter by 1.

Label and Operand

The label and operand fields of an EJECT instruction must be blank.

END (End Assembly)

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

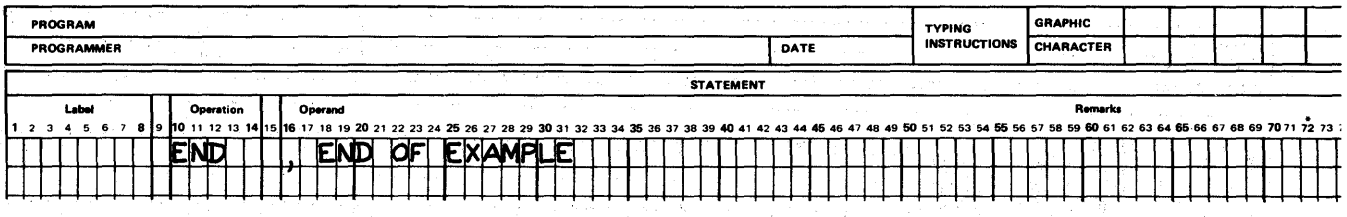


Operand

The operand of this instruction can be blank or contain an expression (usually a label) that 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.

The assembler checks the first 87 bytes of the source statement unless you use ICTL to change the source record format.

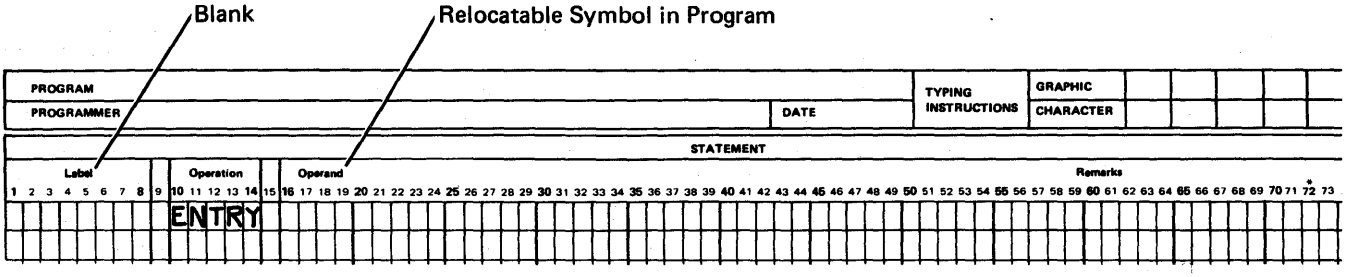
Note: If you have no operand on the END instruction, sequence numbers or comments appearing before byte 87 sometimes cause assembly errors. If the operand is blank and you want to put a comment on the instruction, code a comma as the operand. For example:



ENTRY (Identify Entry-Point Symbol)

The ENTRY instruction identifies linkage symbols that are defined in the program containing the ENTRY instruction and can be referred to from other programs.

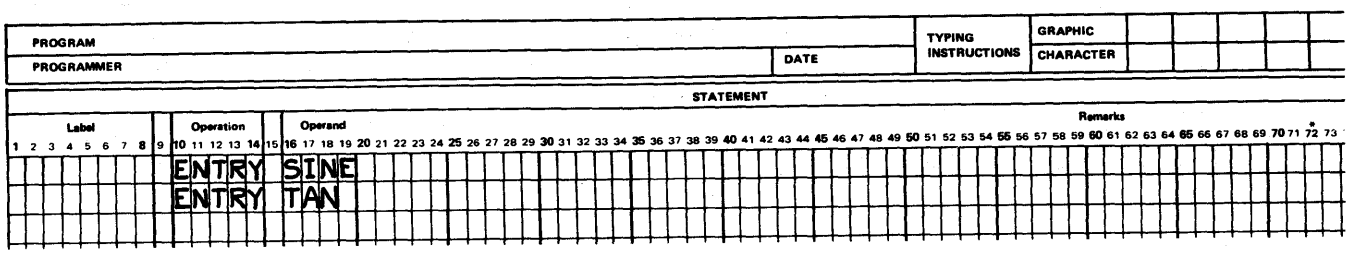
The format of the ENTRY instruction is as follows:



Operand

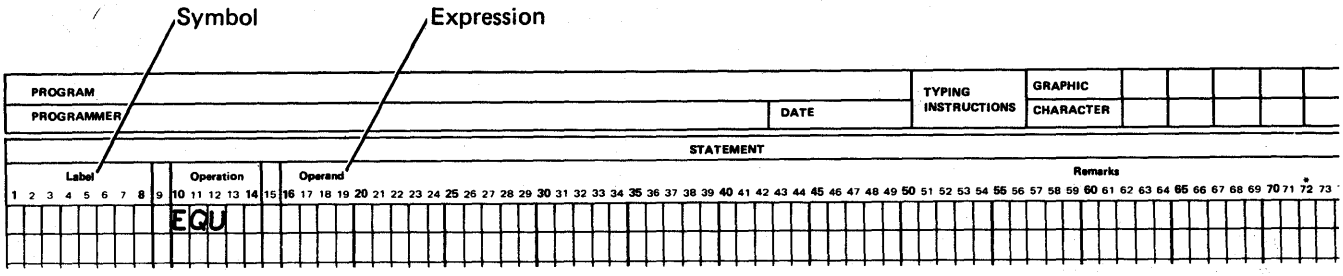
The symbol in an ENTRY instruction operand field can be referred to by another program by using an EXTRN instruction provided that the program uses the same symbol in the operand of the EXTRN instruction. The symbol used in the operand field, for both EXTRN and ENTRY instructions, has a limit of 6 characters. See *EXTRN-Identify External Symbols* later in this chapter.

The following example identifies the statements named SINE and TAN as entry points to the program.



EQU (Equate Symbol)

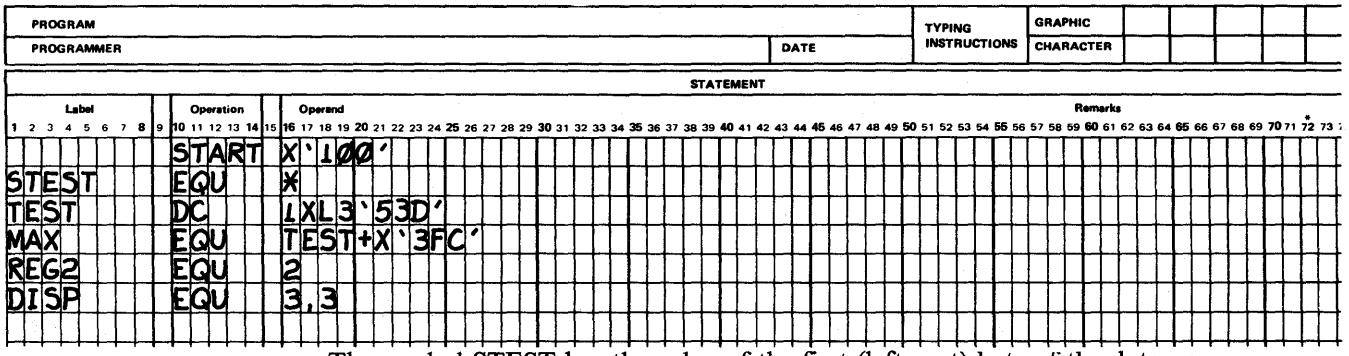
You use the EQU assembler instruction statement to define a label (a symbol) by assigning it to an expression in the operand field. The format is:



The expressions in an operand field can be absolute or relocatable, but you must have defined any symbols you use in the expression. Label and operand field entries are required.

The assembler gives the symbol in the label field the same value and relocatability attributes as the first expression in the operand field. The length of the symbol is the value of the second expression, if present, or the length of the leftmost (or only) term of the expression when the second operand is omitted. When an asterisk (*) or a self-defining term is used as an operand, the default length attribute is 1.

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

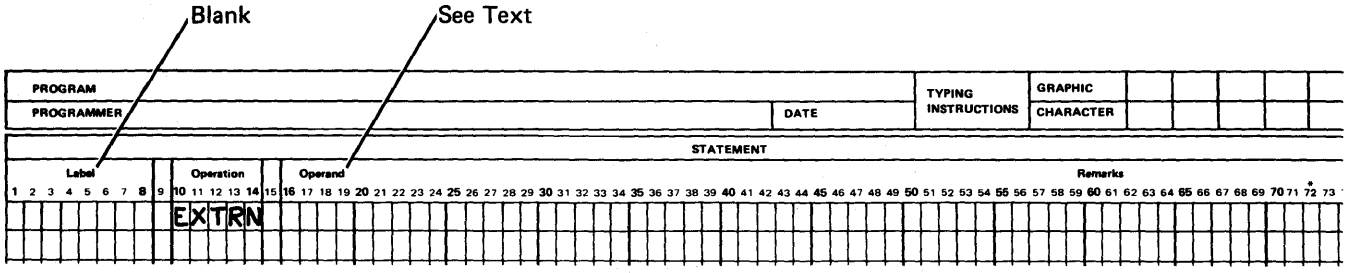


The symbol STEST has the value of the first (leftmost) byte of the data area reserved by the DC instruction. Because the symbol on the DC (TEST) has the value of the last (rightmost) byte, EQU is useful for addressing the leftmost byte. Using the symbol REG2 in any statement has the same effect as using the number 2. DISP has a length of 3 assigned. If DISP is used as the displacement from a register in an instruction, and no length is specified in that instruction, the length value would default to 3.

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

EXTRN (Identify External Symbols)

The EXTRN instruction identifies symbols used in the current program that are defined in another program. Each symbol in the operand of an EXTRN instruction must be identified by an ENTRY instruction or by the module name in some other program. The symbol used in the operand field, for both EXTRN and ENTRY instructions, has a limit of six characters. The format of the EXTRN instruction is as follows:



Label

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

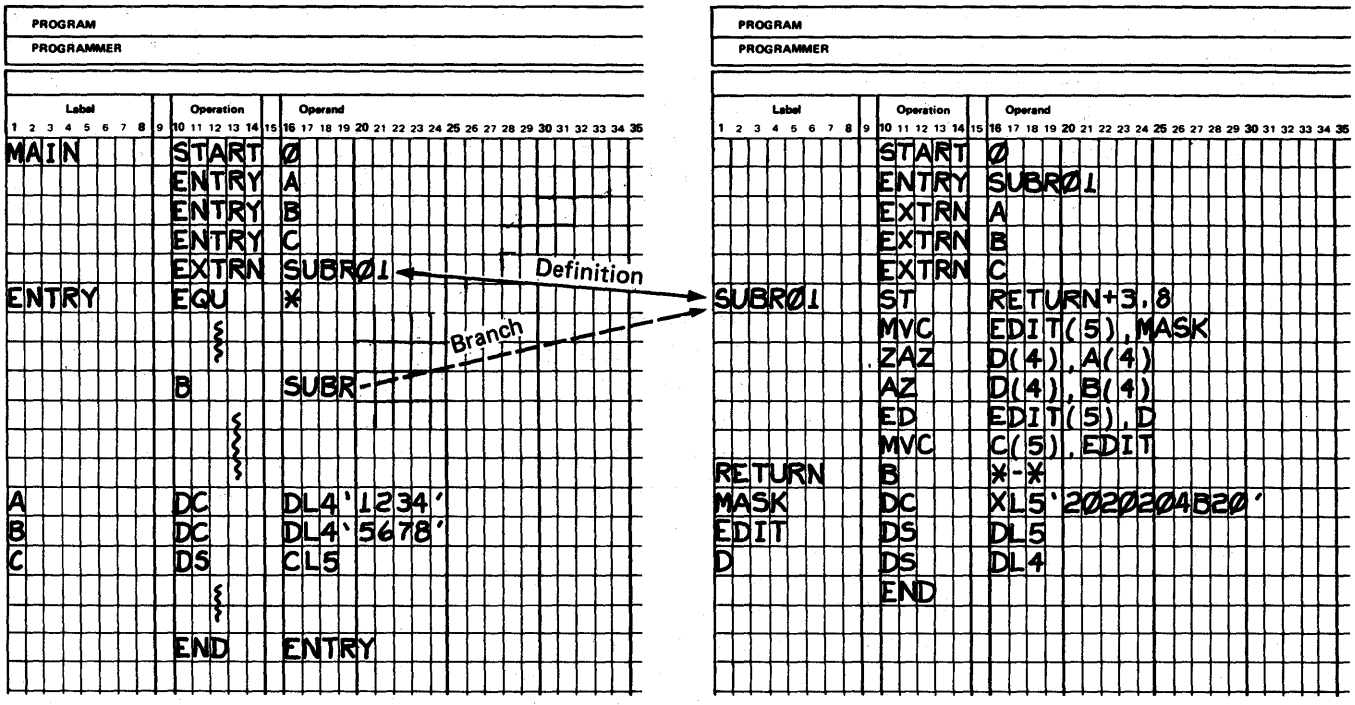
Operand

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 0 nor more than 255. Any symbol in the expression must be previously defined. For an explanation of the subtype values and their meanings, see the *Overlay Linkage Editor Reference Manual.*

An example of Using EXTRN and ENTRY: The example in the following figure shows how these instructions can be used to make two or more programs act as one program by 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 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 external symbol by the main routine. The four symbols (A, B, C, and SUBR01) can now be used interchangeably by both the main routine and the subroutine.

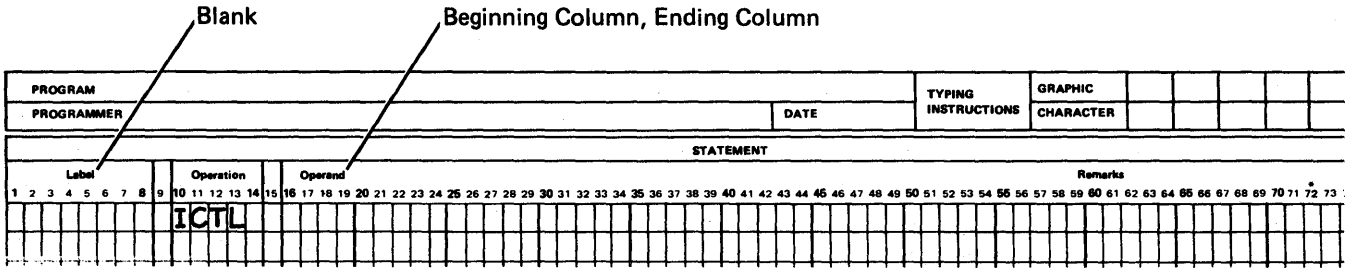
The main routine has control first. It executes instructions, then branches to SUBR01, which is defined as an entry point in the subroutine. Instructions in the subroutine are executed. Notice that the subroutine uses symbols A, B, and C, which were defined in the main routine. Control is then passed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

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



ICTL (Input Format Control)

The ICTL instruction allows the programmer to alter the normal format of the source program statements. If ICTL is used, it must be the first instruction in the source program, including comment statements. An improperly used ICTL instruction ends the assembly. The format of the ICTL instruction is as follows:



Note: When HEADERS or OPTIONS assembler control statements are used, they must be placed before the ICTL instruction, which itself must be placed before all other source statements. The HEADERS and OPTIONS statements are described in Chapter 2.

Operand

The first operand, which can be a number from 1 to 48, specifies the beginning column of the source statement. The second operand, which can be a number from 49 to 96, specifies the ending column of the source statement. The column after the second operand must be blank.

If no ICTL instruction is used in the source program, the assembler assumes that the beginning column is 1 and the ending column is 87.

ISEQ (Input Sequence Checking)

The ISEQ instruction checks the sequence of source records. The assembler does not check the sequence of record numbers unless requested to do so by the ISEQ instruction. Sequence checking begins with the first record after the ISEQ instruction. The sequence number of each following record is compared to the sequence number of the previous record. The ISEQ instruction does the following:

1. Checks the sequence entries on source statement records for ascending order.

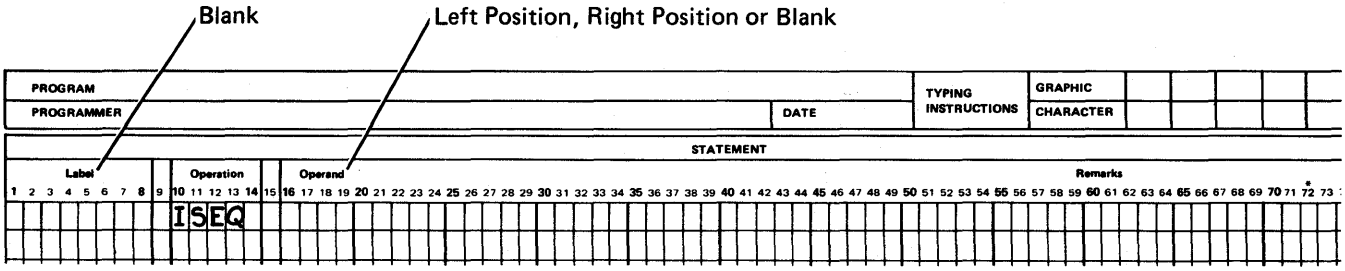
2. Flags statements that are out of order and statements without sequence entries in the assembler listing.

3. Indicates the total number of flagged statements at the end of the listing.

For example, with sequence values of 13, 27, 31, 6, 8, 45, 47, blank, and 48, the record numbered 6 and the record without a sequence value (blank) 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 record numbered 8 would not be flagged, because it is sequential with the record just before it.

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

The following is the ISEQ instruction format:



The label field must be blank.

Operand

The operand identifies the positions of the statements to be read for the sequence number. To check for sequence, specify the leftmost position (no lower than position 73) and the rightmost position (no higher than position 96), separated by a comma. The operand field specified to be checked can be from 1 to 8 positions long.

An ISEQ instruction with no operand ends the checking operation. Checking can be resumed by using another ISEQ instruction.

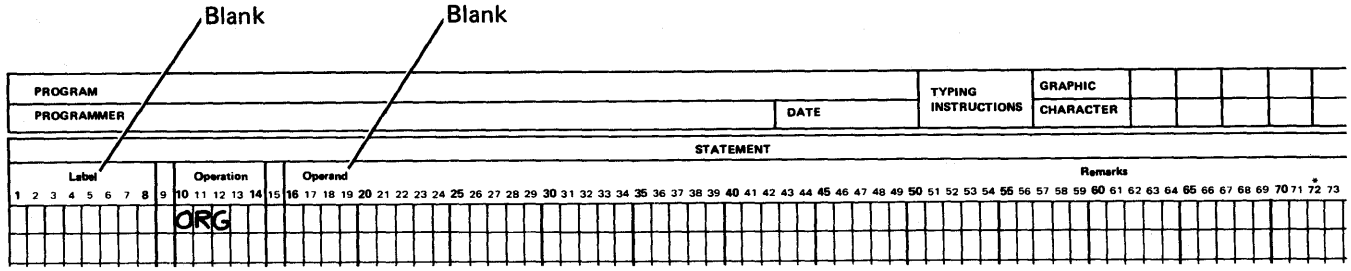
ORG (Set Location Counter)

The ORG instruction alters the setting of the location counter and lets you 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.

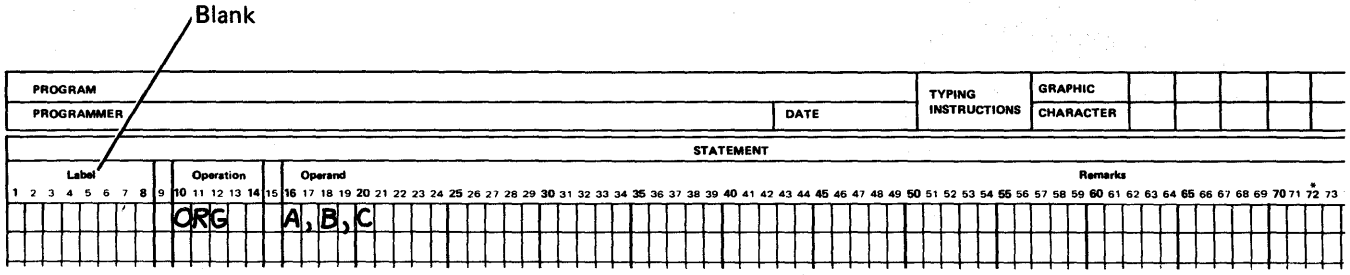
Operand

The operand determines the function of the ORG instruction and can have two formats.

Blank Operand: Restores the location counter to the previous maximum assigned address, plus 1. Do this if the previous ORG statement reduced the location counter to redefine the current program. The format for the blank operand is as follows:



Expression A as Operand: Optionally followed by the two absolute expressions B and C is used in the following way:



The location counter is set to the smallest value that is equal to or greater than A, and is also C more than a multiple of B if B and C are specified. 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:

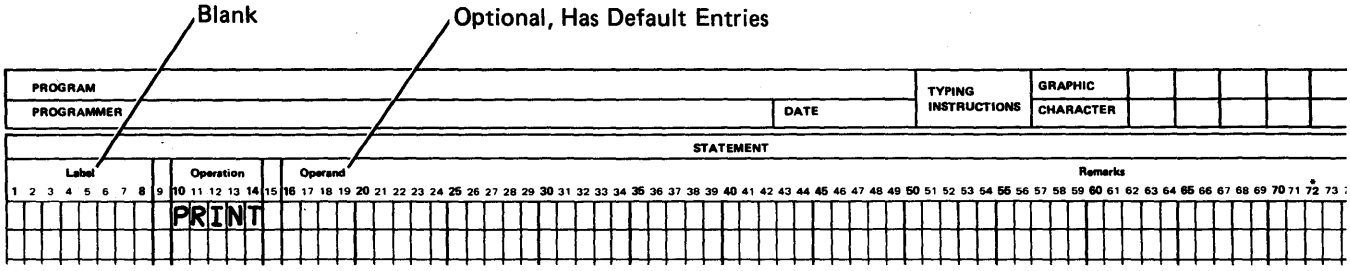
Current Location Counter	A	B	C	New Location Counter
275	*	100	50	350
340	*	100	50	350
350	*	100	50	350
504	*	256	0	512
750	1000	—	—	1000

Any symbols used in the expression must be previously defined. An ORG operation can reduce the location counter for the purpose of redefining the current program, but must not specify a location below the initial location counter value. If an ORG statement reduced the location counter to redefine the current program, an ORG statement with a blank operand can be used to restore the location counter to a value one greater than the previous maximum assigned address.

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

PRINT (Control Program Listing)

You can control the printing of an assembler listing with the PRINT instruction. A program can have any number of PRINT instructions with each one in control until the next PRINT instruction is encountered. The format of the PRINT instruction is as follows:



Label

The label field must be blank.

Operand

The operand describes what will be printed. The operand has three parts, all optional. They have the following descriptions:

ON or OFF: ON, the assembler listing is printed. OFF, no listing is printed. The default is ON.

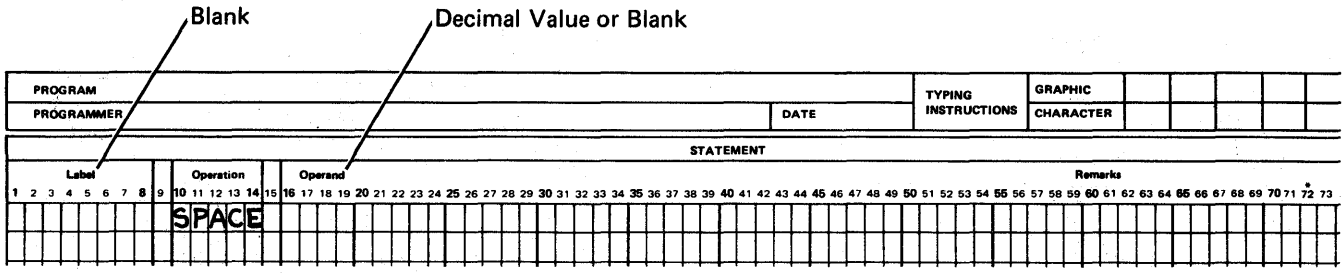
DATA or NODATA: DATA prints complete constants in the assembler listing. NODATA prints only the first (leftmost) 8 bytes of each constant. The default is DATA.

GEN or NOGEN: GEN prints the statements generated by the macroprocessor unless overridden by other print controls, such as PRINT OFF. NOGEN suppresses the printing of statements generated by the macroprocessor. The default is GEN.

Operand entries must be separated by commas. If no operands are entered, the assembler assumes either the previous print instruction or, if this is the first PRINT statement, the default for the previous entry.

SPACE (Line Feed)

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



The SPACE statement is not printed in the assembler listing but does increase the statement counter by one.

Label

The label field must be blank.

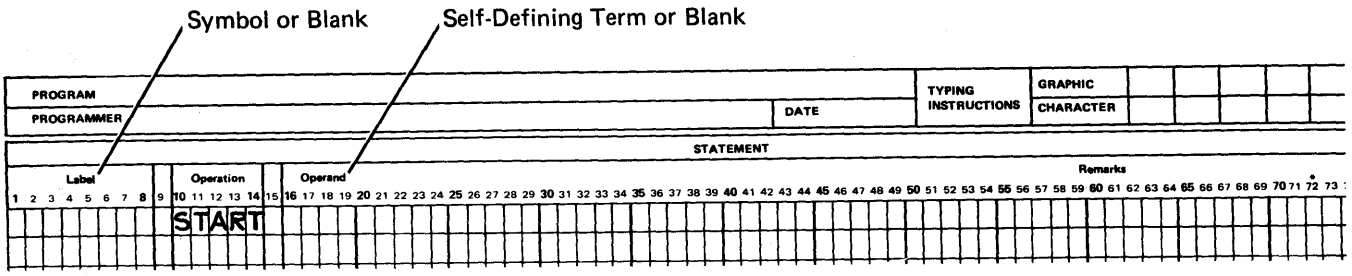
Operand

The operand is an unsigned decimal value that specifies the number of blank lines to be inserted in the listing. If the operand is a blank, a 0, or a 1, one blank line is inserted. If the value of the operand exceeds the number of lines remaining on the current page, the SPACE instruction has the same effect on the listing as an EJECT instruction and will start a new page.

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 in the SPACE instruction, sequence numbers or remarks appearing before the 87th byte cause assembly errors.

START (Start Assembly)

The START instruction specifies an initial location counter value for the program. If no START instruction appears in the program, the initial location counter value is set to 0. The format of the START instruction is as follows:



Note: The START instruction must not be preceded by any type of assembler language statement that can either affect or depend upon the value in the location counter.

Label

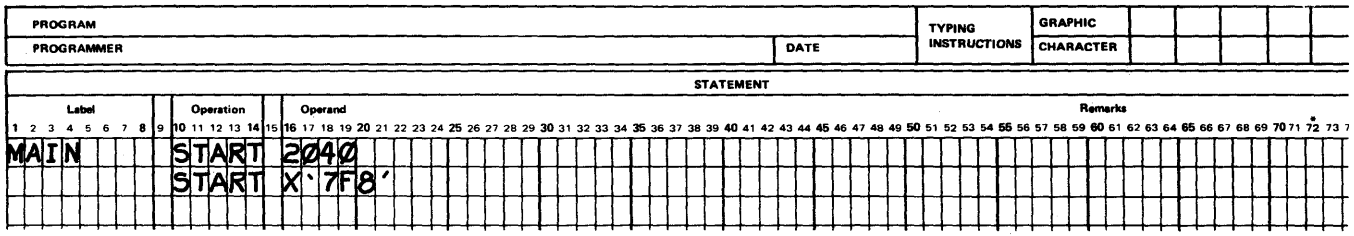
The label is optional and is limited to six characters.

Operand

The operand is a self-defining term that the assembler uses as the initial location counter value for the program. If the operand is omitted, the assembler sets the initial location counter value of the program at 0.

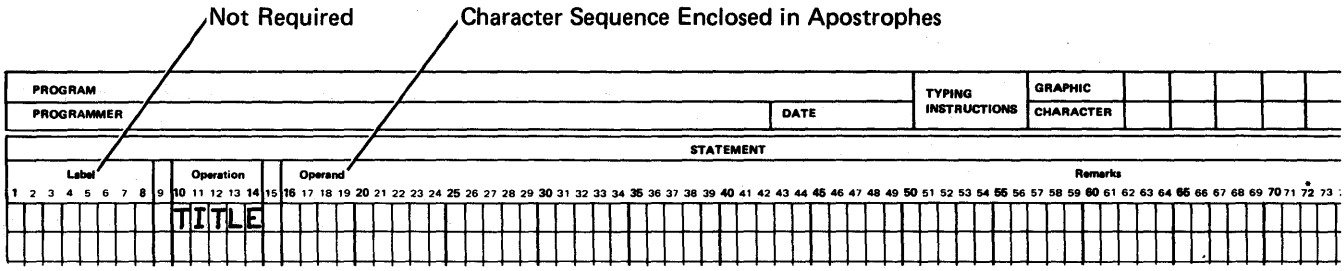
If a symbol names the START instruction, the symbol is established as the name of the object program. If a symbol name is not specified, the object program is assigned the default name ASMOBJ and a diagnostic message is issued.

For example, either of the following START instructions indicate an initial assembly location of 2040. In addition, the first statement establishes MAIN as the object module name.



TITLE (IDENTIFY LISTING)

The TITLE instruction prints the operand field on each page of an assembler listing. The format of the TITLE instruction is as follows:



Label

The label field is not required, but can contain up to six alphabetic or numeric characters in any combination. The label of the first TITLE instruction is used on the title header line of each listing. The labels of all other TITLE instructions are ignored.

Operand

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

You can use more than one TITLE instruction in a program. Each TITLE instruction creates the heading for the pages of the assembler listing that follow it, until another TITLE is encountered. Each TITLE instruction advances the listing (skips) to a new page before the heading is printed. The TITLE statement is not printed in the source listing, but does increase the statement counter by 1.

USING (Use Index Register for Base Displacement Addressing)

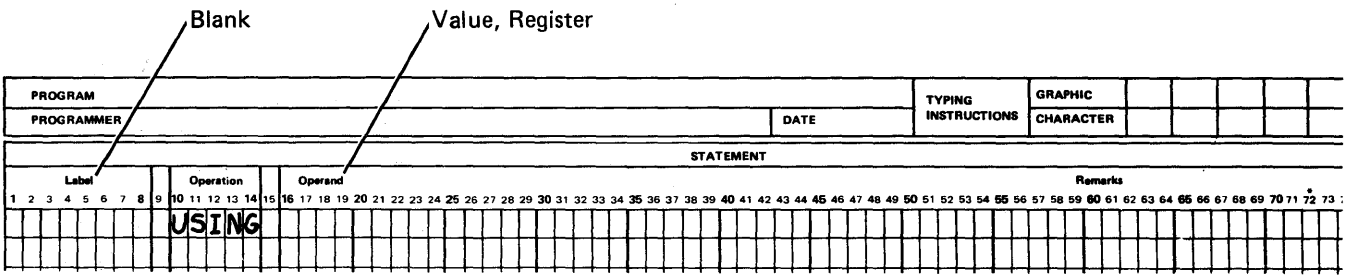
The USING instruction specifies the index register to be used for base displacement addressing on labeled instructions and specifies the relocatable value to be used to compute base displacements during assembly.

Notes:

- A USING instruction does not load the index register because it is executed only during assembly and no code is generated.
- It is the programmer's responsibility to see that the base address value is placed in the index register during program execution. See the text describing the operand.
- The USING statement is not required if you code only absolute displacements.

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

The format of the USING instruction is as follows:



Operand

The operand, Value, is a relocatable expression whose value must be in the range 0 to 65535. The operand, Register, is an absolute expression specifying the index register to contain the base address (represented by value) during program execution. Register must have a value of either 1 or 2.

You can use two USING instructions to specify an index register as a base register for two different portions of main storage. You must change the value in the index register currently used as a base register. The assembler computes displacement from the specified new value (by means of the second USING statement) until another USING or DROP statement is encountered.

Chapter 4. Creating Macroinstructions

A macroinstruction represents a sequence of machine and/or assembler instructions, including other macroinstructions. However, before you can code a macroinstruction, a macroinstruction definition must be previously coded and named, either by a user, or by IBM. That name can be used in an assembler source program to represent the predefined series of instructions. The macroinstruction definition must also reside in the assembler library (#ASMLIB), the system library (#LIBRARY), or the library specified as the MACRO library on the ASM procedure or the ASM help screen.

The IBM-supplied macroinstructions are discussed in Chapter 5 in this manual.

Macroinstruction Definition

A macroinstruction definition resembles a small program. It consists of a prototype or *skeleton* containing symbols, parameters, and statements that specify values for those parameters. The definition and its parameters are used by the macroprocessor to create the set of instructions in the source program during assembly.

The prototype statement is used to assign a name to the macroinstruction and to define the parameters of the corresponding macroinstruction statement. You would use the name of the prototype statement to code the macroinstruction into an assembler source program.

The control statements that follow the prototype statement are similar to assembler language instructions. They usually contain an operation mnemonic, operand, and remarks. Labels can also be assigned to control statements. Control statements must be coded in a specific sequence within the macroinstruction definition. The following illustration shows the relationship between the various parts of a macroinstruction definition.

Prototype statement..	(label) name	operands
Control statement....	(label) operation	operands
Control statement....	(label) operation	operands

You can use standard assembler coding forms or the SEU assembler format to code the components of a macroinstruction definition. The format of a macroinstruction is described in this chapter under *Macroinstruction Format*. The rules, or coding conventions, for coding macroinstructions follow. The rules for coding macroinstruction statements are at the end of this chapter.

Macroinstruction Coding Conventions

The following are the detailed rules or conventions that must be understood and followed to code workable, effective macroinstructions.

Sequence Symbol

Sequence symbols provide labels that can be branched to and, therefore, determine the sequence in which macroinstruction 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.

Character String

A character string is any combination of alphameric, special characters, and blanks and is enclosed in single apostrophes. For every apostrophe that is required as a data character in a character string, two apostrophes must be coded in succession. A character string can be from 1 to 50 bytes long.

Note: Special characters refer to the characters other than alphabetic and numeric that are available in the System/36 character set.

Character Expression

A character expression is a term, null term, or combination of terms enclosed in single apostrophes that can be reduced to a character string from 0 to 50 bytes long. Terms are either literal strings of any of the 256 hexadecimal combinations possible for each byte, except an ampersand or variable symbols. A null term is specified by two consecutive apostrophes. If an apostrophe is required as a data character, it must be entered as two consecutive apostrophes inside the delimiting apostrophes. In expressions with multiple terms, such as:

```
'DEPARTMENT-&DEPT,BUILDING 01'
```

if &DEPT is a variable symbol containing 47A, then the expression will expand to:

```
'DEPARTMENT-47A,BUILDING 01'.
```

All the rules of concatenation apply (see *Concatenation* in this chapter).

Substring

Substring selects specific sequential characters from a character string defined in a character expression. A substring is specified as a character string or its label (m,n) where m and n are each a valid arithmetic expression. The starting character of the substring is m; the length of the substring is n. The following rules apply when you are specifying substrings:

- The value of **m** must be greater than 0.
- The value of **n** must be 1 or greater.
- If the value of **n** is 0 or if the value of **m** is greater than the length of the character string, the substring has no value.
- If the value of **n** were greater than the remaining length of the character string, the substring is all the remaining characters of the character string.

Note: There can be no blanks between the closing single apostrophe of the character string and the left parenthesis of the substring.

The following is an example of creating a substring:

The original character string &CHAR is ABCDEFGHIJKL.

The desired substring contains DEFGH (five characters from position 4).

The substring is coded as 'ABCDEFGHIJKL'(4,5) or &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, commas, blanks, dashes, and equal signs become delimiters. A decoded alphameric value can be up to 50 bytes.

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 $, #, @, _ (no other character or blanks can be used).

Note: The ampersand is a restricted character and cannot be used anywhere else or it will cause an error, ASM-5402.

Attribute

The kind of value assigned to a variable symbol in the variable symbol table is called an attribute. The attributes are:

- Numeric value
- Character string value
- Null value
- Binary value.

There are two types of variable symbols: symbolic parameters and set symbols.

Symbolic Parameter

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

Positional Parameters: Positional parameters are represented by variable symbol names. Positional parameters appear before the keyword parameters in the prototype record. Each positional parameter is written as an & (ampersand) followed by an alphabetic character, $, #, @, or _, followed by as many as five alphabetic or numeric characters, followed by a comma. Positional parameters appear in your macroinstructions as parameter values positioned before keywords and in the same sequence that they had in the prototype.

Keyword Parameters: Keyword parameters are variable symbol names followed by a dash, and immediately following the dash, a parameter value, a comma, or, if the keyword parameter is the last parameter in a macroinstruction, a blank. If a parameter value is included, that value is used. If a parameter value is not included, no default value is used. Keyword parameters follow positional parameters in the prototype statement. Each keyword parameter is written as an & (ampersand) followed by an alphabetic character, $, #, @, or _, followed by five alphabetic or numeric characters.

Keyword parameters on user macroinstruction statements have a label similar to the prototype definition statement; however, the lead ampersand (&) is deleted, &KYWORD becomes KYWRD-, 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 follows:

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

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

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

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

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

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

Set Symbol

A set symbol is a storage area defined by global or local statements. The values assigned to these symbols can be changed by the macroprocessor by use of set statements.

Three different kinds of set symbols can be used:

- Arithmetic set symbols are defined by GBLA (arithmetic global) and LCLA (arithmetic local) statements and are assigned values by SETA (set arithmetic).
- Binary set symbols are defined by GBLB (binary global) and LCLB (binary local) statements and are assigned values by SETB (set binary) statements.
- Character set symbols, which are defined by GBLC (character global) and LCLC (character local) statements and are assigned values by SETC (set character) statements.

Global: A global set symbol is defined by a global statement. This symbol has a storage area assigned to it only once for each program assembled. The same set symbol can be defined in other macroinstruction definitions in the program, but the storage area remains as that of the original. Global set symbols are a primary means of passing information to macroinstruction definitions called later in the program.

Note Be careful when using global set symbols because they retain values and spaces in the symbol table even when they are not being used. Do not use global set symbols when they are not needed; they can cause the symbol table to overflow.

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

&SYSNDX

&SYSNDX is a system variable that might be concatenated with other characters to create unique names for macroinstruction definition statements and generated assembler source instructions. &SYSNDX must not be used as a variable symbol or symbolic parameter. SYSNDX cannot be used as a keyword or positional parameter. The 3-digit number 001 is the value assigned to &SYSNDX when the first macroinstruction definition is processed. The value is increased by 1 for each subsequent macroinstruction definition processed in the program.

&SYSNDX has a maximum value of 999. Therefore, the number of macroinstructions in one job must not exceed 999 when &SYSNDX is used.

Note: No diagnostic messages exist for the incorrect use of &SYSNDX.

Count Function

The count function determines the length, in bytes, of the value assigned to a symbolic parameter. This length is obtained by prefixing K′ to the label of a symbolic parameter. For example, if &LIST equals ABCDEFG, the K′&LIST equals 7.

You can refer to the count function only in the operand of a macroprocessor 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 of 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 for up to three nested levels.

The following rules apply to arithmetic expressions:

- Terms must be separated by operators.
- Operators must be separated by terms.
- No more than three nested levels of parentheses are allowed.
- Parentheses must be balanced; that is, for each left parenthesis there must be a right parenthesis.
- Unless a left parenthesis is the first element in the expression, there must be an operator or another left parenthesis immediately before it.
- A left parenthesis must be immediately followed by a term or another left parenthesis.
- A right parenthesis must be immediately preceded by a term or another right parenthesis.
- 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 -8388608 to 8388607). An expression is reduced to a single value as follows:

- Parenthesized expressions are evaluated from the innermost set of parentheses outward.
- Multiplication and division are performed before addition and subtraction. All operations are performed from left to right.

Continuation

Only prototype statements can be continued. Any character in position 72 following a comma after the last operand indicates that a continuation line of the prototype statement follows. Columns 1 through 15 must be blank. At least one operand, beginning in position 16, must appear on every continuation line of a prototype statement. Only nine continuation lines can be used, giving a maximum statement of ten lines 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:

- A symbolic parameter or set symbol is immediately before or after another symbolic parameter or set symbol with no delimiter between them.
- Characters are immediately before a symbolic parameter or set symbol with no delimiter between them.
- Characters are joined to the symbolic parameter immediately before them or to a set symbol by a period between them.

You can concatenate symbolic parameters, set symbols, and character strings of AIF statements. However, model records and assembler instructions can concatenate only symbolic parameters or set symbols and alphameric values.

Creating Macroinstruction Definitions

You must use definition control statements to create macroinstruction definitions. The values established in the definition control statements are used by the macroprocessor to generate assembler and/or machine instruction statements. The following list shows the definition control statements in the order that they must appear in a macroinstruction definition:

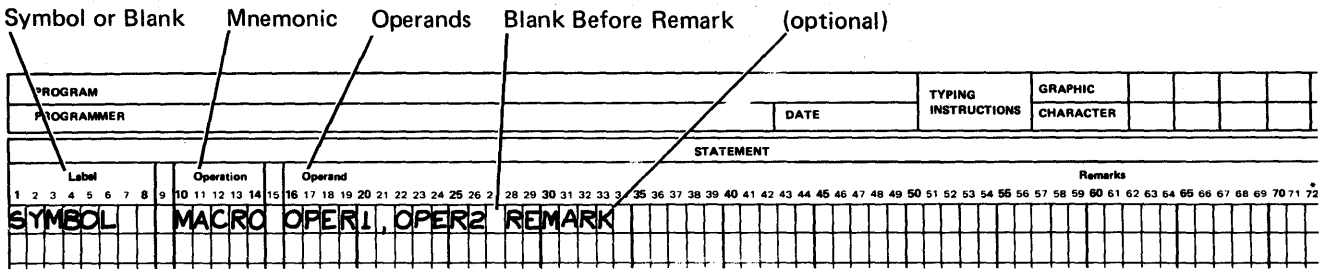
MACRO (required)
Prototype (required)
Global declares
Local declares
Table
Table definitions
TEXT (required)
MEXIT
MEND (required)

Definition Control Statement Format

A definition control statement can contain up to four entries: name, operation, operands, and remarks. Name, operation, and operands are position-dependent and must begin in positions 1, 10, and 16, respectively. The remarks entry can occur in any position following the operands if at least one blank separates it from the operands.

Macroinstruction Format

The format of a macroinstruction follows:



Label

If the label field on the macroinstruction contains a symbol, and if a symbolic parameter is used in the label field of the associated prototype statement, the symbolic parameter is assigned the value of the symbol in the macroinstruction. (See *Prototype* in this chapter.)

If the label field on the macroinstruction contains a symbol, and if the label field of the associated prototype statement does not contain a symbolic parameter, the symbol is ignored.

If the label field on the macroinstruction is not used, and if a symbolic parameter is used in the label field of the associated prototype statement, the symbolic parameter is assigned a null value. The length of the label field is up to 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 statement.

Operand

The operand can contain either keyword or positional parameters, or both. The value assigned a keyword or positional parameter in a macroinstruction is assigned to the corresponding symbolic parameter defined in the associated prototype statement.

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

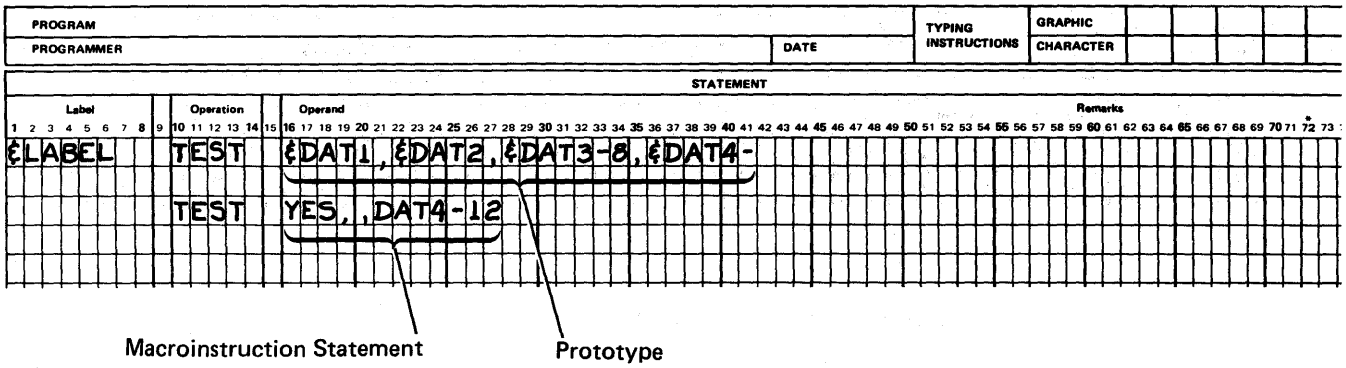
A keyword parameter defined with a value in a prototype statement retains the assigned value, unless an operand containing the corresponding keyword appears in the associated macroinstruction.

The keyword parameters can 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 in the operand must correspond to one of the symbolic parameters in the operand of the associated prototype statement. However, 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, not the keyword. Commas in succession indicate the omission of positional parameters and the assignment of null value. An example of a macroinstruction statement and its relationship to the prototype definition control statement follows:



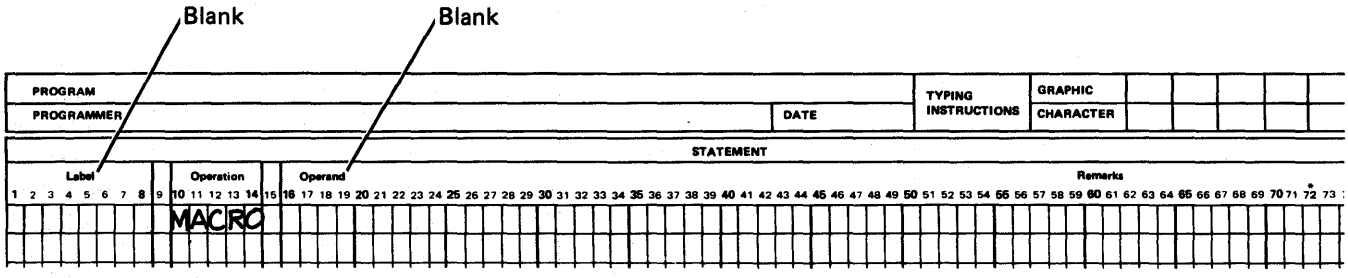
&DAT1is assigned **YES** by the macroinstruction.
&DAT2 is assigned **null** value by omission.
&DAT3 is assigned **8** by prototype default.
&DAT4 is assigned **12** by the macroinstruction.

Macroinstruction Definition Control Statements

You use macroinstruction definition control statements when you are creating macroinstructions to define and describe the specific macroinstruction that you are creating.

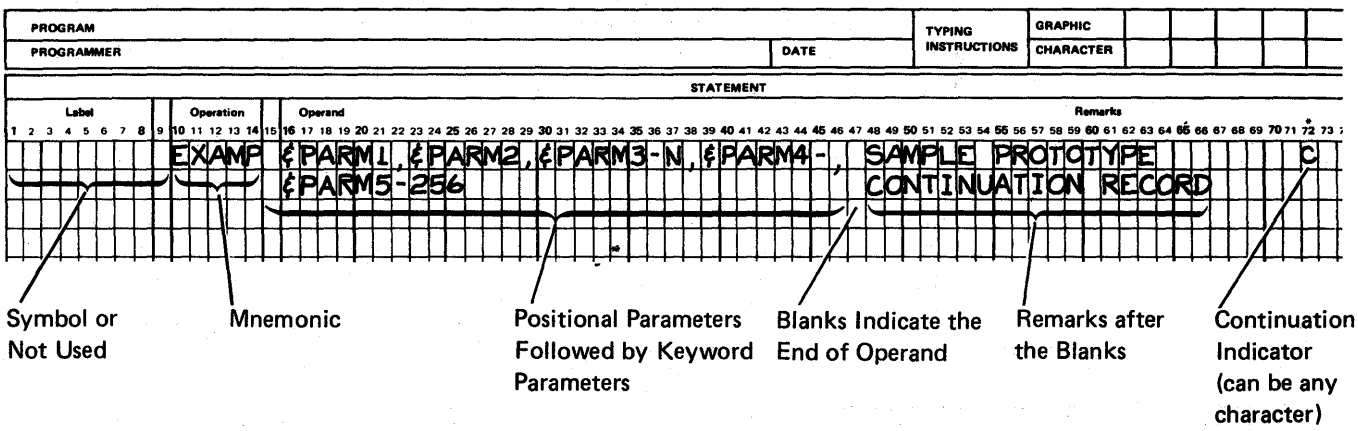
Header

The header statement is required and marks the beginning of a macroinstruction definition. It must be the first control statement in the definition. No more than one comment (an asterisk in position 1) can be before the header. A comment appearing before a header is not generated as source output. The format of the header statement follows:



Prototype

The prototype statement is required and defines both the mnemonic operation code that must be used and any parameters (operands) that can be used 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 change the assembler source instructions generated for each user macroinstruction. The prototype must be the second control statement in a macroinstruction definition. The format of the prototype statement follows:



Label

The name of the prototype statement is optional. If the keyword prototype statement is continued, the label and operation entries must not be repeated on the continuation lines.

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 5 characters long. If the keyword prototype statement is continued, the operation entry must not be repeated on the continuation line.

Operand

The operand consists of either positional or keyword symbolic parameters or both, separated by commas. A blank indicates the end of the operands.

A comment may be entered following the operands as long as at least one blank is placed between it and the operands.

If the prototype statement is continued, at least one operand beginning in position 16 must appear on every continuation line. A prototype statement can have up to nine continuation lines. The preceding example shows a continued prototype statement.

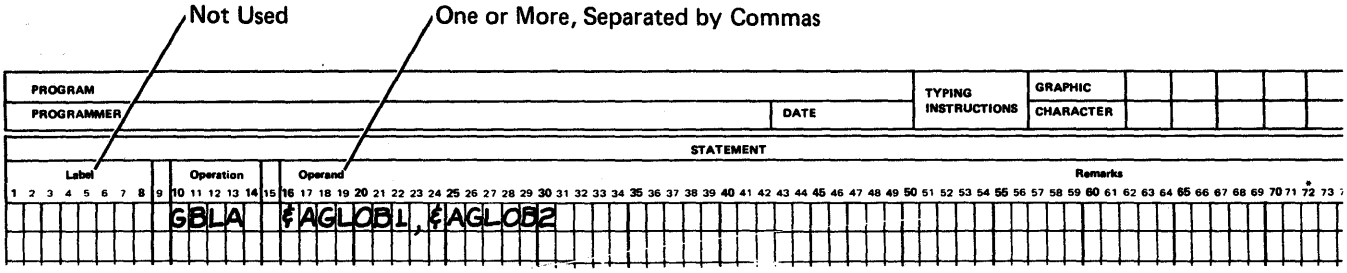
Global

Three types of global statements can be used in macroinstruction definitions to generate global set symbols: arithmetic, binary, and character. A global set symbol 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 statement. Subsequent global statements can specify the same symbol name, but the global set symbol value is not reestablished. When you again declare (use) the symbol, you must specify it as the same type, arithmetic, binary, or character.

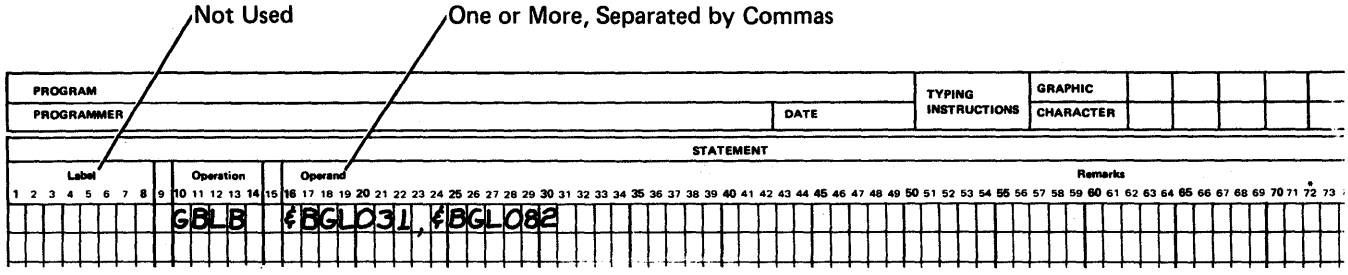
GBLA (Arithmetic Global)

The GBLA global mnemonic specifies an arithmetic set symbol. Arithmetic set symbols are initialized to 3 bytes of hex 0's. The 3-byte field remains through all value assignments. The format of the arithmetic global record follows:



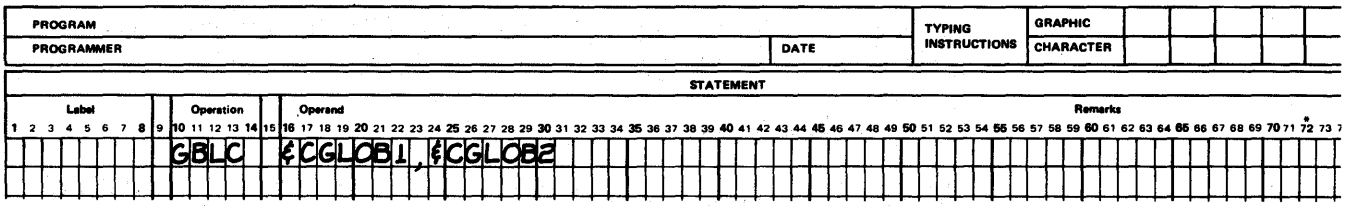
GBLB (Binary Global)

The GBLB global mnemonic specifies a binary set symbol. When a binary set symbol is defined, the defined variable is initialized to 0. The variable can later be set to either 0 or 1 by SETB statements. The format of the binary global statement follows:



GBLC (Character Global)

The GBLC global mnemonic specifies a character set symbol. When a character set symbol is defined, it is given a length of 0. A 0- to 8-byte character field can be assigned by the SETC statement. The assigned characters can be any of the 256 hexadecimal combinations possible for each byte. The format of the character global statement follows:



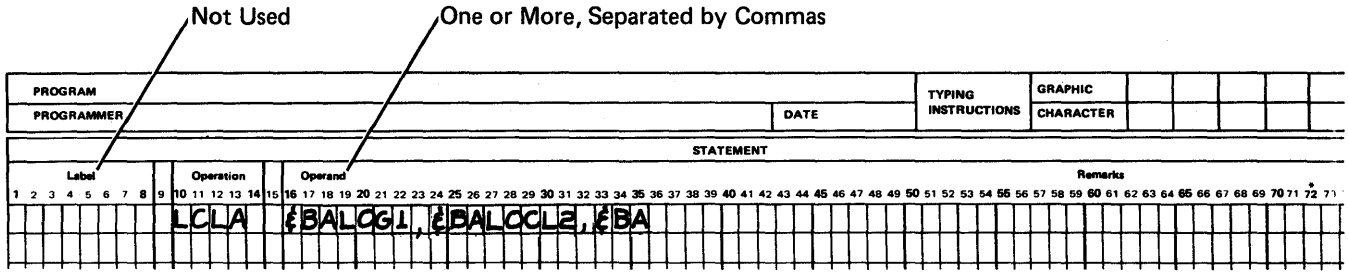
Local

Three types of local statements 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 set symbol statements, if global set symbols are used, or be the first control records following the prototype statement, if global set symbols are not used. Local statements 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.

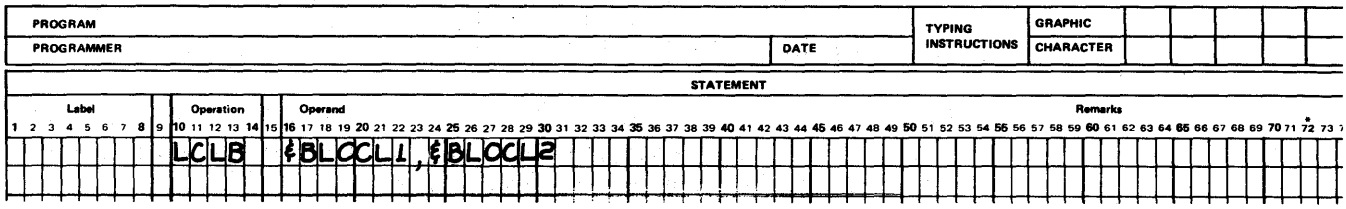
LCLA (Arithmetic Local)

The LCLA local mnemonic specifies an arithmetic set symbol. Each arithmetic set symbol specified is initialized to 3 bytes of hex 0's and remains as a 3-byte field. The format of the arithmetic local statement follows:



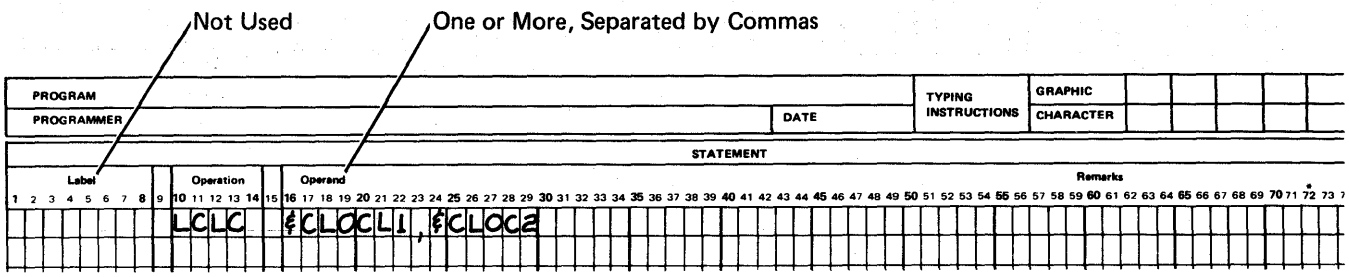
LCLB (Binary Local)

The LCLB local mnemonic specifies a binary set symbol. Each binary set symbol is initialized to 0. The format of the binary local statement follows:



LCLC (Character Local)

The LCLC local mnemonic specifies a character set symbol. Each character set symbol is initialized to a null value and a length of 0. It can then be changed to a character value of from 0 to 8 by SETC. The format of the character local statement follows:

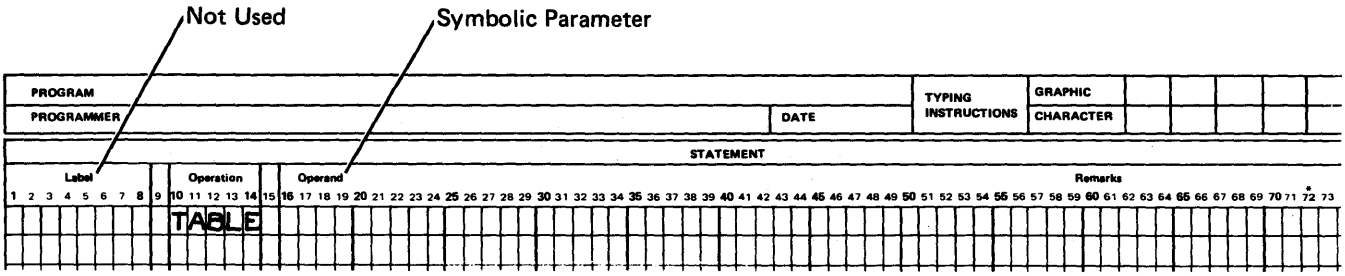


Tables

The TABLE and TABDF (table-definition) statements are used together to define and assign values to tables. These tables are used in and by the assembler program.

TABLE (Table)

The 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 follows:

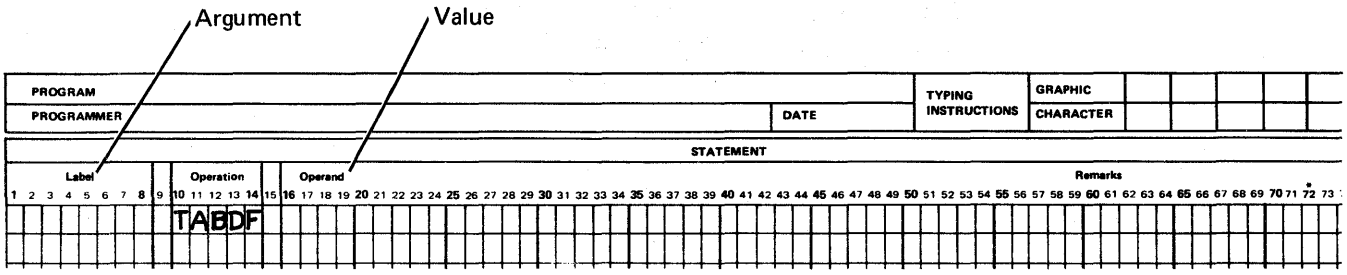


TABDF (Table-Definition)

The TABDF statement assigns values to symbolic parameters specified in table statements. The operand value in a table-definition statement is assigned to the symbolic parameter in the previous table statement, if one of the following conditions is satisfied:

- The label field (argument) of the table-definition statement matches the value previously assigned to the symbolic parameter by the macroinstruction or prototype statement.
- Positions 1 and 2 of the label field (argument) of the table-definition statement are apostrophes, and no value (null) was previously assigned to a symbolic parameter by the macroinstruction or prototype statement.
- The label field (argument) of the table-definition statement is blank. A blank argument assigns the specified value of the operand to a parameter if the parameter does not match an argument specified in an earlier TABDF statement.

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

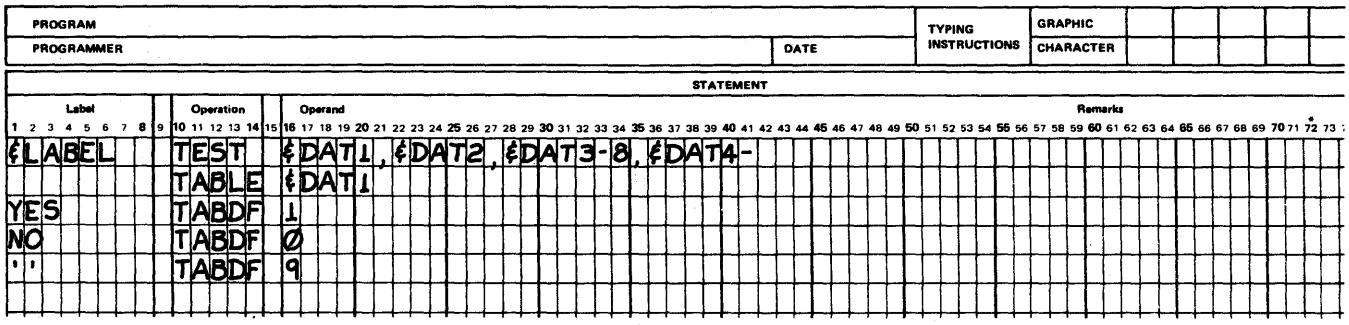


Label

The label is a string of characters with no embedded blanks. The string can be taken from the prototype record or a user macroinstruction.

Operand

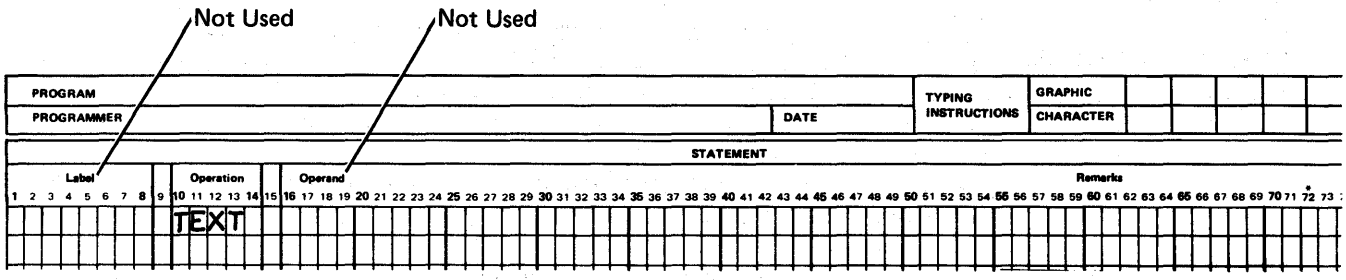
The operand is a character string or an alphameric constant. Following is an example of lines from a macroinstruction definition instruction that define a table or table statement:



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

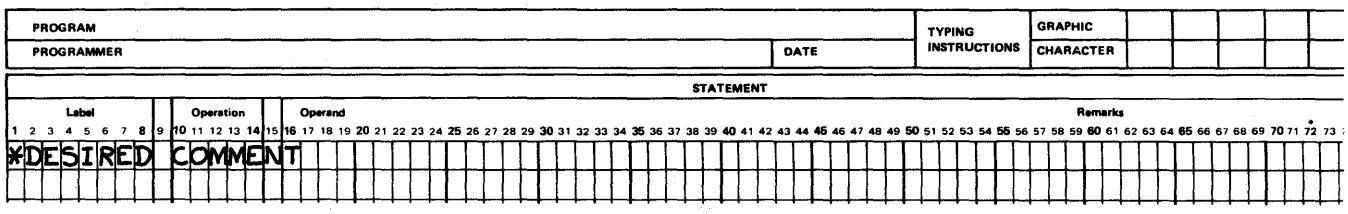
TEXT

The TEXT statement must be present in every macroinstruction definition. The TEXT statement marks the beginning of conditional processing instructions. The definition control statements that can appear before the text statement in the input or output stream are: header, prototype, global, local, table, and table-definition records. Any of these records following the text statement are invalid, and errors result. The format of the text statement follows:



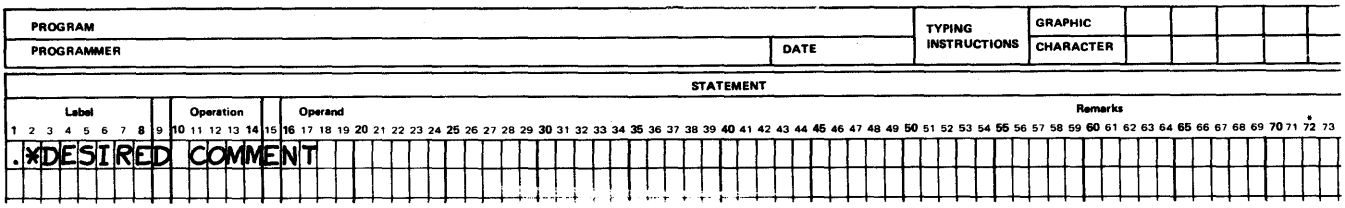
Comment

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



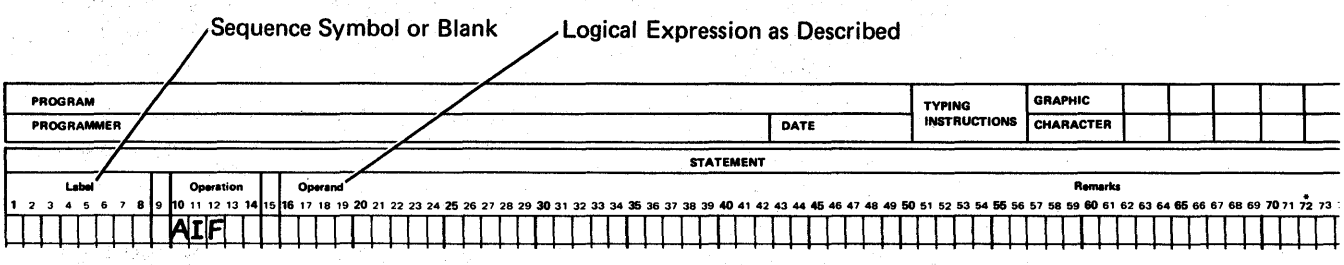
One comment with this format can appear before the header record, but is not generated as source output.

Comments that are internal to the macroinstruction definition can be placed after the header record and before the first trailer record (MEND). These comments are not included in the macroinstruction expansion. The format of an internal macroinstruction comment follows:



AIF (Conditional Branch)

The AIF statement conditionally alters (forward or backward) the sequence in which macroinstruction definition statements are processed. The AIF statement can be used anyplace after the TEXT statement. The format of the AIF statement follows:



Operand

The logical expression in the operand is evaluated to determine whether it is true or false. If the expression is true, the statement named by the sequence symbol in the operand is the next statement processed by the macroprocessor. If the logical expression is false, the next sequential instruction of the macroinstruction definition is processed.

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

- Type attribute checking
- Binary condition checking
- Value checking.

Type Attribute (T') Checking: Permits the user to check the attribute type only in the operand of the AIF record. Attribute checking cannot be performed with set symbols. The following list gives the conditions and meanings:

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 &name and &name1 have the same attribute.

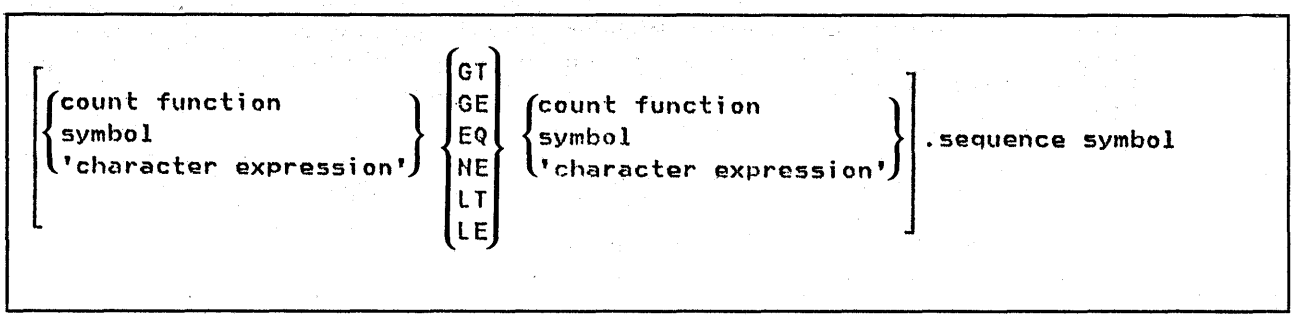
Note: No concatenation of symbols in an AIF operand is supported in T′ processing. If concatenation is specified, an error results.

Binary Condition Checking: Has a format for binary condition checking as follows:

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

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

Value Checking: Has the following format:



Notes:

1. Symbol = any symbolic parameter or set symbol.

2. 'char. expression' = any character expression.

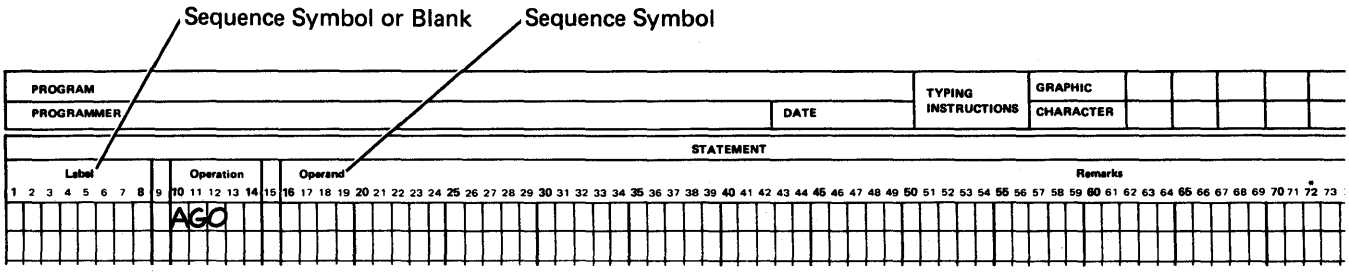
   ```
   GT = greater than
   GE = greater than or equal
   EQ = equal
   NE = not equal
   LT = less than
   LE = less than or equal
   ```

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

AGO (Unconditional Branch Record)

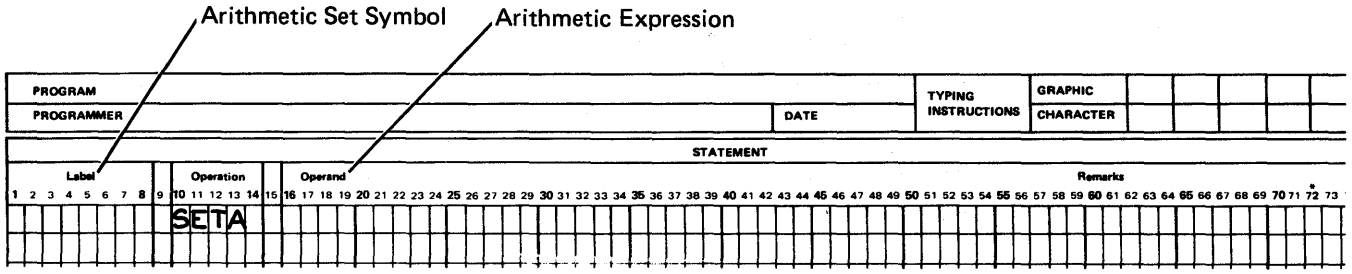
The AGO statement unconditionally alters (forward or backward) the sequence in which macroinstruction definition statements are processed. AGO causes a branch forward or backward to the statement whose name matches the sequence symbol given in the operand of the AGO statement.

The AGO statement can be used anyplace after the TEXT statement and before the MEND statement. The format of the AGO statement follows:



SETA (Set Arithmetic)

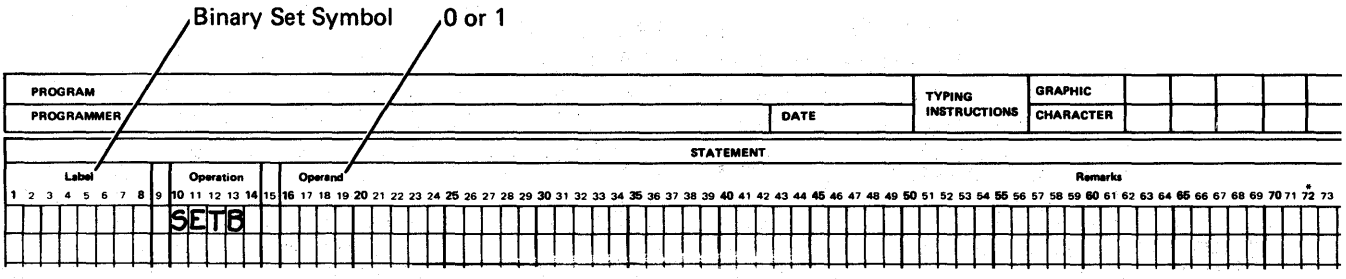
The SETA statement assigns a value to the arithmetic set symbol referred to in the label field. The 3-byte hexadecimal value assigned is derived from an evaluation of the operand field. The SETA statement can be used anyplace after the TEXT statement. The format of the SETA statement follows:



The operand is an arithmetic expression that can contain arithmetic, character, and/or binary set symbols. Any character set symbols used must have a value of from 1- to 8-decimal digits. Binary set symbols are either 0 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 -8388608 to 8388607. If you use the count function as an operand, it must appear alone.

SETB (Set Binary)

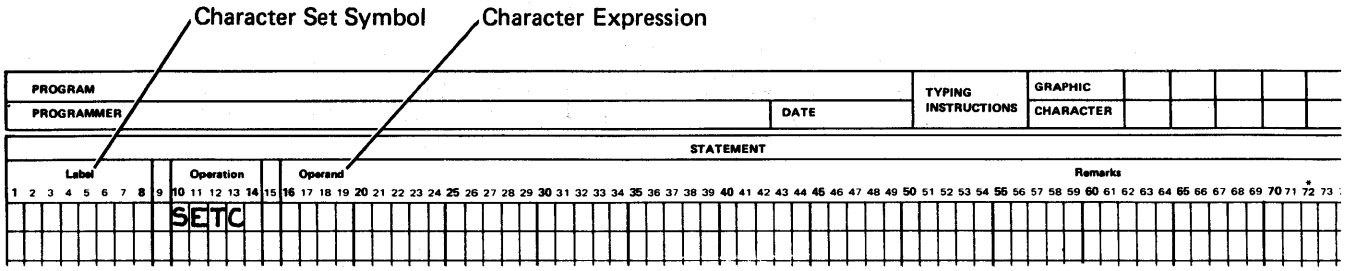
The SETB statement assigns a value of 0 or 1 to the binary set symbol referred to in the label field. The SETB statement can be used anyplace after the TEXT statement. The format of the SETB statement follows:



SETC (Set Character)

The SETC statement assigns an 0- through 8-character expression to the character set symbol referred to in the label 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 statement can be used anyplace after the TEXT statement and before the MEND statement. The assigned characters can be any of the 256 hexadecimal combinations possible for 1 byte. The format of the SETC statement follows:

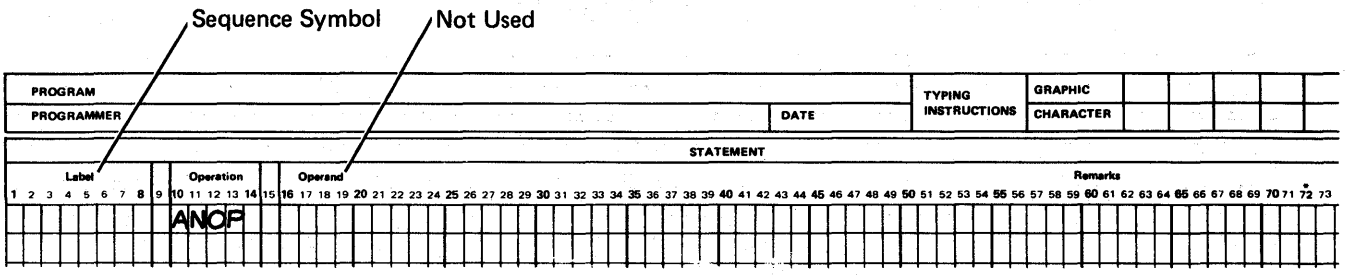


The character expression in the operand, can contain character, arithmetic, or binary set symbols, and can include substring notation. You can assign null values in the character expression by specifying two consecutive single apostrophes or by specifying only variable symbols that are already specified null values.

Arithmetic set symbols you use in the character expression are converted to only their significant decimal digits in the string. All leading 0's are dropped, and, if the value of the arithmetic set symbol is 0, a single decimal 0 is used. Binary set symbols appear as either 0 or 1, and &SYSNDX is given its current value in 3-decimal digits.

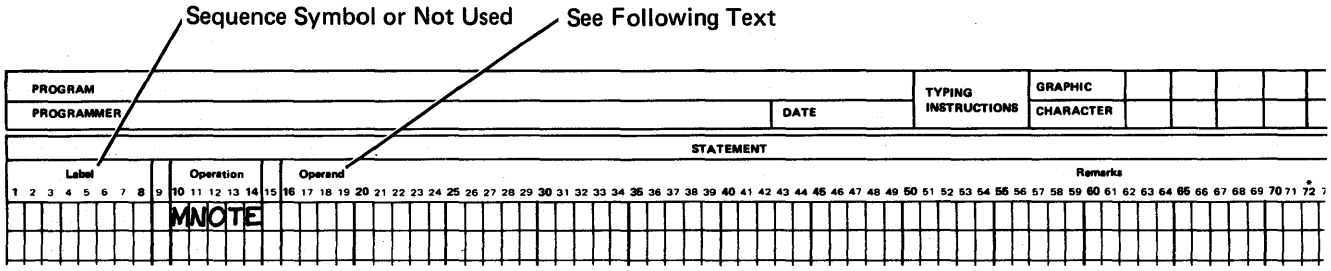
ANOP (Assembly No Operation)

The ANOP statement does not generate any executable machine-language code. The ANOP statement can be used to provide a label (sequence symbol) to which AIF and AGO statements can branch. ANOP can be used anyplace after the TEXT statement and before the MEND statement. The format of the ANOP statement follows:



MNOTE (Message)

The MNOTE statement can be used to generate a message to indicate error severity, if any, associated with the message. The MNOTE statement can be used anyplace after the TEXT statement. The format of the MNOTE statement follows:



The operand consists of a 2-digit severity code, and any information that follows it. The severity code can be optionally followed by either a message or a message identification code (MIC) number and a message ID.

SC = Severity code made up of 2 digits from 00 through 99.

Severity codes are divided into the following classifications:

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

SC = 08 — The macroprocessor generates a special assembler statement that causes the message to be printed on the assembler source listing with a warning (W-error).

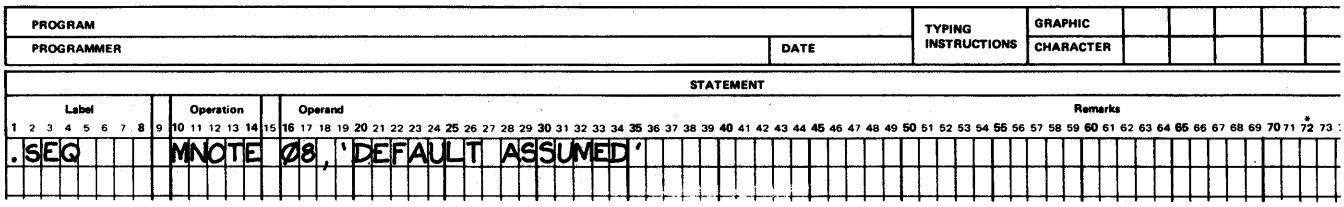
SC > 08 — The macroprocessor generates the message without an * in column 1, which causes the assembler to flag that statement as a hard nonrecoverable error (M-error).

Message: 1 to 50 characters enclosed in apostrophes with no embedded apostrophes. This message appears, as coded, on the assembler source listing.

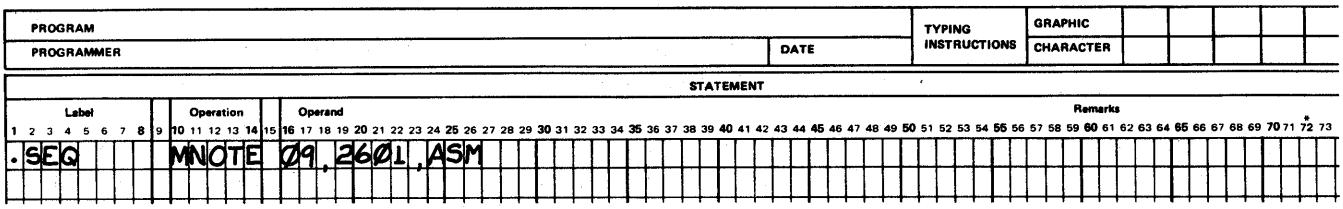
MIC — Message identifier code, a 4-digit code that identifies the message within the message member.

Message ID — This is the 3- or 4-character identifier before the MIC, separated by a dash. For example, if the assembler message ID is ASM and the MIC is 2600, then the printed message would be presented as ASM-2600 followed by the message text.

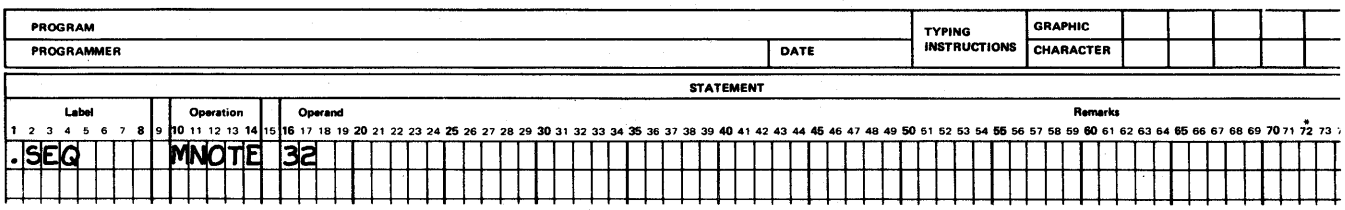
The following shows an MNOTE statement that causes a warning (W-error) and a comment on the source listing:



The following shows an MNOTE statement that causes a hard (M-error) and generates message ASM-2601 as obtained from the assembler message member:

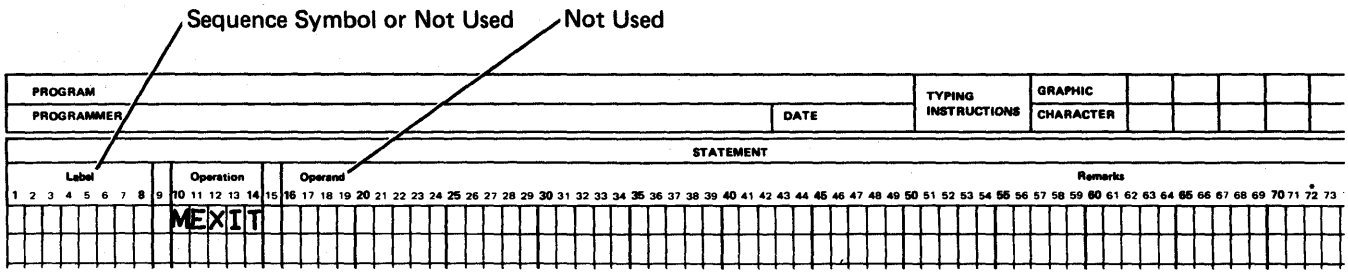


The following shows an MNOTE statement that causes a hard (M-error) but no message:



MEXIT (Logical End)

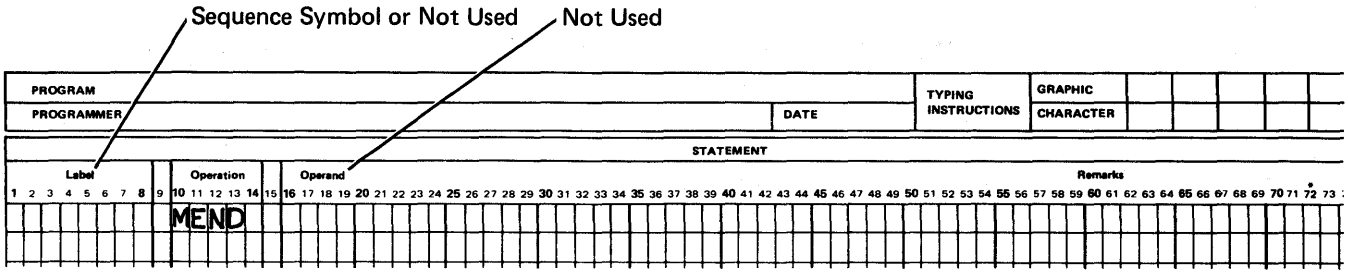
The MEXIT statement ends the macroinstruction definition processing. MEXIT statements can be located anywhere in a macroinstruction definition. The format of the MEXIT statement follows:



MEND (Physical End)

The MEND statement marks the physical end of the macroinstruction definition. The MEND statement is required and must be the last definition control statement in the macroinstruction definition. Processing of the macroinstruction definition ends when this statement is encountered.

The format of the MEND statement follows:



Definition Restrictions

The macroprocessor assumes that any ampersand (&) starts a variable symbol. An ampersand used **anywhere else**, even in a comment, causes error ASM-5402.

Example of A User Macroinstruction Definition

The figure on the next page shows the definition of a user-defined macroinstruction that generates instructions to move more than 256 bytes of data. The following page 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.

IBM-supplied macroinstruction definitions are also shown in Chapter 5.

```
         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 IS ENTERED,
.*                                     ..GO CHECK SECOND PARM, 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
.*
.*********************************************************************
.*                                                                   *
.*       CHECK PARAMETER TWO, 'FROM' ADDRESS LABEL                   *
.*                                                                   *
.*********************************************************************
.MV$001  ANOP
         AIF   (T'&FROM NE 'O').MV$002 IF SECOND PARM IS ENTERED,
.*                                     ..GO CHECK THRID PARM, 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
.*
.*********************************************************************
.*                                                                   *
.*       CHECK PARAMETER THREE, LENGTH OF MOVE.                      *
.*                                                                   *
.*********************************************************************
.MV$002  ANOP
         AIF   (T'&LENGTH NE 'O').MV$003  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$004                 GO SEE IF ALL EDITS PASSED
.MV$003  ANOP
         AIF   (T'&LENGTH EQ 'N').MV$004  IF THE LENGTH PARM IS NUMERIC,
.*                                     ..GO CHECK ERROR SWITCH, ELSE
.*                                     ..WRITE OUT AN ERROR MESSAGE
.*                                     ..AND SET ON THE EDIT SWITCH SO
.*                                     ..THAT NO CODE IS GENERATED.
.*
         MNOTE 08,'PARM 3 (LENGTH) MUST BE NUMERIC.'
&SW      SETB  1
.*
.*********************************************************************
.*                                                                   *
.*       CHECK THE EDIT SWITCH.                                      *
.*                                                                   *
.*********************************************************************
.MV$004  ANOP
         AIF   (&SW).MV$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 2 THROUGH 7.
.*
&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
```

```
ERR LOC  OBJECT CODE      ADDR STMT     SOURCE STATEMENT
    0000                         1 MACSAM   START X'0000'
                                 2 *        @MOVL HERE,THERE,512

    0000 0C FF 022B 042B         4+         MVC   HERE+511(256),THERE+511
    0006 0C FF 012B 032B         5+         MVC   HERE+255(256),THERE+255
                                 6 *        @MOVL HERE,THERE,224,ADDR-HERE

    000C 0C DF 010B 030B         8+         MVC   HERE+223(224),THERE+223
    0012 C2 01 002C              9+         LA    HERE,1
                                10 *        @MOVL HERE,THERE,400,ADDR-@HEREADR

    0016 0C FF 01BB 03BB        12+         MVC   HERE+399(256),THERE+399
    001C 0C 8F 00BB 02BB        13+         MVC   HERE+143(144),THERE+143
    0022 35 01 002B             14+         L     HEREADR,1
                                15 *        @MOVL HERE,,375

W                               17 *08  PARM 2 (FROM ADDR) MAY NOT BE OMITTED.
                                18 *        @MOVL HERE,THERE,HERE

W                               20 *08  PARM 3 (LENGTH) MUST BE NUMERIC.
                                21 *        $EOJ                          END OF JOB
                                22+* LINKAGE TO END OF JOB ROUTINES
    0026 F4 01 04               23+         SVC   X'04',X'01'             EOJ SVC
    0029 04                0029 24+         DC    XL1'04'                 EOJ RIB
                                25+* END OF EXPANSION
    002A 002C              002B 26 HEREADR  DC    AL2(HERE)
                           002C 27 HERE     EQU   *
    002C                   022B 28          DS    2CL256
                           022C 29 THERE    EQU   *
    022C                   042B 30          DS    2CL256
                           0000 31          END   MACSAM

TOTAL STATEMENTS IN ERROR IN THIS ASSEMBLY--     0
```

Using Macroinstructions

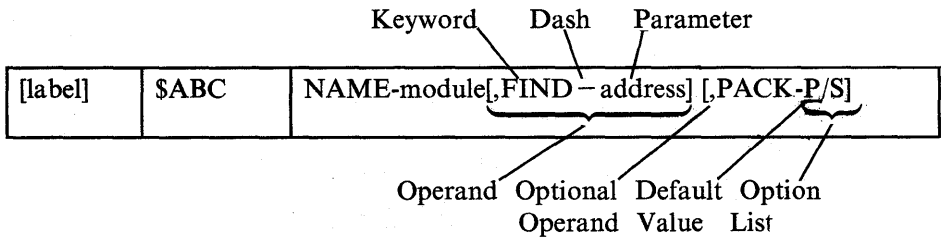
A macroinstruction is written as a source statement. The macroinstruction statement generates a predetermined set of assembler statements when the program is assembled. If a macro library was specified on the ASM procedure or on the second ASM help screen, a comment appears after the commented invocation of the macro call. This comment gives the name of the library where the macro was actually found.

You write macroinstructions as follows:

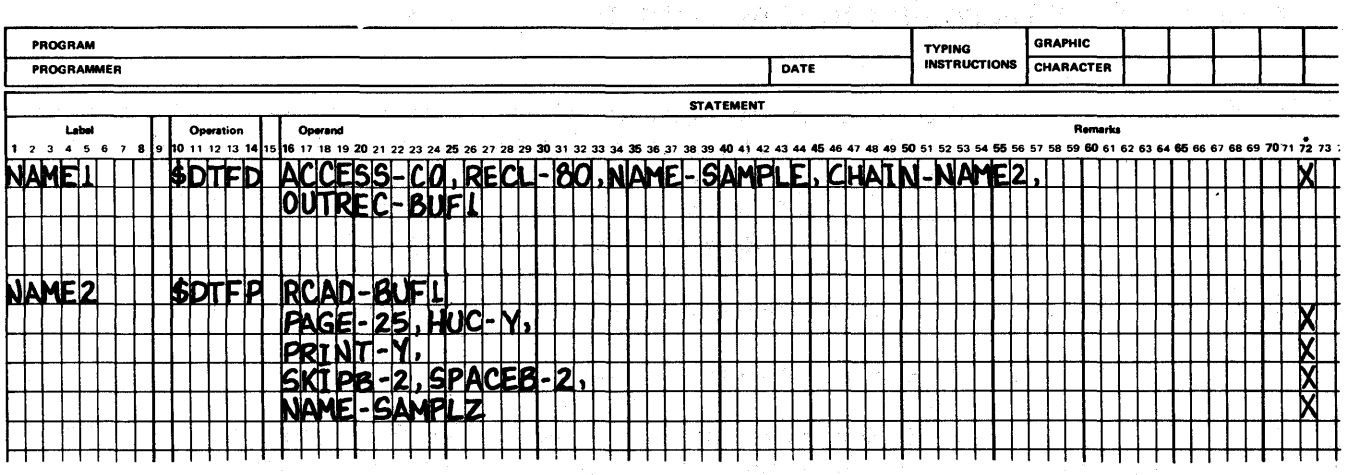
Label	Operation	Operands	Continuation
Symbol or blank	Macro-instruction mnemonic	From none to many- if more than one, separate with commas	Any character in position 72 if continuation is wanted

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

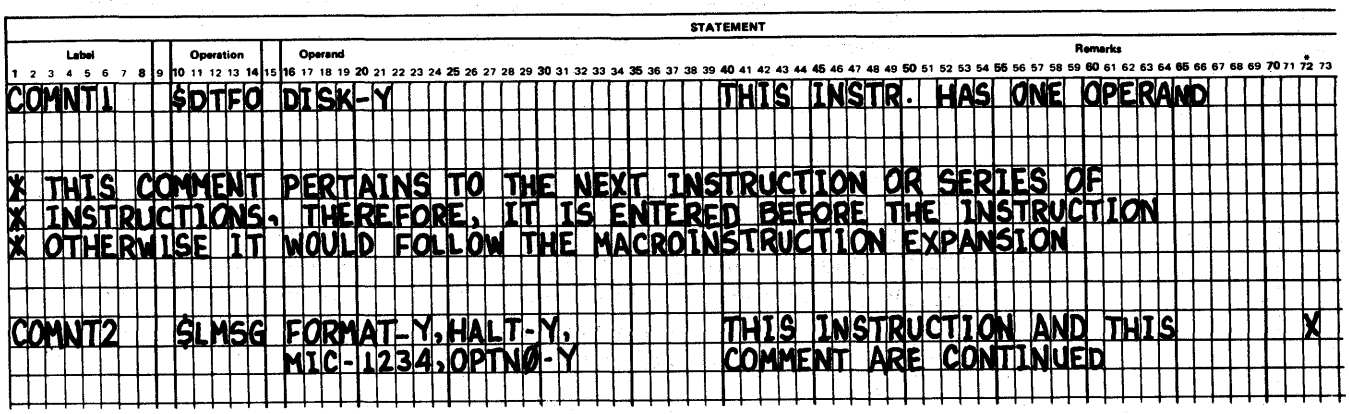
The desired operation mnemonic must appear as specified in that macroinstruction description. The operation code must start in position 10.



The operands specify the options and services that you want to use. Operands must start in column 16. No operands can be entered beyond column 71. If continuation is required, column 72 must contain a character and the last operand before a continuation character must be followed by a comma and at least one 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. To see the use of continuation coding, see the following example:



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 1-line macroinstruction. The following figure shows examples of comments used with macroinstructions. On the assembler listing, all comments on the generated code are aligned by the macroprocessor to begin in column 40. Any comments too long to be contained in columns 40 through 71 are truncated from the right.

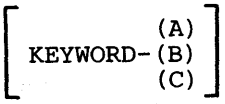


Chapter 5. Macroinstructions Supplied by IBM

The IBM System/36 Assembler and Macro Processor Program Product provides macroinstructions that perform system services and device support. By using IBM-supplied macroinstructions, you can perform these operations with less coding. Scientific macroinstructions are described in the *Scientific Macroinstruction Reference* manual.

The following conventions apply to the IBM-supplied macroinstructions:

- Only keyword operands are used.
- Each operand consists of a keyword followed by a dash and a parameter.



  This list indicates that options A, B, and 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, and N indicates a no response.

- Commas separate the operands; no blanks are allowed between operands.
- Keyword operands can be written in any order.

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

The macroinstructions and functions of the IBM System/36 Assembler and Macro Processor Program Product are shown in the following table. This table is arranged according to device. The macroinstructions that follow each device are in alphabetic order.

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	Creates linkage to system log
General SSP	$FNDP	Generates find parameter list
	$FIND	Finds a directory entry
	$LOAD	Loads or fetches a module
	$SNAP	Performs snap dump of main storage
	$INFO	Retrieves system information
	$INV	Moves inverse data
	$EOJ	Creates linkage to end job
General I/O	$ALOC	Allocates file or device
	$OPEN	Prepares a device or file for access
	$CLOS	Prepares a device or file for termination
	$DTFO	Generates DTF offsets for all devices
Printer	$DTFP	Defines the file for a printer
	$PUTP	Constructs a printer PUT interface
Disk	$DTFD	Defines the file for a disk
	$GETD	Constructs a disk GET interface
	$PUTD	Constructs a disk PUT interface
Disk Sort	$SRT	Generates a loadable sort parameter list
	$SORT	Constructs sort interface

Device Type Supported	Macroinstruction Name	Function
Timer	$TRB	Generates timer request block
	$SIT	Sets timer interval
	$RIT	Returns/cancels timer interval
	$TOD	Returns time and date
Display Station	$DTFW	Defines the file for display station
	$WSIO	Passes I/O requests to display station
	$WIND	Generates indicators for PUT and PUT overrides
	$WSEQ	Generates labels and values for display station device-dependent values
BSC	$DTFB	Defines the files for BSC
	$GETB	Creates GET requests to receive data (move data from BSC I/O buffer to logical buffer)
	$PUTB	Create PUT requests to transmit data (move data from logical buffer to I/O buffer)
	$TRAN	Generates an interface to the translate routine
	$TRL	Generates a parameter list used by the translate routine
	$TRTB	Generates EBCDIC to ASCII or ASCII to EBCDIC translate table

$ALOC (Allocate File or Device)

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

- The DTF is not open.
- The system supports the requested device.
- The device requested is either not being used or is 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, FILE, and PRINTER statements given.
- Load the data management task for data communications DTFs.

When the allocate request is for a disk file, a FILE OCL 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.

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

The normal execution sequence for the 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.

The following output is returned to your program:

The DTF is prepared as required by $OPEN.

The format of the $ALOC macroinstruction follows:

Note: A DTF that was opened cannot be supplied to an allocate request until it is closed. That is, $ALOC must occur before $OPEN.

DTF: Specifies the address of the leftmost byte of the first DTF being allocated. If this operand is entered, an LA instruction is generated to load the specified address into register 2. If this operand is not entered, the address of the DTF is assumed to be in register 2.

$CLOS (Prepare a Device or File for Termination)

The $CLOS macroinstruction prepares a device or file for job termination.

Input to $CLOS consists of the opened DTF.

Output from $CLOS is returned to your program when control is returned. The DTF is returned to the state it was in before $ALOC and $OPEN were used. For example:

- Bit 7 of the second attribute byte of the DTF is set off to indicate the file is closed.
- Bit 5 of the third attribute byte in the DTF is set off to indicate the file is not allocated.

The devices and files that correspond to the open DTFs are prepared either for the job to end or to be allocated and opened again.

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. See $DTFD, $DTFP, and $DTFW later in this chapter.
3. If the DTF is to be reopened, the program must reset the $WSF1A field to be the library F1 address or the program should initialize the field to zero. The $WSF1A field is dually defined with the $WSTU field which is changed by work station data management.

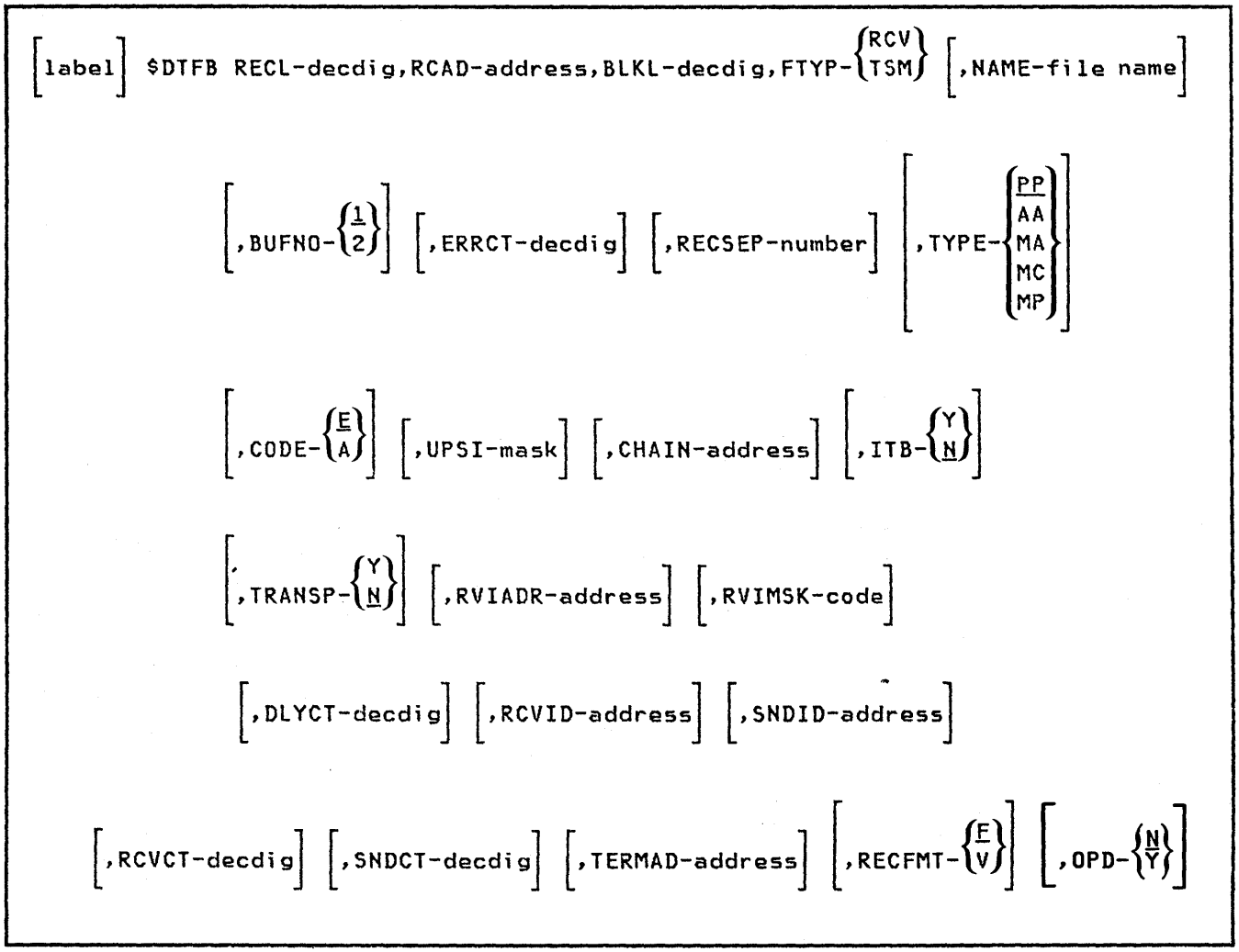
The format of the $CLOS macroinstruction follows:

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

DTF: Specifies the address of the leftmost byte of the first DTF to be closed. If this address is entered, an LA instruction is generated to load the specified address into register 2. If this operand is not entered, the address is assumed to be in register 2.

$DTFB (Define the File for BSC)

The DTF provides information needed to allocate, open, close, and access a BSC file. This macroinstruction generates the code that builds the BSC DTF. The format of the $DTFB macroinstruction follows:



RECL: Specifies in decimal the maximum record length for this file, excluding the transmission control character. The maximum allowable record length is 4075 bytes. However, if data is being blocked (with ITBs or record separators), the record length cannot be so large as to force the physical I/O buffer to be longer than 4096 bytes.

The following algorithm may help you determine a value to use as RECL.

> Buffer size = (record length * number of records per block) + number of bytes needed for ITBs or record separators + 21 (rounded up to a multiple of eight).
>
> Number of bytes needed for ITBs = number of records per block minus 1 (nontransparent), or number of records per block minus 1 times 3 (transparent).
>
> Number of bytes needed for record separator = number of records per block.

Note: For get-a-block operations (OPC-BLK), the record length in the DTF ($BSRCL) is modified by BSC to reflect the length of the block (including transmission control characters) received. See also the RECFMT description in this section.

RCAD: Specifies the symbolic address that identifies the leftmost byte of your logical buffer. The logical buffer must be large enough to contain one record for this file. Records are moved from the logical buffer to the BSC I/O buffers on put requests ($PUTB macroinstruction), and are moved from the BSC I/O buffers to the logical buffer on get requests ($GETB macroinstruction).

BLKL: Specifies in decimal the maximum block length for this file, excluding line control characters. Block length must be equal to or greater than the record length (RECL operand). For maximum block length, see RECL.

FTYP: Specifies whether put requests (TSM) or get requests (RCV) are to be performed on this file.

NAME: Specifies the name of the BSC file to be accessed. If this operand is omitted, no file name is used. The file name is used in certain SSP error messages.

BUFNO: Specifies the number (1 or 2) of physical I/O buffers and IOBs (input/output block) to be contained in the I/O area for this file. If this operand is omitted, 1 is assumed. This operand has no effect on the RECL or BLKL parameters, or the logical buffer length.

ERRCT: Specifies the number of times an unsuccessful BSC operation is retried before an error condition is posted. Valid entries for this parameter are 1 through 255. If this operand is omitted, a value of 7 is assumed. Specifying a retry count of 255 will be treated as an infinite retry count. This will allow BSC to wait forever on a $GETB operation.

RECSEP: Specifies a 1-byte, 2-character hexadecimal value. For put files, BSC inserts the specified byte between blocked records. For get files, this parameter

indicates that the data being received has an intermediate record separator to be removed. Any valid ASCII or EBCDIC character can be used.

The following is a list of *invalid* characters:

ASCII (hex)	EBCDIC (hex)
00	00
01	01
02	02
03	03
04	10
05	1F
11	26
15	2D
16	32
17	37
1F	3D

TYPE: Specifies the type of line connection to be established for this file.

Type	Specification
PP	Point – to – point nonswitched line. PP is assumed if no line type is specified.
AA	Switched line with automatic answer.
MA	Switched line with manual answer.
MC	Switched line with manual call.
MP	Multipoint line; tributary station. MP requires TERMAD parameter.

Note: If you are using an autocall line, the switch type specified has no effect. However, if no phone list is specified in the COMM OCL statement or if the autocall task is not active, the switch type specified here is established unless the ALTERCOM procedure overrides it.

CODE: Specifies whether the character code used on your communications link is EBCDIC (E) or ASCII (A). If this operand is omitted, E is assumed.

UPSI: Specifies the settings of the external (SWITCH statement) indicators used for conditionally opening files. The code must be specified as 8 binary bits. For example, to test bits 0, 3, 5, and 7, you would enter UPSI-10010101. If this operand is omitted, zeros are assumed.

CHAIN: Specifies the symbolic address of the next DTF in the chain. Chained DTFs are allocated, opened, or closed with the first DTF in the chain. To decrease the execution time of your program, all BSC DTFs should be chained together.

ITB: Specifies whether intermediate block checking is requested: Y if yes, N if no. ITB is not valid with transparent transmit files. If this operand is omitted, N is assumed.

TRANSP: Specifies whether data for this file will be transmitted or received in transparent mode: Y if yes, N if no. If this operand is omitted, N is assumed.

RVIADR: Specifies the symbolic address of a 1-byte field you provide. The field is used with the mask specified in the RVIMSK operand (following paragraph) to indicate when a reverse interrupt request (RVI) is received. RVIADR-address requires the RVIMSK operand.

RVIMSK: Specifies 2 hexadecimal digits to represent the reverse interrupt (RVI) mask. The bits represented by the mask are set on by BSC in the RVIADR field (preceding paragraph) if reverse interrupt request is received.

DLYCT: Specifies a decimal delay count. The delay count is the number of seconds after receiving or transmitting a block of data that BSC will wait to receive or transmit another block of data for the same file with no error message. The number must be within the range of 1 through 999. If you do not specify a number, a 180-second delay count is allowed for such things as device errors, halts, and readying I/O devices. The delay count should allow for such things as printing and operator response time. When the delay count is exhausted, EOT is transmitted to the remote station, an error message is displayed for the user, and an error completion code ($BSDLYEX) is returned to your program.

Note: See the BSC Completion Code Table on page 5-12 for an explanation of completion codes.

RCVID: Specifies the symbolic address of the leftmost byte of the identification sequence required from the remote station. RCVID requires the RCVCT operand. Using RCVID and RCVCT may improve security on switched lines; these operands are valid for switched lines only. If the IDs do not match, initialization stops, an error message is displayed, and an error return code is generated.

SNDID: Specifies the symbolic address of the leftmost byte of the identification sequence required by the remote station. SNDID requires the SNDCT operand. Using the SNDID and SNDCT operands may improve security on switched lines; these operands are valid for switched lines only.

RCVCT: Specifies in decimal the length of the identification sequence required from the remote station. The length can be from 1 to 15. If 1 is specified, BSC expects to receive two characters – two duplicates of the character addressed by the RCVID operand (previous paragraph). If no length is specified, 0 is assumed. RCVCT requires the RCVID operand be specified also.

SNDCT: Specifies in decimal the length of the identification sequence required by the remote station. Length can be from 1 to 15. If 1 is specified, BSC transmits two characters – duplicates of the character specified by the SNDID operand (previous paragraph). SNDCT requires the SNDID operand.

TERMAD: Specifies the hexadecimal representation of the 2-character polling or addressing sequence used by this file. If this is a transmit file (FTYP-TSM), TERMAD specifies polling characters; if this is a receive file (FTYP-RCV), TERMAD specifies addressing characters. Each tributary station on a multipoint line must have unique polling and addressing characters. The TERMAD operand is used only when TYPE-MP is specified.

RECFMT: Specifies whether the BSC program will receive records of fixed (F) or variable (V) record length. Fixed (F) is assumed if this parameter is omitted. Transparency, ITB mode, and blank compression or truncation are invalid with variable (V) length records. BSC returns the length of the record received in field $BSRCL of the DTF if RECFMT-V is specified. This parameter has no effect on PUT files. If you specify RECFMT-V, the RECL field must contain the maximum record length you expect to receive.

Note: Relative record numbers (RRN) are specified in an 8-byte field in the DTF. The first RRN is specified in binary, starting with 0 (zero), and in decimal starting with 1 (one). The following labels expanded by the $DTFO macroinstruction define the RRN field in the DTF:

$F1ARG This is a displacement to the rightmost byte of the 8-byte RRN field in the DTF.

$F1RRNB This is a displacement to the rightmost byte of the 3-byte binary RRN field of the DTF. This displacement points to the left 3 bytes of $F1ARG. If you are processing with binary relative record numbers you must specify a right-justified, 3-byte binary number in this field.

$F1RRND This is a displacement to the rightmost byte of the 8-byte decimal RRN field of the DTF. This displacement points to the same field as the $F1ARG field described above. If you are processing a decimal RRN, you must specify a right-justified, 8-byte decimal number in the field.

OPD: Specifies whether the BSC program will do normal end of file processing by sending EOT (N), or by using EXT as the file separator (Y). The Office Product Device support protocol is found in the Program Service Information manual. This mode is only supported for assembler users and only on transmitting. The last record transmitted in a block must be done with a $PUTB OPC-EOB.

BSC Completion Code Table

$BSCMP is a byte in the workstation DTF that contains the completion code. It is referenced by loading an index register with the start address of the DTF and using $BSCMP as an offset. For example:

```
TBN    $BSCMP(,XR2),$BSNRMC    TEST FOR SUCCESSFUL COMPLETION
```

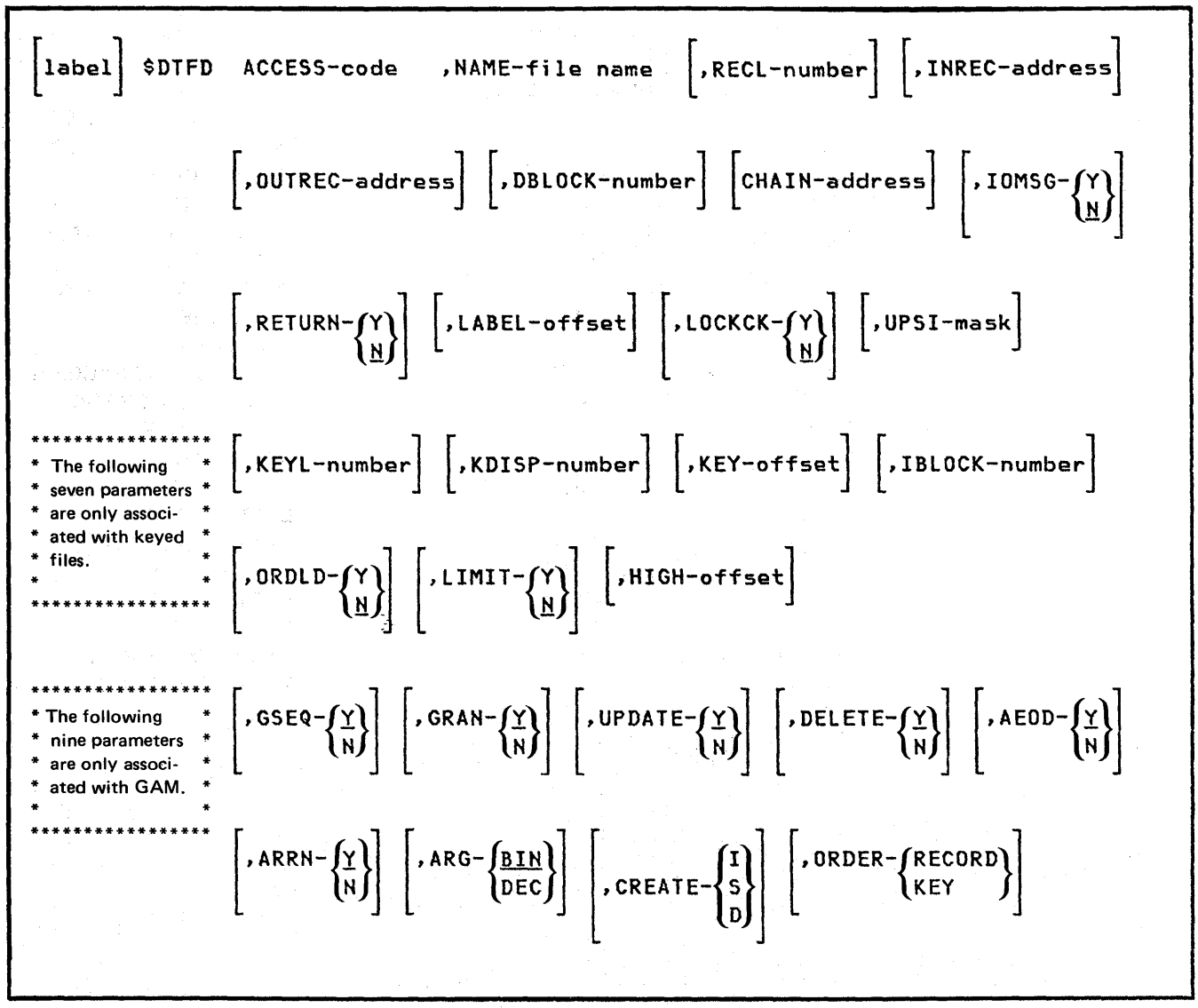
```
$BSCMP      EQU    $BSWKB+1,1          COMPLETION CODE

*           COMPLETION CODES
$BSRQAC     EQU    X'00'               BSC TASK NOT ACTIVE
$BSNRMC     EQU    X'40'               NORMAL COMPLETION
$BSUSER     EQU    X'41'               USER ERROR
$BSEOF      EQU    X'42'               END OF FILE
$BSINVID    EQU    X'43'               INVALID ID ON SWITCHED LINE
$BSREQIG    EQU    X'4A'               REQUEST IGNORED
$BSINVAS    EQU    X'4B'               INVALID ASCII CHARACTER
$BSNOCON    EQU    X'4C'               NO CONNECTION
$BSINVRQ    EQU    X'4D'               INVALID REQUEST
$BSDLYEX    EQU    X'4E'               DELAY COUNT EXCEEDED
$BSPERM     EQU    X'4F'               PERMANENT ERROR
$BSNORSP    EQU    X'50'               NO RESPONSE
$BSDTCK     EQU    X'51'               DATA CHECK
$BSLSTDT    EQU    X'52'               LOST DATA
$BSLSTCN    EQU    X'53'               LOST CONNECTION
$BSINVRS    EQU    X'54'               INVALID RESPONSE
$BSADCK     EQU    X'55'               ADAPTER CHECK
$BSFWDAB    EQU    X'56'               FORWARD ABORT RECEIVED
$BSABRT     EQU    X'57'               EOT ABORT
$BSMLCAT    EQU    X'58'               MLCA TEMP ERROR
$BSMLCAP    EQU    X'59'               MLCA PERM ERROR
```

$DTFD (Define the File for Disk)

The $DTFD macroinstruction generates the code that builds the preopen DTF disk for GET/PUT operations. The disk DTF provides information needed to allocate, open, and access a file on the disk.

Further information access methods for disk files is in Appendix E.

The format of the $DTFD macroinstruction follows:



Note: The above parameters have been grouped according to their function; however, they may be arranged in any order.

ACCESS: Specifies the access method used for the file. This operand is required. The access methods and corresponding codes are:

Access Method	Code
Consecutive add	CA
Consecutive input	CG
Consecutive output	CO
Consecutive update	CU
Direct input (decimal RRN)	DG
Direct output (decimal RRN)	DO
Direct update (decimal RRN)	DU
Direct input addrout (binary RRN)	DGA
Direct output addrout (binary RRN)	DOA
Direct update addrout (binary RRN)	DUA
Generalized access method	GAM
Indexed random add	IA
Indexed output	IO
Indexed sequential input	IS
Indexed sequential add with input capable	ISA
Indexed sequential update	ISU
Indexed sequential update and add	ISUA
Indexed random input	IR
Indexed random add with input capable	IRA
Indexed random update	IRU
Indexed random update and add	IRUA
Record/key length, key displacement in DTF	PSEUDO

Note: Refer to APPENDIX E for additional information on Access Methods.

NAME: Specifies the name of the file. The name cannot exceed eight characters, and must be the same as that specified on the FILE OCL statement. This operand must be specified.

RECL: Specifies the decimal length of the record. The maximum length is 4096. This operand is required for all access types except PSEUDO, which returns the record length of the file. The default record length is 32 bytes.

INREC-address: Specifies the initial address of the leftmost byte of an area that will contain the record from the input operation. The area reserved must be equal to the record length.

OUTREC: Specifies the address of the leftmost byte of the area that will contain the record for any update or add operation. This area must be equal to the record length.

DBLOCK: Specifies the number of records to be moved between main storage and disk with each disk I/O operation. Buffer space is reserved based upon this number and the record length. Data management might change this number based on current file status. The number must be between 1 and 65535. If not specified, 1 is assumed. The DTF value for this field can be overridden by the value specified on the FILE OCL statement.

CHAIN: 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 (hex FFFF is then assumed).

IOMSG: Specifies that an error message should be issued by the System Support Program (SSP) for a permanent disk error. When N is specified, control is returned with the completion code set. If this operand is omitted, N (no) is assumed.

RETURN: Is used only if IOMSG-Y (yes) is also specified to present the options allowed to the operator when a permanent disk error occurs. If RETURN-Y (yes) is specified, the operator is allowed to take option 2 and receive the *permanent disk error* message. If option 2 is taken, control is returned to the user program with the completion code set. If RETURN-N (no) is specified, the operator is allowed option 3 only. If this operand is omitted, N is assumed.

LABEL: Specifies the first byte of the label area from the end of the DTF. the DTF must be 8-bytes long. A file label is returned if:

- When a duplicate key error occurs on the currently used index, the $PUTD macro branches to the routine specified on the DUPREC parameter. When a duplicate key error occurs on an index other than the currently used index, the $PUTD macro branches to the routine specified on the DUPRCO parameter.

- For update key error conditions, the label area contains the label of the file in which the key update is being attempted. When an update key error occurs, the $PUTD macros branches to the routine specified on the KEYERR parameter.

- For a permanent I/O error, the label of the file where the error occurred is returned here. When a permanent I/O error occurs, the $PUTD macro branches to the routine specified on the IOERR parameter.

Note: If alternate indexes are defined on the file that you were processing, this label may be different than the file that you were accessing.

The label field must follow the DTF. The last byte of this field must be within 2048 bytes of the first byte of the DTF.

LOCKCK: Requests a check by data management to see if the requested input record is already owned by this task. If Y (yes) is specified, a completion code is returned if the record is already owned. If N (no) is specified, or the operand is omitted, no check is made.

UPSI: Specifies the settings of the external indicators used for conditionally opening files. The code must be specified as 8 digits. For example, to test bits 0, 3, 5, and 7, you would enter UPSI-10010101. When all corresponding indicators are on, 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.

KEYL: Must be specified for all keyed access methods except PSEUDO to supply the length of the key field. The maximum length is 29, and if this operand is omitted, a length of 1 is assumed. An open with ACCESS-PSEUDO specified returns the key length for an indexed file. If the file has noncontiguous keys, the sum of the lengths of the individual key fields must be specified.

KDISP: Must be specified for all keyed access methods except PSEUDO to indicate the displacement into the record of the rightmost byte of the key field. The displacement of the first byte in the record is 0, the second byte is 1, and so on. The maximum displacement is 4095 and if this operand is omitted, a displacement of 0 is assumed. A pseudo open returns the key displacement for an indexed file. If the file has noncontiguous keys, KDISP must contain decimal 65535 (hexadecimal 'FFFF').

KEY: Specifies the first byte of the key area as the displacement from the end of the DTF. This reserved area must be equal to the key length. This operand is required for KEY, KEYA, and KEYEA operations. The key area must following the DTF. The last byte of the key area must be within 2048 bytes of the first byte of the DTF. If the file has noncontiguous keys, the sum of the lengths of the individual key fields must be specified.

IBLOCK: Specifies the number of index entries moved between main storage and disk with each disk I/O operation. Buffer space is reserved based upon this number and the record length. Data management might change this number based on the access and the current system status. The number must be between 1 and 65535. If not specified, 1 is assumed. The DTF value for this field can be overridden by the value specified on the FILE OCL statement.

ORDLD: Specifies that data management will check that record keys placed in the file are in ascending order. ORDLD can be specified for the following indexed add-capable or output-capable access method (IA, IRA, IRUA, IO, and GAM with AEOD-Y and/or ARRN-Y). If Y (yes) is specified, and the record

keys are not being loaded in ascending order, data management will return a nonsuccessful completion code. If N (no) is specified, data management does not check for ascending order. Duplicate keys are allowed as specified in the FILE OCL statement when the file was created. If ISA or ISUA is specified, the user is forced into ORDLD-Y mode regardless of whether the ORDLD parameter is specified or not. If the operand is omitted for the other indexed-ADD or output methods, N (no) is assumed.

LIMIT: Specifies whether this access is within limits. LIMIT can only be specified for indexed sequential access methods (IS, ISA, ISU, ISUA, and GAM with GSEQ-Y). This allows you to get records in consecutive order from a keyed file by specifying the lowest and highest record key. If while processing within the specified limits, a nonsequential get is issued, the current limits are cancelled. See the HIGH parameter of $DTFD and the $GETD macroinstruction for more information on limits. If this operand is omitted, N (no) is assumed.

HIGH: Specifies the first byte of the limits keys area as the displacement from the end of the DTF, which is lengths long: the low key is in the left half and the high key is in the right half. This field must be after the DTF. The last byte of this field must be within 2048 bytes of the first byte of the DTF. If this operand is omitted, hex FFFF is assumed.

GSEQ: Used only with GAM to specify whether sequential get operations will be issued with this file access. The consecutive get operations that can be specified with the $GETD macroinstruction OP parameter, are NEXT, PREV, PLUS, and MINUS. If this operand is omitted and ACCESS-GAM is specified, Y (yes) is assumed.

GRAN: Used only with GAM to specify whether random get operations will be issued with this file access. Random get operations specified with the $GETD macroinstruction OP parameter are KEY, KEYEA, KEYA, RRN, FIRST, and LAST. If this operand is omitted and ACCESS-GAM is specified, Y (yes) is assumed.

UPDATE: Used only with GAM to specify whether UPDATE operations will be used with this file access. If this operand is omitted and ACCESS-GAM is specified, Y (yes) is assumed.

DELETE: Used only with GAM to specify whether DELETE operations will be issued with the file access. If this operand is omitted and ACCESS-GAM is specified, Y (yes) is assumed.

Note: If a DELETE operation is issued against a file that is *nondelete capable*, an invalid operation completion code is set.

AEOD: Used only with GAM to specify whether add-at-end-of-data operations are issued with this file access. If they are issued, this would cause the added record to be placed in the file at the end of the current records. If this operand is omitted and ACCESS-GAM is specified, Y (yes) is assumed.

ARRN: Used only with GAM to specify whether add-by-RRN operations are issued with this file access. If they are issued the added record is placed in the specified AREA location in the file. If this operand is omitted and ACCESS-GAM is specified, Y (yes) is assumed.

ARG: Used only with GAM or for direct input or update access to specify whether the argument (the RRN or the **plus/minus** value) for this access is binary (BIN) or decimal (DEC). If this operand is omitted, binary (BIN) is assumed.

CREATE: Used only with GAM to specify which file type should be created when the output is put to a new file or to a load-to-old file.

I	creates an indexed file
S	creates a sequential file
D	creates a direct file

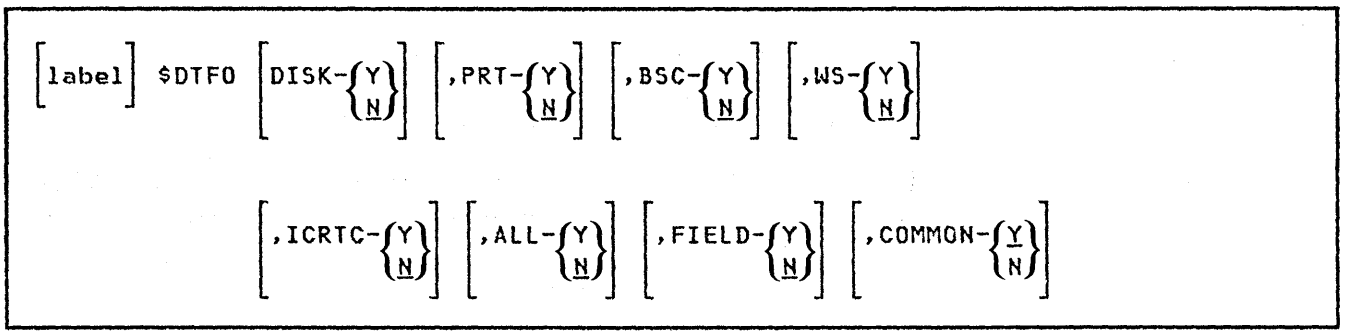
There is no default for the CREATE parameter.

ORDER: Required parameter that must be used with GAM to specify whether the data is to be accessed by key or by record (not by key). There is no default for the ORDER parameter.

$DTFO (Generate DTF Offsets)

This macroinstruction defines the DTF labels, offsets, field contents, and field lengths for all devices and access methods supported by System/36. 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 *Data Areas Handbook*.

The format of the $DTFO macroinstruction is:



DISK: Specifies whether labels are to be generated for the disk devices. If this operand is omitted, N (no) is assumed.

PRT: Specifies whether labels are to be generated for the printer. If this operand is omitted, N (no) is assumed.

BSC: Specifies whether labels are to be generated for BSC. If this operand is omitted, N (no) is assumed.

WS: Specifies whether labels are to be generated for work station and SSP-ICF devices. If this operand is omitted, N (no) is assumed.

ICRTC: 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: Specifies whether labels are to be generated for all devices supported. If this operand is omitted, N (no) is assumed.

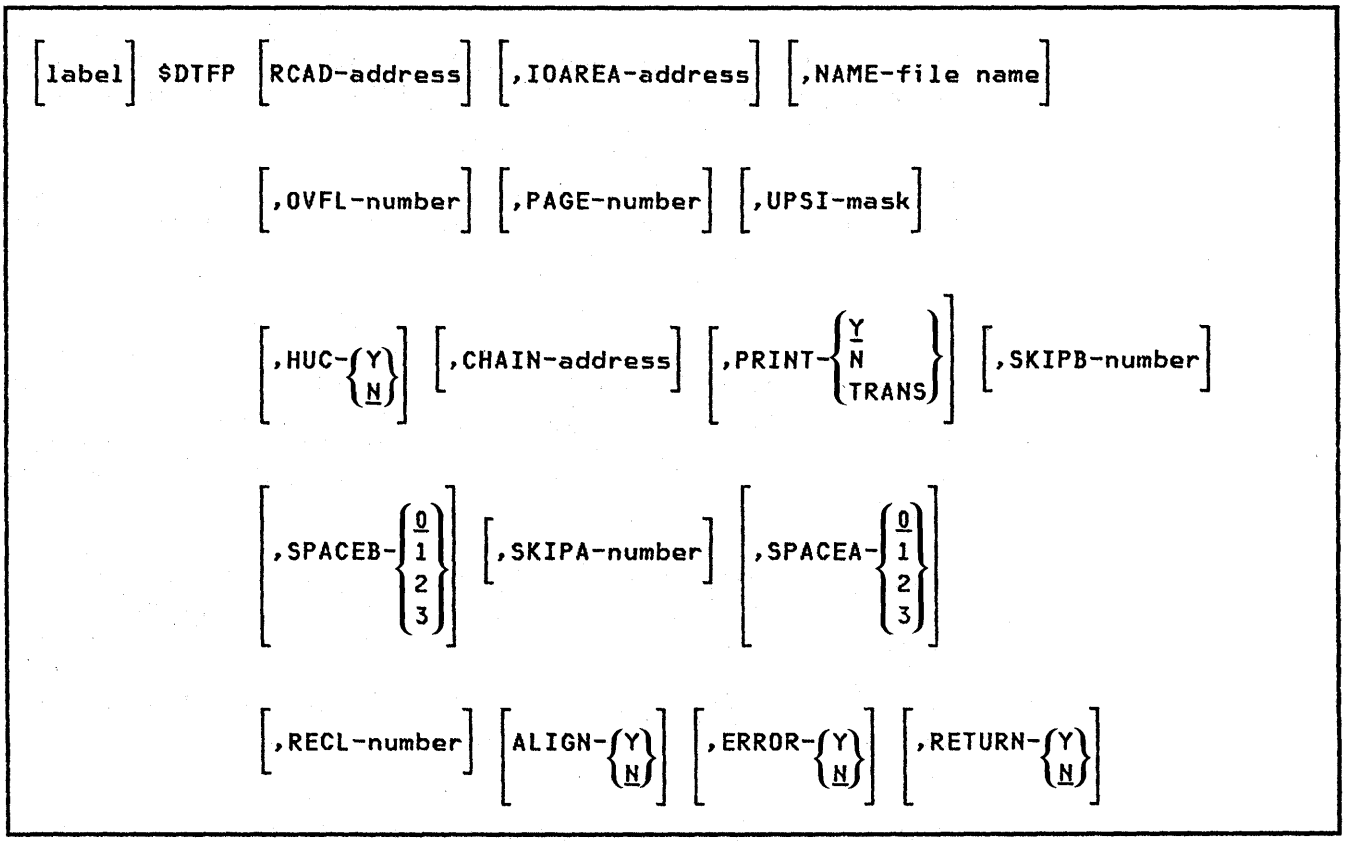
FIELD: Specifies whether labels are to be generated to define the contents of the DTF fields. If this operand is omitted, N (no) is assumed.

COMMON: Specifies whether labels are to be generated to define the field content on the common portion of the DTF (from the start of the DTF, and ending with the name field). If this operand is omitted, Y (yes) is assumed.

$DTFP (Define the File for a Printer)

$DTFP builds a DTF for a printer and assigns its offsets. 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 follows:



RCAD: A required operand that specifies the address of the leftmost byte of the logical record.

IOAREA: This parameter is not required and is provided for System/34 compatibility only.

NAME: Specifies the name of the print file. This name must be the same as the name specified on the PRINTER OCL statement. This operand defaults to FILE NAME.

OVFL: Specifies the print line after which the overflow completion code will be returned. If this operand is omitted, the value defaults to six lines less than the number specified for the PAGE operand.

PAGE: Specifies the number of printed lines to print per page. If this operand is omitted, it defaults to the system value for the number of lines per page or to the LINES parameter of the PRINTER OCL statement.

UPSI: 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 1 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, 0's are assumed.

HUC: Specifies whether to halt if an unprintable character is detected. If N (no) is specified or if this operand is omitted, no halt occurs, and unprintable characters appear as blanks.

CHAIN: Indicates the address of the next DTF. If there is no DTF chain or if this is the last DTF in a chain, this operand should be omitted (hex FFFF is then assumed).

PRINT: Specifies with Y (yes) to perform both a print and the specified skip or space, or with N (no) only a skip or space. The default is Y, meaning that a print is performed as well as the other operation.

PRINT-TRANS (transparent mode): Requires the before forms feed commands at the beginning of the record. The after forms feed command of the printer DTF must be 0 or 1.

SKIPB: Specifies the line to skip to before the print operation. If this operand is omitted the existing value is used. If the operand is an invalid number (too large) the parameter is ignored and no skip is performed, until a valid value is used.

SPACEB: Specifies the number of lines to space before the print operation. If this operand is omitted the existing value is used. If the operand is an invalid number (too large) the parameter is ignored and no space is performed, until a valid value is used.

SKIPA: Specifies the line to be skipped to after a print operation. The maximum allowed is 255. If this operand is omitted the existing value is used. If the operand is an invalid number (too large) the parameter is ignored and no skip is performed, until a valid value is used.

Note: If the SKIP or SPACE values exceed the value of PAGE (lines per page), no operation is performed.

SPACEA: Specifies the number of lines to space after the print operation. If this operand is omitted the existing value is used. If the operand is an invalid number (too large) the parameter is ignored and no space is performed, until a valid value is used.

Note: If the SKIP or SPACE values exceed the value of PAGE (lines per page), no operation is performed.

RECL: Specifies the length of the line to be printed, from 1 through 198 positions. If this operand is omitted, the default is 132 positions. When a value greater than 132 positions is specified, the output must be printed at 15 characters per inch.

ALIGN 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: 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, such as a forms jam or an out-of-forms condition, are not considered permanent errors.

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

$DTFW (Define the File for Display Station)

The $DTFW 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.

All communication with the display stations or system console is done via work station management. 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 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 *Creating Displays: Screen Design Aid and System Support Program* manual.

Work station data management 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: 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 follows:

```
[label] $DTFW [DEV-code] [,UPSI-mask] [,CHAIN-address] [,OUTLEN-number]
              [,RESET-{Y}] [,RCAD-address] [,INLEN-number]
                       {N}
              [,ROLINE-number] [,STRTLN-number] [,ENDLN-number]
              [,VARLIN-number] [,INDA-address] [,MEMBER-name]
              [,TERMID-name] [,PRNT-{Y}] [,ROLL-{Y}]
                                    {N}        {N}
              [,CLEAR-{Y}] [,RECBKS-{Y}] [,HELP-{Y}] [,FKDATA-{Y}]
                      {N}          {N}        {N}          {N}
              [,TIDTAB-address] [,ENTLEN-number] [,TNUM-number]
              [,RPGEXT-address] [,HALTS-{Y}] [,CMDKEY-mask]
                                        {N}
              [,CKMASK-{DTF   }] [,FKMASK-{DTF   }]
                       {FORMAT}           {FORMAT}
              [,EXTEND-{Y}]
                       {N}
              [,IDDUCM-iddu-communications-file-definition-name]
              [,DICTCM-iddu-communications-data-dictionary-name]
```

DEV specifies the file type for which this DTF is to be used. If this operand is omitted, 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: 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, 0's are assumed, and the file is unconditionally opened.

CHAIN: 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 (hex FFFF is then assumed).

OUTLEN: Only required for OPMODs of ERROR and UNF; or OPCs of PUT, PTG, PNW, and PTI of the $WSIO macroinstruction. If the operation is ERROR , the OUTLEN value must be between 1 and 78, and the value 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 the value represents the exact length of the data stream. If the operation is a PUT, PTG, PNW, or PTI, 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 execution-time output fields.

An execution-time output field is a field that was declared as output of $SFGR field definition specifications (columns 23 and 24) and does not have data specified on the $SFGR field definition specifications in columns 57 through 79. The data for this field is specified at execution time by the user program. If this operand is omitted, an OUTLEN value of hex 0000 is assumed. After a successful input operation, the actual length of data returned is in this field; therefore, OUTLEN should be specified again after every input operation.

Note: For each $SFGR D-specification that requires MIC data from the user's logical record area, 6 bytes must be added to the total OUTLEN value. These bytes contain the 4-character message identification code followed by a 2-character message member identifier.

RESET: Specifies whether to reset the active format index address. If Y is specified, a new format index is built, and the old index is lost. If N is specified and there is an active format index, the new index is added to the old. 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. If this parameter is omitted, N is assumed.

RCAD: Specifies the symbolic address of the leftmost byte of the logical record area.

Note: If the operation being performed involves GET, ACI, or UNF, the record area must be on an 8-byte boundary.

INLEN: 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 display.

ROLINE: Specifies in decimal the number of lines to roll the displayed data on a roll operation. The maximum number 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 23. If this operand is omitted, 01 is assumed.

ENDLN: Specifies in decimal the number of the last line of the roll area on a roll operation. The minimum number is 02; the maximum number is 24. If this operand is omitted, 24 is assumed.

VARLIN: Specifies in decimal the actual start line number if a variable start line number was specified in SFGR for the format associated with this request. The maximum number is 24. If this operand is omitted, 01 is assumed.

INDA: 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 address hex 0000 because WSDM assumes no indicator area exists at address hex 0000, and the indicators are assumed to be off. If this operand is omitted, address hex 0000 is assumed.

MEMBER: 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.

TERMID: 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, blanks are assumed. For an SRT program, blank means the requesting display station is assumed. For MRT programs, a halt is issued unless the operation does not need TERMID such as ACI (accept), SIQ (status inquiry) INQ, (status inquiry), GTA (get attributes) operation, or EGTA (extended get attributes) operation.

The following parameters, PRNT, ROLL, CLEAR, RECBKS, CMDKEY, CKMASK, FKMASK, and HELP are the function-control-key mask specifications.

PRNT: Specifies whether your program will process the Print key. If Y (yes) is specified, the print key indicator is placed in the attention identification (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 current display with the optional heading and border 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 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 when the operator presses the Clear key. If this operand is omitted, N (no) is assumed.

RECBKS: Specifies whether your program can process record backspace (that is, the Home key when the cursor is *only* 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: Specifies whether your program can 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 and your program must support the key. If yes is specified, application help is not available. If N (no) is specified and the operator presses the Help key, either your application help or an error message is displayed. If this operand is omitted, N (no) is assumed.

FKDATA: Specifies whether input data is to be returned along with a function key indicator for all enabled function keys. If Y (yes) is specified, the appropriate function key indicator is placed in the AID byte field of your program DTF when the operator presses an enabled function 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. (See Note 1.)

Notes:

1. The FKDATA parameter has no effect on the operation of the *Roll Up* and *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 display of the remote work station when a function key is pressed while the FKDATA parameter is active.

TIDTAB: 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 acquired by the program plus additional entries up to the program's MRTMAX value. MRTMAX is specified in a COMPILE statement and can be overridden by an ATTR statement. The entire table must be initialized to hex 00 before OPEN is called. After open is complete, the user program must maintain the table. If this operand is omitted, address hex 0000 is assumed, and no table is built. (For a description of ATTR, COMPILE, and WORKSTN statements, see the *System Reference* manual.)

ENTLEN: 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: 2 bytes for an ID and a third byte for status indicators.

TNUM: 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.

HALTS: Valid only if this DTF is used with the interactive communications feature, which is described in the manual, *Interactive Communications Feature: Reference*. This 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.

CMDKEY: Specifies the command key mask to be placed into DTF. The mask is made up of 24 binary bits (bit 0 = CMD1, bit 23 = CMD24) entered as 6 hexadecimal digits. If this operand is omitted, hex FFFFFF is assumed.

CKMASK: Specifies whether WSDM should use the command key mask from the display format or from the DTF. If this operand is omitted, FORMAT is assumed.

FKMASK-FORMAT/DTF: Specifies whether WSDM should use the function key mask from the display format and from the DTF (FORMAT) or just from the DTF (DTF). If this operand is omitted, FORMAT is assumed. If FORMAT is specified, the function key must be masked ON in both the format and the DTF for the function key to be enabled.

EXTEND: Specifies whether the extended DTF is generated for Communications Data Dictionary purposes. If Y or YES is specified, the workstations DTF is extended allowing the IDDUCM and DICTCM parameters to be used. If N or NO is specified or if the EXTEND parameter is not specified, the workstation DTF will not be extended. Refer to *Getting Started with the Interactive Data Definition Utility (IDDU)* or IDDU Online Information for information on using IDDU.

IDDUCM: Specifies the name of the Interactive Data Definition Utility (IDDU) file to be used for communications. Refer to *Getting Started with the Interactive Data Definition Utility (IDDU)* or IDDU Online Information for information on using IDDU.

Note: If the DTF is extended and the IDDUCM parameter is not specified at compile time, it is expected the parameter will be entered as data at the time the program is run.

DICTCM: Specifies the name of the data dictionary to be used for IDDU communications. Refer to *Getting Started with the Interactive Data Definition Utility (IDDU)* or IDDU Online Information for information on using IDDU.

Note: If the DTF is extended and the DICTCM parameter is not specified at compile time, it is expected the parameter will be entered as data at the time the program is run.

$EOJ (End of Job)

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

The format of the $EOJ macroinstruction follows:

```
[label]  $EOJ     no operands
```

$FIND (Find a Directory Entry)

You can use the $FIND macroinstruction to locate library members for your program.

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

The format of the $FIND macroinstruction follows:

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

PLIST: Specifies the address of the leftmost byte of the 17- or 54-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.

$FIND uses the parameter list generated by the $FNDP macroinstruction.

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.

A successful $FIND can be determined by checking the field $FNDDTOT. ($FNDDTOT is an equate generated by the $FNDP macro that is the offset from the start of the $FNDP parameter list.)

The following instructions could be used:

```
CLC $FNDDTOT (2,XR2),ZEROES
JE  ERROR
```

(where ZEROES is defined as ZEROES DC XL2'0000')

If the total length of the found module is zero, then the module is not found.

Note: When a module is not found by $FIND:

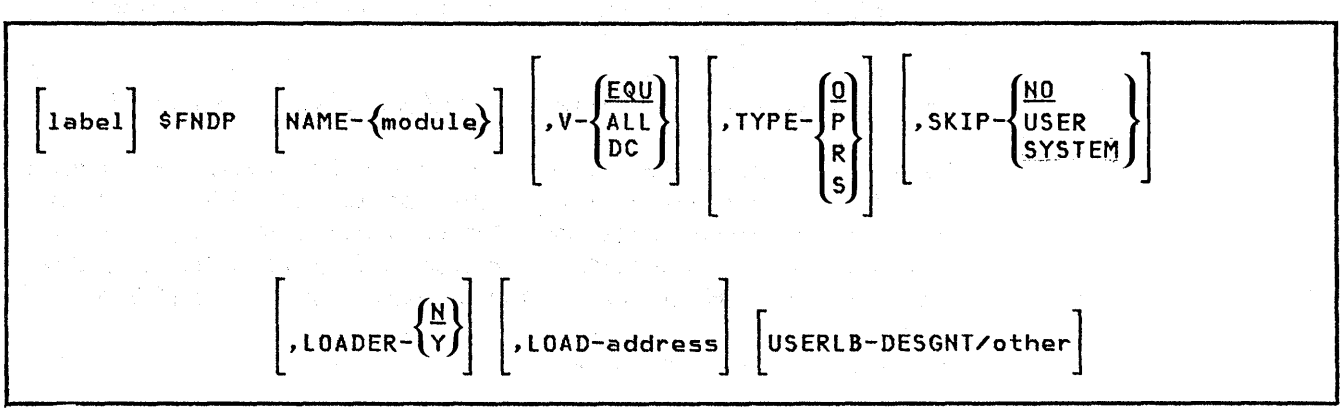
- If LOADER-Y is specified in the $FNDP macroinstruction, a cancel-only halt is issued and control is not returned to your program.
- If LOADER-N is specified in the $FNDP macroinstruction, control is returned to your program for determination of appropriate action.

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

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

The $FNDP macroinstruction generates a load parameter list and 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 follows:



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

V: Specifies whether the parameter list, labels, or both are to be generated. If this operand is omitted, EQU is assumed.

DC: Generates a 17- or 54-byte parameter list used by the $FIND macroinstruction.

EQU: Generates the displacement labels for the $FIND parameter list. If V-EQU is specified or supplied as the default, all other operands are ignored.

ALL: Generates both the parameter list and the corresponding displacement labels.

TYPE: Specifies the library member type. If this parameter is omitted, O is the default. The codes have the following meaning:

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

SKIP: Specifies the type of library search to perform by specifying which library to skip. The codes have the following meaning:

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, and both libraries are searched.

LOADER: Specifies whether the parameter list is used by $LOAD. If Y (yes) is specified, a 17-byte parameter list is generated for use by $LOAD. If N (no) is specified, a parameter list containing 17 bytes of loader information, 33 bytes of directory, and 4 bytes of find information overlays the input. 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: Specifies the main storage address where the module is to be loaded. 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.

USERLB: Specifies the library to be searched.

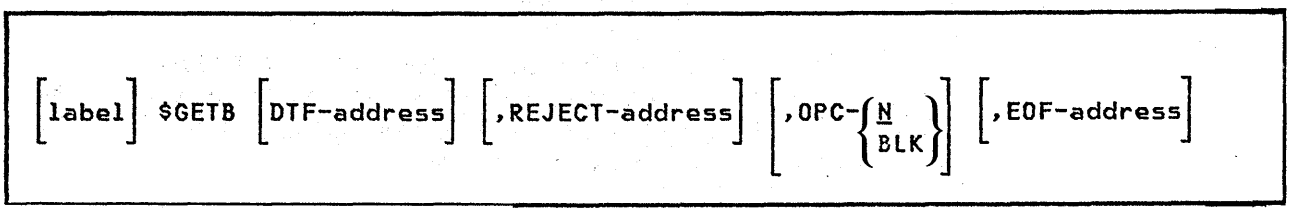
DESGNT: Specifies a search of the current (designated) user library.

Other: Specifies a search of the library specified in $FNDDF1A.

$GETB (Issue a Get Request)

The $GETB macroinstruction generates code to move data from a BSC I/O buffer to your logical buffer. To use this macroinstruction, construct a BSC DTF for the file (using the $DTFB macroinstruction) and use the $DTFO macroinstruction to generate the labels and establish the offsets for the DTF.

The format of the $GETB macroinstruction follows:



DTF: Specifies the address of the DTF for which the get was issued. If this operand is omitted, the address of the DTF is assumed to be in register 2.

REJECT: Specifies the routine to receive control if this get request is rejected by BSC. If this operand is omitted, control is returned to the user program at the next sequential instruction after the $GETB.

OPC: Specifies how BSC handles the record received for this program. **N** indicates normal deblocking by BSC before the record is passed to the receiving program. That is, BSC removes transmission control characters and moves the data to the logical buffer (RCAD in $DTFB) one record at a time. **BLK** indicates the entire block (including control characters) is passed to the receiving program. BSC places the length of the block in $BSRCL in the DTF if OPC = BLK or when receiving variable length records. If this operand is omitted, N is assumed.

Note: If you specify OPC-BLK, be sure your logical buffer (RCAD in $DTFB) is large enough to hold an entire block of data plus transmission control characters (maximum 4096).

EOF: Specifies your end-of-file routine. If this operand is omitted, control is returned to the user program at the next sequential instruction after the $GETB.

If EOF or REJECT addresses are not specified, your program should check the return code in the DTF ($BSOPC) to determine the outcome of the operation.

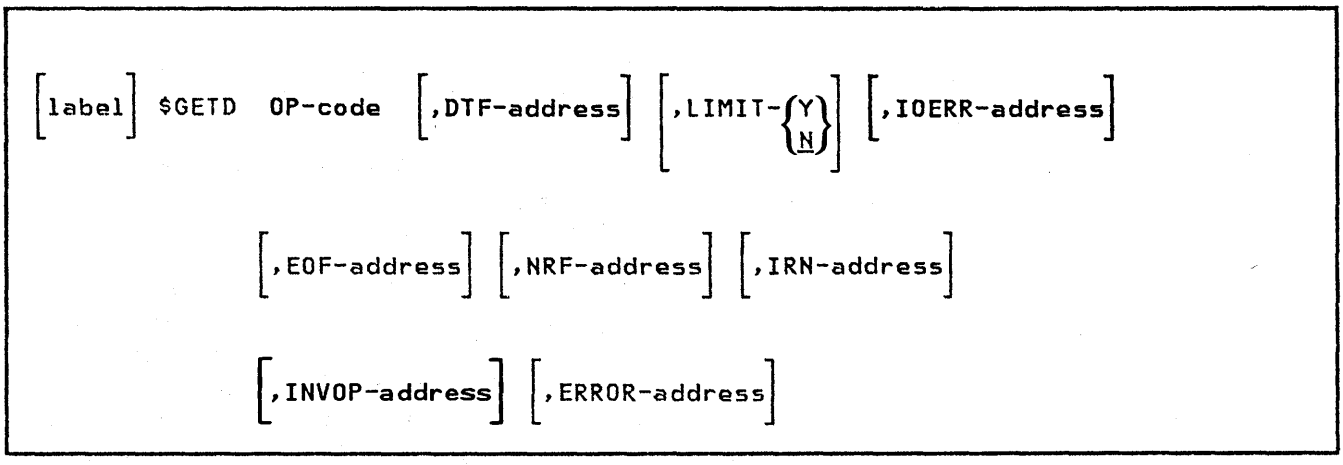
$GETD (Construct a Disk Get Interface)

The $GETD macroinstruction generates the interface needed to communicate with 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 need the data in register 2 later, save the contents of that register before issuing $GETD.

Data management operates in move mode for input operations. In move mode, disk data management moves a record into the logical buffer (INREC) identified in the disk DTF from the 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 follows:



DTF: 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.

LIMIT: Specifies whether new limits are to be set for this file. If Y (yes) is specified, the low and high limit keys must be in the area specified by the HIGH parameter of $DTFD. If N (no) is specified, the DTF is unchanged. If this operand is omitted, N is assumed.

Note: If a GET PREV is going to be used, only the low limit key needs to be specified; the high limit key is ignored.

OP: Must be specified as in the following list.

For consecutive access:

Code	Meaning
NEXT	Get next
PREV	Get previous
PLUS	Get forward by argument value **Note**: The argument value is placed in the DTF at label $F1ARG.
MINUS	Get backward by argument value **Note**: The argument value is placed in the DTF at label $F1ARG.

For direct access:

Code	Meaning
RRN	Get random record by RRN **Note**: The RRN value is placed in the DTF at the label $F1RRNB (in binary format) or $F1RRND (in decimal format).
FIRST	Get first in file
LAST	Get last in file

For indexed random access:

Code	Meaning
KEY	Get random by key
KEYEA	Get key equal or above
KEYA	Get key above

FIRST	Get first by key
LAST	Get last by key

For indexed sequential access:

Code	Meaning
NEXT	Get next by key
PREV	Get previous by key
READE	Get key equal. (Do a NEXT and return record if its key is equal to the key in the field specified by the key parameter in the $DFTD.)

For GAM (record order):

Code	Meaning
RRN	Get random record by RRN **Note:** The RRN value is placed in the DTF at the label $F1RRNB (in binary format) or $F1RRND(in decimal format).
FIRST	Get first in file
LAST	Get last in file
NEXT	Get next
PREV	Get previous
PLUS	Get forward by argument value **Note:** The argument value is placed in the DTF at label $F1ARG.
MINUS	Get backward by argument value **Note:** The argument value is placed in the DTF at label $F1ARG.

For GAM (key order):

Code	Meaning
KEY	Get random by key
FIRST	Get first by key
LAST	Get last by key
NEXT	Get next by key

PREV	Get previous by key
READE	Get key equal. (Do a NEXT and return record if its key is equal to the key in the field specified by the key parameter in the $DFTD.)
KEYEA	Get key equal or above
KEYA	Get key above

For any access (key or record order):

Code	Meaning
NULL	No-op is moved into the DTF (see note)

Note: Null can be used if the operation code is changed by the program. The programmer is responsible to assure that some operation code is moved into the DTF before $GETD is run.

IOERR: Specifies the address that receives control if the controlled cancel option is taken in response to a permanent disk error. If this operand is omitted, there is no check for a permanent disk error completion code.

EOF: 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. Do not use this operand with random or direct access methods.

NRF: Specifies the address in your program that receives control if a no-record-found condition occurs. Do not use NRF with consecutive or indexed sequential access methods.

IRN: Specifies your program address receiving control if an invalid-record-number condition occurs on a PLUS or MINUS or RRN operation.

INVOP: The address in your program that receives control if an invalid operation condition occurs.

ERROR: Supplies the address in your program that receives control if an unsuccessful completion code is detected. The successful completion code is hex 40. Any other hex value is an unsuccessful completion code.

Note: If an IOERR, EOF, NRF, IRN, or INVOP occurs but is not specified, and you do not specify ERROR, you should check the return code in your program to determine the outcome of the operation.

$INFO (Information Retrieval)

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

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

Follow this format to generate the labels (DEFINITION PASS):

```
$INFO
```

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.

Follow this format to generate the SVC (EXECUTION PASS):

```
[label]  $INFO  [PLIST-{2      }]
                    {address}
```

PLIST: Specifies the address of the left-most byte of the parameter list generated by the third expansion of this macro. (This would be the *LABEL* on the *LIST* pass expansion of $INFO.) A *2* indicates that this address is available in XR2.

This second expansion of the $INFO macro could be considered the *EXECUTION* pass of the macro. (This is the pass which generates the executable SVC call. If the PLIST parameter is omitted, labels would be generated again as in the *DEFINITION* pass (first expansion) of the $INFO macro.)

The third expansion of the $INFO macro is the *LIST* pass which generates the parameter list to be executed by the SVC generated in pass 2, and defines the function desired.

Follow this format to generate the parameter list (LIST PASS):

Note: This invocation creates a parameter list. It should not be placed within executable code or an execution-time error may occur.

```
[label] $INFO [GET-code]  [,BUFFER-address] [,ID-name]
              [PUT-code]

              [,LEN-number] [,OFFSET-number] [,CIB] [,CD]
```

GET: Specifies the value to be retrieved from the system and placed in the buffer you supply. If this operand is omitted, UPSI is assumed. A description of each GET function follows. The number of bytes returned in the buffer and the contents of those bytes is also given.

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

JULIAN: Returns the Julian date in the format YYDDD, which is based on the program date.

PROGDATE: Returns 3 bytes containing the program date field. This is a 6-digit date in YYMMDD format.

SDATE: Returns 3 bytes containing the session date field. This is a 6-digit date in YYMMDD format.

UPSI: Returns the 1-byte UPSI switch value.

INQUIRY: Returns the 1-byte inquiry switch value. The character Y indicates an inquiry request is pending; N indicates an inquiry request is not pending. If a Y is returned, the inquiry request indicator is reset. A pending inquiry request indicates that the requesting operator has pressed the inquiry key and then selected option 4 from the inquiry menu. If your program is an MRT, option 4 is not available to the operator.

LOCSYS: Returns up to 512 bytes of the system local data area as specified by the LEN and OFFSET operands.

Note: LOCSYS and LOCUSER are mutually exclusive.

LOCUSER: Returns up to 512 bytes of the user local data area as specified by the LEN and OFFSET parameters.

NEP: Returns the 1-byte program attribute. The character Y indicates the program is a never-ending program (NEP); N indicates the program is not a never-ending program.

MRTMAX: Returns the 1-byte hexadecimal value for the maximum number of requesters allowed.

LINES: Returns the 1-byte 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.

SLIST: Returns 3 characters showing the status of the current system list device.

OFF: System list device is off.

CRT: System device is the display station.

ID: The system list device identification is returned as the printer ID.

SYS: The system list device is the system printer.

SPID: When the *$INFO GET-SPID, BUFFER-address* is issued, the buffer should contain the printer filename. If the file is found, the 6-character spool ID is returned in the format SPXXXX, where XXXX is a four- digit number. If the printer file is not found, a return code is issued. See the label $INFRET in the first expansion of $INFO for the possible returned codes, returned at offset $INFRNT from the start of the parameter list.

CIB: Returns the 33-byte block of complier information. This would only be used by a compiler module.

CD: Returns a 2-byte compilation code to the user buffer.

PUT-code: Specifies that the value in the specified buffer is the data used for updating. A description of each PUT function follows. The number of bytes updated and the contents of those bytes is also included.

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

PROG1: Updates the 3-byte, program-1 message member disk address of the job control block (JCB) with values from the $FIND parameter list, which is the 3-byte sector address and the 3-byte F1 address of the library.

PROG2: Updates the 3-byte, program-2 message member disk address of the JCB with values from the $FIND parameter list, which is the 3-byte sector address and the 3-byte F1 address of the library.

USER1: Updates the 3-byte, user-1 message member disk address with values from the $FIND parameter list, which is the 3-byte sector address and the 3-byte F1 address of the library.

LOCSYS: Updates up to 512 bytes of the system local data area as specified by the LEN and OFFSET operands.

Note: LOCSYS and LOCUSER are mutually exclusive.

CIB: Puts a 33-byte block of compiler information into the compiler information block. This would only be used by a compiler module.

CD: Puts a 2-byte compilation code in the JCBDTRCD field of the job control block.

LOCUSER: Updates up to 512 bytes of the user local data area as specified by the LEN and OFFSET parameters.

BUFFER: Specifies the address of the leftmost byte of the buffer in which the data is placed for a GET operation or acquired for a PUT operation. If this operand is omitted, address hex FFFF is assumed.

ID: 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: Specifies a decimal value from 1 to 512, 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.

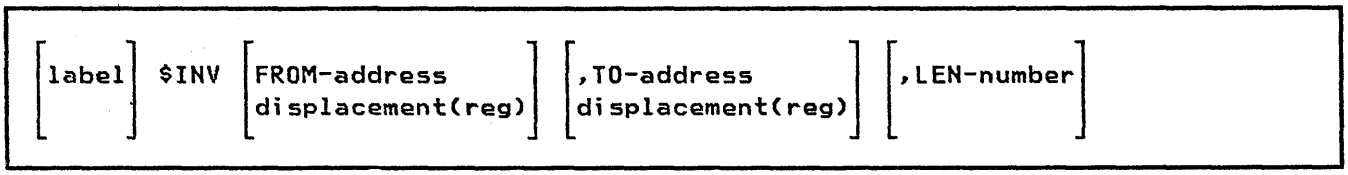
Note: The sum of the values of LEN and OFFSET cannot exceed 513.

OFFSET: Specifies a value from 1 to 512, which is used as the offset for this local request. If this operand is omitted, 1 is assumed.

$INV (Inverse Data Move)

The $INV 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:



FROM: 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 indicated by displacement(reg).

TO: 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 indicated by displacement(reg).

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

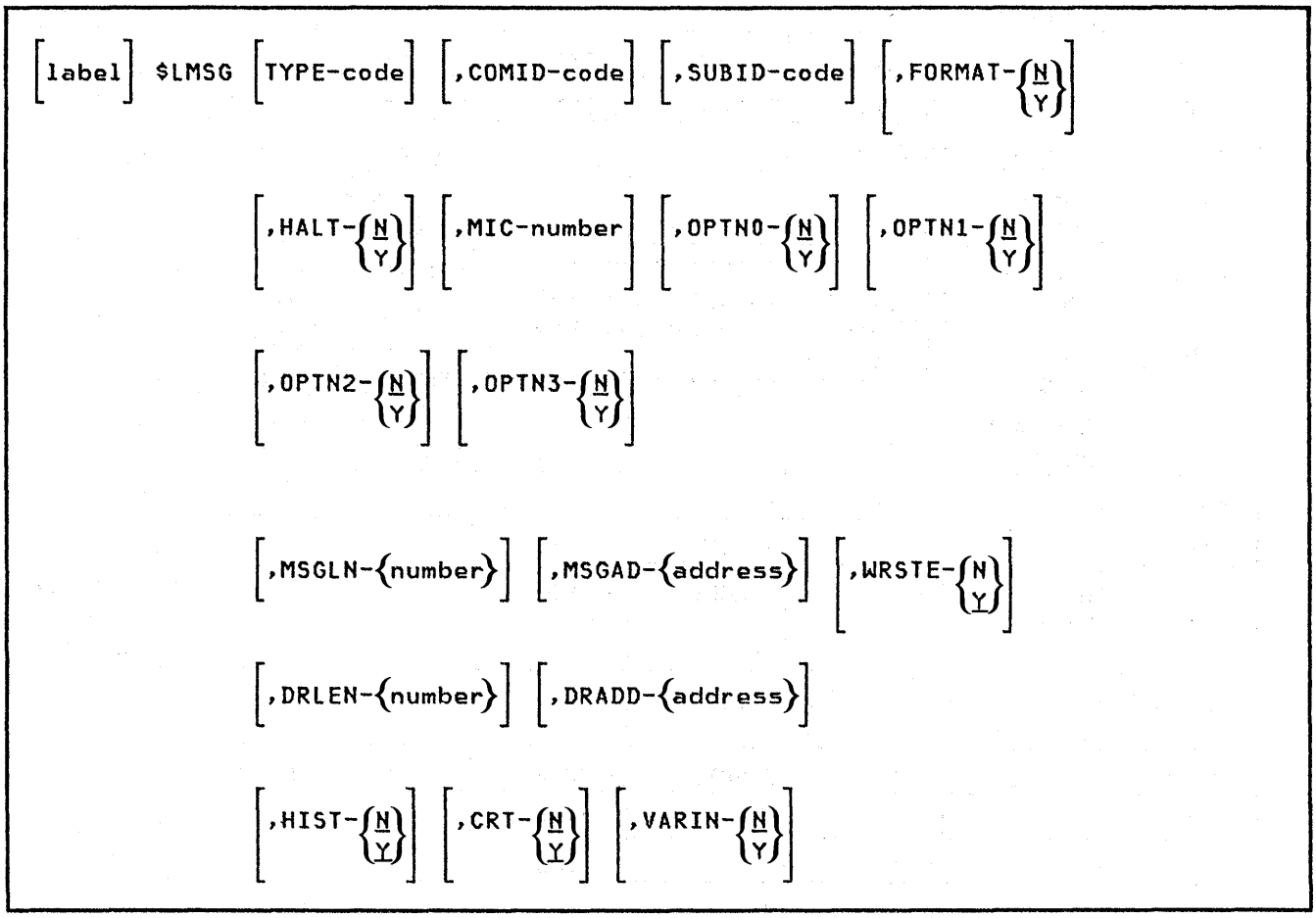
LEN: Specifies the decimal length (in bytes) of the field to be moved.

$LMSG (Generate a Parameter List for a Displayed Message

The $LMSG 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.

For a description of how to use system log, refer to $LOG.

The format of the $LMSG macroinstruction follows:



$LMSG Parameter Use Chart

Parameter	Message Type						Default Values
	1	1R	2	2R	3	4	
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:

No entry = Parameter not used with corresponding message type.

R = Parameter is used with corresponding message type under noted circumstance or diagnostic MIC number issued if not entered.

E = Parameter invalid with corresponding message type and diagnostic MIC number issued if entered.

S = Parameter used with corresponding message type and default assumed if not entered.

TYPE: 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
R	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: 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: 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: Specifies whether 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: Specifies whether the operator is supposed to enter an option number). If this operand is omitted, N (no) is assumed.

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

OPTN0: 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: Specifies whether option 1 is allowed. If Y (yes) is entered, option 1 is allowed; if N (no) is entered or the operand is omitted, option 1 is not allowed.

OPTN2: Specifies whether option 2 is allowed. If Y (yes) is entered, option 2 is allowed; if N (no) is entered or the operand is omitted, option 2 is not allowed.

OPTN3: Specifies whether option 3 is allowed. If Y (yes) is specified, option 3 is allowed; if N (no) is specified or the 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.

MSGLN: 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: 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 values less than hex 40.

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

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

DRLEN: Length of the reply area in decimal. This area must be either 1, 8, 60, or 120 bytes long.

DRADD: Specifies the address of the leftmost byte of the reply area.

HIST: Specifies whether the message is to be recorded on the history file. If this operand is omitted, Y (yes) is assumed.

CRT: Specifies whether the message is to be displayed on the display screen. If this operand is omitted, Y (yes) is assumed.

VARIN: Specifies a variable length data insert (1 through 32) for type-1 messages. The system log function allows you to insert variable length data anywhere in 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.

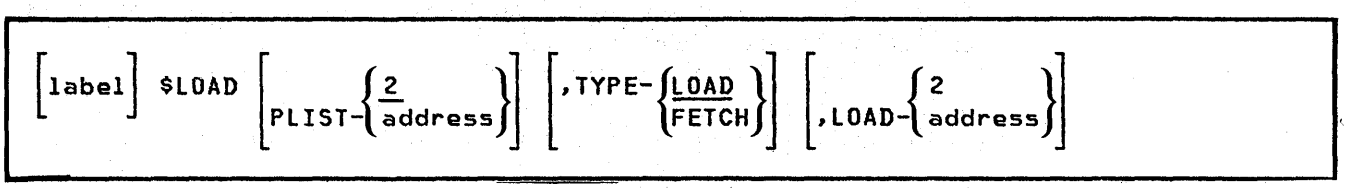
Note: The inserted data should contain only printable characters. For example, the data string should not contain values less than hex 40.

For a description of how to use system log, refer to $LOG.

$LOAD (Load or Fetch a Module)

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 can have control returned after the module is loaded, or you can pass control to the module. If you will need the data in register 2 later, save the contents of register 2 before issuing $LOAD.

The format of the $LOAD macroinstruction follows:



PLIST: This address identifies the directory entry of the module in main storage and specifies the address of the leftmost byte of the parameter list built by $FNDP. Register 2 can be specified; however, if this operand is omitted, the address is assumed to be register 2.

TYPE: 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: Specifies either the address at which the module is to be loaded in main storage, or the address in register 2. The address must be on an 8-byte boundary. Use this parameter only if the load address is to be filled or changed. There is no default for LOAD. If an address is specified for LOAD, then PLIST must be specified and cannot be the value 2.

$LOG (Generate the Linkage to the System Log)

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

Use the $LOG macroinstruction 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.

Formatted and unformatted output are available through system log. Both types are displayed on the system log device.

- A formatted message has two lines. The first line is the format line, which contains the message ID and available options. The second line contains the message text.
- An unformatted message has one line. It indicates errors or issues instructions to the operator.

To use system log, you must do the following:

1. Build the log parameter list using the $LMSG macroinstruction.
2. Use the $LOGD macroinstruction 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.

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:

```
[label] $LOG [LIST-address] [,OPTN0-address] [,OPTN1-address]

             [,OPTN2-address]
```

LIST: 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: 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: 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: 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.

$LOGD (Generate Displacements for System Log)

The $LOGD 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.

For a description of how to use system log, refer to $LOG.

The format of the $LOGD macroinstruction follows:

```
[label]  $LOGD
```

$OPEN (Prepare a Device or File for Access)

This macroinstruction prepares a device or file for data transfer. Use the $ALOC macroinstruction before preparing (opening) the file. One or more of the following functions are performed for each file opened:

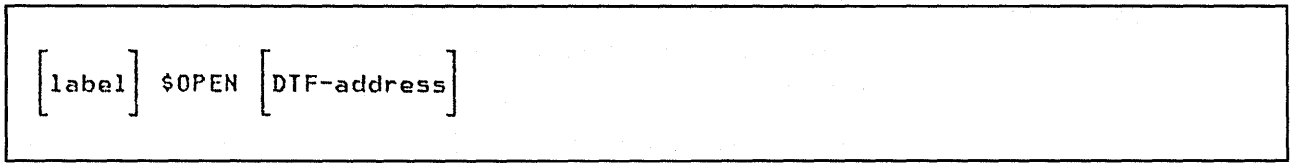
- The DTF is formatted.
- Data (I/O) buffers and data management control blocks have space allocated.
- Bit 7 in the second attribute byte of the DTF is set on to indicate that the DTF is open.
- Data management control blocks are initialized.
- Diagnostic tests are performed to ensure that the access method and the file organization are compatible and that other information in the DTF is correct.

Note: When a DTF is opened ($OPEN) it cannot be moved or overlaid until it is closed ($CLOS). More than one DTF can be opened at one time by chaining the DTFs. See $DTFB, $DTFD, $DTFP, and $DTFW.

Input: The preopen DTF is input to the open routine. Before issuing $OPEN, you must be sure to allocate the device or file by issuing the $ALOC macroinstruction.

Output: The open routine returns control to your program after the requested file is opened. All register contents are restored. The devices and files corresponding to open DTFs are prepared for use.

The format of the $OPEN macroinstruction follows:

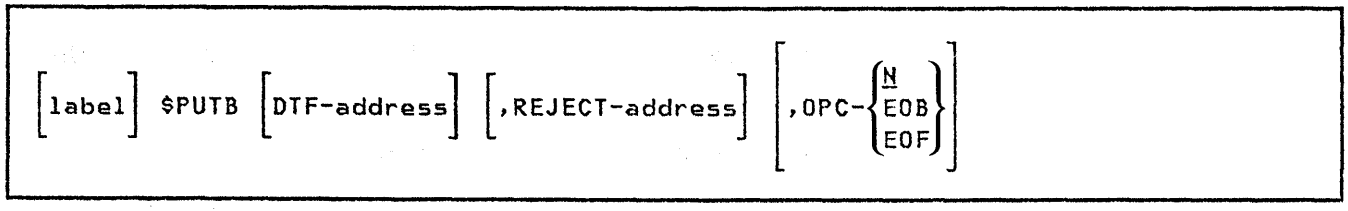


DTF: Specifies the address of the leftmost byte of the first DTF to be opened. If this operand is entered, an LA instruction is generated to load the specified address into register 2. If this operand is not entered, the address is assumed to be in register 2.

$PUTB (Issue a Put Request)

The $PUTB macroinstruction generates code to move data from your logical buffer to a BSC I/O buffer. To use this macroinstruction, construct a BSC DTF for the file (using the $DTFB macroinstruction) and use the $DTFO macroinstruction to generate the labels and establish the offsets for the DTF.

The format of the $PUTB macroinstruction follows:



DTF: Specifies the address of the DTF for which the put was issued. If this operand is omitted, the address is assumed to be in register 2.

REJECT: Specifies the routine to receive control if the put request is rejected by BSC. If this operand is omitted, control is returned to the user program at the next sequential instruction after the $PUTB. You should check the return code to determine the outcome of the operation.

Note: To prevent issuing BSC requests after a BSC error has occurred, the REJECT parameter should always be coded.

OPC: Specifies how BSC should send this record.

N: Specifies normal record blocking before the record is sent. If this operand is omitted, N (no) is assumed.

EOB: Specifies that the block is ended with this record and should be sent as it is.

EOF: Specifies end of file. The put file is closed by transmitting the last block of data with end of text (ETX), then transmitting end of transmission (EOT). If operation is in 3740 multiple file mode, the last block of data is transmitted with end-of-text block (ETB), and System/36 waits for the next user operation.

Note: No new data is sent when the EOF operation code is issued. The ETX or ETB is placed at the end of the previous block of data.

$PUTD (Construct a Disk Put Interface)

The $PUTD macroinstruction generates the interface needed to communicate with data management when putting a record on 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 (OUTREC) 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 follows:

```
[label] $PUTD  OP-code  [,DTF-address] [,DUPRCO-address] [,IOERR-address]
      [,EOX-address] [,DUPREC-address] [,SEQERR-address]
      [,KEYERR-address] [,INVDRP-address] [,DIRNDR-address] [,INVOP-address]
      [,IRN-address] [,ERROR-address]
```

OP: Must be specified as follows:

Code	Meaning
AEOD	Adds a record at the end of data (consecutive, indexed, and GAM with AEOD capable).
ARRN	Adds a record at the RRN location (direct and GAM with ARRN capable).
UPDATE	Updates the record at the current record pointer location.
DELETE	Deletes the record at the current record pointer location.

RELEASE Releases the record at the current record pointer location.

NULL Does not move any operation code into the DTF (see note).

Note: Null can be used if the operation code is changed by the program. The programmer is responsible to assure that some operation code is moved into the DTF before $PUTD is run.

DTF: 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.

DUPRCO: Specifies the address that receives control when you attempt to add a duplicate key to an alternative index of this file, and that other index does not allow duplicates.

IOERR: Specifies the address that receives control if the controlled cancel option is taken in response to a permanent disk error. If this operand is omitted, there is no check for a permanent disk error completion code.

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

DUPREC: Provides the address in your program that receives control when you attempt to add a duplicate key to this file and duplicates are not allowed.

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

KEYERR: Specifies the address in 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: 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. An invalid put is signaled if the record to be added to or updated in the file contains hex FF in the first byte.

DIRNDR: Used only for the direct access method and GAM. 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 (the record does not contain hex FF in the first byte).

IRN: Specifies your program address that receives control when an invalid-record-number condition occurs. For example, issuing $PUTD OP-ARRN for a record not in the file.

INVOP: The address in your program that receives control if an invalid operation condition occurs.

ERROR: Supplies the address in your program that receives control if any unsuccessful completion code is detected. The successful completion code is hex 40.

Note: If DUPRCO, IOERR, EOX, DUPREC, SEQERR, KEYERR, INVDRP, DIRNDR, IRN, or INVOP occurs but is not specified, and ERROR is not

specified, you should check the return code in your program to determine the outcome of the requested operation.

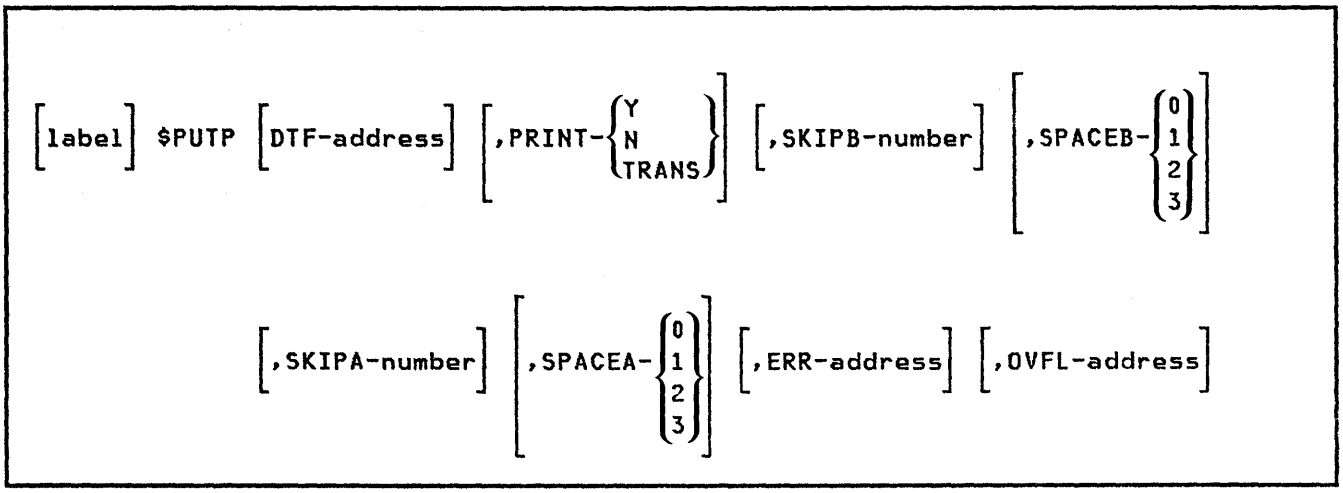
$PUTP (Construct a Printer Put Interface)

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

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:



DTF: 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: Specifies with Y (yes) to perform both a print and the specified skip or space, or with N (no) only a skip or space. If this operand is omitted, the DTF is unchanged.

PRINT-TRANS (transparent mode): Requires the before forms feed commands at the beginning of the record. The after forms feed command of the printer DFT must be 0 or 1.

SKIPB: Specifies the line to skip to before the print operation. The maximum must be less than the number of lines per page as specified in $DTFP. If this operand is omitted, the DTF is unchanged.

SPACEB: Specifies the number of lines to space before the print operation. The maximum must be less than the number of lines per page as specified in $DTFP. If this operand is omitted, the DTF is unchanged.

SKIPA: Specifies the line to be skipped to after the print operation. The maximum must be less than the number of lines per page as specified in $DTFP. If this operand is omitted, the DTF is unchanged.

SPACEA: Specifies the number of lines to space after the print operation. The maximum must be less than the number of lines per page as specified in $DTFP. If this operand is omitted, the DTF is unchanged.

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

ERR: 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.

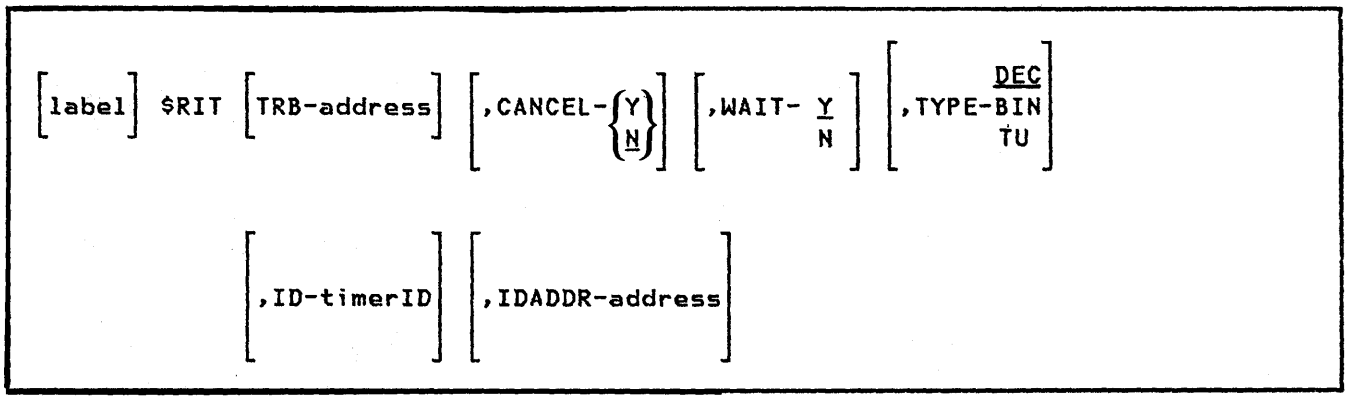
OVFL: Specifies the address in your program that should receive control if page overflow occurs.

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 program must reset the DTF if this is required.

$RIT (Return Interval Time)

The $RIT 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 by the TYPE parameter in the $RIT macroinstruction.

The format of the $RIT macroinstruction follows:



TRB: 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: Specifies whether the remaining time in the interval is to be canceled. If this operand is omitted, N is assumed.

WAIT: Specifies whether or not the task issuing the $RIT macroinstruction is to be put 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.

TYPE: The method of stating the format of the time returned.

DEC — A 6-byte decimal number specifying the time in hours, minutes, and seconds (HHMMSS) until the timer expires. The time is based on a 24 hour clock.

BIN — A 32-bit binary number specifying the time in seconds until the timer expires. The binary value is right-adjusted in bytes 4 through 7 of the timer request block time field. The time is based on a 24 hour clock.

TU — A 32-bit binary number specifying the time in timer units until the timer expires. One timer unit is 8.192 milliseconds. The binary value is right-adjusted in bytes 4 through 7 of the timer request block time field. The time is based on a 24 hour clock.

The default is DEC.

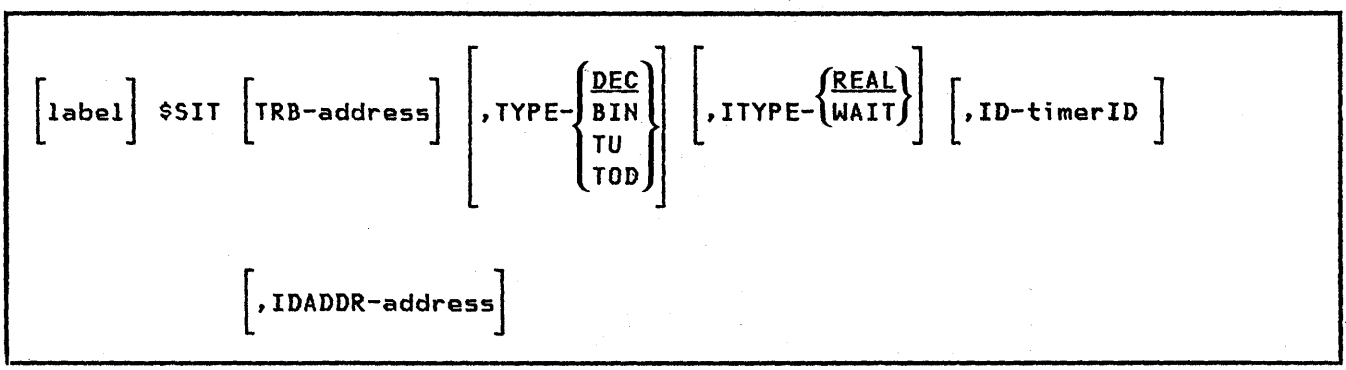
ID: A 1-byte self-defining expression, whose value is from 1 through 255, used to identify task timer intervals. This parameter is not needed if the time intervals are synchronous. This parameter is ignored if IDADDR is also specified. The default value is 1.

IDADDR: The address of the location containing a self-defining expression of the value 1 through 255 that is used to identify task timer intervals.

$SIT (Set Interval Timer)

The $SIT 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 follows:



TRB: 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: 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) on a 24-hour clock that can pass before the timer interrupt.

BIN — A 32-bit binary number specifying the number of seconds that can pass before the timer interrupt. The binary value must be right-adjusted in bytes 4 through 7 of the timer request block time field.

TU — A 32-bit binary number specifying the number of timer units that can pass before the timer interrupt. One timer unit is 8.192 milliseconds. The binary value must be right-adjusted in bytes 4 through 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). The time is with respect to a 24 hour clock.

ITYPE: 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.

ID: A 1-byte self-defining expression, whose value is from 1 through 255, used to identify task timer intervals. This parameter is not needed if the time intervals are synchronous. This parameter is ignored if IDADDR is also specified. The default value is 1.

IDADDR: The address of the location containing a self-defining expression of the value 1 through 255 that is used to identify task timer intervals.

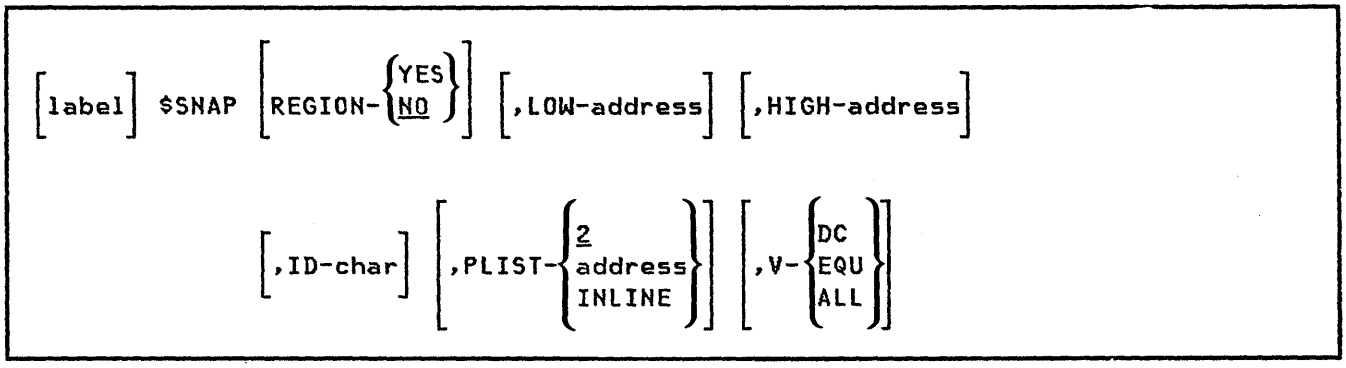
$SNAP (Snap Dump of Main Storage)

The $SNAP macroinstruction provides a system storage dump. You must specify the region or the limits of the area to be dumped. The program continues unaffected at the end of the dump. 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 work registers W4, W5, W6, and W7
- 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 follows:



REGION: 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.

LOW: 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.

HIGH: 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.

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

ID: Specifies the four characters used as a dump identifier. If this operand is omitted, blanks are assumed.

PLIST: Specifies the address of the $SNAP parameter list. If this operand is omitted, 2 is assumed. The parameters have the following meanings:

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

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-INLINE. The parameters have the following meanings:

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.

$SORT (Construct a Loadable Sort Interface)

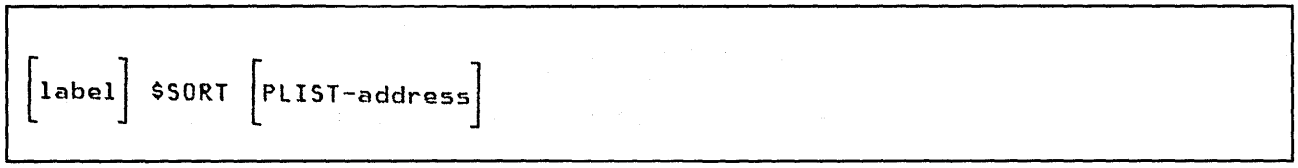
The $SORT macroinstruction generates an interface to the sort utility. The sort utility is part of the SSP. The sort utility is described in the *Sort Reference Manual*. Before you issue $SORT, you must generate a sort parameter list by issuing the $SRT macroinstruction. $SRT is described in the following pages.

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. 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 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.

The format of the $SORT macroinstruction is:



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

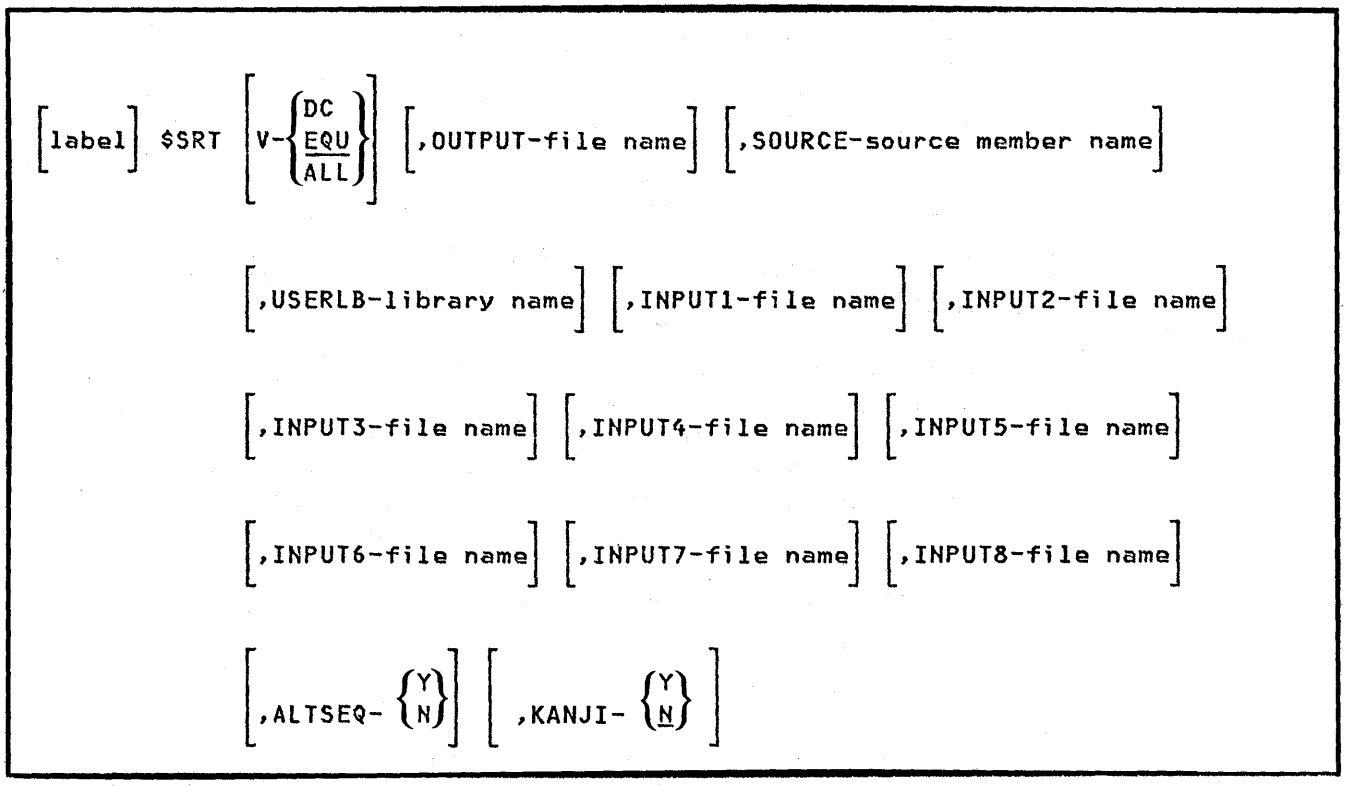
$SRT (Generate a Loadable Sort Parameter List)

The $SRT macroinstruction generates the parameter list used by the sort utility when it is called by the $SORT macroinstruction.

The sort utility is part of the SSP. The sort utility and the parameter list are described in the *Sort Guide* manual.

The maximum size of the parameter list is 2048 bytes, including 125 bytes reserved as a work area for the sort utility.

The format of the $SRT macroinstruction follows:



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

DC — Generates the sort parameter list used by the sort utility when it is called 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: Specifies the name of the file that is to contain the sorted data. If this operand is omitted, blanks are assumed. See Notes 1 and 2.

SOURCE: Specifies the name of the source member that contains the sort specifications. If this operand is omitted, no entry is created for it in the generated parameter list, and the 34-byte sort specifications must be placed immediately after the generated portion of the sort parameter list.

Omit this operand if you want to supply the sort specifications in the sort parameter list. See Note 1.

USERLB: 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. #LIBRARY is assumed if a source name is specified and USERLIB is omitted. Omit this operand if you want to supply the sort specifications in the sort parameter list. See Note 1.

INPUT1: Specifies the name of the first, or only, input file to sort. If this operand is omitted, blanks are assumed. See Notes 1 and 2.

INPUT2: 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. See Note 2.

INPUT3: 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. See Note 2.

INPUT4: 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. See Note 2.

INPUT5: 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. See Note 2.

INPUT6: 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. See Note 2.

INPUT7: 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. See Note 2.

INPUT8: 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. See Note 2.

ALTSEQ: Specifies whether an alternative 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.

KANJI: Specifies whether to invoke the extended sort utility, for sorting ideographic data: Y if yes, N if no. If this operand is omitted, N is assumed. Omit this parameter if non-ideographic data is to be sorted.

Notes:

1. Space is always reserved in the generated parameter list for an OUTPUT file name and an INPUT1 file name. 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. Files named must correspond to the NAME parameter on the FILE OCL statements for the respective files. The files must be closed before $SORT is used.

Constructing a SORT Parameter List

The following example shows how to use $SRT 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 in a source member.
- The specified alternative collating sequence sorts all characters except blanks, uppercase alphabetic characters, and numeric characters to the end of the file.

```
PROGRAM                                                    TYPING INSTRUCTIONS   GRAPHIC
PROGRAMMER                                  DATE           TYPING INSTRUCTIONS   CHARACTER

                                                STATEMENT
Label      Operation  Operand                                                    Remarks
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 7
SORTPARM   $SRT       Y-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'Ø7ØØ'                      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
```

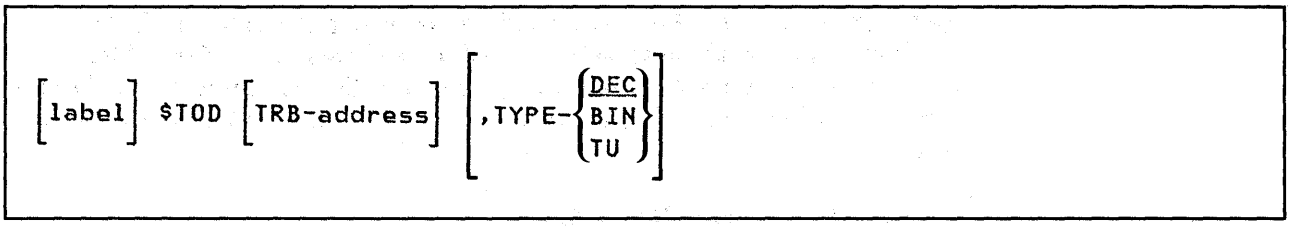
The following operation calls the loadable sort:

```
PROGRAM                                                    TYPING INSTRUCTIONS   GRAPHIC
PROGRAMMER                                  DATE           TYPING INSTRUCTIONS   CHARACTER

                                                STATEMENT
Label      Operation  Operand                                                    Remarks
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 7
           $SORT     PLIST-SORTPARM
```

$TOD (Return Time and Date)

The $TOD 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 is returned in the date field. The time and date fields are at displacements $TRBTIME and $TRBDATE, respectively, in the timer request block generated by $TRB. The date is returned in the format specified during system configuration.

The format of the $TOD macroinstruction follows:



TRB: 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: Specifies how the time is to be returned in the timer request block. The time is with respect to a 24 hour clock. 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 through 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 through 7 of the time field of the timer request block.

If this operand is omitted, DEC is assumed.

$TRAN (Generate an Interface to the Translate Routine)

The $TRAN macroinstruction generates an interface to the translate routine for EBCDIC-ASCII translation. See $TRTB and $TRL macros.

```
[label] $TRAN [TRL-address]
```

TRL: Specifies the symbolic address of the translate parameter list. If this operand is omitted, the address is assumed to be in register 1. If the $TRL macroinstruction is used to generate the parameter list, this address should be the label assigned to the $TRL macroinstruction. The parameter list is described as follows:

Field Length	Field Description
2	Address of the translate table (Your program must define the translate table.)
2	FROM field address, for translation
2	TO field address, for translation
2	Number of bytes to translate
1	Completion code: Hex 00 indicates translation complete, no errors; hex FF indicates invalid character encountered

$TRB (Generate Timer Request Block)

The $TRB 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 follows:

V-DC/EQU/ALL: Specifies whether the parameter list labels, DCs, or both, are generated for the $RIT, $SIT, and $TOD macroinstructions. If this operand is omitted, DC is assumed. The following is the parameters and their meanings:

DC — Generates the DC's for the timer request block parameter list.

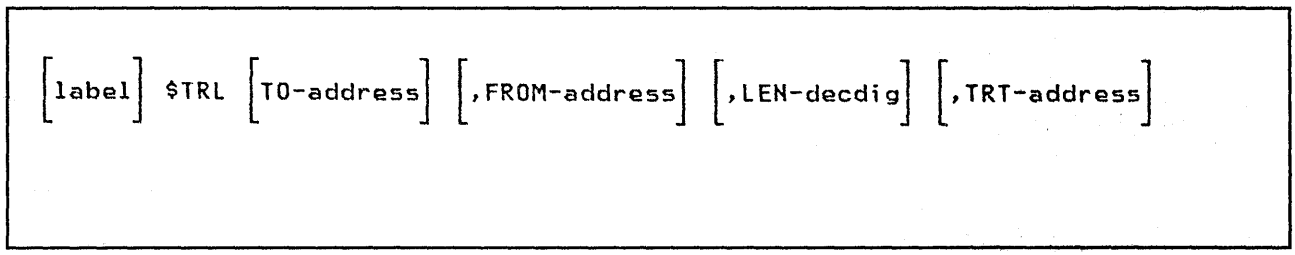
EQU — Generates the displacement labels for the timer request block.

ALL — Generates the timer request block and the corresponding displacement labels.

$TRL (Generate a Translation Parameter List)

The $TRL macroinstruction generates a parameter list used by the translation routine for EBCDIC-ASCII translation. See $TRAN and $TRTB macros. $TRL does not generate executable code.

The format of the $TRL macroinstruction follows:



TO: Specifies the symbolic address of the leftmost byte of the field to which the translated data will be moved.

FROM: Specifies the symbolic address of the leftmost byte of the data field to be translated. This address may be the same as the address specified in the TO operand.

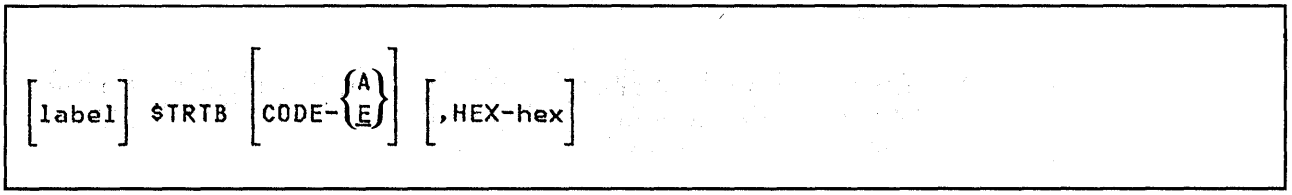
LEN: Specifies in decimal the number of characters to be translated.

TRT: Specifies the symbolic address of the leftmost byte of the translate table. If the $TRTB macroinstruction is used to generate the translate table, this address should be the label assigned to the $TRTB.

$TRTB (Generate a Translation Table)

This macroinstruction generates an EBCDIC to ASCII or ASCII to EBCDIC translation table. The table is generated in the format required by the $TRL macroinstruction, and the table can be addressed by $TRL when you translate data.

The format of the $TRTB macroinstruction follows:



CODE: Specifies whether the data is to be translated from EBCDIC to ASCII (E) or from ASCII to EBCDIC (A). If this operand is omitted, E is assumed. If CODE-E is specified, $TRTB generates a 258-byte translation table; if CODE-A is specified, $TRTB generates a 130-byte translation table.

HEX: Specifies the hexadecimal digits with which to replace any invalid characters found during translation. If the HEX operand is not specified, the replacement character is hex 3F for ASCII to EBCDIC or hex 1A for EBCDIC to ASCII.

Translation tables generated by the $TRTB macroinstruction are generated in the following format:

Byte	Field Description
0	Identifies a character that is not to be translated.
1	Substituted for characters that are not to be translated.
2 through 257	256-byte EBCDIC to ASCII translation table.
2 through 129	128-byte ASCII to EBCDIC translation table.

Construct the translation table so that the displacement from the beginning of the table equals the hexadecimal representation of the untranslated character. The contents of the location indicated by the displacement is the character to be translated to. (For example, if you want to translate hex C1 to hex 41, you should construct a translation table in which the value at displacement hex C1 in the table is hex 41.)

The translate routine processes a field, specified by the $TRL macroinstruction, 1 byte at a time. The byte at a given displacement is compared with the first byte in the translate table (byte 0). If they are equal, the character is considered to be invalid, and the following actions are performed:

- The completion code in the parameter list is set to indicate that an invalid character was detected.
- The second byte of the translate area (byte 1) is substituted for the original character.
- Translation continues with the next character. After the translate routine is finished, control is returned to your program with a completion code in the translate routine parameter list.

$WIND (Generate Override Indicators for Display Station)

The $WIND 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 follows:

```
[label] $WIND [MAXIND-number]
```

MAXIND: Specifies in decimal the highest number used by SFGR as an override indicator for your program. If this operand is omitted, 99 is assumed.

$WSEQ (Generate Labels for Display Station)

This macroinstruction generates labels and offsets to reference certain work station device-dependent values, such as identification (AID) bytes and bit representations for the display screen attribute bytes and write control characters.

The format of the $WSEQ macroinstruction follows:

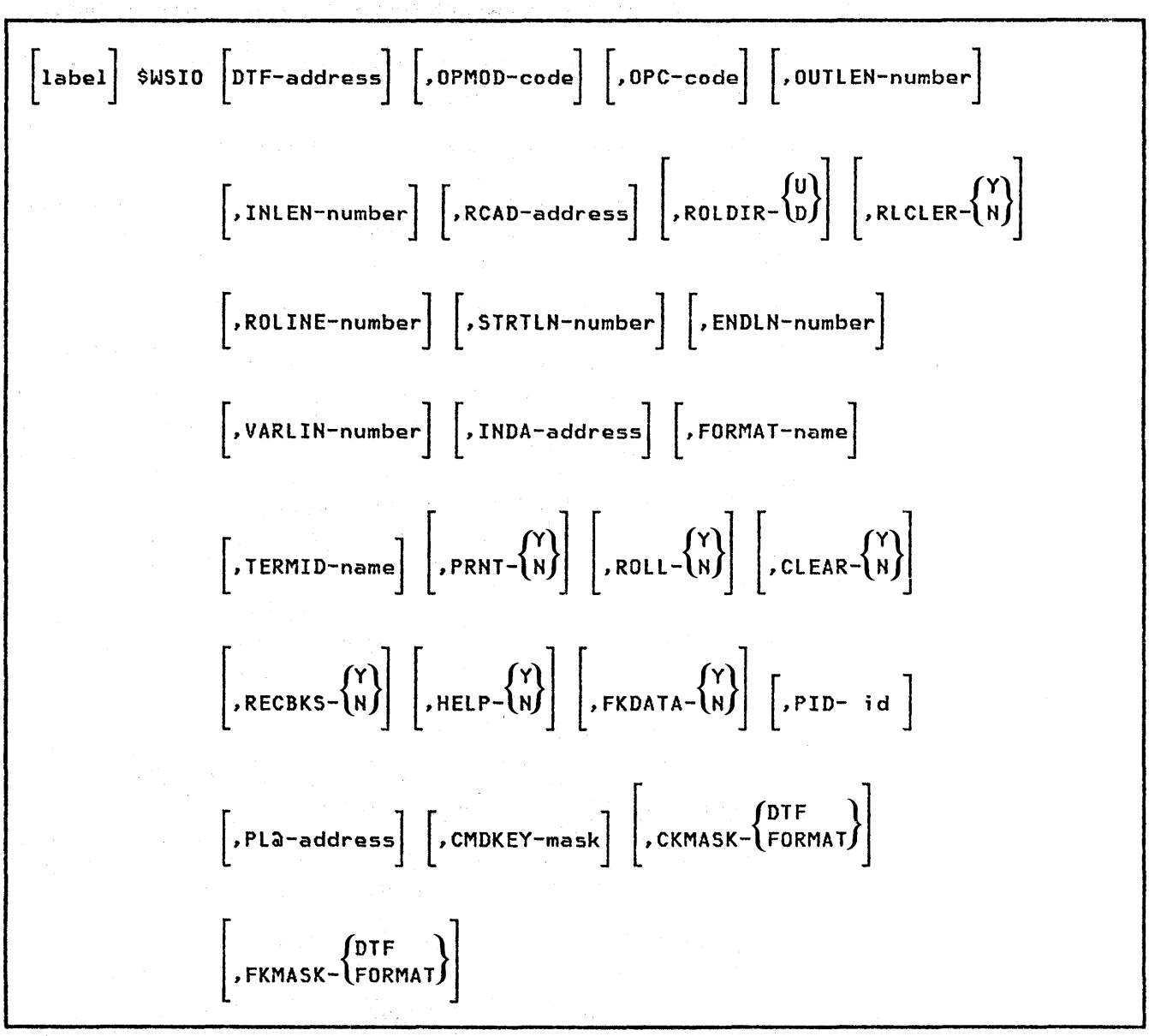
```
[label]  $WSEQ
```

$WSIO (Construct a Display Station Input/Output Interface)

The $WSIO macroinstruction builds the executable code to modify a display station DTF using only the specified parameters, then issues a call to work station data management to perform 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, save the contents of those registers before issuing $WSIO. For a description of how to code $WSIO for the interactive communications feature, see the manual, *Interactive Communications Feature: Reference Manual.*

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 in *Appendix F.*

The format of the $WSIO macroinstruction follows:



DTF: 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: Specifies the operation code modifier to be generated. The codes and their meanings are as follows:

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: Rolls the display with the specified operation.

UNF: The FORMAT parameter need not be specified. The stream of data and control commands in the user's program logical record area, beginning at the RCAD specified address, is sent to the work station. The OUTLEN parameter specifies the number of bytes to be sent. If an unformatted PUT is specified and there are input fields defined in the data stream, the INLEN value must be specified on the $WSIO macroinstruction.

Note: See the *Functions Reference Manual* for more information on display station data streams.

PRINT: Prints the displayed data on the printer specified in the PID parameter.

PRUF: PUT for read under format.

FMH: Use only with the interactive communications feature, which is described in the *Interactive Communications Feature: Reference Manual*, SC21-7910. This 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.

CONFIRM: Use only with the interactive communications feature, which is described in the *Interactive Communications Feature: Reference Manual*, SC21-7910. This code indicates that a confirm indication is to be sent with the data associated with the EVOKE, PUT, GET (in the send state only), and INVITE (in the send state only) operations. This code is valid only for the APPC subsystem.

ZERO: Clear any previous OPMOD specification.

Notes:

1. The OPMOD keyword can be coded as OPM.

2. An OPC of PUT, PTG, PNW, or PTI must also be specified for OPMOD values of OVR, UNF, or PRUF.

OPC: 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 manual *Interactive Communications Feature: Reference*):

ZERO: Sets the operation code field to hex 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: Receives 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: Sends data to the display station specified by the TERMID parameter. Control is returned to your program when data transfer is complete.

PTG: Sends 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: Enables 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 or GET operation. Control returns to your program as soon as the invite input (INV) is scheduled.

PNW: Sends 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 scheduled. 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: Sends 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: 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. This operation ignores the OPMOD value.

ACQ: Allocates the display station specified by the TERMID parameter for this program. This operation ignores the OPMOD value.

REL: Releases from this program the display station specified by the TERMID parameter. This operation ignores the OPMOD value.

GTA: Gets the attributes of the display station specified by the TERMID parameter, and places 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' Nondisplay type

All remaining letters are reserved.

Byte 1 **Display Size**

C'1' 1920-character display

Byte 2 **Attachment Type**

C'L' Local

C'R' Remote

The attachment type is C'R' for a display station pass-through or DHCF device.

Byte 3 **Online/Offline Status**

C'O' 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	**Completion Status of Device Invite**
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: Gets any other attributes of the display station specified by the TERMID parameter, and places them in the program's record area. This operation also ignores the OPMOD value.

Following a get attribute operation, the program's record area appears as follows:

Bytes 0-7	Same as for GTA.

Note: If the device is offline (Byte 3 = C'F'), the values found in Bytes 8-15 may not be accurate.

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.
Byte 11	Application help facility.
C'Y'	Facility enabled.
C'N'	Facility not enabled.
Byte 12	27x132 status.
C'1'	27x132 capable display station is in 24x80 mode.

C'2'	27x132 capable display station is in 24x132 mode.
C'N'	24x80 capable only.
Bytes 13-15	Reserved. Hex A is returned.

GST: Gets the Advanced-Program-to-Program Communication session status. See the *Interactive Communications Feature: Reference Manual*, SC21-7910 for more information.

STI: Cancels a previously issued invite input request to the display station specified by the TERMID parameter. If the stop invite fails (the operator already pressed the Enter/Rec Adv key, a function key, or command key), your program is informed by a return code, and the data remains at the display station and is available for a subsequent request. If the program issues a get or accept after the stop invite fails, the system handles any disabled command or function key. The system waits until the Enter/Rec Adv key or an enabled command or function key is pressed before giving data or control back to the program. However, if an output request is issued to the display station, the input data is lost.

Note: A stop invite is not required to override an existing invite input. WSDM performs a stop invite when necessary. 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 *Creating Displays: Screen Design Aid and System Support Program.*

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 of the display station that was 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, hex 00 means no invites outstanding; hex 10 means at least one invite outstanding; hex 30 means at least one invite outstanding, and at least one completed invite. In the low-order byte, hex 00 means stop

system is not in effect; hex 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 not changed. 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 uses 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, and at least one completed invite. 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.

STM: Specifies the time interval to wait before issuing a timer expired return code. The first 6 bytes of the user record area specify the interval in the format HHMMSS. A timer expired return code is returned on the first accept following the expiration of the timer. When this return code is given, a TERMID is not returned, and the TERMID field of the DTF is unchanged. If a previous set timer has not yet expired, the old time interval is replaced with the new.

OUTLEN: Only required for OPMOD parameters ERROR and UNF, or OPC parameters PUT, PTG, PNW, and PTI. If the operation is ERROR, the OUTLEN value must be between 1 and 78. OUTLEN represents the amount of data written from the logical record area to the error line at the display station. If the operation has an OPMOD of UNF, the OUTLEN value must be between 2 and 4096. It represents the exact length of the data stream. If the operation is a PUT, PTG, PNW, or PTI, OUTLEN represents the maximum amount of data that can be written from the logical record area to the output fields in the display format. The OUTLEN value must be at least as large as the sum of the lengths of all program output fields. If the operand is omitted, the DTF value is unchanged. After a successful input operation, the actual length of data returned is stored in this field. Therefore, OUTLEN should be respecified after every input operation.

Note: If the execution time output data from the user's logical record area also contains MIC data, the user must reserve 6 bytes for each MIC to contain the 4-character digits and the 2-character message member identifier. This 6-byte length must be included in the total OUTLEN value.

INLEN: Specifies in decimal the size of your 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 is unchanged. The INLEN and PID(printer ID) 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, INLEN must equal the total length of all input fields defined on the display.

RCAD: Specifies the symbolic address of the leftmost byte of the logical record area. If this operand is omitted, the DTF is unchanged.

Note: If the operation being performed involves GET or ACI or UNF, the record area must be on an 8-byte boundary.

ROLDIR: Specifies the direction to roll the display when requested. This operand must be specified in the first $WSIO you issue with a roll operation. If this operand is subsequently omitted, the DTF is unchanged.

RLCLER: 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. If this operand is subsequently omitted, the DTF is unchanged.

ROLINE: Specifies in decimal the number of lines a roll operation should roll the data being displayed. The maximum number is 24. If this operand is omitted, the ROLINE-number in the DTF is unchanged.

STRTLN: Specifies in decimal the first line of the roll area on a roll operation. The maximum number is 23. If this operand is omitted, the DTF is unchanged.

ENDLN: Specifies in decimal the last line of the roll area on a roll operation. The minimum number is 02. The maximum number is 24. If this operand is omitted, the DTF is unchanged.

VARLIN: 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 24. If this operand is omitted, the DTF is unchanged.

INDA: 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 address hex 0000 because WSDM assumes no indicator area exists at address hex 0000, and the indicators are assumed to be off. If this operand is omitted, address hex 0000 is assumed.

FORMAT: Specifies the name of the display format to be used for this operation. This operand is required only for formatted PUT operations. If this operand is omitted, the DTF is unchanged.

TERMID: Specifies the symbolic name of the display station. This is the 2-character ID, which the user assigned either during system configuration or in 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 is unchanged.

PRNT: Specifies whether your program can 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 current display with the optional heading and border on the printer associated with the display station. If the operand is omitted, N (no) is assumed.

ROLL: Specifies whether your program is able to process the Roll Up and 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. Data is

returned as if the Enter/Rec Adv key was pressed. If N (no) is specified, an error message is displayed when the operator presses either roll key (see Note 1).

CLEAR: Specifies whether your program can 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 when the operator presses the Clear key.

RECBKS: Specifies whether your program can 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 when the operator presses the Home key.

HELP: Specifies whether your program can 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 when the operator presses the Help key.

FKDATA: Specifies whether input data is 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. Input data is returned regardless of whether the operator modified any of the fields. This function 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 you press an enabled function control key. No input data is returned (see Note 1).

Notes:

1. The FKDATA parameter has no effect on the operation of the Roll Up and Roll Down 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 display of the remote work station when a function control key is pressed while the FKDATA parameter is active.

PID: Specifies the ID of the desired printer on a print request. Allowable values are:

Code	Meaning
SYSTEM	The system printer.
WSTN	The printer associated with the display station specified by the TERMID parameter nn.
XX	Where XX is the 2-character ID of the desired printer.

If this operand is omitted, the DTF is unchanged. The INLEN and PID parameters use the same field in the DTF; therefore, PID must be specified after each input operation.

PL@: Used with the interactive communications feature. This parameter specifies the address of an associated evoke parameter list, which is generated by the $EVOK macroinstruction. $EVOK is described in the manual *Interactive Communications Feature: Reference.* This operand must be specified for the first evoke operation and is unchanged if not specified again.

CMDKEY: Specifies the command key mask to be placed in the DTF. The mask is made up of 24 binary bits (bit 0 = CMD1 through bit 23 = CMD24) entered as 6 hexadecimal digits. If this operand is omitted, hex FFFFFF is assumed.

CKMASK: Specifies whether WSDM should use the command key mask from the display format or from the DTF. If this operand is not specified on any $WSIO call, FORMAT is assumed. If it is specified on any $WSIO call, any future $WSIO calls will leave the DTF unchanged if this parameter is omitted.

FKMASK: Specifies whether WSDM should use the function key mask from the display format and the DTF (format is specified in the DTF), or just from the DTF (DTF is specified). If this operand is not specified on any $WSIO call, FORMAT is assumed. If it is specified on any $WSIO call, any future $WSIO calls will leave the DTF unchanged if this parameter is omitted.

Programming Considerations

Coding Restrictions

The generated code for some macroinstructions uses register 1 and/or register 2. The contents of the register 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	$LOAD	$RIT
$CLOS	$LOG	$SIT
$FIND	$OPEN	$SNAP
$GETB	$PUTB	$SORT
$GETD	$PUTD	$TOD
$INFO	$PUTP	

Disk, printer, and work station data managements use work registers 4, 5, 6, and 7. Unless the contents of these registers are no longer needed, they must be saved before issuing any of the following macroinstructions:

$GETD
$PUTD
$PUTP
$WSIO

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.

Binary Synchronous Communications

Macroinstructions

BSC macroinstructions can cause the IBM System/36 to function as any of the following station types:

- Receive only (receive data from a remote terminal)
- Transmit only (transmit data to a remote terminal)
- Transmit and receive (no conversational reply) in one of three modes of operation:
  - Transmit a file, then receive another file
  - Receive a file, then transmit another file
  - Transmit records from one file while receiving records from another file.

  Note: Because BSC closes the file in use before another file is to be used, there is a delay between each transmit and receive operation. The remote station might not be tolerant of this delay.

Every BSC program you write with the assembler language must do these two things:

- Prepare BSC DTFs for data reception, data transmission, or both.
- Begin and end the transfer of data (receive data, transmit data, or both).

Preparing BSC DTFs For Data Transfer

When writing a program for data transfer, always include the following three steps:

1. Generate field displacements and labels for the BSC DTFs by using the $DTFO macroinstruction coded with BSC-Y and FIELD-Y.

2. Prepare BSC data files. Define each BSC file ($DTFB), allocate it ($ALOC), and open it ($OPEN).

3. If data in your BSC files requires translation, either before it is transmitted or after it is received, you must provide for data translation by constructing translate tables ($TRTB macroinstruction for EBCDIC/ASCII tables) and generating a translate parameter list ($TRL). When you translate data, generate the interface to the translate routine ($TRAN).

Note: If you want to transmit or receive ASCII data, be sure to give the polling and addressing characters and station identification sequences in ASCII.

Initiating and Terminating the Transfer of Data

To initiate data transfer, you must issue the following requests:

- Get requests to receive data ($GETB)
- Put requests to transmit data ($PUTB).

The first get or put request causes BSC to establish line connection with the remote station. How the data transfer is ended depends on whether the System/36 is receiving data ($GETB) or transmitting data ($PUTB). If System/36 is transmitting, then stop sending the data to the current file by one of the following means:

- $PUTB with OPC-EOF. This transmits the last block of data ending with ETX. The System/36 then transmits EOT. In 3740 mode, the System/36 waits for the next user operation and then sends either STX ETX or EOT.
- $PUTB to another transmit file. This transmits the last block of data from the current file ending with ETX. System/36 sends EOT, and line initialization for the new file takes place. The block ends with ETB when in 3740 mode. In 3740 multiple file mode, STX ETX replaces the EOT.
- $GETB to a receive file. This transmits the same sequences as issuing a $PUTB to another transmit file.
- $CLOS to the current file. This transmits the last block of data ending with ETX and EOT (or DISC if switched lines). The last block ends with ETB when in 3740 mode. In the case of 3740 multiple file mode, use $CLOS to transmit EOT.

If the System/36 is receiving, the remote station initiates the end of data transmission. You can detect this by coding EOF on the $GET macroinstruction or by checking for hex 42 ($BSEOF) in the $BSCMP field of the BSC DTF after each $GETB request.

Issue successive $GETB requests until you detect EOF or an error. You can detect a BSC error by coding REJECT on the $GETB macroinstruction. The error code is returned in $BSCMP.

Using Move Mode

System/36 performs all BSC get and put requests in move mode. BSC moves data from the BSC I/O buffers to the logical buffer on get requests, and from the logical buffer to the BSC I/O buffers on put requests.

A single get or put request does not necessarily result in the actual data transmission over the communications line. For a get request, the remote station transmits data only when its BSC I/O buffer is filled.

A put request transmits data to the remote station only if the record to be moved to a BSC I/O buffer cannot be contained in the current I/O buffer. The first put request begins line initialization. Data transfer begins after the second put request, so your program is always at least one put request ahead of BSC.

Blank Truncation

System/36 BSC can transmit and receive data with the trailing blanks removed. For put files, BSC moves data from the logical buffer to the BSC I/O buffer with all trailing blanks removed. After each record, BSC inserts an IRS character.

For get files, BSC scans the data in the BSC I/O buffer for an IRS. BSC then moves all data up to the IRS character to the logical buffer and blanks the remainder of the logical buffer.

To use blank truncation, run the ALTERCOM procedure with the TRUNCATE parameter or the $SETCF utility with a SETR utility control statement with BLANK-T parameter specified before running the BSC program.

Be aware of the following:

- Blank truncation will not operate in ITB mode. You can specify blank truncation with transparent mode; however, the truncation will not be performed.

- When you use blank compression/expansion or blank truncation with blocked records, the number of records per block vary depending on the number of blanks in each record.

Blank Compression/Expansion

In order to use the line more effectively and decrease communications line costs, the System/36 BSC offers assembler users the capability of transmitting and receiving data with all contiguous blanks (groups of 2 or more blanks) removed. This is done by using the same format used by the IBM 3780.

For put files, BSC moves data from the logical buffer to the BSC I/O buffer with contiguous blanks removed and compression control characters inserted. After each record, BSC inserts an IRS.

If the record is to be printed from the logical buffer, it should be printed before a put because BSC alters the record with IGS characters and count characters while compressing the record.

For get files, the procedure is reversed as follows. The System/36 BSC removes compression control characters, inserts blanks removed at the remote station, recognizes the intermediate record separator and moves the record from the BSC I/O buffer to the logical buffer.

To use blank compression/expansion, either run an ALTERCOM procedure with the COMPRESS parameter before running the BSC program, or run a SETR utility control statement with BLANK-C specified.

When you use blank compression/expansion or blank truncation with blocked records, the number of records per block vary depending on the number of blanks in each record.

Note: You cannot use blank compression/expansion with transparent or ITB mode.

Data Formats

System/36 BSC support uses the following data formats for transmission of data. Use these formats when sending data to System/36 from a processing unit.

- Nontransparent, non-ITB:

  STX-data-ETX(ETB)

- Nontransparent, non-ITB, blocked:

  STX-rec 1/rec 2/.../rec n-1/rec n-ETX(ETB)

- Nontransparent, ITB:

  STX-data-ITB-data-ITB-data-ETX(ETB)

- Transparent, non-ITB:

  DLE-STX-data-DLE-ETX(ETB)

- Transparent, non-ITB, blocked:

  DLE-STX-rec 1/rec 2/.../rec n-1/rec n-DLE-ETX(ETB)

- Transparent, ITB (receive files only):

  DLE-STX-data-DLE-ITB-DLE-STX-data-DLE-ITB-DLE-STX-data-DLE-ETX(ETB)

Changing the BSC Environment

BSC configuration information is changed by the System/36 ALTERCOM or SETCOMM procedure. When you run BSC programs from the job queue, the configuration information from the system console is used for the job. The SSP gets this information at the same time the job is run. If you want to change the BSC environment when running from the JOBQ, first run ALTERCOM from the system console before starting your job.

The ALTERCOM procedure runs the $SETCF utility. Instead of using this procedure to change the BSC configuration, you can use the SETB and SETR utility control statements of the $SETCF utility. For information on coding System/36 procedure commands and utility control statements, see the *System Reference* manual.

Errors

If an error occurs at either the sending or receiving station, System/36 retries the operation the number of times specified by the $DTFB macroinstruction, or the number of retries specified by the ALTERCOM procedure command, or the SETB utility control statement. (See the *System Reference* manual for information on the SETB utility control statement and the ALTERCOM procedure.)

Note: Refer to the expansion within your program of the $DTFO macro for possible error codes (following label $BSCMP). These will appear only when the parameter BSC-Y is coded on the $DTFO macro.

Automatic Call Support

When System/36 is configured with the MLCA (multiline communications adapter) and the Autocall feature or the X.21 feature, remote locations can be called without operator intervention. Because there is no reference to the autocall or X.21 capabilities in user programs, existing programs can add autocall or X.21 without other modification. You specify autocall or X.21 by using the PHONE parameter on the COMM OCL statement. The COMM statement is described in the *System Reference* manual.

The phone list specified in the COMM statement can contain up to 120 phone numbers and is generated by the DEFINEPN or the DEFINX21 procedure described in the *System Reference* manual. When the first request during any BSC job step is made to BSC data management, the phone list is searched for a number to call. The first time the list is referred to, the search begins with the first number. For each succeeding reference, the search begins with the next available number. If a number cannot be reached, the value of the number of retries is reduced by one and the next number is called. If no numbers in the list can be reached, a no-line connection return code is passed to the user program. A message is displayed to the system console indicating each number that could not be reached. When a number is reached, a message is displayed indicating the number reached, and communication proceeds in the same manner as for a manual call line. When the job step ends, you can use the OCL statement with the LISTDONE parameter to perform the step again and call the next number. You can use the same phone list in a later step of the job.

If a batch BSC job is run on an autocall line and no phone list is specified in the COMM statement (or there is no COMM statement), the call mode defaults to the mode specified in the user's DTF or the display station communications configuration record. The mode can be manual answer, manual call, or automatic answer. If the phone list is specified in the COMM statement but the line is not an autocall line, or the autocall task was not loaded at IPL time, the line is considered to be a manual or automatic answer line, depending on the switch type defined for the line.

If a batch BSC job is run on a switched line under X.21 and no phone list is specified, switch type automatic answer is assumed. If the X.21 task is not active, an error message is displayed and the BSC program is not run. You must IPL the system to make the X.21 task active.

The ability to call multiple locations within a single BSC job step is useful primarily when the System/36 is receiving data from multiple locations. Because any number may be called during a request, transmission of data to a particular location should be performed using a phone list containing a single number.

If, during the receiving of data, a permanent error occurs, the phone number associated with the data link is not reset. Because the number is not reset, it cannot be called again on subsequent passes through the list. The recovery associated with that particular job step is the responsibility of the user.

Chapter 6. Assembler Problem Determination

If a problem occurs while you are using assembler, the cause of the problem may not be obvious. An error in your application or in system operation could have caused the problem. The problem determination procedure in this chapter can help you solve or circumvent the problem. If you need more information refer to the *System/36 System Problem Determination* manual, SC21-7919 for the 5360 System Unit, or to the *System/36 System Problem Determination* manual, SC21-9063 for the 5362 System Unit, or to chapter 13 in *System/36 Operating Your Computer - 5364*, SC21-9085 for the 5364 System Unit, before contacting your service representative.

How to Use this Procedure

This procedure is arranged in a sequence of questions that you can answer with a *Yes* or *No*. Based on your answer, you are directed to another question or to a recommendation for action.

Start at the beginning of the procedure and follow the question-and-answer sequence, answering each question to which you are directed based on your previous answer. If the problem is a condition that requires more detailed procedures, you are referred to those procedures.

Identifying Assembler Problems

When a assembler problem occurs, you can use the following series of questions to pinpoint its possible cause:

1 **Did you receive a message indicating that an operator needs to do something to a device such as a printer or a display station?**

No **Yes**

Take the actions indicated by the message and save any automatic dumps printed as a result of the message. If the action requires operator action, call your system operator. If the action requires you to call for help, see Contacting Your Service Representative on page 6-7.

When you examine a message for indicated actions, check the following:

- Second-level message text, which describes the message in more detail. To get the second-level message text press the Help key.
- Some messages contain a number of options for possible recovery actions. These options are explained in Chapter 1 of the *Assembler Messages Manual*, SC21-7942.

If you still cannot solve your problem after fully examining the message, see Contacting Your Service Representative on page 6-7.

2 **Are other system users having problems communicating with the system?**

No **Yes**

Call your system operator and describe the problem. Have your operator use the procedures in the appropriate *System/36 System Problem Determination* manual.

3 **Is this the first time you have ever run the job or subroutine?**

Yes **No**

You may have a system problem. Call your system operator, describe your problem, and have the operator use the appropriate *System/36 System Problem Determination* manual.

4 **Are you having a nonprogramming problem, such as spooled output that is not produced or a device that is not working?**

No **Yes**

You may have a system problem. Call your system operator and have the operator use the appropriate procedure in the appropriate *System/36 System Problem Determination* manual.

5 **Are you using the current release of SSP?**

Yes **No**

Install the current release of SSP.

6 **Have all IBM-supplied program changes you have received that apply to the current release of SSP been installed?**

Yes **No**

Install the program changes you have received that have not yet been applied.

7 **Are you using the current release of assembler? The release number is printed on the first line of the source listing for any assembler program.**

Yes No

Install the current release of assembler and compile or run the program again.

8 **Have all IBM-supplied program changes you have received that apply to the current release of assembler been installed? (Check with your system operator)**

Yes No

Install the program changes you have received that have not yet been applied and run the program again.

9 **Have any non-IBM changes been made to assembler or to SSP?**

No Yes

If assembler has been changed, install its current release and program changes, and run the program again. If SSP has been changed, install its current release and program changes.

10 **Have changes been made to the user program since the last time it ran successfully?**

No Yes

Read on, but consider what has been changed. For example: have operating procedures changed, has the data within the files changed, are new device files being used, or have program changes been applied recently? A good starting point for problem determination is a changed item.

Assembly Time Problems

11 **Was unexpected assembler output produced?**

No Yes

Check if:

- The NOLIST option was used. NOLIST specifies that the assembler is not to produce the assembler listing. Specify LIST to produce the complete assembler listing.
- The program has the NOGEN option. NOGEN suppresses the printing of statements generated by the macroprocessor. Specify GEN to print statements generated by the macroprcessor.
- The program has the PRINT OFF option. This option overrides The GEN option. Specify PRINT ON in your program.

12 **Does the program have poor performance during assemble-time?**

No **Yes**

Check if:

- There is space allocated for the work area. Examine the ASM procedure for allocation and enlarge the allocated space.
- The macro processor was called, but no macroinstructions were used. Examine the listing and specify NOMAC in the ASM procedure.
- There are required macroinstructions in #ASMLIB. Do a LISTLIBR OF #ASMLIB and move user written macroinstructions to #ASMLIB.
- If you cannot solve or circumvent the problem contact your service representative.

Link Time Problems

13 **Were errors encountered during linkage?**

No **Yes**

Check if:

- There are any coding errors that occurred during assembly. The assembler message should indicate what the error is. Correct and re-assemble the program until all the errors are corrected.
- The program has the NOOBJ option. NOOBJ specifies that the assembler is not to place the object (assembled) program in the library. Specify OBJ in your program to place the object (assembled) program in the library as a subroutine member.
- Refer to the *IBM System/36 Overlay Linkage Editor Guide*, SC21-9041 for other considerations.

Execution Time Problems

14 **Did errors occur when loading the program (via //LOAD OCL)?**

No **Yes**

Check if:

- The load module exists by specifying LISTLIBR of the load module.
- The object module (R module) is linked before executing.

15 **Was unexpected execution-time output produced?**

No **Yes**

Check if:

- There is incorrect program data.
- There are assembler coding errors.
- The program is in an infinite loop.

16 **Did a task dump occur?**

No **Yes**

- Follow the system prompt to get a listing of the dump. Examine the dump to find the cause of the problem. Go to the next question about messages.

17 **Did you get Message SYS—0015?**

No **Yes**

- If the message SYS—0015 appears, make sure your program does not try to execute data. The following example of a program demonstrates this.

```
          BH        LABEL
DATA      DC        XL2'0000'
          .
          .
          .
LABEL     EQU       *
```

If the branch does not take place, X'0000' is interpreted as an instruction, but 00 is an invalid main storage instruction.

- Make sure a valid instruction was not modified by the program. This often happens when the base registers XR1 and XR2 contain incorrect data. The following example of a program, demonstrates this.

```
location
                MOVE    MVI     0(,XR1),X'00'
                        .
                        .
                        .
  02A0                  J       LABEL
                        .
                        .
                        .
                LABEL   EQU     *
```

If location X'02A0' of the program is a JUMP instruction, but at the point of execution of the MVI instruction at label MOVE, XR1 contains X'02A0'. The MVI instruction has modified the Jump instruction to '00', but 00 is an invalid instruction.

18 **Did you get Message SYS–0013 or Message SYS–0014 ?**

No **Yes**

- If the messages SYS–0013 or SYS–0014 appear, the program tried to access an address outside the region size of the program. Check if the index registers contain correct values. The following example of a program demonstrates this.

```
                MVC     DATA1(2),0(,XR1)
                .
                .
                .
  DATA1         DC      XL2'0000'
```

If the program size is X'0400' bytes, but when executing the MVC instruction, XR1 contains X'0600', SYS–0013 will occur.

19 **Did a system message occur?**

No **Yes**

The message should provide some information. Carefully, check the usage of system macros, DTF's, device file, OCL, and return codes.

If you still cannot solve your problem after fully examining the message, your program and procedures, see Contacting Your Service Representative on page 6-7.

20 **Does the program have poor performance during execution-time?**

No **Yes**

Check if:

- There is over utilization of some system resources, for example disk usage. Your system operator can run the System Measurement Facility to find the utilization of system devices. Refer to the *System Measurement Facility Guide*, SC21-9025 to find the optimal configuration for your system.

If after using this procedure you or your system operator have not solved the problem, consult the appropriate *System/36 System Problem Determination* manual for your system unit before calling the service representative.

Contacting Your Service Representative

If you cannot solve a problem by the problem determination procedures listed in this chapter, and the appropriate *System/36 System Problem Determination* manual, you may want to contact your service representative. Before contacting your service representative, you will be asked to provide the following:

- For compile time problems:
  - A task dump at the time of the failure
  - Run the APAR procedure and include the entire history file
  - A diskette copy of the user program source and macro source
  - A diskette copy of the user procedure
  - An assembled source listing with cross-references
- For execution time problems, the above and the following:
  - A diskette copy of the user files
  - A diskette copy of the user display screens
  - A diskette copy of the user load module
  - A diskette copy of the history file immediately after the problem occurs

The procedures for obtaining the above information are explained in the appropriate *System/36 System Problem Determination* manual.

Appendix A. Programming Examples

This appendix contains assembler programming examples, macroinstruction definition examples, and related macroinstruction expansions.

BSC Programming Example

The following programming example illustrates the use of the BSC macroinstructions in assembler programs.

Transmit

This program reads a file from disk (BSCFIL) and transmits it to another System/36.

Note: The following BSC examples (A2 to A8) are only representative portions of larger programs; therefore, they are incomplete and should be used for illustrative purposes only.

IBM System/34, System/36, Assembler Coding Form — GX21-9279-1 Printed in U.S.A.

```
XMITL    START 0
*
*    ALLOCATE AND OPEN CHAIN OF DTF'S
*
         $ALOC DTF-BSCDTF                    ALLOCATE ALL DTF'S
         $OPEN DTF-BSCDTF                    OPEN ALL DTF'S
*
*         TRANSMIT THE FILE
*
LOOP1    EQU   *
         $GETD DTF-DSKDTF,IOERR-DSKERR,EOF-CLOSE,OP-NEXT GET A RECORD
         $PUTB DTF-BSCDTF,REJECT-BSCERR
         B     LOOP1                         LOOP UNTIL END OF FILE
*
*    PRINT RESULTS OF TRANSMISSION
*
DSKERR   EQU   *
         MVC   PTBUF+39(40),ERR1+39          DISK ERROR MESSAGE  (see A-8 for definition)
         J     CLOSE
BSCERR   EQU   *
         MVC   PTBUF+39(40),ERR2+39          BSC ERROR MESSAGE   (see A-8 for definition)
CLOSE    EQU   *
         $PUTP DTF-PRTDTF
         $CLOS DTF-BSCDTF
         $EOJ
*
*    DTF'S, BUFFERS, AND EQUATES
*
BSCDTF   $DTFB RECL-80,BLKL-80,RCAD-BUFFR1,FTYP-TSM,TYPE-MC,                C
               CHAIN-PRTDTF,RCVID-RCVX,RCVCT-4,SNDID-TRNX,SNDCT-4
BUFFR1   EQU   *
         DC    1XL80'00'
PRTDTF   $DTFP RCAD-PTBUF,IOAREA-PTIO,RECL-40,                              C
               PRINT-Y,CHAIN-DSKDTF,SPACEA-1,NAME-PRINT
DSKDTF   $DTFD ACCESS-CG,RECL-80,NAME-BSCFIL,INREC-BUFFR1
PTBUF    EQU   *
         DC    CL40'FILE SUCCESSFULLY TRANSMITTED' PRINT RECORD AREA
         END
```

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.

Receive Program

The following program receives data and prints that data:

IBM System/34, System/36, Assembler Coding Form — GX21-9279-1 Printed in U.S.A.

PROGRAM BSASM2 — PROGRAMMER — DATE — TYPING INSTRUCTIONS — GRAPHIC CHARACTER — PAGE 1 OF 3

```
RECV     START 0
*
*        ALLOCATE AND OPEN DTF'S
*
         $ALOC DTF-BSCDTF                ALLOCATE BOTH DTF'S
         $OPEN DTF-BSCDTF                OPEN BOTH DTF'S
*
*        RECEIVE AND PRINT FILE
*
LOOP1    EQU   *
         $GETB DTF-BSCDTF,EOF-CLOSE,REJECT-BSCERR GET A RECORD
         $PUTP DTF-PRTDTF,ERR-CLOSE      PRINT THE RECORD
         B     LOOP1                     LOOP UNTIL END OF FILE
*
*        IF ERROR PRINT MESSAGE
*
```

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.

IBM System/34, System/36, Assembler Coding Form — GX21-9279-1 Printed in U.S.A.

PROGRAM BSASM2 — PROGRAMMER — DATE — TYPING INSTRUCTIONS — GRAPHIC CHARACTER — PAGE 2 OF 3

```
BSCERR   EQU   *
         MVC   PTBUF+79(80),ERRMSG+79    BSC ERROR MESSAGE
         $PUTP DTF-PRTDTF,SKIPB-1        PRINT MESSAGE
*
*        CLOSE DTF'S AND END JOB
*
CLOSE    EQU   *
         $CLOS DTF-BSCDTF                CLOSE BOTH DTF'S
         $EOJ                            END OF JOB
*
*        DTF'S, BUFFER, AND EQUATES
*
BSCDTF   $DTFB RECL-80,BLKL-80,RCAD-PTBUF,FTYP-RCV,TYPE-MA,RCVID-RCVX,C
               RCVCT-4,SNDID-TRNX,SNDCT-4,CHAIN-PRTDTF
PRTDTF   $DTFP RCAD-PTBUF,RECL-80,PRINT-Y,SPACEA-1,NAME-PR
*
PTBUF    EQU   *
         DC    80CL1' '                  BSC/PRINT RECORD AREA
```

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.

IBM

IBM System/34, System/36, Assembler Coding Form

GX21-9279-1
Printed in U.S.A.

PROGRAM: BSASM2 | TYPING INSTRUCTIONS: GRAPHIC / CHARACTER | PAGE 3 | OF 3

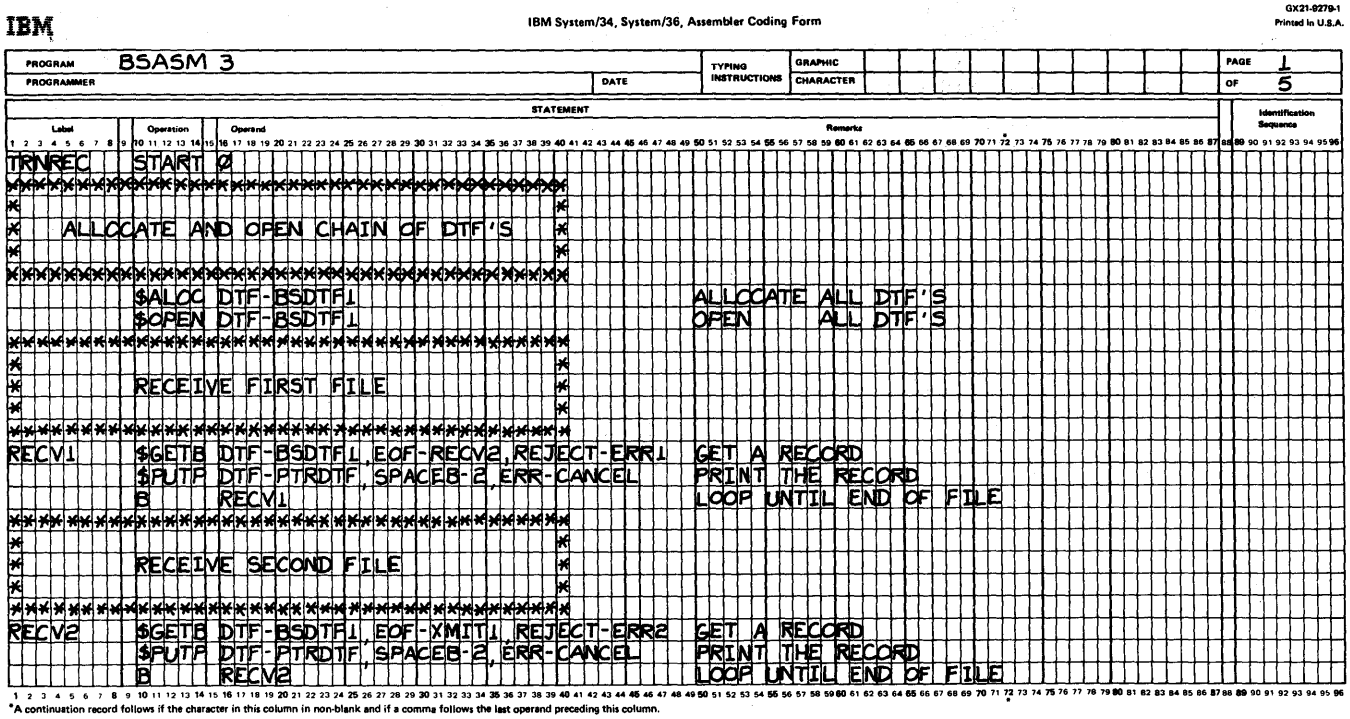
PROGRAMMER | DATE

STATEMENT				Identification Sequence
Label	Operation	Operand	Remarks	
ERRMSG	EQU	*		
	DC	CL80'BSC ERROR WHILE RECEIVING FILE'		
*				
TRNX	EQU	*		
	DC	CL4'XX01'		
RCVX	EQU	*		
	DC	CL4'XX02'		
	$DTF0	PRT-Y,BSC-Y,FIELD-Y	ALIGN ON 8-BYTE BOUND	
	END		PRINT I/O AREA	

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.

Transmit and Receive Program

This program receives two files from a 3741, then transmits two different files in return. The data is transmitted from two disk files (BSFIL1 and BSFIL2). The data received is printed. Before running this program, run the ALTERCOM procedure with MULTFILE specified.



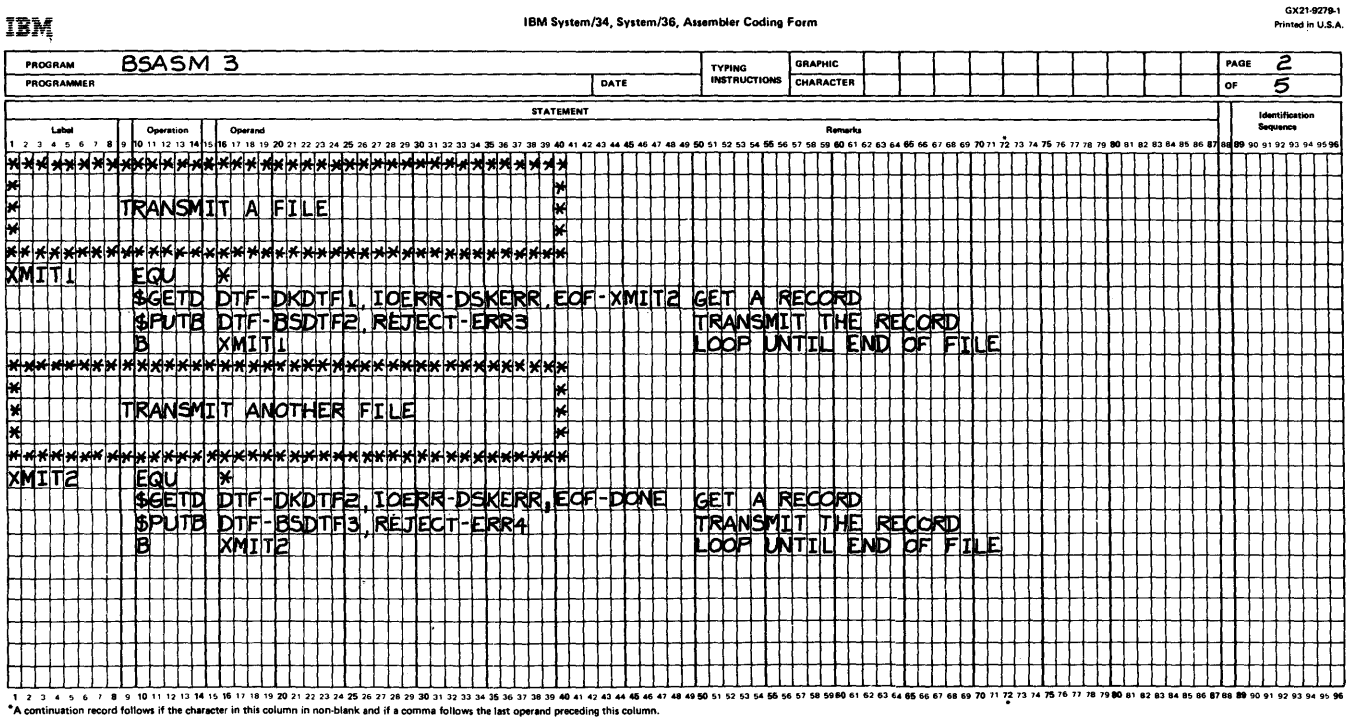
IBM System/34, System/36, Assembler Coding Form — GX21-9279-1 Printed in U.S.A.

PROGRAM: BSASM 3 — PROGRAMMER — DATE — TYPING INSTRUCTIONS — GRAPHIC CHARACTER — PAGE 1 OF 5

STATEMENT: Label | Operation | Operand | Remarks — Identification Sequence

```
TRNREC   START 0
*
*   ALLOCATE AND OPEN CHAIN OF DTF'S
*
         $ALOC DTF-BSDTF1                    ALLOCATE ALL DTF'S
         $OPEN DTF-BSDTF1                    OPEN     ALL DTF'S
*
*        RECEIVE FIRST FILE
*
RECV1    $GETB DTF-BSDTF1,EOF-RECV2,REJECT-ERR1  GET A RECORD
         $PUTP DTF-PTRDTF,SPACEB-2,ERR-CANCEL    PRINT THE RECORD
         B     RECV1                         LOOP UNTIL END OF FILE
*
*        RECEIVE SECOND FILE
*
RECV2    $GETB DTF-BSDTF1,EOF-XMIT1,REJECT-ERR2  GET A RECORD
         $PUTP DTF-PTRDTF,SPACEB-2,ERR-CANCEL    PRINT THE RECORD
         B     RECV2                         LOOP UNTIL END OF FILE
```

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.



IBM System/34, System/36, Assembler Coding Form — GX21-9279-1 Printed in U.S.A.

PROGRAM: BSASM 3 — PROGRAMMER — DATE — TYPING INSTRUCTIONS — GRAPHIC CHARACTER — PAGE 2 OF 5

STATEMENT: Label | Operation | Operand | Remarks — Identification Sequence

```
*
*        TRANSMIT A FILE
*
XMIT1    EQU   *
         $GETD DTF-DKDTF1,IOERR-DSKERR,EOF-XMIT2 GET A RECORD
         $PUTB DTF-BSDTF2,REJECT-ERR3            TRANSMIT THE RECORD
         B     XMIT1                         LOOP UNTIL END OF FILE
*
*        TRANSMIT ANOTHER FILE
*
XMIT2    EQU   *
         $GETD DTF-DKDTF2,IOERR-DSKERR,EOF-DONE  GET A RECORD
         $PUTB DTF-BSDTF3,REJECT-ERR4            TRANSMIT THE RECORD
         B     XMIT2                         LOOP UNTIL END OF FILE
```

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.

IBM

IBM System/34, System/36, Assembler Coding Form

GX21-9279-1
Printed in U.S.A.

PROGRAM: BSASM 3 | PROGRAMMER | DATE | TYPING INSTRUCTIONS | GRAPHIC | CHARACTER | PAGE 3 | OF 5

STATEMENT

Label	Operation	Operand	Remarks
*			
*	PERFORM EOJ AND ERROR PROCESSING		*
*			*
DONE	EQU	*	
	MVC	PTBUF+39(40),EØJMSG+39	MOVE GOOD MESSAGE
	J	PRINT	JUMP TO PRINT
ERR1	EQU	*	
	MVC	PTBUF+39(40),EBUF1+39	BSC ERROR RECV FILE 1
	J	PRINT	JUMP TO PRINT
ERR2	EQU	*	
	MVC	PTBUF+39(40),EBUF2+39	BSC ERROR RECV FILE 2
	J	PRINT	
ERR3	EQU	*	
	MVC	PTBUF+39(40),EBUF3+39	BSC ERROR XMIT FILE 1
	J	PRINT	JUMP TO PRINT
ERR4	EQU	*	
	MVC	PTBUF+39(40),EBUF4+39	BSC ERROR XMIT FILE 2
	J	PRINT	JUMP TO PRINT
DSKERR	EQU	*	
	MVC	PTBUF+39(40),DKMSG+39	DISK ERROR MESSAGE
PRINT	EQU	*	
	$PUTP	DTF-PTRDTF,SPACEA-2	PRINT MESSAGE

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.

IBM

IBM System/34, System/36, Assembler Coding Form

GX21-9279-1
Printed in U.S.A.

PROGRAM: BSASM 3 | PROGRAMMER | DATE | TYPING INSTRUCTIONS | GRAPHIC | CHARACTER | PAGE 4 | OF 5

STATEMENT

Label	Operation	Operand	Remarks
*			
*	CLOSE DTF'S AND END JOB		*
*			*
CANCEL	$CLOS	DTF-BSDTF1	CLOSE ALL DTF'S
	$EOJ		END OF JOB
*			
*	DTF'S, BUFFERS, AND EQUATES		*
*			*
BSDTF1	$DTFB	RECL-40,BLKL-40,FTYP-RCV,TYPE-MA,RCAD-PTBUF,CHAIN-BSDTF2	
BSDTF2	$DTFB	RECL-40,BLKL-40,FTYP-TSM,TYPE-MA,RCAD-DKBUF,CHAIN-BSDTF3	
BSDTF3	$DTFB	RECL-40,BLKL-40,FTYP-TSM,TYPE-MA,RCAD-DKBUF,CHAIN-DKDTF1	
DKDTF1	$DTFD	ACCESS-CG,RECL-40,NAME-BSFIL1,IOAREA-DKIO,CHAIN-DKDTF2	
DKDTF2	$DTFD	ACCESS-CG,RECL-40,NAME-BSFIL2,IOAREA-DKIO,CHAIN-PTRDTF	
PTRDTF	$DTFP	RCAD-PTBUF,PRINT-Y,RECL-40,NAME-PRINT	
*			
DKBUF	EQU	*	
	DC	CL40' '	

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.

PROGRAM BSASM 3 | TYPING INSTRUCTIONS | GRAPHIC CHARACTER | PAGE 5
PROGRAMMER | DATE | OF 5

Label	Operation	Operand	Remarks	Identification Sequence
PTBUF	EQU	*		
	DC	CL40' '		
EBUF1	EQU	*		
	DC	CL40'ERROR WHILE RECEIVING FIRST FILE'		
EBUF2	EQU	*		
	DC	CL40'ERROR WHILE RECEIVING SECOND FILE'		
EBUF3	EQU	*		
	DC	CL40'ERROR WHILE TRANSMITTING FIRST FILE'		
EBUF4	EQU	*		
	DC	CL40'ERROR WHILE TRANSMITTING SECOND FILE'		
EOJMSG	EQU	*		
	DC	CL40'JOB SUCCESSFULLY COMPLETED'		
DKMSG	EQU	*		
	DC	CL40'DISK ERROR WHILE GETTING A RECORD'		
*				
	$DTF0	DISK-Y,BSC-Y,FIELD-Y,PRT-Y		
	END			

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.

System Date/Time Program

```
DTIM     START 0
*************************************************************************
*  PROGRAM: DTIM  -  PRINT THE SYSTEM DATE/TIME                         *
*  DESC   : THE PROGRAM USES THE MACRO $TOD TO ACCESS THE SYSTEM        *
*           DATE AND TIME, IT PRINTS THEM IN THE FORMAT                 *
*           TIME =  HH.MM                                               *
*           DATE =  MM/DD/YY                                            *
*  INPUT  : SYSTEM DATE AND TIME                                        *
*  OUTPUT : PRINT DATE AND TIME                                         *
*************************************************************************
*************************************************************************
*       ALLOCATE THE PRINTER FILE                                       *
*************************************************************************
*
         $ALOC DTF-PRT                        ALLOCATE THE PRINTER FILE
         EJECT
*
*************************************************************************
*        OPEN THE PRINTER FILE                                          *
*************************************************************************
*
         $OPEN DTF-PRT                        OPEN THE PRINTER FILE
         EJECT
*
```

```
*****************************************************************************
*       GET THE TIME/DATE AND PRINT THEM                                    *
*****************************************************************************
*
         $TOD  TRB-TIMDAT                      CALL MACRO
         MVC   PTIME(2),$TRBTIME-2(,XR2)       GET MINUTES
         MVC   PTIME-3(2),$TRBTIME-4(,XR2)     GET HOURS
         MVC   PDATE(2),$TRBDATE(,XR2)         GET YEAR
         MVC   PDATE-3(2),$TRBDATE-2(,XR2)     GET DAY
         MVC   PDATE-6(2),$TRBDATE-4(,XR2)     GET MONTH
*
         MVC   PRTBUFL+5(6),DTIME              MOVE TIME DESC TO PRINTER BUF
         MVC   PRTBUFR(5),PTIME                MOVE THE TIME TO PRINTER BUFF
         $PUTP DTF-PRT                         PRINT THE TIME
         MVC   PRTBUFL+5(6),DDATE              MOVE DATE DESC TO PRINTER BUF
         MVC   PRTBUFR(8),PDATE                MOVE THE DATE TO PRINTER BUFF
         $PUTP DTF-PRT                         PRINT THE DATE
         EJECT
*
*****************************************************************************
*       CLOSE THE PRINTER FILE AND GO TO END OF JOB                         *
*****************************************************************************
*
         $CLOS DTF-PRT                         CLOSE THE PRINTER FILE
         $EOJ
*
*****************************************************************************
*       DEFINE THE DATA AREAS                                               *
*****************************************************************************
*
PRT      $DTFP NAME-PRTFILE,RCAD-PRTBUFL,IOAREA-PRTIO,RECL-20,SPACEB-1
```

```
*
PRTBUFL  EQU   *                          PRINTER BUFFER
PRTBUFR  DC    XL20'00'                   PRINTER BUFFER INITIALIZED
*
PRTIO    EQU   *                          PRINTER INPUT/OUTPUT
         DC    XL20'00'                   PRINTER INPUT/OUTPUT INITIALIZED
*
DTIME    DC    CL6'TIME ='                TIME DESCRIPTION
DDATE    DC    CL6'DATE ='                DATE DESCRIPTION
PTIME    DC    CL5'  .  '                 TIME FIELD
PDATE    DC    CL8'  /  /  '              DATE FIELD
*
XR1      EQU   1                          INDEX REGISTER 1
XR2      EQU   2                          INDEX REGISTER 2
         EJECT
*
TIMDAT   $TRB  V-ALL                      MACRO FOR TIME/DATE REQUEST BLOC
         EJECT
*
         $DTFO PRT-Y                      GENERATE DTF OFF-SETS
         END
```

Workstation and Print Program

```
WSASM    START 0                  SET LOCATION COUNTER VALUE
         PRINT NOGEN
*************************************************************************
* PROGRAM: WSASM - WORK STATION OPERATION                                *
* DESC   : THIS PROGRAM ASSUMES THE EXISTENCE OF A DISPLAY FORMAT  *
*          'FMTNM' IN A FORMAT LOAD MEMBER 'WSFMT'.                      *
*          THE PROGRAM ISSUES A 'PUT AND GET' OPERATION TO THE     *
*          WORK STATION MANAGEMENT WHICH PUTS OUT A DISPLAY SCREEN *
*          AND PASSES THE INPUT DATA FROM THE SCREEN TO THE        *
*          PROGRAM.  THE PROGRAM WILL THEN PRINT OUT THE SCREEN    *
*          INPUT.  THE ABOVE PROCESS CONTINUES UNTIL THE WORK      *
*          STATION OPERATOR INDICATES SO ON THE DISPLAY SCREEN.    *
* INPUT  : THERE ARE FOUR INPUT FIELDS FROM THE DISPLAY FORMAT     *
*                AN EOJ INDICATOR - 1 BYTE                         *
*                                   'Y' IF END OF JOB IS DESIRED   *
*                NAME FIELD       - 3 BYTES                        *
*                STREET FIELD     - 19 BYTES                       *
*                CITY FIELD       - 20 BYTES                       *
* OUTPUT : THE NAME FIELD, STREET FIELD, AND CITY FIELD FROM THE   *
*          SCREEN INPUT WILL BE PRINTED.                           *
* ENTRY  : DISPLAY FORMAT 'FMTNM' HAVE BEEN CREATED AND COMPILED.  *
* EXIT   : NORMAL                                                  *
*************************************************************************
*                    CONSTANTS, BUFFER, AND EQUATES                     *
*************************************************************************
```

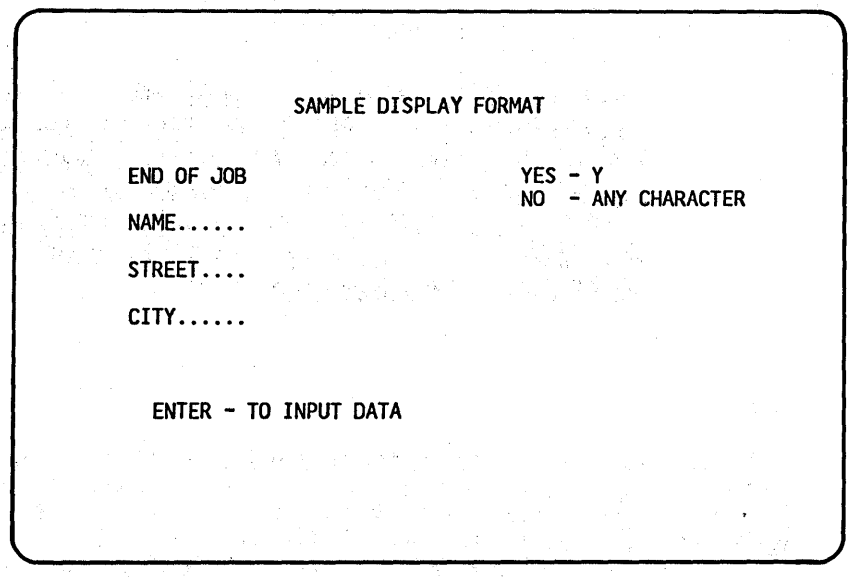
```
         SPACE 1
XR2      EQU   2                 INDEX REGISTER 2
         SPACE 2
*                ************************************
*                *         DTF DISPLACEMENTS        *
*                ************************************
         SPACE 1
         $DTFO PRT-YES,          DISPLACEMENTS FOR PRINTER            C
               WS-YES            DISPLACEMENTS FOR WORK STATIONS
         EJECT
*                ************************************
*                *           PRINTER DTF            *
*                ************************************
         SPACE 1
PRTDTF   EQU    *                  ADDR OF LEFMOST BYTE OF PRT DTF
         $DTFP RCAD-PRTBUF,         ADDRESS OF LOGICAL BUFFER          C
               IOAREA-PRTAREA,      ADDRESS OF PHYSICAL BUFFER         C
               NAME-PRTFILE,        NAME OF PRINT FILE                 C
               CHAIN-WSDTF,         POINTER TO WORK STATION DTF        C
               RECL-70              RECORD LENGTH
         SPACE 2
*                ************************************
*                *        LOGICAL PRINT BUFFER        *
*                ************************************
         SPACE 1
         ORG   *,8                SET LOCATION COUNTER TO 8 BYTE BOUNDARY
PRTBUF   EQU   *                  POINTER TO LEFT BYTE OF PRT BUFFER
         DS    CL10               BUFFER POSITIONAL PADDING
PRTNM    DS    CL3                NAME FIELD
         DS    CL7                POSITIONAL PADDING
PRTST    DS    CL19               STREET FIELD
```

```
         DS    CL11              POSITIONAL PADDING
PRTCT    DS    CL20              CITY/STATE FIELD
         SPACE 2
*                ************************************
*                *      PHYSICAL PRINT BUFFER       *
*                ************************************
         SPACE 1
PRTAREA  EQU   *                 LEFT ADDRESS OF PHYSICAL PRINT BUFFER
         DS    CL70              PHYSICAL PRINT BUFFER
         DS    CL19              + ROOM FRO IOB
         EJECT
*                ************************************
*                *                                  *
*                *           WORK STATION           *
*                *               DTF                *
*                *                                  *
*                ************************************
         SPACE 1
WSDTF    EQU   *                 WORK STATION DTF
         $DTFW MEMBER-WSFMT,     FORMAT LOAD MEMBER NAME                C
               INLEN-43          TOTAL LENGTH OF ALL INPUT FIELDS
         SPACE 2
*                ************************************
*                *                                  *
*                *    WORK STATION INDEX AREA AND   *
*                *          LOGICAL BUFFERS         *
*                *                                  *
*                ************************************
         SPACE 1
WSINDX   EQU   *                 WORK STATION INDEX AREA
         DS    CL16              EACH FORMAT REQUIRES 16 BYTES
```

```
         SPACE 2
         ORG   *,8                 LOC CTR SET TO 8 BYTE BOUND FOR GET OP
WSLBUF   EQU   *                   LEFT ADDR WORK STATION LOGICAL BUFFER
WSIND    DS    CL1                 WS OPERATOR END OF JOB INDICATOR
NAME     DS    CL3                 NAME INFORMATION STORAGE
STREET   DS    CL19                ADDRESS INFORMATION STORAGE
CITY     DS    CL20                ADDRESS INFORMATION STORAGE
         EJECT
*************************************************************************
*                         MAINLINE ROUTINE                              *
*************************************************************************
         SPACE 1
START    $ALOC DTF-PRTDTF          ALLOCATE PRINTER
         $OPEN DTF-PRTDTF          OPEN PRINTER
NXTREC   B     GETWS               GET RECORD FROM WORK STATION
         B     PRINT               GO MOVE DATA TO PRINT BUFFER
         CLI   EOJIND,X'01'        PROGRAM END OF JOB INDICSATOR ON?
         BNE   NXTREC              NO - GO PROCESS NEXT RECORD
         $CLOS DTF-PRTDTF          CLOSE PRINTER
         $EOJ                      GO TO END OF JOB
         SPACE 2
*************************************************************************
*                    ROUTINE 1 - PRINT ROUTINE                          *
*************************************************************************
         SPACE 1
PRINT    ST    RETURN1+3,ARR       STORE RETURN ADDR
         USING PRTBUF,XR2          SYMBOLVALUE TO USE IN DISP CALC
         LA    PRTBUF,XR2          LOAD @ PRT BUFFER INTO BASE REGISTER
         B     CLRBUF              GO CLEAR PRINT BUFFER
         MVC   PRTNM(3,XR2),NAME     MOVE NAME TO NAME FIELD
         MVC   PRTNM(19,XR2),STREET  MOVE ADDRESS TO STREET FIELD
```

```
         MVC   PRTCT(20,XR2),CITY    MOVE CT/STATE TO CITY FIELD
         DROP  XR2                 STOP BASE DISP CALCULATION
         $PUTP DTF-PRTDTF,         POINT TO PRINTER DTF                  C
               SKIPB-20,           SKIP TO LINE 20 BEFORE PRINTING       C
               SPACEA-2            SPACE TWO SPACES AFTER PRINTING
RETURN1  B     *-*                 RETURN TO CALLER'S NSI
         SPACE 2
*************************************************************************
*                    ROUTINE 2 - CLEAR BUFFER                           *
*************************************************************************
         SPACE 1
ARR      EQU   8                   ADDRESS RECALL REGISTER VALUE
CLRBUF   ST    RETURN2+3,ARR       SAVE CALLER'S RETURN ADDRESS
         MVI   PRTCT,BLANK         PUT BLANK IN RTMOST POS OF BUFFER
         MVC   PRTCT-1,PRTCT(69)   PROPAGATE THRU REST OF BUFFER
RETURN2  B     *-*                 RETURN TO CALLER'S NSI
BLANK    EQU   X'40'
         EJECT
*              ****************************************
*              *                                      *
*              *   PUT AND GET FROM DISPLAY STATION   *
*              *   CALL TO WORK STATION MANAGEMENT    *
*              *                                      *
*              ****************************************
         SPACE 1
GETWS    ST    RETURN3+3,ARR       STORE RETURN ADDR
         $WSIO DTF-WSDTF,          ADDRESS OF LEFT BYTE OF WS DTF        C
               OPC-PTG,            PUT UP FORMAT AND GET RECORD          C
               INLEN-43,           MAX AMOUNT OF DATA FROM WS            C
               RCAD-WSLBUF,        LEFTMOST ADDR OF WS BUFFER            C
               FORMAT-FMTNM        FORMAT NAME IN LOAD MEMBER
         CLI   WSIND,C'Y'          END OF JOB IND BY OPERATOR?
         BNE   RETURN3             NO - GO PROCESS THE NEXT RECORD
         SBN   EOJIND,B'00000001'  YES - SET PROG END OF JOB INDICATOR
RETURN3  B     *-*                 RETURN TO CALLER'S NSI
EOJIND   DS    BL1                 PROGRAM END OF JOB INDICATOR
         END   START               PROGRAM ENTRY POINT
```

The program uses the following display format:



This display was created from the following format listing:

```
WSFMT     S   FMT     11/09/84  14.38  000003

          SFMTNM
          DFL0001     21 630Y                          CSAMPLE DISPLAY FORMAT
          DFL0002     10 916Y                          CEND OF JOB
          DFL0003      1 929   Y
          DFL0004      7 949Y                          CYES - Y
          DFL0005     191049Y                          CNO  - ANY CHARACTER
          DFL0006     101216Y                          CNAME......
          DFL0007      31229   Y           Y
          DFL0008     101416Y                          CSTREET....
          DFL0009     191429   Y
          DFL0010     101616Y                          CCITY......
          DFL0011     201629   Y
          DFL0012     212018Y                          CENTER - TO INPUT DATA
```

Alternative Index and Noncontiguous Keys Program

This is a sample program that illustrates some of the features of disk support. This program does a keyed access to retrieve a record by key from a file that has a 39 byte noncontiguous key. The program assumes the existence of a file that has three noncontiguous keys. This file was created as a sequential file by a COBOL program and had an alternative index built using the BLDINDEX procedure. For more information on the BLDINDEX procedure, please refer to the *System Reference Manual*, SC21-9020.

```
KEYASM   START 0
*
*************************************************************************
* PROGRAM: KEYASM - KEYED ACCESS NONCONTIGUOUS AND GREATER THAN 29 BYTES   *
* DESC   : THIS PROGRAM ASSUMES THE EXISTENCE OF AN INDEXED FILE WITH AN   *
*          ALTERNATIVE INDEX WITH NONCONTIGUOUS KEYS.                      *
*          THIS PROGRAM DOES AN INDEXED GET USING THE GENERALIZED ACCESS   *
*          METHOD (GAM) WITH A KEY                                         *
*          'SMITH    ' + 'LUMBERJACK          ' + 'IC' + '1234567'         *
*          (SMITH    LUMBERJACK          ', 'IC' AND '1234567' ARE THE 3   *
*          KEYS)                                                           *
*          IF THE RECORD IS FOUND, IT IS PRINTED.                          *
* INPUT  : INDEX FILE                                                      *
*          ALTERNATIVE INDEX FILE WITH NONCONTIGUOUS KEY                   *
*          FORMAT:                                                         *
```

```
*                                                                               *
*                                     INITIALS                                  *
*                                       |                                       *
*          1111111122222222222222222234456666666     LAST NAME  (KEY1)         *
*           NAME          OCCUPATION   |  | NUMBER     OCCUPATION (KEY1)         *
*                                     BLANKS           BLANK                     *
*                                                      INITIALS   (KEY2)         *
*          ============================              BLANK                     *
*                   KEY 1                            NUMBER     (KEY3)         *
*                                       ==                                      *
*                                     KEY 2                                     *
*                                          =======                              *
*                                           KEY 3                               *
*                                                                               *
* OUTPUT : PRINT THE RECORD                                                     *
* ENTRY  : PROCEDURE THAT CREATES THE INDEX FILE THEN BUILDS THE INDICES        *
* EXIT   : NORMAL                                                               *
*           ERROR - THE KEY IS NOT FOUND                                        *
*********************************************************************************
*********************************************************************************
*       ALLOCATE THE DISK AND PRINTER FILES                                     *
*********************************************************************************
*                                                                               *
         $ALOC DTF-IND                        ALLOCATE THE DISK FILE
         $ALOC DTF-PRT                        ALLOCATE THE PRINTER FILE
         EJECT
*
*********************************************************************************
*        OPEN THE DISK AND PRINTER FILES                                        *
*********************************************************************************
*
         $OPEN DTF-IND                        OPEN THE DISK FILE
         $OPEN DTF-PRT                        OPEN THE PRINTER FILE
         EJECT
*
```

```
*****************************************************************************
*       GET THE RECORD FROM THE INDEXED FILE                                *
*****************************************************************************
*
         MVC   PRTBUFR(80),IMSG              INFO MSG:  INDEXED FILE
         $PUTP DTF-PRT                       PRINT THE MSG
         $GETD DTF-IND,OP-KEY,NRF-NOFOUND    GET THE RECORD
         MVC   PRTBUFR(39),IDSKBUFR          MOVE THE RECORD TO PRINTER BUF
         $PUTP DTF-PRT                       PRINT THE RETRIEVED RECORD
         MVC   PRTBUFR(80),SMSG              INFO MSG:  TEST SUCESSFUL
         $PUTP DTF-PRT                       PRINT THE MSG
         EJECT
*****************************************************************************
*       CLOSE DISK AND PRINTER FILES AND GO TO END OF JOB                   *
*****************************************************************************
*
EOJ      $CLOS DTF-IND                       CLOSE THE DISK FILE
         $CLOS DTF-PRT                       CLOSE THE PRINTER FILE
         $EOJ
*
*****************************************************************************
*       IF THE RECORD WAS NOT FOUND, PRINT THE "NOT FOUND" MESSAGE AND END  *
*****************************************************************************
*
NOFOUND  MVC   PRTBUFR(80),NMSG              INFO MSG: TEST FAILED
         $PUTP DTF-PRT                       PRINT THE MESSAGE
         J     EOJ
         EJECT
*
```

```
*****************************************************************************
*      DATA AREAS                                                           *
*****************************************************************************
*      DISK DTF - NOTE THAT THE RECORD LENGTH IS 39 BYTES.  THE KEY DIS-    *
*                 PLACEMENT (KDISP) IS X'FFFF'.  THIS KDISP VALUE IS A      *
*                 REQUIREMENT TO TELL THE SYSTEM THAT THIS FILE HAS         *
*                 NONCONTIGUOUS KEYS.  ALSO, THE KEY FOR THE DESIRED        *
*                 RECORD IS PASSED IN DATA AREA INDKEY.  NOTE THAT THE      *
*                 KEY IS PASSED AS IF IT WAS A CONTIGUOUS FIELD WHEN        *
*                 THE FILE CONTAINS BLANKS BEFORE AND AFTER 'IC'.           *
*****************************************************************************
*
IND      $DTFD NAME-KEYNCK,ACCESS-GAM,KEY-0,RECL-39,                    X
               INREC-IDSKBUFL,IOMSG-Y,KEYL-37,KDISP-NCKEY,ORDER-KEY
*
INDKEY   DC    CL8'SMITH'                 INITIALIZE KEY - NAME
         DC    CL20'LUMBERJACK          ' INITIALIZE KEY - JOB
         DC    CL9'IC1234567'             INITIALIZE KEY - INITIALS, #
NCKEY    EQU   X'FFFF'                    KEY DISPLACEMENT
*
PRT      $DTFP NAME-PRTFILE,RCAD-PRTBUFL,IOAREA-PRTIO,RECL-80,SPACEB-1
*
IDSKBUFL EQU   *                            RECORD BUFFER
IDSKBUFR DC    XL39'00'                     INITIALIZE RECORD BUFFER
*
PRTBUFL  EQU   *                            PRINTER BUFFER
PRTBUFR  DC    XL80'00'                     INITIALIZE PRINTER BUFFER
*
PRTIO    EQU   *                            PRINTER INPUT/OUTPUT AREA
         DC    XL80'00'                     INITIALIZE I/O AREA
*****************************************************************************
*        PRINTED MESSAGES                                                   *
*****************************************************************************
IMSG     DC    CL80'INDEXED FILE -- NONCONTIGUOUS KEY'
NMSG     DC    CL80'TEST FAILED!!                  RECORD NOT FOUND'
SMSG     DC    CL80'TEST SUCCESSFUL!!                     RECORD FOUND'
         EJECT
*
         $DTFO DISK-Y,PRT-Y                 GENERATE THE DTF OFFSETS
         END
```

Appendix B. Character Sets

The coded character set for EBCDIC (extended binary coded decimal interchange code) and ASCII (American National Standard Code for Information Interchange) in the following tables.

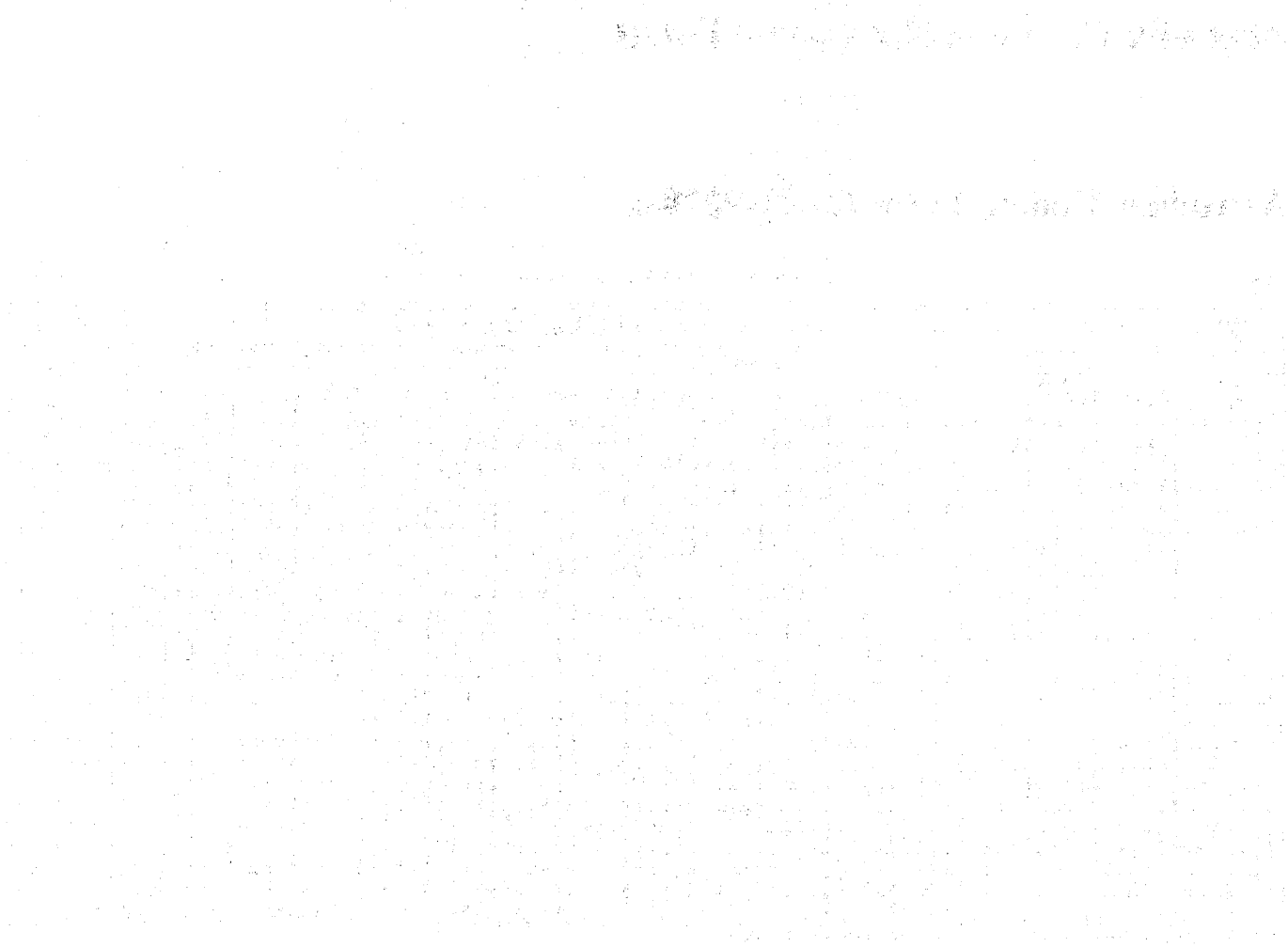
EBCDIC

Main Storage Bit Positions 4, 5, 6, 7	Main Storage Bit Positions 0, 1, 2, 3															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		ð	&	-						{	}	\	0
1	SOH	DC1	SOS				/		a	j	~		A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	DC3	TM / DC3						c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	EOB / ETB	UC					f	o	w		F	O	W	6
7	DEL	IL	PRE / ESC	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9	RLF	EM							i	r	z		I	R	Z	9
A	SMM	CC	SM		¢	!	¦									LVM
B	VT	CU1	CU2	CU3		$	,	#								
C	FF	IFS		DC4	<	*	%	@								
D	CR	IGS	ENQ	NAK	(	)	_	'								
E	SO	IRS	ACK		+	;	>									
F	SI	IUS	BEL	SUB	[	]	?	"								E0

/ Duplicate Assignment

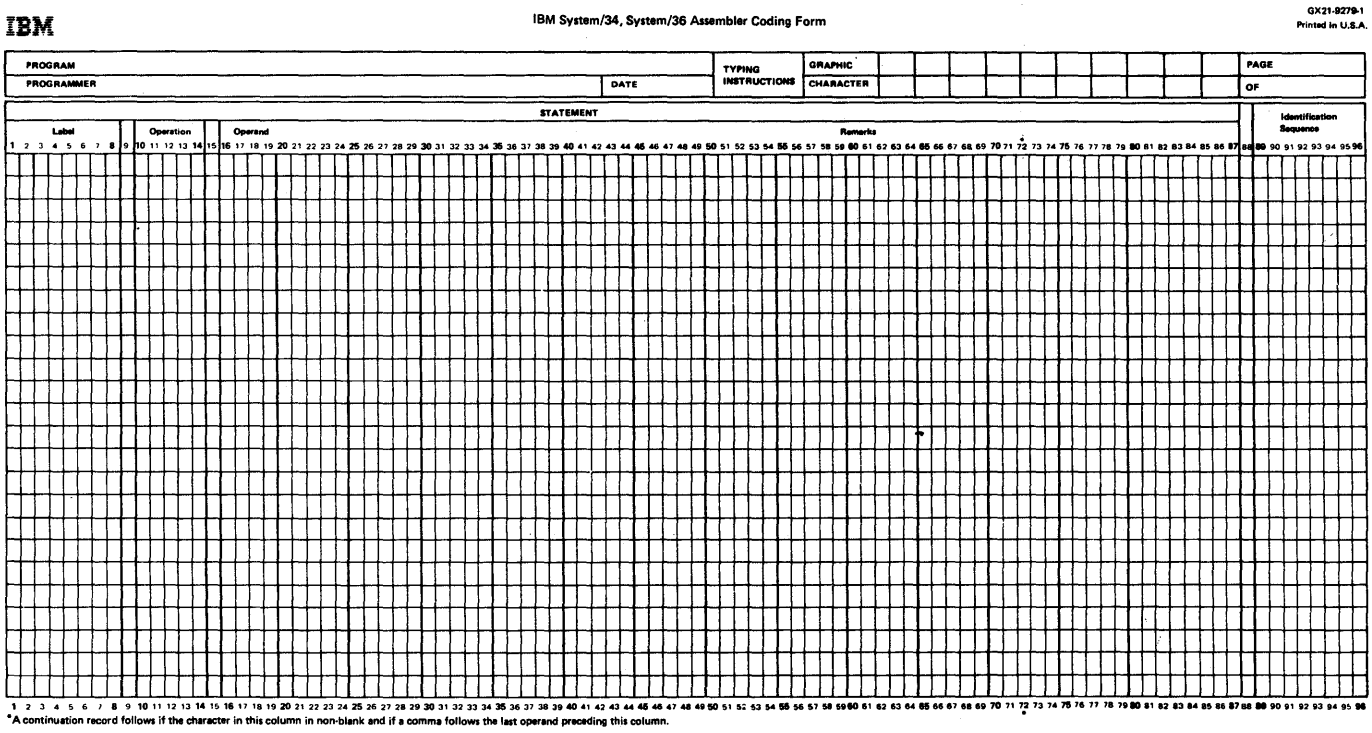
ASCII

Main Storage Bit Positions 4, 5, 6, 7	Main Storage Bit Positions 0, 1, 2, 3															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P		p								
1	SOH	DC1	!	1	A	Q	a	q								
2	STX	DC2	"	2	B	R	b	r								
3	ETX	DC3	#	3	C	S	c	s								
4	EOT	DC4	$	4	D	T	d	t								
5	ENQ	NAK	%	5	E	U	e	u								
6	ACK	SYN	&	6	F	V	f	v								
7	BEL	ETB	'	7	G	W	g	w								
8	BS	CAN	(	8	H	X	h	x								
9	HT	EM	)	9	I	Y	i	y								
A	LF	SUB	*	:	J	Z	j	z								
B	VT	ESC	+	;	K	[	k	{								
C	FF	FS	,	<	L	\	l	¦								
D	CR	GS	-	=	M	]	m	}								
E	SO	RS	.	>	N	¬	n	~								
F	SI	US	/	?	O	_	o	DEL								



Appendix C. Assembler Coding Forms

Assembler Coding Form GX21-9279-2


IBM

IBM System/34, System/36 Assembler Coding Form

GX21-9279-1
Printed in U.S.A.

PROGRAM
PROGRAMMER
DATE
TYPING INSTRUCTIONS
GRAPHIC
CHARACTER
PAGE
OF

STATEMENT

Label | Operation | Operand | Remarks | Identification Sequence

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96

*A continuation record follows if the character in this column in non-blank and if a comma follows the last operand preceding this column.

Notes:

Appendix D. Assembler Machine Instruction Formats

Assembler Instruction Formats

Mnemonic	Function	Op Code											
		Two-Address Instructions (Operand 1/Operand 2)								One Address			
		Direct/ Direct	Direct/ XR1	Direct/ XR2	XR1/ Direct	XR1/ XR1	XR1/ XR2	XR2/ Direct	XR2/ XR1	XR2/ XR2	Direct	XR1	XR2	Control
A	Add to register										36	76	B6	
ALC	Add logical character	OE	1E	2E	4E	5E	6E	8E	9E	AE				
AZ	Add zoned decimal	06	16	26	46	56	66	86	96	A6				
BC	Branch on condition										CO	DO	EO	
BC	Branch on ARR													FO
CLC	Compare logical character	0D	1D	2D	4D	5D	6D	8D	9D	AD				
CLI	Compare logical immediate										3D	7D	BD	
ED	Edit	0A	1A	2A	4A	5A	6A	8A	9A	AA				
ITC	Insert and test characters	0B	1B	2B	4B	5B	6B	8B	9B	AB				
JB	Jump backward													F1
JC	Jump on condition													F2
L	Load register										35	75	B5	
LA	Load address										C2	D2	E2	
LPMR	Load program mode register													F6
MVC	Move characters	0C	1C	2C	4C	5C	6C	8C	9C	AC				
MVI	Move logical immediate										3C	7C	BC	
MVX	Move hex character	08	18	28	48	58	68	88	98	A8				
S	Subtract from register										37	77	B7	
SBF	Set bits off masked										3B	7B	BB	
SBN	Set bits on masked										3A	7A	BA	
SLC	Subtract logical character	0F	1F	2F	4F	5F	6F	8F	9F	AF				
SLI	Subtract logical immediate										3F	7F	BF	
SRC	Shift right character										3E	7E	BE	
ST	Store register										34	74	B4	
SVC	Supervisor call													F4
SZ	Subtract zoned decimal	07	17	27	47	57	67	87	97	A7				
TBF	Test bits off masked										39	79	B9	
TBN	Test bits on masked										38	78	B8	
XFER	Transfer control													F5
ZAZ	Zero and add zoned	04	14	24	44	54	64	84	94	A4				

Notes:

Appendix E. Disk Data Management Considerations

Access Methods

Figure E-1 lists the actions caused by Allocate and Open when the various access methods are used to access the three types of files. The following situations are covered on the chart:

- The combination of the file type and the access method is allowed. These situations are indicated by a blank entry in the chart.
- The combination of the file type and the access method is not allowed either by allocate or by open. For these situations, the issuer and the message number of the message issued are given in the chart.
- In several situations, a load-to-old will occur to the file. Load-to-old includes the following:
  - The contents of the old file are destroyed.
  - A new file is created using the current file's location and space.
  - The new file's type is determined by the access method and other parameters specified in the DTF.

For these situations, load-to-old and the type of file created – sequential, direct, or indexed – are given in the chart.

Note: Please refer to the *Distributed Data Management Guide*, SC21-8011 for remote file considerations when using Assembler.

SEQUENTIAL FILES

	DISP-OLD	DISP-NEW	DISP-SHRRR	DISP-SHRRM	DISP-SHRMR	DISP-SHRMM or DISP-SHR	DISP Not Specified, Existing File	DISP Not Specified, New File
CG Access		Sequential File Created						Aloc 1356
CU Access		Sequential File Created	Open 2217	Open 2217				Aloc 1356
CA Access		Sequential File Created	Open 2217	Open 2217				Sequential File Created
CO Access	Load-to-old Sequential File Created	Sequential File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Sequential File Created
DG/DGA Access		Direct File Created						Aloc 1356
DU/DUA Access		Direct File Created	Open 2217	Open 2217				Aloc 1356
DO/DOA Access	Load-to-old Direct File Created	Direct File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Direct File Created
IR Access	Open 2203	Indexed File Created	Open 2203	Open 2203	Open 2203	Open 2203	Open 2203	Aloc 1356
IRU Access	Open 2203	Indexed File Created	Open 2217	Open 2217	Open 2203	Open 2203	Open 2203	Aloc 1356
IA/IRA/IRUA Access	Open 2203	Indexed File Created	Open 2217	Open 2217	Open 2203	Open 2203	Open 2203	Aloc 1356
IO Access	Load-to-old Indexed File Created	Indexed File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Indexed File Created
IS Access	Open 2203	Indexed File Created	Open 2203	Open 2203	Open 2203	Open 2203	Open 2203	Aloc 1346
ISU Access	Open 2203	Indexed File Created	Open 2217	Open2217	Open 2203	Open 2203	Open 2203	Aloc 1356

Figure E-1. (Part 1 of 6). Access Method and File Type Combinations

SEQUENTIAL FILES

	DISP-OLD	DISP-NEW	DISP-SHRRR	DISP-SHRRM	DISP-SHRMR	DISP-SHRMM or DISP-SHR	DISP Not Specified, Existing File	DISP Not Specified, New File
ISA/ISUA Access	Open 2203	Indexed File Created	Open 2217	Open 2217	Open 2203	Open 2203	Open 2203	Aloc 1356
GAM Access ORDER-RECORD		Sequential File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N		Direct File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD AEOD-N, ARRN-N		Sequential File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N, ARRN-N, GSEQ-N		Direct File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N, ARRN-N, GSEQ-N, GRAN-N		Sequential File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, CREATE-S	Load-to-old Sequential File Created	Sequential File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Sequential File Created
GAM Access OWNER-RECORD, CREATE-D	Load-to-old Direct File Created	Direct File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Direct File Created
GAM Access ORDER-RECORD, CREATE-I	Load-to-old Indexed File Created	Indexed File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Indexed File Created
GAM Access ORDER-KEY	Open 2203	Indexed File Created	Open 2217	Open 2217	Open 2203	Open 2203	Open 2203	Aloc 1356
GAM Access ORDER-KEY, CREATE-S	Load-to-old Sequential File Created Open 2203	Sequential File Created Open 2203	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Sequential File Created Open 2203
GAM Access ORDER-KEY, CREATE-D	Load-to-old Direct File Created Open 2203	Direct File Created Open 2203	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Direct File Created Open 2203
GAM Access ORDER-KEY, CREATE-I	Load-to-old Indexed File Created	Indexed File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Indexed File Created

Figure E-1. (Part 2 of 6). Access Method and File Type Combinations

	DISP-OLD	DISP-NEW	DISP-SHRRR	DISP-SHRRM	DISP-SHRMR	DISP-SHRMM or DISP-SHR	DISP Not Specified, Existing File	DISP Not Specified, New File
CG Access		Sequential File Created						Aloc 1356
CU Access		Sequential File Created	Open 2217	Open 2217				Aloc 1356
CA Access	Open 2202	Sequential File Created	Open 2217	Open 2217	Open 2202	Open 2202	Open 2202	Sequential File Created
CO Access	Load-to-old Sequential File Created	Sequential File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Sequential File Created
DG/DGA Access		Direct File Created						Aloc 1356
DU/DUA Access		Direct File Created	Open 2217	Open 2217				Aloc 1356
DO/DOA Access	Load-to-old Direct File Created	Direct File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Direct File Created
IR Access	Open 2203	Indexed File Created	Open 2203	Open 2203	Open 2203	Open 2203	Open 2203	Aloc 1356
IRU Access	Open 2203	Indexed File Created	Open 2217	Open 2217	Open 2203	Open 2203	Open 2203	Aloc 1356
IA/IRA/IRUA Access	Open 2203	Indexed File Created	Open 2217	Open 2217	Open 2203	Open 2203	Open 2203	Aloc 1356
IO Access	Load-to-old Indexed File Created	Indexed File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Indexed File Created
IS Access	Open 2203	Indexed File Created	Open 2203	Open 2203	Open 2203	Open 2203	Open 2203	Aloc 1356
ISU Access	Open 2203	Indexed File Created	Open 2217	Open 2217	Open 2203	Open 2203	Open 2203	Aloc 1356
ISA/ISUA Access	Open 2203	Indexed File Created	Open 2217	Open 2217	Open 2203	Open 2203	Open 2203	Aloc 1356

Figure E-1. (Part 3 of 6). Access Method and File Type Combinations

DIRECT FILES

	DISP-OLD	DISP-NEW	DISP-SHRRR	DISP-SHRRM	DISP-SHRMR	DISP-SHRMM or DISP-SHR	DISP Not Specified, Existing File	DISP Not Specified, New File
GAM Access ORDER-RECORD		Sequential File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N		Direct File Created	Open 2217	Open 2217				Aloc 1356
Gam Access ORDER-RECORD, AEOD-N, ARRN-N		Sequential File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N, ARRN-N, GSEQ-N		Direct File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N, ARRN-N, GSEQ-N, GRAN-N		Sequential File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, CREATE-S	Load-to-old Sequential File Created	Sequential File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Sequential File Created
GAM Access ORDER-RECORD, CREATE-D	Load-to-old Direct File Created	Direct File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Direct File Created
GAM Access ORDER-RECORD, CREATE-I	Load-to-old Indexed File Created	Indexed File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Indexed File Created
GAM Access ORDER-KEY	Open 2203	Indexed File Created	Open 2217	Open 2217	Open 2203	Open 2203	Open 2203	Aloc 1356
GAM Access ORDER-KEY, CREATE-S	Load-to-old Sequential File Created Open 2203	Sequential File Created Open 2203	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Sequential File Created Open 2203
GAM Access ORDER-KEY, CREATE-D	Load-to-old Direct File Created Open 2203	Direct File Created Open 2203	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Direct File Created Open 2203
GAM Access ORDER-KEY, CREATE-I	Load-to-old Indexed File Created	Indexed File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Indexed File Created

Figure E-1. (Part 4 of 6). Access Method and File Type Combinations

	DISP-OLD	DISP-NEW	DISP-SHRRR	DISP-SHRRM	DISP-SHRMR	DISP-SHRMM or DISP-SHR	DISP Not Specified, Existing File	DISP Not Specified, New File
CG Access		Sequential File Created						Aloc 1356
CU Access		Sequential File Created	Open 2217	Open 2217				Aloc 1356
CA Access		Sequential File Created	Open 2217	Open 2217				Sequential File Created
CO Access	Load-to-old Sequential File Created	Sequential File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Sequential File Created
DG/DGA Access		Direct File Created						Aloc 1356
DU/DUA Access		Direct File Created	Open 2217	Open 2217				Aloc 1356
DO/DOA Access	Load-to-old Direct File Created	Direct File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Direct File Created
IR Access		Indexed File Created						Aloc 1356
IRU Access		Indexed File Created	Open 2217	Open 2217				Aloc 1356
IA/IRA/IRUA Access		Indexed File Created	Open 2217	Open 2217				Aloc 1356
IO Access	Load-to-old Indexed File Created	Indexed File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Indexed File Created
IS Access		Indexed File Created						Aloc 1356
ISU Access		Indexed File Created	Open 2217	Open 2217				Aloc 1356

Figure E-1. (Part 5 of 6). Access Method and File Type Combinations

INDEX FILES

	DISP-OLD	DISP-NEW	DISP-SHRRR	DISP-SHRRM	DISP-SHRMR	DISP-SHRMM or DISP-SHR	DISP Not Specified, Existing File	DISP Not Specified, New File
	ISA/ISUA Access	Indexed File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD		Sequential File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N		Direct File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N, ARRN-N		Sequential File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N, ARRN-N, GSEQ-N		Direct File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, AEOD-N, ARRN-N, GSEQ-N, GRAN-N		Sequential File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-RECORD, CREATE-S	Load-to-old Sequential File Created	Sequential File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Sequential File Created
GAM Access ORDER-RECORD, CREATE-D	Load-to-old Direct File Created	Direct File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Direct File Created
GAM Access ORDER-RECORD, CREATE-I	Load-to-old Indexed File Created	Indexed File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Indexed File Created
GAM Access ORDER-KEY		Indexed File Created	Open 2217	Open 2217				Aloc 1356
GAM Access ORDER-KEY, CREATE-S	Load-to-old Sequential File Created	Sequential File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Sequential File Created
GAM Access ORDER-KEY, CREATE-D	Load-to-old Direct File Created	Direct File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Direct File Created
GAM Access ORDER-KEY, CREATE-I	Load-to-old Indexed File Created	Indexed File Created	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1360	Aloc 1359	Indexed File Created

Figure E-1. (Part 6 of 6). Access Method and File Type Combinations

Data Management Control Blocks and Interface Areas

To use data management to process disk files, you are required to provide storage space for interface information.

DTF

The DTF is the major control block for communication between you and data management. The DTF provides the information needed to allocate, open, access, and close a file on a disk. It also contains pointers to other control blocks and areas needed to interface with data management. The DTF must be available to the system from the time it is allocated until it is closed, and must not be moved or overlaid from the time it is opened until it is closed. The DTF is 78 bytes long for record (nonkeyed) accesses and 98 bytes long for keyed accesses. For more information on generating a disk DTF, see $DTFD (Define the File for Disk) in Chapter 5.

Input Record Area

When data is being read from disk through any type of get operation, you must provide an input record area. This is the location in your program where data management will place the record read from disk. This area can be the same area as the output record area described below. The location of this area (as specified in the DTF) can be changed at any time. This area corresponds to the INREC parameter of the $DTFD macroinstruction.

Output Record Area

When data is being written to disk through an output, an add, or the output portion of an update, you must provide an output record area which is the location in your program where data management will get the record to write to disk. This area can be the same area as the input record area described above. The location of this area (as specified in the DTF) can be changed at any time. This area corresponds to the OUTREC parameter of the $DTFD macroinstruction.

Key Area

While processing under an indexed random input-capable access method, you use the key area to provide to data management the key of the record to read from disk. These access methods are IR, IRU, IRA, IRUA, and GAM with GRAN-Y and order-key specified or assumed. The length of the key area must equal the key length. This area corresponds to the KEY parameter of the $DTFD macroinstruction.

Key Limits Area

When you request the use of key limits by using the LIMIT-Y parameter of the $DTFD macroinstruction, you must provide an area to contain the low and high key limits. The length of this area must equal two times the key length. The location of this area should not change after the file is opened. The low key is in the left half and the high key is in the right half. Limits are established when the first get-next operation is issued. This area corresponds to the HIGH parameter of the $DTFD macroinstruction.

Label Return Area

When processing disk files, data management can return a file label when certain conditions occur. If you want this file label returned, you must provide an 8-byte label return area. The location of this area should not change after the file is opened. A file label is returned in the area under the following conditions:

- Duplicate key in another index. An add or update operation would cause the creation of a duplicate key in another index over the file, or in this index if the file is being accessed by a nonkeyed access method. The label of the file in which the duplicate would have been created is returned in the label area.

- Update key error. An update operation would cause a key to be changed in a file that does not allow key updates. The label of that file is returned in the label area.

- Permanent I/O error. The label of the file where the error occurred is returned in the label area.

This area corresponds to the LABEL parameter on 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 for the file is reserved and initialized on the disk.
- Tests are performed to ensure that the access method and file organization are compatible and that all necessary information about the file was provided.
- Space in main storage (but not in your program) is allocated for buffers and data management required control blocks. The control blocks are initialized.
  - The DTF is formatted to a post open state.

For more information on the $ALOC and $OPEN macroinstructions, see $ALOC (Allocate File or Device) and $OPEN (Prepare a Device or File for Access) in Chapter 5.

Accessing Records in a File

After the file is allocated and opened, you can begin accessing records in that file. The interface between your program and data management is the same DTF that was used for allocating and opening your file. Some fields in the DTF communicate from your program to data management, some communicate from data management to your program, some are bidirectional and communicate both ways, and still others are for data management internal use only.

Figure E-2 describes the DTF fields that make up the data passed back and forth. Each field in the DTF has a name as defined in the $DTFO macroinstruction expansion. Those field names are used in Figure E-2 to identify specific DTF fields. The length of each DTF field is given with the initial field description. Several DTF fields are pointers or offsets to main storage areas in your program, and are identified as such. All DTF fields not described in this figure are reserved for internal data management use and should not be altered or otherwise used by any calling program.

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1DEV	Disk DTF device code, 1 byte. Set by the $DTFD macroinstruction to hex AO to indicate this is a disk DTF.	All	None	None	No	To
$F1CCQ	Completion code qualifier, 2 bytes. In certain error situations, data management issues error messages to the operator. If control is returned to he program, the number of the message issued is returned in this field.	All	None	None	Yes	From
$F1UPS	External indicators (UPSI), 1 byte. Used to condition open files. Masked against the external switch settings set by the // switch OCL statement.	All	$DTFD	UPSI	No	To
$F1CHB	DTF forward chain field, 2 bytes. Contains a pointer to the next DTF in a chain of DTFs if the program chooses to allocate, open or close multiple DTFs with one call. The last DTF in the chain should not specify the CHAIN parameter.	All	$DTFD	CHAIN	No	To

Figure E-2 (Part 1 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1OUT	Output Record area address, 2 bytes. Contains a pointer to the output record area in your program. Data management gets the record to write to disk from this area for output, add, and update operations. This DTF field can be changed at any time. The output record area address can be the same as the record area address ($F1INP) described below.	GAM, CA CO, CU, DO DOA, DU DUA, IA, IO ISA, ISU ISUA, IRA IRU, IRUA	$DTFD	OUTREC	Yes	To
$F1CMP	Completion code, 1 byte. Set by data management to indicate successful or unsuccessful completion of the operation requested of data management.	All	None	None	Yes	From
$F1OPC	Operation code, 1 byte. Set by the program to indicate what operation data management is to perform.	All	$DTFD $PUTD	OP OP	Yes	To
$F1AT1	Attribute byte 1, 1 byte. Defines in general the type of access, and the operation codes allowed under the access method. The other attribute bytes (2-5) described below can further define the access and allowable operation codes.	All	$DTFD	ACCES GRAN GSEQ AEOD ARRN UPDATE DELETE	No	To

Figure E-2 (Part 2 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1AT2	Attribute byte 2, 1 byte. Further defines the type of access. Indicates if the access is by record or key. Indicates if this is ACCESS-PSEUDO. Indicates if the file has been opened.	All	$DTFD	ACCESS ORDER	No	To
$F1AT3	Attribute byte 3, 1 byte. Further defines the type of access. Indicates what type of file to create (sequential, direct, or indexed) for output accesses. Indicates if data management should ensure keys are in ascending order when keyed output or add is done. Indicates whether the relative record number or argument value for ARRN, RRN, PLUS, or MINUS operations is binary or decimal. Default for ACCESS-CG is binary if $DTFD ARG parameter is not specified. ACCESS-DGA /DOA/DUA implies binary RRNs/ values, ACCESS-DG, DO/DU implies decimal RRNs/ values. Indicates if the file has been allocated.					

Figure E-2 (Part 3 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1AT4	Attribute byte 4, 1 byte. Indicates whether or not return permanent disk errors to the program. Indicates whether to allow option 2 on permanent disk error messages issued. Indicates if key limits are used for this keyed access. Indicates if data management is to check if the requested record is already owned by the task.	All	$DTFD	IOMSG RETURN LIMIT LOCKCK	No	To
$F1RCL	Record length field, 2 bytes. Defines the record length of the records in the file being accessed through this DTF.	All	$DTFD	RECL	No	To
$F1NAM	File name field. 8 bytes. Indicates the name of the file being accessed through this DTF. The name specified in the DTF must be the same as the name specified in the NAME parameter on the // FILE OCL statement for the file. The name is left-justified in this field.	All	$DTFD	NAME	No	To

Figure E-2 (Part 4 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1INP	Input record area address, 2 bytes. Contains a pointer to the input record area in your program. Data management places the record read from the disk in the area for all input operations. This DTF field can be changed at sny time. The input record area address can be the same as the output record area address ($F1OUT) described previously.	GAM, CG, CU, DG, DGA, DU, DUA, IS, ISA, ISU, ISUA, IR, IRA, IRU, IRUA	$DTFD	INREC	Yes	To

Figure E-2 (Part 5 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1DBF	Data blocking factor, 2 bytes. Specifies the number of records to be moved between main storage and disk for each disk I/O operation. A default of 1 is assumed if the $DTFD DBLOCK parameter is not specified. Allowed blocking factors are from 1 to 65535. Buffer space is reserved by the open function based on this factor, the index blocking factor ($F1IBF described below), the record length, and the type of access. The maximum size buffer space is 45056. If the blocking factors, record length, and access type dictate a buffer space larger than the maximum allowed, the buffer space is set to 45056 bytes, and divided as equally as possible between data buffers and index buffers (if any).	All	$DTFD	DBLOCK	No	To
$F1MD1	Modifier byte 1, 1 byte. Set by $GETD LIMIT parameter to indicate if new key limits should be set by data management for this GET request.	GAM (GSEQ-Y and ORDER -KEY) IS, ISU	$GETD	LIMIT	Yes	To

Figure E-2 (Part 6 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1ARG	Argument field, 8 bytes. Relative record number (RRN) or argument value. For certain I/O operations, the program is required to pass an RRN or an argument value to data management in the field. This value can be specified in binary or in zoned decimal. $F1RRNB, described below, redefines this field for decimal values. See descriptions following for additional information.	GAM (GRAN-Y and ORDER -RECORD) DG, DGA, DU, DUA, DO, DOA	None	None	Yes	To/Fr
$F1RRNB	Binary argument field, 3 bytes. This field is defined over the leftmost 3 bytes of the $F1ARG field, described above. If you are passing a binary argument, place it as a 3-byte number in this field.See the $DTFD ARG parameter for information on how to specify that the argument is in binary.	GAM (GRAN-Y & ORDER -RECORD) DGA, DUA, DOA	None	None	Yes	To/Fr
$F1RRND	Decimal argument field, 8 bytes. This field is defined over the entire 8 bytes of the $F1ARG field, described above. If you are passing a decimal argument, place it as an unsigned, 8-byte decimal number in this field. See the $DTFD ARG parameter for information on how to specify that the argument is in decimal.	GAM (GRAN-Y and ORDER-RECORD), DG, DU, DA	None	None	Yes	To/Fr

Figure E-2 (Part 7 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1FBL	Feedback label offset field, 2 bytes. This field is an offset from the end of the DTF to the leftmost byte of the 8-byte feedback label area in your program. This area must be after the DTF. The last byte of this area must be less than 2048 bytes away from the first byte of the DTF. In some situations, data management can return a file label in this area.	All	$DTFD	LABEL	No	To
$F1ATS	Attribute byte 5, 1 byte. This attribute byte is for expansion purposes only.	None	None	None	No	-
$F1KEY	Key area offset field, 2 bytes. This field is an offset from the last byte of the DTF to the first byte of the key area in your program. The area length must be equal to the key length. This area must be after the DTF. The last byte of this area must be less than 2048 bytes from the first byte of the DTF. The key area must contain the key of the record to be read from disk for indexed random input-capable access methods.	GAM (GRAN-Y and ORDER -KEY) IR, IRA, IRU, IRUA	$DTFD	KEY	No	To
$F1KL	Key length field, 2 bytes. This field contains the key length of the file being accessed through this DTF.	GAM (ORDER -KEY), IO, IS, ISA, ISU, ISUA, IR, IRA, IRU, IRUA	$DTFD	KEYL	No	To

Figure E-2 (Part 8 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1KD	Key displacement field, 2 bytes. This field contains the displacement into the record of the rightmost byte of the key in the record. The displacement of the first byte in the record is 0, the second is 1, and so on. The maximum displacement is 4095 bytes.	GAM (ORDER -KEY), IO, IS, ISA, ISU, ISUA, IR, IRA, IRU, IRUA	$DTFD	KDISP	No	To

Figure E-2 (Part 9 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1IBF	Index blocking factor field, 2 bytes. Gives the number of index entries (key length +3) moved between main storage and disk for each I/O operation. Default is 1 if the $DTFD IBLOCK parameter is specified. Allowed blocking factors are from 1 to 65535. The actual blocking factor may be larger because the index buffer is always a multiple of 256 bytes and thus may hold more entries than requested. (Index entries do not cross 256 byte boundaries in the index buffer.) Buffer space is reserved by $OPEN on this factor, the data blocking factor ($F1DBF described above), the record length, and the type of access. Maximum size buffer space is 45056 bytes. If the blocking factors, record length, and access type dictate a buffer space larger than the maximum allowed, the buffer space is set to 45056 bytes, and divided as equally as possible between data and index buffers.	GAM (ORDER -KEY), IO, ISA, ISU, ISUA, IR, IRA, IRU, IRUA, IS	$DTFD	IBLOCK	No	To

Figure E-2 (Part 10 of 11). Disk DTF Field Description

Field Name	Field Description	Access Methods Applic-able	Macro Used	Key-word Used	Can Be Altered After Allocate	Interf To/Fr Disk D/M
$F1LIM	Key limits area offset field, 2 bytes. This field is an offset from the end of the DTF to the leftmost byte of the key limits area in your program. The key limits area length must be equal to two times the key length. This area must be after the DTF. The last byte of this area must be less than 2048 bytes away from the first byte of the DTF. Use this field to pass the low and high key limits to data management if this access is keyed within limits.	GAM (GSEQ and ORDER -KEY), IS, ISU	$DTFD	HIGH	No	To

Figure E-2 (Part 11 of 11). Disk DTF Field Description

Completion Conditions

Figure E-3 describes all currently defined completion conditions, and the access methods and I/O operations to which they apply. The completion condition indicates whether the I/O operation was successful or not, and is returned to your program by disk D/M in the completion code field ($F1CMP) in the DTF. The labels and hex values generated by the $DTFO macroinstruction for the completion conditions are used in Figure E-3.

$DTFO Label	Value (hex)	Completion Condition Description	Applicable Access Methods	Applicable Operation Codes
$F1CCSUC	40	Normal. The operation completed normally.	All	All
$F1CCPER	41	Permanent I/O Error. An unrecoverable software or hardware error occurred. Refer to the $DTFD IOMSG and RETURN parameters for message options that can be requested for this error. Also, if the $DTFD LABEL parameter is specified, the file label is returned in the label return area.	All except PSEUDO	All except RELEASE
$F1CCEOF	42	End of File (EOF). This $GETD OP-NEXT parameter is attempting to read past the last record in the file, or this $GETD OP-PREV parameter is attempting to read before the first record in the file. A $GETD OP-NEXT issued after a $GETD OP-NEXT, which received an EOF, also received an EOF. A $GETD OP-PREV issued after a $GETD OP-PREV which received an EOF, will also received an EOF. This $GETD OP-READE is attempting to read a record whose key is not equal to the key specified by the key parameter in the $DTFD.	GAM (GSEQ-Y), CG, CU, IS, ISA, ISU, ISUA	Get NEXT, PREV, READE

Figure E-3 (Part 1 of 6). Completion Condition Descriptions

$DTFO Label	Value (hex)	Completion Condition Description	Applicable Access Methods	Applicable Operation Codes
$F1CCIOP	43	Invalid Operation Code. The operation requested is invalid for the access method in the DTF or for the type of file being accessed. $GETD, OP-KEY/KEYEA/KEYA /RRN/FIRST/LAST were issued, but the access is not random-get-capable. $GETD, OP-NEXT/PREV/PLUS/ MINUS issued, but the access is not sequential-get-capable. $GETD, LIMIT-Y was issued, but LIMIT-Y was not specified on $DTFD macroinstruction. $GETD, OP-UPDATE were issued, but access is not update-capable. $GETD, OP-DELETE issued, but access or file is not delete-capable. $PUTD OP-AEOD was issued, but the access is not add-at-end-data-capable. $PUT OP-ARRN issued, to a sequential file that is not delete-capable, to a direct file that is not delete-capable, to an indexed access DTF, or to an access that is not random-add-capable. Issue $PUTD OP-AEOD to a direct file.	All except PSEUDO	All except RELEASE

Figure E-3 (Part 2 of 6). Completion Condition Descriptions

$DTFO Label	Value (hex)	Completion Condition Description	Applicable Access Methods	Applicable Operation Codes
$F1CCNRF	44	Record Not Found. The requested record was not found in the file. $GETD OP-FIRST /LAST issued, but the file is empty. $GETD OP-PLUS/MINUS /RRN issued, but the record at that position in the file is deleted. $GETD OP-KEY /KEYEA/KEYA issued, but the requested key does not exist in the file.	GAM (GRAN-Y) DG, DGA, DU,DUA, IR, IRA, IRU, IRUA	Get FIRST, LAST, PLUS, MINUS, KEY, KEYA, KEYEA, RRN
$F1CCNPR	45	No Pending Record. For a nonshared file, the program has not read a valid record before issuing a $GETD OP-UPDATE/DELETE. For a shared file, the operation immediately preceding a $GETD OP-UPDATE/DELETE was not a valid read of a record.	GAM (UPDATE-Y), CU, DU, DUA,IRU, IRUA,ISU, ISUA	Put UPDATE, DELETE
$F1CCIRN	48	Invalid Relative Record Number (RRN). The requested RRN is not within the file. $GETD OP-PLUS/MINUS/RRN issued, but no record exists with that RRN. $PUTD OP-ARRN issued, but the RRN is beyond the extents of the file. $PUTD OP-UPDATE issued, and the completion code from the previous $GETD or $PUTD was 48.	GAM (GRAN-Y and ORDER-RECORD), or (ARRN-Y and ORDER-RECORD) DG, DGA, DU, DUA, DO, DOA	Get PLUS, MINUS, RRN Put ARRN, UPDATE

Figure E-3 (Part 3 of 6). Completion Condition Descriptions

$DTFO Label	Value (hex)	Completion Condition Description	Applicable Access Methods	Applicable Operation Codes
$F1CCIUA	49	Invalid Data Record. The program is attempting to put a record with hex FF in the first byte into a delete-capable file. $PUTD OP-UPDATE/ARRN/ AEOD issued, the record to be written has hex FF in the first byte, and the file is delete-capable.	GAM (with any combination of UPDATE-Y, ARRN-Y, AEOD-Y) CU, DU, DUA, DO, DOA, IO, IA, IRU, IRUA, ISU, ISUA	Put UPDATE, ARRN, AEOD
$F1CCKER	50	Update Key Error. The program is attempting to change a key in the index for a file (parent index if this is a multiple index file). $PUTD OP-UPDATE issued, and the key in the record to be written is different from the key for that record in the index for that file (parent index, if this is a multiple index file).	GAM (UPDATE-Y) CU, DU, DUA, IRU, IRUA, ISU, ISUA	Put UPDATE
$F1CCNDR	53	No Deleted Record Found. $PUT OP-ARRN issued to a delete-capable file, but the record at the RRN location is not a deleted record.	GAM (ARRN-Y), DU,DUA, DO, DOA	Put ARRN
$F1CCDUP	60	Duplicate Key. The $PUTD OP-AEOD/ARRN /UPDATE being attempted will cause a duplicate key in the index being used to access the file, and that index does not allow duplicate keys. If this is an AEOD and BYPASS-YES was specified on the // FILE OCL statement, the add will be allowed.	GAM (with any combination of UPDATE-Y, ARRN-Y, AEOD-Y) CU, DU, DUA, IA, IO, IRU, IRUA, ISU, ISA, ISUA	Put AEOD, ARRN, UPDATE

Figure E-3 (Part 4 of 6). Completion Condition Descriptions

$DTFO Label	Value (hex)	Completion Condition Description	Applicable Access Methods	Applicable Operation Codes
$F1CCDPO	61	Duplicate Key in Another Index. $PUTD OP-AEOD/ARRN /UPDATE being attempted which causes a duplicate key in an index not being used to access the file, and that index does not allow duplicate keys.	GAM (with any combination of UPDATE-Y, ARRN-Y, AEOD-Y) CU, DU, DUA, IA, IO, IRU, IRUA, ISU, ISA, ISUA	Put AEOD, ARRN, UPDATE
$F1CCSEQ	62	Key Out of Sequence. The program is adding a key less than the previous key that was added, and an ordered load ($DTFD ORDLD-Y or $DTFD ACCESS-ISA/ISUA specified) was requested for this access.	GAM (AEOD-Y), IA, IO, ISA, ISUA	Put AEOD
$F1CCEOX	70	End of Extent. The program is issuing a $PUTD OP-AEOD to a file, the file is full, and either the EXTEND // FILE OCL statement parameter was not specified for the file, or an extend was attempted but could not be completed.	GAM (AEOD-Y), CA, CO, IA, IO, IRA, IRUA, ISA, ISUA	Put AEOD
$F1CCUAT	75	Undefined Access Type. Currently never issued.		
$F1CCRAL	80	Record Already Locked. The program is attempting to read a record, or to add a record by a $PUTD OP-ARRN, through a DTF that has LOCKCK-Y specified, and that record is already owned (read with an update-capable access method) by another DTF in the program.	GAM (with any combination of UPDATE-Y, ARRN-Y, AEOD-Y), CG, CU, DG,DU, DUA, IR, IRU, IRUA, IS, ISU, ISA, ISUA	All Get operations Put ARRN

Figure E-3 (Part 5 of 6). Completion Condition Descriptions

$DTFO Label	Value (hex)	Completion Condition Description	Applicable Access Methods	Applicable Operation Codes
$F1CCNOP	99	File Not Opened. The program is attempting to access a file, and the DTF for that file has not been opened.	All except PSEUDO	All

Figure E-3 (Part 6 of 6). Completion Condition Descriptions

Closing the File

When you are finished processing records in a file, you should close ($CLOSE) the file. Close performs the following operations:

- Writes to disk any data buffers that need to be written.
- Releases the main storage space allocated in open for buffers and data management required control blocks.
- Resets the disk DTF to a preallocate state.

Once a DTF has been closed, it must be allocated ($ALOC) and opened ($OPEN) again before it can be used to access records in a file. For more information on the $CLOS macroinstruction, see $CLOS (Prepare a Device or File for Termination) in Chapter 5.

Appendix F. 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 Programmer's Guide*.

GET and ACI Return Codes

After a GET or ACI operation, the following return codes are possible at $WSRTC:

Label	Value (hex)	Explanation
$WSROK	00	Operation was successful.
$WSRACC	01	New requester. **Note**: If the user program does ACI as the first operation in order to accept program data, and their input buffer is not large enough to accept all of the program data, a return code of X'01' is returned. $WSOUTL will contain X'0000'.
$WSRSTP	02	Stop system was requested by system operator.
$WSRCTL	03	No data was returned – control information only.
$WSRACR	11	ACI was rejected. No invites outstanding.
$WSRNAV	24	Display station was released by display station operator.
$WSRREL	28	GET was rejected. Display station previously released by program.

Label	Value (hex)	Explanation
$WSRIRJ	34	Input was rejected. Input buffer (INLEN parameter) is too small.
$WSPOST	60	Posted with user-defined address.
$WSPPRE	80	Permanent I/O error occurred at the display station. In response to the error, the system operator selected option 2.

ACQ Return Codes

After an ACQ operation, the following return codes are possible at $WSRTC:

Label	Value (hex)	Explanation
$WSROK	00	ACQ was successful.
$WSRQO	08	ACQ was successful. Display station was already allocated to the task.
$WSRAFW	18	ACQ failed. Display station was allocated to a non-NEP.
$WSRAFN	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 a NEP. • The previous release operation for the display station is still being processed.

STI Return Codes

After an STI operation, the following return codes are possible at $WSRTC:

Label	Value (hex)	Explanation
$WSROK	00	STI was successful.
$WSRNAV	24	Display station was released by display station operator.
$WSRREL	28	STI was ignored. Display station was previously released by program.
$WSRSPF	44	STI failed. Display station operator entered data, which should be read by a GET or ACI operation.

Label	Value (hex)	Explanation
$WSPRE	80	Permanent I/O error occurred at the display station. In response to the error, the system operator selected option 2.

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 (hex)	Explanation
$WSROK	00	Operation was successful.
$WSRNAV	24	Display station was released by display station operator.
$WSRREL	28	Operation was ignored. Display station previously released by program.
$WSRIRJ	34	Input was rejected. Input buffer (INLEN parameter) too small.
$WSROFL	40	Requested terminal was offline.
$WSPOGE	45	Invalid ideographic data was found on a print operation.
$WSRPAL	48	Print operation was issued to the allocated printer.
$WSRGRF	50	On an output operation, a display station ideographic character table full of ideographic characters was detected. The user selected a 2 option.
$WSRGI	51	On an output operation an invalid ideographic character was found.The user selected a 2 option.

Label	Value (hex)	Explanation
$WSRGU	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	80	Permanent I/O error occurred at the display station. In response to the error, the system operator selected option 2.

Notes:

Glossary

#LIBRARY. The library, provided with the system, that contains the System Support Program Product. See *system library.*

abnormal termination. A system failure or operator action that causes a job to end unsuccessfully.

access method. The way that records in files are referred to by the system. The reference can be consecutive (records are referred to one after another in the order in which they appear in the file), or it can be random (the individual records can be referred to in any order).

address. A name, label, or number that identifies a location in storage, a device in a network, or any other data source.

address recall register (ARR). A register in the main storage processor that is used for temporary storage of an address to be used later by the program being run.

advanced program-to-program communications (APPC). Communications support that allows System/36 to communicate with other systems having the same support. APPC is the way that System/36 puts the IBM SNA LU-6.2 protocol into effect.

alarm. An audible signal at a display station or printer that is used to get the operator's attention.

allocate. To assign a resource, such as a disk file or a diskette file, to perform a specific task.

alphabetic character. Any one of the letters A through Z (uppercase and lowercase). Assembler extends the alphabet to include the special characters #, $, and @.

alphameric. Consisting of letters, numbers, and often other symbols, such as punctuation marks and mathematical symbols.

alphanumeric. See *alphameric.*

alternative system console. A command display station that can be designed as the system console.

American National Standard Code for Information Interchange (ASCII). The code developed for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

APPC. See *advanced program-to-program communications (APPC).*

application. (1) A particular business task, such as inventory control or accounts receivable. (2) A group of related programs that apply to a particular business area, such as the Inventory Control or the Accounts Receivable application.

application program. A program used to perform an application or part of an application.

ASCII. See *American National Standard Code for Information Interchange (ASCII).*

assembler. A program that converts assembler language statements to an object program.

assembler instruction statement. A statement that controls what the assembler does, rather than what the user's program does.

assembler language. A symbolic programming language in which the set of instructions includes the instructions of the machine and whose data structures correspond directly to the storage and registers of the machine.

attribute. A characteristic.

autoanswer. In data communications, the ability of a station to receive a call over a switched line without operator action. Contrast with *manual answer.*

autocall. In data communications, the ability of a station to place a call over a switched line without operator action. Contrast with *manual call.*

autocall unit. A common carrier device that allows System/36 to automatically call a remote location.

base displacement addressing. In assembler language, an addressing method that involves setting up a base address from which other addresses can be calculated.

base number. The part of a self-check field from which the check digit is calculated.

BASIC (beginner's all-purpose symbolic instruction code). A programming language designed for interactive systems and originally developed at Dartmouth College to encourage people to use computers for simple problem-solving operations.

batch. Pertaining to activity involving little or no operator action. Contrast with *interactive.*

batch BSC. The System Support Program Product support that provides data communications with BSC computers and devices via the RPG T specification or the assembler $DTFB macroinstruction.

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

binary synchronous communications (BSC). A form of communications line control that uses transmission control characters to control the transfer of data over a communications line. Compare with *synchronous data link control (SDLC).*

bit. Either of the binary digits 0 or 1. See also *byte.*

block. (1) A group of records that is recorded or processed as a unit. Same as *physical record.* (2) Ten sectors (2560 bytes) of disk storage.

branch instruction. An instruction that changes the sequence in which the instructions in a computer program are performed. The sequence of instructions continues at the address specified in the branch instruction.

branching. Performing a statement other than the next one in sequence.

BSC. See *binary synchronous communications (BSC).*

byte. The amount of storage required to represent one character; a byte is 8 bits.

call. (1) To activate a program or procedure at its entry point. Compare with *load.* (2) In data communications, the action necessary in making a connection between two stations on a switched line.

cancel. To end a task before it is completed.

character. A letter, digit, or other symbol.

character key. A keyboard key that allows the user to enter the character shown on the key. Compare with *command keys* and *function key.*

character string. A sequence of consecutive characters.

check. (1) An error condition. (2) To look for a condition.

check digit. The rightmost digit of a self-check field used to check the accuracy of the field.

close. To end the processing of a file.

COBOL (common business-oriented language). A high-level programming language, similar to English, that is used primarily for commercial data processing.

code. (1) Instructions for the computer. (2) To write instructions for the computer. Same as *program.* (3) A representation of a condition, such as an error code.

command. A request to perform an operation or a procedure.

command display station. A display station from which an operator can start and control jobs. A command display station can become an alternative system console, can be designated as a subconsole, and can also be used as a data display station. See also *alternative system console, data display station,* and *subconsole.*

command keys. The 12 keys on the top row of the display station keyboard that are used with the Cmd key (and optionally the Shift key) to request up to 24 different actions defined for program products and user programs. Compare with *character key* and *function key.*

command mode. A mode that allows a display station operator to request or start jobs.

command text, command source or load member. The command to be processed when an operator selects an option on a menu.

comment. Words or statements in a program or procedure that serve as documentation rather than as instructions.

compilation time. The time during which a source program is translated from a high-level language to a machine language program.

compile. To translate a program written in a high-level programming language into a machine language program.

constant. A data item with a value that does not change. Contrast with *variable*.

constant field. A field that is defined by a display format to contain a value that does not change.

continuation line. A line of a source statement into which characters are entered when the source statement cannot be contained on the previous line or lines.

control command. A command used by an operator to control the system or a work station. A control command does not run a procedure and cannot be used in a procedure.

control station. The primary or controlling computer on a multipoint line. The control station controls the sending and receiving of data.

cursor. A movable symbol (such as an underline) on a display, usually used to indicate to the operator where to type the next character.

data display station. A display station from which an operator can only enter data. A data display station is acquired and controlled by a program. Contrast with *command display station*.

data file utility (DFU). The part of the Utilities Program Product that is used to create, maintain, display, and print disk files.

data item. A unit of information to be processed.

data type. A category that identifies the mathematical qualities and internal representation of data.

decimal. Pertaining to a system of numbers to the base ten; decimal digits range from 0 through 9.

default value. A value stored in the system that is used when no other value is specified.

define-the-file (DTF). A control block containing information that is passed between data management routines and users of the data management routines.

delete. To remove.

DFU. See *data file utility (DFU)*.

direct file. A disk file in which records are referenced by the relative record number. Contrast with *indexed file* and *sequential file*.

disk. A storage device made of one or more flat, circular plates with magnetic surfaces on which information can be stored.

display. (1) A visual presentation of information on a display screen. (2) To show information on the display screen.

display control specification. A record within the display format specifications, it provides information about the entire display format that, in general, is unrelated to the specific fields being defined. Also known as the S-specification.

display format. Data that defines (or describes) a display.

display layout sheet. A form used to plan the location of data on the display.

display screen. The part of the display station on which information is displayed.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent or receive information from the system.

display text source or load member. Describes the information displayed on a menu.

DTF. See *define-the-file (DTF)*.

edit. (1) To modify the form or format of data; for example, to insert or remove characters such as for dates or decimal points. (2) To check the accuracy of information that has been entered, and to indicate if an error is found.

embedded blanks. Blanks that are surrounded by any other characters.

enter. To type in information on a keyboard and press the Enter key in order to send the information to the computer.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

field. One or more characters of related information (such as a name or an amount).

file. A set of related records treated as a unit.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) To arrange such things as characters, fields, and lines.

FORTRAN (formula translation). A high-level programming language used primarily for scientific, engineering, and mathematical applications.

function key. A keyboard key that requests an action but does not display or print a character. The cursor movement and Help keys are examples of function keys. Compare with *command keys* and *character key.*

GAM. See *generalized access method (GAM).*

generalized access method (GAM). A disk file access method in assembler allowing random and consecutive processing, update, delete, and add.

hex. See *hexadecimal.*

hexadecimal. Pertaining to a system of numbers to the base sixteen; hexadecimal digits range from 0 (zero) through 9 (nine) and A (ten) through F (fifteen).

host system. The primary or controlling computer in the communications network. See also *control station.*

index. (1) A table containing the key value and location of each record in an indexed file. (2) A computer storage position or register, the contents of which identify a particular element in a set of elements.

indexed file. A file in which the key and the position of each record is recorded in a separate portion of the file called an index. Contrast with *direct file* and *sequential file.*

index key. The field within a record that identifies that record in an indexed file.

indicator. An internal switch that communicates a condition between parts of a program or procedure.

informational message. A message that provides information to the operator, but does not require a response.

input. Data to be processed.

input/output (I/O). Pertaining to either input or output, or both.

instruction. A statement that specifies an operation to be performed by the computer and the locations in storage of all data involved in that operation.

instruction address register (IAR). A register in the main storage processor that contains the address of the next instruction to be performed.

instruction fetch. The act of getting an instruction from storage and loading it into the correct registers.

integer. A positive or negative whole number; that is, an optional sign followed by a number that does not contain a decimal point.

interactive. Pertaining to activity involving requests and replies as, for example, between an operator and a program or between two programs. Contrast with *batch.*

Interactive Communications Feature (SSP-ICF). A feature of the System Support Program Product that allows a program to interactively communicate with another program or system.

interchange record separator (IRS). Same as *record separator.*

intermediate-text-block (ITB) character. In binary synchronous communications, the transmission control character used to indicate the end of a section of data to be checked.

IRS (interchange record separator). Same as *record separator.*

ITB. See *intermediate-text-block character.*

K-byte. 1024 bytes.

key mask. A string of numbers and alphabetic characters that identify the function keys and command keys that the operator can use to control program operations.

left-adjust. To place or move an entry in a field so that the leftmost character of the field is in the leftmost position. Contrast with *right-adjust.*

library. (1) A named area on disk that can contain programs and related information (not files). A library consists of different sections, called library members. (2) The set of publications for a system.

library member. A named collection of records or statements in a library. The types of library members are *load member, procedure member, source member,* and *subroutine member.*

link-editing. To combine, by the overlay linkage editor, a number of load members and/or subroutine members into one program.

literal. A symbol or a quantity in a source program that is itself data, rather than a reference to data.

load. To move data or programs into storage.

load member. A library member that contains information in a form that the system can use directly, such as a display format. Contrast with *source member.*

load module. A program in a form that can be loaded into main storage and run. The load module is the output of the overlay linkage editor.

local. Pertaining to a device, file, or system that is accessed directly from your system, without the use of a communications line. Contrast with *remote.*

machine instruction. An instruction of the machine language that can be performed by the computer.

machine language. A language that can be used directly by a computer without intermediate processing.

macro. See *macro definition, macro instruction.*

macro call. Synonym for *macro instruction.*

macro definition:. A set of statements that defines the name of, format of, and conditions for generating a sequence of assembler language statements from a single source statement.

macroinstruction. A single instruction that represents a set of instructions.

macro library. A library of macro definitions used during macro expansion.

magnetic stripe reader. A device, attached to a display station, that reads data from a magnetic stripe on a badge before allowing an operator to sign on.

manual answer. In data communications, a line type requiring operator actions to receive a call over a switched line. Contrast with *autoanswer.*

manual call. In data communications, a line type requiring operator actions to place a call over a switched line. Contrast with *autocall.*

menu. A displayed list of items from which an operator can make a selection.

message. Information sent to an operator or programmer from a program. A message can be either displayed or printed.

message identification. A field in the display or printout of a message that directs the user to the description of the message in a message guide or a reference manual. In Assembler, this field consists of the alphabetic characters ASM, followed by a dash, followed by the message identification code.

message identification code (MIC). A four-digit number that identifies a record in a message member. This number can be part of the message identification.

message identifier. A field in the display or printout of a message that directs the user to the description of the message in a message guide or reference manual. This field consists of up to four alphabetic characters, followed by a dash, followed by the message identification code.

message member. A library member that defines the text of each message and its associated message identification code.

MIC. See *message identification code (MIC).*

modulus 10/modulus 11 checking. Formulas used to calculate the check digit for a self-check field.

noncontiguous. Not being in actual contact.

null character. The character hex 00, used to represent the absence of a printed or displayed character.

numeric. Pertaining to any of the digits 0 through 9.

object module. A set of instructions in machine language. The object module is produced by a compiler from a subroutine or source program and can be input to the overlay linkage editor.

object program. In COBOL, a set of instructions in machine – runnable form. The object program is produced by a compiler from a source program.

OCL. See *operation control language (OCL).*

open. To prepare a file for processing.

operand. (1) A quantity of data that is operated on, or the address in a computer instruction of data to be operated on. (2) In COBOL, the object of a verb or an operator; that is, an operand is the data or equipment governed or directed by a verb or operator.

operation. A defined action, such as adding or comparing, performed on one or more data items.

operation code. A code used to represent the operations of a computer.

operation control language (OCL). A language used to identify a job and its processing requirements to the System Support Program Product.

output. The result of processing data.

overlay. (1) To write over (and therefore destroy) an existing file. (2) A program segment that is loaded into main storage and replaces all or part of a previously loaded program segment.

overlay linkage editor. The part of the System Support Program Product that combines object programs to produce code that can be run and allows the user to determine overlays for programs.

overlay region. A continuous area of main storage in which segments can be loaded independently of other regions.

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

parameter. A value supplied to a procedure or program that either is used as input or controls the actions of the procedure or program.

printout. Information from the computer that is produced by a printer.

procedure. A set of related operation control language statements (and, possibly, utility control statements and procedure control expressions) that cause a specific program or set of programs to be performed.

procedure member. A library member that contains the statements (such as operation control language statements) necessary to perform a program or set of programs.

program. (1) A sequence of instructions for a computer. See *source program* and *load module*. (2) To write a sequence of instructions for a computer. Same as *code*.

program product. A licensed program for which a fee is charged.

prompt. A displayed request for information or operator action.

record. A collection of fields that is treated as a unit.

record separator. In binary synchronous communications, a character used to indicate the end of one record and the beginning of another.

recovery procedure. (1) An action performed by the operator when an error message appears on the display screen. Usually, this action permits the program to continue or permits the operator to run the next job. (2) The method of returning the system to the point where a major system error occurred and running the recent critical jobs again.

register. A storage area, in a computer, usually intended for some special reason, capable of storing a specified amount of data such as a bit or an address.

relative record number. A number that specifies the location of a record in relation to the beginning of the file.

remote. Pertaining to a system or device that is connected to your system through a communications line. Contrast with *local.*

restore. Return to an original value or image. For example, to restore a library from diskette.

return code. In data communications, a value generated by the system or subsystem that is returned to a program to indicate the results of an operation issued by that program.

right-adjust. To place or move an entry in a field so that the rightmost character of the field is in the rightmost position. Contrast with *left-adjust.*

RPG. A programming language specifically designed for writing application programs that meet common business data processing requirements.

RRN. See *relative record number.*

run. To cause a program, utility, or other machine function to be performed.

screen design aid (SDA). The part of the Utilities Program Product that helps the user design, create, and maintain displays and menus. Additionally, SDA can generate specifications for RPG and WSU work station programs.

SDA. See *screen design aid (SDA).*

SDLC. See *synchronous data link control (SDLC).*

self-check field. A field, such as an account number, consisting of a base number and a check digit.

sequential access. An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

sequential file. A file in which records occur in the order in which they were entered. Contrast with *direct file* and *indexed file.*

SEU. See *source entry utility (SEU).*

significant digit. Any digit of a number that follows the leftmost digit which is not a zero and that is within the accuracy allowed.

source entry utility (SEU). The part of the Utilities Program Product used by the operator to enter and update source and procedure members.

source member. A library member that contains information in the form in which it was entered, such as RPG specifications. Contrast with *load member*.

source program. A set of instructions that are written in a programming language and that must be translated to machine language before the program can be run.

special character. A character other than an alphabetic or numeric character. For example; *, +, and % are special characters.

special registers. In COBOL, compiler-generated data items used to store information produced by specific COBOL features (for example, the DEBUG-ITEM special register).

split key. A key, for an indexed file, defined from more than one field within each record.

SSP. See *System Support Program Product (SSP)*.

SSP-ICF. See *Interactive Communications Feature (SSP-ICF)*.

statement. An instruction in a program or procedure.

storage index. A table in main storage that contains the address of the lowest key on each track in the file index.

subconsole. A display station that controls a printer or printers.

subroutine member. A library member that contains information that must be combined with one or more members before being run by the system.

synchronous data link control (SDLC). A form of communications line control that uses commands to control the transfer of data over a communications line. Compare with *binary synchronous communications (BSC)*.

system library. The library, provided with the system, that contains the System Support Program Product and is named #LIBRARY.

System Support Program Product (SSP). A group of licensed programs that manage the running of other programs and the operation of associated devices, such as the display station and printer. The SSP also contains utility programs that perform common tasks, such as copying information from diskette to disk.

terminal error. Any error that causes termination of the current program.

transaction. (1) An item of business. The handling of customer orders and customer billing are examples of transactions. (2) In interactive communications, the communication between the application program and a specific item (usually another application program) at the remote system.

TRANSACTION file. In COBOL, an input/output file used to communicate with display stations and SSP-ICF sessions.

truncate. To shorten a field or statement to a specified length.

turnaround time. The time interval required to reverse the direction of transmission over a communication line.

unique. The only one.

unprotected field. A displayed field for which operators can enter, modify, or delete data.

Utilities Program Product. A program product that contains the data file utility (DFU), the source entry utility (SEU), the work station utility (WSU), and the screen design aid (SDA).

utility control statement. A statement that gives a utility program information about the way the program is to perform or the output it is to produce.

utility program. A System Support Program Product program that allows you to perform a common task, such as copying information from diskette to disk.

variable. A name used to represent a data item whose value can change while the program is running. Contrast with *constant*.

work station. A device that lets people transmit information to or receive information from a computer; for example, a display station or printer.

work station data management. The part of the System Support Program Product that enables a program to present data on a display screen by providing a string of data fields and a format name.

work station utility (WSU). The part of the Utilities Program Product that helps you to write programs for data entry, editing, and inquiry.

WSU. See *work station utility (WSU)*.

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your nearest IBM branch office. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

☐ If your comment does not need a reply (for example, pointing out a typing error) check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your nearest IBM branch office to request additional publications.

Name _____

Company or Organization _____

Address _____

_____

City State Zip Code

No postage necessary if mailed in the U.S.A.

SC21-7908-3

Cut Along Line

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
Information Development
Department 532
Rochester, Minnesota, U.S.A. 55901

Fold and tape Please do not staple Fold and tape

IBM

International Business Machines Corporation

Programming with Assembler

Contents

File Number
S36-21

Order Number
SC21-7908-3

Part Number
59X3987

Printed in U.S.A.