SYSTEM/32

IBM System/32
System Control Programming
Macroinstructions
Reference Manual

IBM System/32
SCP Macroinstructions

Programming Information

GC21-5157-0
File No. S32-36

Program Number
5725-SC1

IBM System/32
System Control Programming
Macroinstructions
Reference Manual

Preface

This manual describes the macroinstructions provided by the IBM System/32. The publication is intended for persons who are programming in the Basic Assembler Language or its equivalent and who are familiar with the concept of macroinstructions and system programming for the IBM System/32.

The following topics are discussed in this publication:

- Coding macroinstructions
- Descriptions of the various macroinstructions
- OCL necessary to call the macroinstruction processor
- Error conditions detected by the macroinstruction processor
- A sample program showing how macroinstructions are used

Related Publications

These publications contain information that further describes topics discussed in this manual:

- *IBM System/32 Basic Assembler and Macro Processor Reference Manual*, SC21-7673
- *IBM System/32 Functions Reference Manual*, GA21-9176
- *IBM System/32 System Control Programming Reference Manual*, GC21-7593
- *IBM System/32 System Logic Manual*, SY21-0567
- *IBM System/32 System Data Areas and Diagnostic Aids*, SY21-0532
- *IBM System/32 Data Communications Reference Manual*, GC21-7691

First Edition (March 1977)

This edition applies to version 06, modification 00 of IBM System/32 (Program Number 5725-SC1) and to all subsequent versions and modifications unless otherwise indicated in new editions or technical newsletters.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, refer to the latest *IBM System/32 Bibliography* for the editions that are applicable and current.

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

A Reader's Comment Form is at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. Comments become the property of IBM.

Contents

Chapter 1. Introduction

A macroinstruction is a source statement that causes generation of a predetermined set of assembler statements each time the macroinstruction is used. The System/32 system control program provides macroinstructions that perform both system services and input/output device support. By using these macroinstructions, you can minimize the coding for both system and input/output operations.

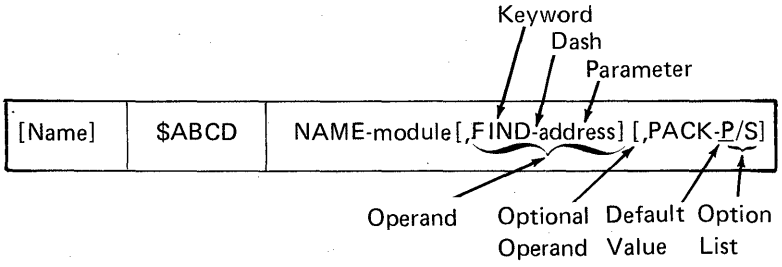
WRITING MACROINSTRUCTIONS

You code macroinstructions as follows:

Starting Column	1	10	16	72
	Name	Operation	Operands	Continuation
	Symbol or blank	Macro name	No operands or one or more separated by commas	Any nonblank characters if continuation is being used

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

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



Operands specify the available services and options. The operands must start in column 16 and are written as follows:

- Each operand consists of a keyword followed by a dash and a parameter.
- Commas separate the operands; no blanks should be left between operands.
- Keywords–those shown in capital letters–are coded exactly as shown. The keyword part of each operand must correspond to one of the keywords in the macroinstruction description.
- The parameter part of the operand must immediately follow the dash.
- Parameters–those shown in lowercase letters–indicate information you must supply. Some operands are not required; these optional operands are enclosed in brackets [KEYWORD-parameter].

  An option list for a keyword parameter is specified as follows:

  KEYWORD-A/B/C

  This list indicates that the keyword has the options A, B, or C. In this example these are the only valid options.

  When the options Y/N are given as parameters in a macroinstruction, Y indicates a yes response, N indicates a no response.

- The operands may be written in any order. If an operand is not specified, the default value is used. The default value is indicated in the macroinstruction description by a line under the default option. For example, [KEY-A/B/C] indicates that option A is the default value.

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

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

```
NAME1    $DTFD ACCESS-CO,RECL-8Ø,NAME-SAMPLE,BLKL-512,CHAIN-NAME2,               X
               RCAD-BUF1,IOAREA-BUF2


NAME2    $DTFP RCAD-BUF1,IOAREA-BUF2                                              X
               PAGE-25,HUC-Y,                                                     X
               CHAIN-NAME3,PRINT-Y,                                               X
               SKIPB-Z,SPACEB-ZH
```

Figure 1. Continuation Coding Examples

```
COMNT1   $DTFO DISK-Y                  THIS INSTR. HAS ONE OPERAND

* THIS COMMENT PERTAINS TO THE NEXT INSTRUCTION OR SERIES OF
* INSTRUCTIONS, THEREFORE, IT IS ENTERED BEFORE THE INSTRUCTION
* OTHERWISE IT WOULD FOLLOW THE MACROINSTRUCTION EXPANSION
* IN THE LISTING

COMNT2   $LMSG FORMAT-Y,HALT-Y,        THIS INSTRUCTION AND THIS                  X
               MIC-1234,OPTNØ-Y        COMMENT ARE CONTINUED
```

Figure 2. Comments on Macroinstructions

SYSTEM CONFIGURATION

The system control programming macroinstructions are provided as a part of the system control program and can be used on any model of the IBM System/32 with the IBM System/32 Basic Assembler and Macro Processor Program Product (Program Number 5725-AS1) or equivalent.

MACROINSTRUCTIONS PROVIDED

The macroinstructions provided by the system control program (SCP) and the functions they perform are shown in Figure 3.

All macroinstructions you want to use must be in the library. You may want to delete some macroinstructions from your library to reduce the amount of disk space required for the macroinstructions. For instance, if your system does not include the binary synchronous communications (BSC) support, the BSC macroinstructions would be of no use to you. You can delete macroinstructions from your library by using the library maintenance utility program, $MAINT. See the *IBM System/32 System Control Programming Reference Manual*, GC21-7593.

Device Type Supported	Macroinstruction Name	Function
System Log	$LMSG	Generate parameter list for halt message on system log
	$LOG	Generate linkage to system log
	$LOGD	Generate offsets in log parameter list
General SCP	$CSLD	Load a function into the control storage increment
	$FIND	Find a directory entry
	$FNDP	Generate parameter list and offsets for $FIND
	$LOAD	Load a module
	$EOJ	End of job
	$COMN	Generate commonly used equates
General I/O	$ALOC	Allocate disk space or device
	$OPEN	Prepare an I/O device
	$CLOS	Prepare a device for controlled termination
	$DTFO	DTF offsets for all devices
Printer	$DTFP	Define the file for disk
	$PUTP	Construct a printer PUT interface
Disk	$DTFD	Define the file for disk
	$GETD	Construct a disk GET interface
	$PUTD	Construct a disk PUT interface
BSC	$DTFB	Define a BSC file
	$GETB	Construct a BSC GET interface
	$PUTB	Construct a BSC PUT interface
	$TRL	Generate a translate parameter list
	$TRTB	Generate a translate table
	$TRAN	Generate an interface to the translate routine
Display Screen/ Keyboard	$DTFS	Define the file for display screen
	$GETS	Construct a keyboard/display screen GET interface
	$PUTS	Construct a display screen PUT interface
	$PGS	Construct a PUT, then a GET request to display screen/keyboard

Figure 3. Macroinstructions

Chapter 2. Macroinstruction Statements

You code macroinstructions to generate a block of assembler statements that perform a certain function. Some functions may be the same each time they are used; others may be modified by different operands specified by the user. This chapter explains the System/32 macroinstructions in detail.

The macroinstructions are grouped in this chapter according to the functions they perform:

- System services
- Input/output support

Input/output support macroinstructions are further divided according to the device supported.

PROGRAMMING CONSIDERATIONS

When you use the macroinstruction processor you should remember the following restrictions:

The generated code for some macroinstructions may alter the contents of register 1 and register 2 depending on the parameters specified. You should save the contents of the register used by the generated code before issuing the macroinstruction; otherwise, the contents may be destroyed. The macroinstruction $TRAN uses register 1. The macroinstruction $CSLD uses registers 1 and 2. These macroinstructions use register 2:

$ALOC	$LOG
$CLOS	$OPEN
$FIND	$PGS
$GETB	$PUTB
$GETD	$PUTD
$GETS	$PUTP
$LOAD	$PUTS

The code generated by the macroinstructions is assigned labels that begin with a dollar sign ($). To avoid duplicate label errors, you should not use a dollar sign as the first character of a label.

SYSTEM SERVICES MACROINSTRUCTIONS

By using system services macroinstructions, you can communicate with the system control program. These macroinstructions can do the following:

- Log and write error messages
- Determine the location of an object module on disk
- Terminate the current job
- Load a function into extended control storage

The system services macroinstructions are divided into two groups:

System log macroinstructions provide support and linkage to the following system log functions:

$LMSG
$LOG
$LOGD

General SCP macroinstructions provide linkage to the following system functions:

$CSLD
$EOJ
$FIND
$FNDP
$LOAD
$COMN

System Log Support

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

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

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

- A formatted message is identified by a 4-character statement that indicates the type and source of an error. Formatted messages reside in a message member.

  For information about building a message member see the *IBM System/32 System Control Programming Reference Manual,* GC21-7593.

- An unformatted message is a statement that can be used to indicate the presence of errors or to issue instructions to the operator; for example, requesting that a disk file be loaded onto disk from diskette. Unformatted messages reside in the user's program.

Messages can be issued with or without an accompanying halt.

Note: A halt must be accompanied by a formatted line and an option.

Two devices can be used as the system log device: the printer or the display screen. If you do not specify a device, the display screen is used when you perform initial program load (IPL). You can change devices by entering a LOG OCL statement in your job stream.

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 equates for the log parameter list.
3. Issue the $LOG macroinstruction.
4. Process the operator's reply in your program.

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

This macroinstruction generates a system log parameter list for a message to the operator.

The format of the $LMSG macroinstruction instruction is:

```
[Name]    $LMSG    [TYPE-1/2] [,MEMBER-code] [,MINOR-code]
                   [,SUBID-code] [,FORMAT-Y/N] [,HALT-Y/N]
                   [,MIC-number] [,OPTN0-Y/N] [,OPTN1-Y/N]
                   [,OPTN2-Y/N] [,OPTN3-Y/N] [,SKIP-Y/N]
                   [,SPACE-1/2/3] [,MSGLN-number]
                   [,MSGAD-address]
```

TYPE-1/2 specifies the type of system log parameter list. TYPE-1 specifies output from a message member; TYPE-2 specifies output from a user program. If this operand is omitted, TYPE-1 is assumed.

The following operands apply when output from a message member is specified (TYPE-1):

MEMBER-code is the first character of the 4-character ID code specifying the program that issued the message. If this operand is omitted, the default is P.

Code	Meaning
C	SCP
D	Data management
I	IOS
R	RPG, FORTRAN, Assembler
V	SCP nucleus
E	SCP linkage editor, word processing, and the overlay linkage editor
U	SCP utilities
S	Sort
F	Data file utility
L	SCP librarian
P	User-defined message access (default)
K	SEU
B	BSC and SNA/SDLC
T	SCP system services
X	MRJE and SCP messages contained in program product message members
H	Heading and miscellaneous text

MINOR-code is the second of the 4-character ID. This code specifies the module within the program. If this operand is omitted, blank is assumed. (Any alphameric character is valid.)

SUBID-code includes the third and fourth characters of the ID. This code is used to further identify the module within the program. If this operand is omitted, blank is assumed. (Any alphameric character is valid.)

Note: MINOR and *SUBID* should be chosen so that the 4 characters of the ID are unique for each module.

FORMAT-Y/N specifies whether or not to include the format line for output from a message member. If this operand is omitted, Y (yes) is assumed.

HALT-Y/N specifies whether or not an operator response is required. If this operand is omitted, Y (yes) is assumed.

Note: The FORMAT parameter must be Y (yes) when HALT parameter is Y (yes); otherwise an error is issued. If HALT-Y is specified, one or more of the OPTN operands must be specified; otherwise an error is issued.

MIC-number is a decimal number from 1 to 9999 used to identify a specific message within the message member. If this operand is omitted, 1 is assumed.

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

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

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

OPTN3-Y/N specifies whether option 3 is allowed. If Y (yes) is specified, option 3 is allowed; if N (no) is specified or if this operand is omitted, option 3 is not allowed.

Note: If option 3 is allowed and selected by the user, control will not be returned to the user's program.

The following operands apply when output from the user program is specified (TYPE-2):

SKIP-Y/N specifies skip to line one of the next page before printing. If this operand is omitted, N (no) is assumed. This operand is valid only for printed messages.

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

MSGLN-number specifies the text length. This entry is a decimal entry from 1 to 132. If this operand is omitted, 001 is assumed.

MSGAD-address specifies the address of the leftmost byte of the text. If this operand is omitted, zeros are assumed.

Generate Displacements for System Log ($LOGD)

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

The format of the $LOGD macroinstruction is:

```
$LOGD
```

Generate the Linkage to the System Log ($LOG)

This macroinstruction generates the linkage required to use the system log function, and checks the response returned. The $LOGD macroinstruction must be used with this macroinstruction to establish offsets in the system log parameter list.

If you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $LOG macroinstruction.

The format of the $LOG macroinstruction is:

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

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

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

OPTN1-address specifies the address of the routine that should receive control if option 1 is the response. If this operand is not specified, no check is made for a response of 1. You would use this operand only if the $LMSG macroinstruction used to generate the system log parameter list was coded *OPTN1-Y*.

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

General SCP Support

The general SCP macroinstructions allow you to provide linkage to system functions by communicating with the system control program.

Find a Directory Entry ($FIND)

The $FIND macroinstruction generates the interface that searches the library directory for the requested module name. If the module name is found, the directory entry is placed in the parameter list; if the name cannot be found, the parameter list remains unchanged.

If you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $FIND macroinstruction.

The format of the $FIND macroinstruction is:

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

PLIST-name is the label of the 14-byte parameter list built by $FNDP. After execution, it contains the directory entry of the module. If this operand is omitted, register 2 is assumed to contain the address of the parameter list.

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

The $FNDP macroinstruction generates a 14-byte parameter list and/or the equates for the displacements into the parameter list. This parameter list is used as input to the supervisor by $FIND.

The format of the $FNDP macroinstruction is:

```
[Name]    $FNDP    [NAME-module] [,V-DC/EQU/ALL] [,TYPE-O/P/R/S]
```

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

V-DC/EQU/ALL specifies whether the DCs, equates, or both are to be generated. If this operand is omitted, *EQU* is assumed.

DC generates a 14-byte parameter list used by the $FIND macroinstruction.

EQU generates the equates for the displacements into the $FIND parameter list.

ALL generates the parameter list and corresponding displacements.

TYPE-O/P/R/S specifies the library member type.

Code	Meaning
O	Load module (default)
R	Subroutine module
S	Source module
P	Procedure module

Load or Fetch a Module ($LOAD)

The $LOAD macroinstruction generates the linkage to load a module into main storage at the address you specify. You may have control returned after the module is loaded, or you may pass control to the module. If you will need to use the data in register 2 at a later time, you should save the contents of register 2 before issuing the $LOAD instruction.

The format of the $LOAD macroinstruction is:

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

PLIST-address is the address used in the previous $FIND macroinstruction. It identifies the directory entry of the module in main storage. If this operand is omitted, the address is assumed to be in register 2.

LOAD-address specifies the address where the module is to be loaded in main storage. If this operand is omitted, the address is assumed to be in the parameter list generated by $FNDP and updated by $FIND.

TYPE-code specifies which program receives control after the requested program is loaded:

LOAD loads the module and returns control to the requesting program.

FETCH loads the module and passes control to the module.

SYSFETCH loads the module and passes control to the module. In addition, the disk and module relocation factors are updated.

Note: If both *LOAD* and *TYPE* operands are omitted, then equates and offsets are generated for the relocation loader parameter list. These equates and offsets are used for RIB values for the relocating loader and for parameter list displacements.

End of Job ($EOJ)

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

The format of the $EOJ macroinstruction is:

```
[Name]    $EOJ
```

COMMON Equates ($COMN)

This macroinstruction generates equates for various labels and values, such as register equates, which may be used by other macroinstructions in the program. This macroinstruction is not required for $CSLD, $DTFB, $DTFD, $DTFO, $DTFP, $DTFS, $EOJ, $LMSG, and $LOGD.

The format of the $COMN macroinstruction is:

```
          $COMN
```

Load a Function into Extended Control Storage ($CSLD)

This macroinstruction provides the linkage to load a function into the control storage increment.

CAUTION
Do not issue this macroinstruction from the transient area because unpredictable results will occur.

The format of the $CSLD macroinstruction is:

```
[Name]    $CSLD    [FUNC-function] [,ERR-address]
```

FUNC-function specifies the function to be loaded. The default is FORTRAN.

ERR-address specifies the address of the user's code that receives control if the function to be loaded cannot be found. If this operand is omitted, the default address is X'0000'.

INPUT/OUTPUT MACROINSTRUCTIONS

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

- General macroinstructions are used with all device types. The following macroinstructions are in this group:

  - $ALOC
  - $CLOS
  - $DTFO
  - $OPEN

- Printer macroinstructions support printer devices. The following macroinstructions are in this group:

  - $DTFP
  - $PUTP

- Disk macroinstructions provide support and linkage to disk data management. The following macroinstructions are in this group:

  - $DTFD
  - $GETD
  - $PUTD

- BSC macroinstructions provide support and linkage to BSC data management. The following macroinstructions are in this group:

  - $DTFB
  - $GETB
  - $PUTB
  - $TRAN
  - $TRL
  - $TRTB

- Display screen/keyboard macroinstructions support the display screen and keyboard devices. The following macroinstructions are in this group:

  - $DTFS
  - $GETS
  - $PGS
  - $PUTS

General I/O Support

The general I/O support macroinstructions are used with all devices. The normal sequence for using these macroinstructions is:

1. $ALOC to allocate the file(s) or device(s) to be used in the user's program
2. $OPEN to prepare the file(s) or device(s) for use in the user's program
3. I/O operations and any processing required
4. $CLOS to prepare the file(s) and/or device(s) for end of job

Allocate Space ($ALOC)

The routines called by the $ALOC macroinstruction allocate various input/output devices and space on the disk for each disk file. These routines check to ensure that:

- The system supports the requested device.
- The device requested is available to the requesting program and/or is capable of multiple allocations.
- The LOCATION parameters of the OCL file statements (if given) are valid.
- Space is available on the disk for the requested data files.
- No more than 52 DTFs are present in the calling program.

An allocate request requires that preopen DTFs be supplied as input to the routine. For a description of DTFs, see $DTFB, $DTFD, $DTFP, and $DTFS. You can allocate more than one DTF 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 has hex FFFF entered in place of the address. If your program contains an interrupt handler, such as a binary synchronous communications program, all DTFs in the program should be chained together and allocated in one operation. When an error condition occurs, the allocate routine calls halt/syslog to display the proper halt code.

If you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $ALOC macroinstruction.

The following output is produced when control is returned to your program:

- The contents of register 1 are saved when ALLOCATE is called, and are restored before control is returned to the user.
- The format-1 labels in the scheduler work area are updated.
- For a nondisk DTF, bit 1 in the second byte of the attribute bytes of the preopen DTF is set on to indicate device allocation.
- The address of the first DTF allocated is returned in register 2.

Note: If your program uses telecommunications, $ALOC must be issued prior to any telecommunications operation.

The format of the $ALOC macroinstruction is:

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

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

Prepare an I/O Device ($OPEN)

This macroinstruction prepares an input/output file for data transfer. The file to be prepared (opened) must previously have been allocated by the allocate macroinstruction. Depending on the device, one or more of the following functions are performed for each file opened:

- The preopen DTF is formatted to a postopen DTF (see Figure 4).
- Preopen DTF information is preserved in the format-1 label as required.
- Data I/O buffers, index I/O buffers, and the IOB(s) are formatted as needed.
- Buffers are initialized as required.
- The index area on disk for indexed files and the data area on disk for direct files are formatted as required.
- Diagnostics are performed to ensure that the access method and the file organization are compatible.

Note: You can open more than one DTF 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 has hex FFFF entered in place of the address. See $DTFB, $DTFD, $DTFP, and $DTFS.

Preopen Conditions	Postopen Conditions
1. Unformatted disk files are present for output files.	1. Formatted disk files are created.
2. The I/O buffer is in the unformatted mode.	2. I/O buffers, IOBs, and various work areas are formatted as required.
	3. A bit is set on in the DTF attribute bytes to indicate an opened file.
	4. Backward chain pointers are built.

Figure 4. Comparison of Preopen and Postopen DTFs and Data Areas

Input: The preopen DTF and format-1 label are input to the open routine. Before the open macroinstruction is issued, you must be sure to have the device allocated by previously issuing the allocate macroinstruction. Also, if you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $OPEN macroinstruction. You must also consider the following in preparing the DTF:

- The disk access method must be compatible with the disk file organization of the file being opened.
- The access method must be compatible with the access method of the same file opened or for an inquiry program.
- The record length, block length, and key length must be specified correctly.

Output: The open routine returns control to your program when the requested file has been opened. The following output is produced:

- The contents of register 1 are saved when OPEN is called, and restored before control is returned to the user.
- The format-1 labels are updated.
- Bit 7 in the second attribute byte in the postopen DTF is set on to indicate that the file has been opened.
- The buffers are initialized as needed.
- The address of the last DTF opened is returned in register 2.

The format of the $OPEN macroinstruction is:

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

DTF-address specifies the address of the leftmost byte of the DTF for the file to be opened. If a series or chain of DTFs is to be opened, this operand specifies the address of the leftmost byte of the first DTF in the chain. If this operand is not entered, it is assumed that the address is in register 2.

Prepare a Device for Termination ($CLOS)

The $CLOS macroinstruction prepares a file(s) or device(s) for job termination. The routine returns postopen DTFs to their preopen state and updates format-1 labels to reflect the current file status. For devices other than disk, only the entries related to the requested functions are restored. If you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $CLOS macroinstruction.

Input to the close routine consists of the postopen DTF and the format-1 labels. The allocate and open macroinstructions must have previously been issued.

Output created by $CLOS is returned to your program when control is returned. The output requires the following operations:

- The contents of register 1 are saved when CLOSE is called, and restored before control is returned to the user.
- The postopen DTFs are reinitialized to the preopen state.
- Any pending I/O operations are performed.
- The format-1 label for disk files is updated to indicate current file status.
- The buffer contents scheduled for disk output or update operations are written.
- The data and index are written to disk if needed, and an indicator is set if key sorting is required at end of job for indexed output files and file additions to indexed files.

Note: You can close more than one DTF at one time by chaining the DTFs. Each DTF to be closed must contain the address of the next DTF in the chain. The last DTF in a chain has hex FFFF entered in place of the address.

The format of the $CLOS macroinstruction is:

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

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

Generate DTF Offsets ($DTFO)

This macroinstruction defines the DTF labels, offsets, field contents, and field lengths for all devices and access methods supported by System/32. To avoid duplicate labels, this macroinstruction should be used only once in each program; you should also set the operands to indicate any devices you plan to use in the program.

The format of the $DTFO macroinstruction is:

```
[Name]    $DTFO    [DISK-Y/N] [,PRT-Y/N] [,BSC-Y/N] [,CRT-Y/N]
                   [,ALL-Y/N] [,FIELD-Y/N]
```

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

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

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

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

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

FIELD-Y/N specifies whether labels are to be generated that define the contents of a DTF field for the devices specified. If this operand is omitted, N (no) is assumed.

Printer Support

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

- Build a preopen DTF for a printer and format its offsets. The DTF provides information to printer data management routines that perform input/output operations.
- Build the interface needed to print data.

Define the File for Printer ($DTFP)

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

The format of the $DTFP macroinstruction is:

```
[Name]    $DTFP    RCAD-address,IOAREA-address [,OVFL-number]
                   [,PAGE-number] [,UPSI-mask] [,HUC-Y/N]
                   [,CHAIN-address] [,PRINT-Y/N] [,RECL-number]
```

RCAD-address is a required operand that gives the address of the leftmost byte of the logical record.

IOAREA-address is a required operand that specifies the address of the leftmost byte of the I/O area. This area must be at least 146 bytes long.

OVFL-number specifies the line on the printer after which the overflow completion code will be returned. If this operand is not specified, default is made to 6 lines less than the number specified for the PAGE operand.

PAGE-number specifies the number of lines to print per page. If this operand is not specified, default is made to the system value for the number of lines per page.

UPSI-mask specifies the settings of the external (SWITCH statement) indicators used for conditionally opening files. The code must be specified as 8 binary bits. For example, to set on bits 0, 3, 5, and 7, you would enter UPSI-10010101. If this operand is not entered, zeros are assumed.

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

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

PRINT-Y/N specifies whether to perform a print operation. Default is N (no), meaning that a print is not performed.

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

Construct a Printer Put Interface ($PUTP)

This macroinstruction generates the interface needed to communicate with printer data management. You must provide a DTF for the file and use the $DTFO macroinstruction to establish the offsets in the DTF. You must also provide, through an EXTRN statement in your program, the label #$BDMC for continuous and noncontinuous forms. (This label is necessary for the printer data management module to perform the printer output operation.)

If you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $PUTP macroinstruction.

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.

The format of the $PUTP macroinstruction is:

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

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

PRINT-Y/N specifies whether to perform a print. If this operand is omitted, the DTF remains unchanged.

SKIPB-number specifies the line to skip to before the print operation. This DTF field is set to zero when the file is opened. If this operand is omitted, the DTF remains unchanged (maximum = 84).

SPACEB-number specifies the number of lines to space before the print operation. This DTF field is set to zero when the file is opened. If this operand is omitted, the DTF remains unchanged (maximum is 3 lines, 3H if half spacing is valid). Half spacing is valid only with the printer set at 6 lines per inch and the half space feature installed.

SKIPA-number specifies the line to be skipped to after the print operation. This DTF field is set to zero when the file is opened. If this operand is omitted, the DTF remains unchanged (maximum = 84).

SPACEA-number specifies the number of lines to space after the print operation. This DTF field is set to zero when the file is opened. If this operand is omitted, the DTF remains unchanged (maximum is 3 lines, 3H if half spacing is valid). Half spacing is valid only with the printer set at 6 lines per inch and the half space feature installed.

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

OVFL-address specifies the address in your program that should receive control if page overflow occurs. If this operand is omitted, no check is made for an overflow condition.

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

Disk Device Support

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

- Build a preopen DTF for disk I/O operations and assign its offsets
- Build the interfaces required to get records from the disk via a get or a read
- Build the interfaces required to put records to the disk via a put or a write

Define the File for Disk ($DTFD)

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

The format of the $DTFD macroinstruction is:

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

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

Access Code	System Module	Access Method
CA	#$CSOP	Consecutive add
CG	#$CSIP	Consecutive input
CO	#$CSOP	Consecutive output
CU	#$CSUP	Consecutive update
DG	#$DAID	Direct input with decimal record numbers
DO	#$DAUD	Direct output with decimal record numbers
DU	#$DAUD	Direct update with decimal record numbers
DGA	#$DAIB	Direct input with binary record numbers
DOA	#$DAUB	Direct output with binary record numbers
DUA	#$DAUB	Direct update with binary record numbers
IA	#$IOAD	Indexed add
IO	#$IOUT	Indexed output
IS	#$ISIP	Indexed sequential input
ISA	#$ISAD	Indexed sequential add
ISU	#$ISUP	Indexed sequential update
ISUA	#$ISUA	Indexed sequential update and add
IR	#$IRIP	Indexed random input
IRA	#$IRAD	Indexed random add
IRU	#$IRUP	Indexed random update
IRUA	#$IRUA	Indexed random update and add
DUMMY		Dummy open to obtain information about how a file was created

Note: You must provide, through an EXTRN statement in your program, the label (under *System Module* column above) for the appropriate access code.

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

NAME-filename specifies the name of the file. The name must be no more than 8 characters in length. This operand must be specified.

BLKL-number specifies the number of bytes in the buffer. The minimum number of bytes is 256. Larger lengths may be specified, but must be in multiples of 256.

IOAREA-address provides the address of the leftmost byte of an area in main storage allocated to contain all buffers and IOB(s) for the access method specified in the ACCESS operand. This operand must be specified. The amount of main storage required is shown in the following chart:

Access Method	Formula
Consecutive, Direct	If the record length is an integral power of 2 (2, 4, 8, 16, 32, 64, 128, . . . ,4096), then: IOAREA = (22 x BUFNO) + [(record length + 255 rounded down to the next multiple of 256) x BUFNO] If the record length is not an integral power of 2 then: IOAREA = (22 x BUFNO) + [(record length + 255 rounded up to the next multiple of 256) x BUFNO]
Indexed	If the record length is an integral power of 2 (2, 4, 8, 16, 32, 64, 128, . . . 4096), then: IOAREA = 22 + [(record length + 255 rounded down to the next multiple of 256) x BUFNO] + 22 + 256 If the record length is not an integral power of 2 then: IOAREA = 22 + [(record length + 255 rounded up to the next multiple of 256) x BUFNO] + 22 + 256

UPSI-mask 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 set on bits 0, 3, 5, and 7, you would enter UPSI-10010101. If this operand is not entered, zeros are assumed.

BUFNO-1/2 allows you to specify either one or two buffers for the file. You can use two buffers only with #$CSIP and #$CSOP consecutive access methods. If this operand is omitted, one buffer is assumed.

LIMIT-Y/N is specified only for indexed sequential get and indexed sequential update. It specifies whether the sequential access is within limits. If this operand is not entered, N (no) is assumed.

ORDLD-Y/N specifies whether an ordered load is to be used with the indexed output access method. This operand can be specified only with the indexed output access method. If this operand is not entered, N (no) is assumed.

CHAIN-address specifies the address of the next DTF in the chain of DTFs. If there is no DTF chain or if this DTF is the last DTF in the chain, this operand should be omitted and hex FFFF assumed.

RCAD-address specifies the address of the leftmost byte of the logical record. If this operand is not entered, hex 0000 is assumed. Depending on the disk access method being used for an input operation, either move mode or locate mode is used. If move mode is used, the record is provided at the address specified in the RCAD parameter. If locate mode is used, the address of the input record is contained at the displacement of $DTFWKB in the DTF and this operand is not used. For information on the mode used by the different access methods, see *IBM System/32 System Logic Manual*, SY21-0567.

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

KDISP-number is entered for all indexed access methods. It indicates the displacement into the record of the leftmost byte of the key field. The displacement of the first byte in the record is zero, the second byte is one, and so on. If this operand is omitted, zero is assumed.

KEYADD-address specifies the following:

- Main storage address of the leftmost byte of the key field for indexed random access methods. This area must be one key length long.

- Main storage address of the leftmost byte of the relative record number field for direct access methods. This area must be 23 bytes when decimal relative record numbers are used, with the relative record number located right-justified in the rightmost 15 bytes of the field. This area must be 8 bytes when binary relative record numbers are used, with the relative record number located right-justified in the rightmost 3 bytes of the field. If omitted, a default address of zero is supplied.

HIGH-address specifies the address of the leftmost byte of the user's save area, two key lengths long, with the low key in the left half and the high key in the right half. This operand is used in conjunction with indexed sequential processing within limits. If omitted, a default address of zero is supplied.

MSTNDX-address specifies the address of the leftmost byte of the master sector index in main storage. This operand can be specified only for indexed random access. If omitted, a default address of zero is supplied. You must allocate space in main storage for the master sector index.

MSTBYT-number specifies the number of bytes reserved for the master sector index. If omitted, the default value is zero. This parameter should be used in conjunction with the MSTNDX parameter.

CURENT-address specifies the address of the leftmost byte of the user's save area for current and last keys for indexed sequential access method. If omitted, a default address of zero is supplied.

Construct a Disk Get Interface ($GETD)

The $GETD macroinstruction generates the interface needed to communicate with disk data management when a record is being read from a disk file. To use this macroinstruction, construct a disk DTF for the file and use the $DTFO macroinstruction to establish the offsets for the DTF. You must also provide the labels for the necessary data management routines through EXTRN statements in your programs. If you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $GETD macroinstruction.

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 tests the completion codes returned by data management.

The format of the $GETD macroinstruction is:

```
[Name]    $GETD    ACCESS-code [,DTF-address] [,ERR-address]
                   [,EOF-address] [,NRF-address]
```

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

Access Code	**System Module**	**Access Method**
CA	#$CSOP	Consecutive add
CG	#$CSIP	Consecutive input
CO	#$CSOP	Consecutive output
CU	#$CSUP	Consecutive update
DG	#$DAID	Direct input with decimal record numbers
DO	#$DAUD	Direct output with decimal record numbers
DU	#$DAUD	Direct update with decimal record numbers
DGA	#$DAIB	Direct input with binary record numbers
DOA	#$DAUB	Direct output with binary record numbers
DUA	#$DAUB	Direct update with binary record numbers
IA	#$IOAD	Indexed add
IO	#$IOUT	Indexed output
IS	#$ISIP	Indexed sequential input
ISA	#$ISAD	Indexed sequential add
ISU	#$ISUP	Indexed sequential update
ISUA	#$ISUA	Indexed sequential update and add
IR	#$IRIP	Indexed random get
IRA	#$IRAD	Indexed random add
IRU	#$IRUP	Indexed random update
IRUA	#$IRUA	Indexed random update and add

Note: You must provide, through an EXTRN statement in your program, the label (under *System Module* column above) for the appropriate access code.

DTF-address indicates the address of the leftmost byte of the DTF for this file. If this operand is not specified, the address is assumed to be in register 2.

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

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

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

Note: If ERR, NRF, or EOF addresses are not specified, your program should check the return code in the DTF to determine the outcome of the operation.

Construct a Disk Put Interface ($PUTD)

The $PUTD macroinstruction generates the interface needed to communicate with disk data management when the program is putting a record to disk or updating a previously retrieved record. You must provide a DTF for the file and use the $DTFO macroinstruction to establish the offsets in the DTF. You must also provide, through EXTRN statements in your program, the labels of the disk data management modules necessary to perform the I/O operation. If you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $PUTD macroinstruction.

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 and control is returned to your program.

The format of the $PUTD macroinstruction is:

```
[Name]    $PUTD    ACCESS-code [,DTF-address] [,ERR-address]
                   [,EOX-address] [,DUPREC-address]
                   [,SEQERR-address] [,KEYERR-address]
                   [,UPDATE-Y/N]
```

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

Access Code	System Module	Access Method
CA	#$CSOP	Consecutive add
CG	#$CSIP	Consecutive input
CO	#$CSOP	Consecutive output
CU	#$CSUP	Consecutive update
DG	#$DAID	Direct input with decimal record numbers
DO	#$DAUD	Direct output with decimal record numbers
DU	#$DAUD	Direct update with decimal record numbers
DGA	#$DAIB	Direct input with binary record numbers
DOA	#$DAUB	Direct output with binary record numbers
DUA	#$DAUB	Direct update with binary record numbers
IA	#$IOAD	Indexed add
IO	#$IOUT	Indexed output
IS	#$ISIP	Indexed sequential input
ISA	#$ISAB	Indexed sequential add
ISU	#$ISUP	Indexed sequential update
ISUA	#$ISUA	Indexed sequential update and add
IR	#$IRIP	Indexed random input
IRA	#$IRAD	Indexed random add
IRU	#$IRUP	Indexed random update
IRUA	#$IRUA	Indexed random update and add

Note: You must provide, through an EXTRN statement in your program, the label (under *System Module* column above) for the appropriate access code.

DTF-address specifies the address of the leftmost byte of the DTF associated with this file. If this operand is not specified, the address is assumed to be in register 2.

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

EOX-address supplies the address in your program that is to receive control when an end of extent is reached during the operation. This operand is not used when UPDATE = Y.

DUPREC-address provides the address in your program that is to receive control when an attempt to add a duplicate record has occurred. This operand is used only with an indexed add access method.

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

KEYERR-address specifies the address of your routine to be called when an attempt has been made to update a record in an indexed file and the attempt would destroy the record key.

UPDATE-Y/N indicates whether an update is to be performed. If this operand is not entered, N (no) is assumed.

Note: If ERR, EOX, DUPREX, SEQERR, or KEYERR addresses are not specified, your program should check the return code in the DTF to determine the outcome of the operation.

Display Screen/Keyboard

This section describes the macroinstructions that support the display screen/keyboard. This support can be grouped in two categories: display support and program function key support. It provides the following capabilities:

- Builds a preopen DTF for the display screen/keyboard data management
- Builds the interface to get a record from the keyboard
- Builds the interface to put a record to the display screen
- Builds the interface to put a record to the display screen, and then to get a record from the keyboard

The display screen/keyboard macroinstructions provide information to the display screen/keyboard data management routines that perform the input/output operations.

Display Support

Define the File for Display Screen/Keyboard ($DTFS)

The $DTFS macroinstruction provides information needed to allocate, open, access, and close a display screen/keyboard file. This macroinstruction generates the code that builds a display screen/keyboard DTF.

The format of the $DTFS macroinstruction is:

```
[Name]    $DTFS    [PUTDAT-address] [,PUTLOC-number] [,UPSI-mask]
                   [,CHAIN-address] [,PUTLEN-number] [,OPC-code]
                   [,GETDAT-address] [,GETLOC-number]
                   [,GETLEN-number] [,FUNKEY-number]
                   [,CMDKEY-number] [,SHIFT-A/N] [,CURSOR-number]
                   [,SPACE-number] [,WAIT-Y/N] [,IOBST-address]
```

PUTDAT-address specifies the leftmost byte of the logical record for a put request. For a $PGS request, this area is used for the output. If this operand is not specified, hex 0000 is assumed, and the address must be updated with (or prior to) the first $PGS or $PUTS request issued.

PUTLOC-number specifies a number that represents the starting location on the display screen for a put request. Valid entries for this operand are from 1 through 240. If this operand is not specified, 1 (the first display screen position) is assumed. If the number exceeds 240, no data is written.

UPSI-mask specifies the settings of the external (SWITCH statement) indicators used for conditionally opening the file. The mask must be specified as 8 binary bits. For example, to set on bits 0, 3, 5, and 7, you would enter UPSI-10010101. If this operand is not specified, zeros are assumed for all 8 bits.

CHAIN-address specifies the address of the next DTF in the DTF chain. If there is no DTF chain or if this DTF is the last one in the chain, this operand should be omitted and end of chain (hex FFFF) assumed.

PUTLEN-number specifies the number of bytes to process for a put request. If this operand is not specified, the missing information must be supplied with (or prior to) the first $PGS or $PUTS request. Valid entries for this operand are from 1 through 240. If this number plus the entry for the PUTLOC operand exceeds 241, the data written is truncated at location 240.

OPC-code specifies the operation code to be set. If this operand is not specified, the information is supplied with the first $GETS, $PUTS, or $PGS request. The codes and their meanings are:

Code	Meaning
OUTPUT	Display prompt or data on display screen
BDE	Basic data entry
SDE	Sequential data entry
CSDE	Controlled sequential data entry

GETDAT-address specifies the leftmost byte of the area into which the input data will be placed for a get request; for a $PGS request, this area is used for the input. If this operand is not specified, hex 0000 is assumed, and the information must be supplied with (or prior to) the first $GETS or $PGS request issued.

GETLOC-number specifies a number that indicates the starting location on the display screen for a get request. Valid entries for this operand are 1 through 240. If this operand is not specified, 1 (the first display screen position) is assumed. If a number greater than 240 is specified, no data will be read.

GETLEN-number is a decimal number that represents the number of bytes to get. If this operand is not specified, hex 0000 is assumed, and the missing information must be supplied with (or prior to) the first $PGS or $GETS request issued. Valid entries for this operand are 1 through 240. If this number plus the entry specified for the GETLOC operand exceeds 241, the data read is truncated after location 240.

FUNKEY-number is a 3-byte hex number you can use to redefine the use of the function keys for your program. If this operand is omitted, hex 000000 is assumed. For further information see *IBM System/32 Functions Reference Manual*, GA21-9176.

CMDKEY-number is a 3-byte hex number you can use to identify the command keys acceptable as input for your program. If this operand is omitted, hex 000000 is assumed. For further information see *IBM System/32 Functions Reference Manual*, GA21-9176.

SHIFT-A/N indicates whether your input can be alphameric or numeric only. If this operand is omitted, A is assumed.

CURSOR-number specifies the cursor position within the display screen. 1 identifies the first position, 2 identifies the second position, etc. If zero is entered or the operand is omitted, the cursor is not displayed.

SPACE-number specifies the number of lines the display screen should be rolled before input is accepted from the keyboard. Valid entries are 0 through 6. If this operand is omitted, zero is assumed.

WAIT-Y/N–Y (yes) waits for keyboard input before returning control to the user. N (no) returns control to the user before keyboard input completion is checked in the first call to data management, and the second call to data management waits for completion of the keyboard input before returning control to the user. If this operand is omitted, Y (yes) is assumed.

IOBST-address specifies the address of the leftmost byte of the keyboard IOB. If this operand is omitted, the default address is hex 0000.

Get a Record from the Keyboard ($GETS)

The $GETS macroinstruction generates the interface needed to communicate with display screen data management when a record is being read from the display screen. To use this macroinstruction, construct a display screen DTF for the file and use the $DTFO macroinstruction to establish the offsets for the DTF. You must include an EXTRN for #$BDMC. If you will need to use the data in register 2 at a later time, you should save the contents of that register before using the $GETS macroinstruction.

The format for the $GETS macroinstruction is:

```
[Name]    $GETS    [DTF-address] [,GETDAT-address] [,GETLEN-number]
                   [,GETLOC-number] [,OPC-code] [,FUNKEY-number]
                   [,CMDKEY-number] [,SHIFT-A/N] [,WAIT-Y/N]
                   [,CURSOR-number] [,SPACE-number]
```

DTF-address specifies the leftmost byte of the DTF for this file. If this operand is not specified, the address of the DTF is assumed to be in register 2.

GETDAT-address specifies the leftmost byte of the area into which the data will be placed. If this operand is omitted, the current address in the DTF remains unchanged.

GETLEN-number specifies the number of bytes to get. Valid entries for this operand are 1 through 240. If the sum of this number plus the number specified for the GETLOC operand exceeds 241, the data read is truncated after location 240. If this operand is omitted, the current length in the DTF remains unchanged.

GETLOC-number specifies a number representing the starting location on the display screen for this get. Valid entries for this operand are 1 (the first display screen position) through 240. If this entry exceeds 240, no data is read. If this operand is omitted, the current location in the DTF remains unchanged.

OPC-code specifies the operation code to be set. If this operand is omitted, basic data entry (BDE) is assumed.

Code	Meaning
BDE	Basic data entry
SDE	Sequential data entry
CSDE	Controlled sequential data entry

FUNKEY-number is a 3-byte hex number used to redefine the use of the function keys. If this operand is omitted, the function key mask in the DTF remains unchanged.

CMDKEY-number is a 3-byte hex number used to identify the command keys acceptable as input. If this operand is omitted, the command key mask in the DTF remains unchanged.

SHIFT-A/N indicates whether the input is alphameric or numeric only. If this operand is omitted, the shift indicator in the DTF remains unchanged.

WAIT-Y/N–Y (yes) waits for keyboard input before returning control to the user. N (no) returns control to the user before keyboard input completion is checked in the first call to data management, and the second call to data management waits for completion of the keyboard input before returning control to the user. If this operand is omitted, Y (yes) is assumed.

CURSOR-number specifies the cursor position within the display screen (maximum = 240). If this operand is omitted, the cursor position in the DTF remains unchanged.

SPACE-number specifies the number of lines the display screen should be rolled before input is accepted (maximum = 6). If this operand is omitted, the space count in the DTF remains unchanged.

Generate a PUT/GET Operation through Display Screen Data Management ($PGS)

This macroinstruction generates a PUT/GET data request to display screen data management. To use this instruction, you must construct a display screen DTF for the file and use the $DTFO macroinstruction to establish the offsets in the DTF. You must also provide the labels for the necessary data management routines through an EXTRN for #$BDMC. If you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $PGS macroinstruction.

The format for the $PGS macroinstruction is:

```
[Name]    $PGS    [DTF-address] [,OPC-code] [,PUTDAT-address]
                  [,PUTLEN-number] [,PUTLOC-number]
                  [,GETDAT-address] [,GETLEN-number]
                  [,GETLOC-number] [,FUNKEY-number]
                  [,CMDKEY-number] [,SHIFT-A/N] [,CURSOR-number]
                  [,SPACE-number] [,WAIT-Y/N]
```

DTF-address specifies the address of the DTF for this file. If the operand is not specified, the address is assumed to be in register 2.

OPC-code specifies the operation code to be set. If this operand is omitted, basic data entry (BDE) is assumed:

Code	**Meaning**
BDE	Basic data entry
SDE	Sequential data entry
CSDE	Controlled sequential data entry

PUTDAT-address identifies the leftmost byte of the user area from which the data will be taken. If this operand is omitted, the current address in the DTF remains unchanged.

PUTLEN-number specifies the number of bytes to put to the display screen. Valid entries for this operand are 1 through 240. If the sum of this number plus the entry specified for the PUTLOC operand exceeds 241, the data written is truncated after location 240. If this operand is omitted, the current length in the DTF remains unchanged.

PUTLOC-number specifies the starting location on the screen for this put request. Valid entries for this operand are 1 (the first display screen position) through 240. If this number exceeds 241, no data is written. If this operand is omitted, the current location in the DTF remains unchanged.

GETDAT-address specifies the leftmost byte of the area into which the data will be placed for a get request. If this operand is omitted, the current address in the DTF remains unchanged.

GETLEN-number specifies the number of bytes to get from the display screen. Valid entries for this operand are 1 through 240. If the sum of this number plus the entry specified for the GETLOC operand exceeds 241, the data is truncated after location 240. If this operand is omitted, the current length in the DTF remains unchanged.

GETLOC-number specifies the starting location on the display screen for this get request. Valid entries for this operand are 1 (the first display screen position) through 240. If this number exceeds 240, no data is read. If this operand is omitted, the current location in the DTF remains unchanged.

FUNKEY-number is a 3-byte hex number used to redefine the use of the function keys. If this operand is omitted, the function key mask in the DTF remains unchanged.

CMDKEY-number is a 3-byte hex number used to identify the command keys acceptable as input. If this operand is omitted, the command key mask in the DTF remains unchanged.

SHIFT-A/N indicates whether the input is alphameric or numeric only. If this operand is omitted, the shift indicator in the DTF remains unchanged.

CURSOR-number specifies the cursor position within the display screen (maximum = 240). If this operand is omitted, the cursor positon in the DTF remains unchanged.

SPACE-number specifies the number of lines the display screen should be rolled before accepting input (maximum = 6). If this operand is omitted, the space count in the DTF remains unchanged.

WAIT-Y/N–Y (yes) waits for the completion of keyboard input before returning control to the user. N (no) returns control to the user before keyboard input completion is checked in the first call to data management, and the second call to data management waits for completion of the keyboard input before returning control to the user. If this operand is omitted, Y (yes) is assumed.

Note: If the operands PUTDAT, PUTLOC, PUTLEN, GETDAT, GETLOC, or GETLEN are not specified, you must supply the missing information in the DTF before issuing the first $PGS request.

Put a Record to the Display Screen via Data Management ($PUTS)

This macroinstruction generates a put data request to display screen data management. To use this macroinstruction, you must construct a display screen DTF for the file and use the $DTFO macroinstruction to establish the offsets in the DTF.

If you will need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $PUTS macroinstruction. You must also provide the labels for the necessary data management routines through an EXTRN to #$BDMC.

The format for the $PUTS macroinstruction is:

```
[Name]    $PUTS    [DTF-address] [,PUTDAT-address] [,PUTLOC-number]
                   [,PUTLEN-number] [,SPACE-number]
```

DTF-address specifies the address of the DTF for this file. If this operand is not specified, the address is assumed to be in register 2.

PUTDAT-address specifies the leftmost byte of the area from which the data will be taken. If this operand is omitted, the current address in the DTF remains unchanged.

PUTLOC-number specifies the starting location on the display screen for this put. Valid entries for this operand are 1 (the first display screen position) through 240. If this number exceeds 240, no data is written. If this operand is omitted, the current location in the DTF remains unchanged.

PUTLEN-number specifies how many bytes to put to the display screen. Valid entries for this operand are 1 through 240. If the sum of this number plus the entry specified for the PUTLOC operand exceeds 241, the data written is truncated at location 240. If this operand is omitted, the DTF remains unchanged.

SPACE-number specifies the number of lines the display screen should be rolled before accepting input (maximum = 240). If this operand is omitted, the space count in the DTF remains unchanged.

Note: If the operands PUTDAT, PUTLOC, or PUTLEN are missing, the missing information must be supplied in the DTF before the first put request is issued.

BSC Support

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

- Build a DTF for BSC GET/PUT operations and its offsets
- Build the interface to get a BSC record
- Build the interfaces required to translate data from ASCII to EBCDIC or EBCDIC to ASCII
- Build the interface to put a BSC record

Define the File for BSC ($DTFB)

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 is:

```
[Name]    $DTFB    RECL-number,RCAD-address,BLKL-number,
                   FTYP-RCV/TSM [,BUFNO-1/2] [,BUFST-address]
                   [,IOBST-address] [,TYPE-PP/AA/MA/MP/MC]
                   [,CODE-E/A] [,UPSI-mask] [,CHAIN-address]
                   [,ITB-Y/N] [,TRANSP-Y/N] [,RVIADR-address]
                   [,RVIMSK-code] [,DLYCT-number] [,RCVID-address]
                   [,RCVCT-number] [,SNDID-address] [,SNDCT-number]
                   [,TERMAD-number]
```

RECL-number specifies, in decimal, the maximum record length for this file, excluding line control characters. Record length is limited by available storage and terminal characteristics.

RCAD-address specifies the symbolic address identifying 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 ($PUTB macroinstruction), and moved from the BSC I/O buffers to the logical buffer on GET requests ($GETB macroinstruction).

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

FTYP-RCV/TSM indicates whether the first operation for this file is receive (RCV) or transmit (TSM). If you define a receive file (RCV), the first I/O request for the file must be a GET request; if you define a transmit file (TSM), the first I/O request for the file must be a PUT request or a request for an online test.

BUFNO-1/2 specifies the number of I/O buffers and IOBs to be contained in the I/O area for this file. If this operand is omitted, one is assumed.

BUFST-address specifies the address of the leftmost byte of the I/O buffer. If BUFST is omitted, the $DTFB macroinstruction generates a name and buffer area. If shared buffering between $DTFBs is desired, the user must supply the buffer via the BUFST parameter. The buffer should be large enough to satisfy the requirements of the $DTFB needing the largest area. The area needed can be calculated as follows:

(buffer length)*(number of buffers) 1 or 2 buffers allowed

buffer length = BLKL + 21 + (number of ITB characters)
number of ITB characters = ((BLKL/RECL)-1)*(ITB count)
ITB count =1 for ITB nontransparent
3 for ITB transparent receive
0 for non-ITB

IOBST-address of the leftmost byte of the IOB. In a *one IOB* DTF, this address must point to a 22-byte area. In a *two IOB* DTF, this address must point to a 44-byte area. If IOBST is omitted, the $DTFB macroinstruction generates a name and IOB area.

Note: IOBST must address a different IOB area for each $DTFB.

TYPE-PP/AA/MA/MP/MC: specifies the type of line connection to be established for this file. You must have the appropriate network attachment feature installed before specifying one of the following line types:

PP specifies that this file will use a point-to-point nonswitched line. PP is assumed if no line type is specified.

AA specifies that this file will use a switched line, auto answer.

MA specifies that this file will use a switched line, manual answer.

MP specifies that this file will use a multipoint line, and that this station is a tributary station. TYPE-MP requires the TERMAD operand.

MC specifies that this file will use a switched line, manual call.

CODE-E/A specifies whether the character code of your data is EBCDIC (E) or ASCII (A). If this operand is omitted, E is assumed.

UPSI-mask 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 set on bits 0, 3, 5, and 7, you would enter UPSI-10010101. If this operand is omitted, zeros are assumed.

CHAIN-address specifies the symbolic address of the next DTF in the chain. Chained DTFs are allocated, opened, or closed at the same time as the first DTF in the chain. An end-of-chain indicator, hex FFFF, is entered in the last DTF, or in a DTF if no chain operation is needed.

ITB-Y/N 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 (no) is assumed.

TRANSP-Y/N 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 (no) is assumed.

RVIADR-address 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-code specifies 2 hexadecimal characters to represent the reverse interrupt (RVI) mask. The bits represented by the mask are set on by IOS in the RVIADR field (preceding paragraph) if reverse interrupt request (RVI) is received.

DLYCT-number specifies a decimal delay count. The delay count is the number of seconds after receiving or transmitting a block of data that System/32 will wait for you to receive or transmit another block of data for the same file (1-999). If you do not specify a number, a 180-second delay count is assumed. If you do not specify a delay count, consider the time that may be required for such things as device errors, halts, and readying I/O devices.

RCVID-address specifies the symbolic address of the leftmost byte of the identification sequence required from the remote station. RCVID-address requires the RCVCT operand. Using RCVID and RCVCT improves data security on switched lines; these operands are valid for switched lines only.

RCVCT-number specifies, in decimal, the length of the identification sequence required from the remote station. Length can be from 1 to 15. If 1 is specified, IOS expects to receive 2 characters–duplicates of the character addressed by the RCVID operand (preceding paragraph). If no length is specified, 0 is assumed. RCVCT-number requires the RCVID operand.

SNDID-address specifies the symbolic address of the leftmost byte of the identification sequence required by the remote station. SNDID-address requires the SNDCT operand. Using the SNDID and SNDCT operands improves data security on switched lines; these operands are valid for switched lines only.

SNDCT-number 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, IOS transmits 2 characters–duplicates of the character addressed by the SNDID operand (preceding paragraph). SNDCT-number requires the SNDID operand.

TERMAD-number 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/or addressing characters. The TERMAD operand is used only when TYPE-MP is specified. For further information about polling and/or addressing characters see *IBM System/32 Data Communications Reference Manual*, GC21-7691.

Issue a GET Request ($GETB)

The $GETB macroinstruction generates code to move data from an IOS buffer to your logical buffer. To use this macroinstruction, construct a BSC DTF for the file and use the $DTFO macroinstruction to generate the labels and establish the offsets for the DTF. You must also provide, through an EXTRN statement in your program, the label #$BSDB for BSC data management.

The format of the $GETB macroinstruction is:

```
[Name]    $GETB    [DTF-address] [,REJECT-address] [,EOF-address]
```

DTF-address specifies the address of the DTF (file) for which the GET was issued. If this operand is omitted, the address of the DTF is assumed to be in register 2.

EOF-address specifies the user's end-of-file routine. If this operand is omitted, control is returned to the caller at the next sequential instruction after the $GETB.

REJECT-address specifies the routine to receive control if this GET request is rejected by BSC. If this operand is omitted, control is returned to the caller at the next sequential instruction after the $GETB.

If EOF or REJECT addresses are not specified, you should check the return code in your program to determine the outcome of the operation.

Issue a PUT Request ($PUTB)

The $PUTB macroinstruction generates code to move data from your logical buffer to an IOS buffer. To use this macroinstruction, construct a BSC DTF for the file and use the $DTFO macroinstruction to generate the labels and establish the offsets for the DTF. You must also provide, through an EXTRN statement in your program, the label #$BSDB for BSC data management.

The format of the $PUTB macroinstruction is:

```
[Name]    $PUTB    [DTF-address] [,REJECT-address]
```

DTF-address specifies the address of the DTF (file) for which the PUT was issued. If this operand is omitted, the address is assumed to be in register 2.

REJECT-address specifies the routine to receive control if the PUT request is rejected by BSC. If this operand is omitted, control is returned to the caller at the next sequential instruction after the $PUTB, and the return codes should be checked to determine the outcome of the operation.

Generate a Translate Parameter List ($TRL)

This macroinstruction generates a parameter list used by the translate routine. This list is specified in the $TRAN macroinstruction. $TRL does not generate executable code. Figure 4 shows the format of the translate parameter list.

Translate Routine Operation

To use the translate routine, you must provide a translate control area. To construct a translate control area you can use the $TRTB macroinstruction. The format of the area is:

Byte	Field Description
0	Byte contents used to identify an invalid character (character is not to be translated)
1	Byte contents substituted for characters that are not to be translated
2-257	256-byte translate table for EBCDIC to ASCII
2-129	128-byte translate table for ASCII to EBCDIC

The translate routine processes a field, specified by the $TRAN macroinstruction, 1 byte at a time.

The translate table must be constructed so that the value the character is to be translated to is located at the displacement (from the beginning of the table) equal to the hexadecimal representation of the untranslated character. (For example, if you want to translate hex C1 to hex 41, you should construct a translate table in which the value at displacement hex C1 in the table is hex 41.)

The contents of the byte at a given displacement are compared with the contents of the first byte in the translate area (byte 0). If an equal compare results, 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 hexadecimal value in the second byte of the translate area (byte 1) is substituted for the original character.
- Translation continues with the next character.

The format of the $TRL macroinstruction is:

```
[Name]    $TRL      TO-address,FROM-address,LEN-number,TRT-address
```

TO-address specifies the symbolic address of the leftmost byte of the data field to which the translated data will be moved.

FROM-address 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-number specifies the decimal length of the number of characters to be translated.

TRT-address specifies the symbolic address of the leftmost byte of the translate control area. If the $TRTB macroinstruction is used to generate the translate control area, this address should be the label assigned to the $TRTB macroinstruction.

All four operands are required.

Generate a Translate Table ($TRTB)

This macroinstruction generates an EBCDIC to ASCII or an ASCII to EBCDIC translate table. The table is generated in the format required by the $TRL macroinstruction, and can be addressed by $TRL when you translate data.

The format of the $TRTB macroinstruction is:

```
[Name]    $TRTB     [CODE-E/A] [,HEX-hex]
```

CODE-E/A specifies whether the data is to be translated from EBCDIC to ASCII (E) or ASCII to EBCDIC (A). If this operand is omitted, EBCDIC (E) is assumed. If CODE-E is specified, $TRTB generates a 258-byte control area; if CODE-A is specified, $TRTB generates a 130-byte control area.

HEX-hex specifies the hexadecimal pattern with which to replace any invalid characters found during translation. If the HEX operand is not specified, the replacement character is hex 3F for EBCDIC or hex 1A for ASCII.

Generate an Interface to the Translate Routine ($TRAN)

This macroinstruction generates an interface to the translate routine. After the translate routine has finished, control is returned to your program with a completion code in the translate routine parameter list. The address of the parameter list is in register 1. If you will need to use the data in register 1 at a later time, you should save the contents of the register before issuing the $TRAN instruction. You should check the completion code to see whether any invalid characters were encountered.

The format of the $TRAN macroinstruction is:

```
[Name]    $TRAN    [TRL-address]
```

TRL-address specifies the symbolic address of the translate parameter list. If this operand is not entered, 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 below:

Field Length	Field Description
2	Address of the translate control area (your program must define the translate control area)
2	FROM field address, for translation
2	TO field address, for translation
2	Number of bytes to translate
1	Completion code: Hex 00 = translation complete, no errors Hex FF = invalid character encountered

Figure 5. Translate Parameter List

Chapter 3. OCL and Sample Program

OCL FOR MACRO PROCESSOR

OCL statements used to call the macro processor can be entered through the system input device or be called as a procedure from the source library. The OCL statements necessary to call the macro processor are shown below. The COMPILE statement shown is only necessary when input is in a source library.

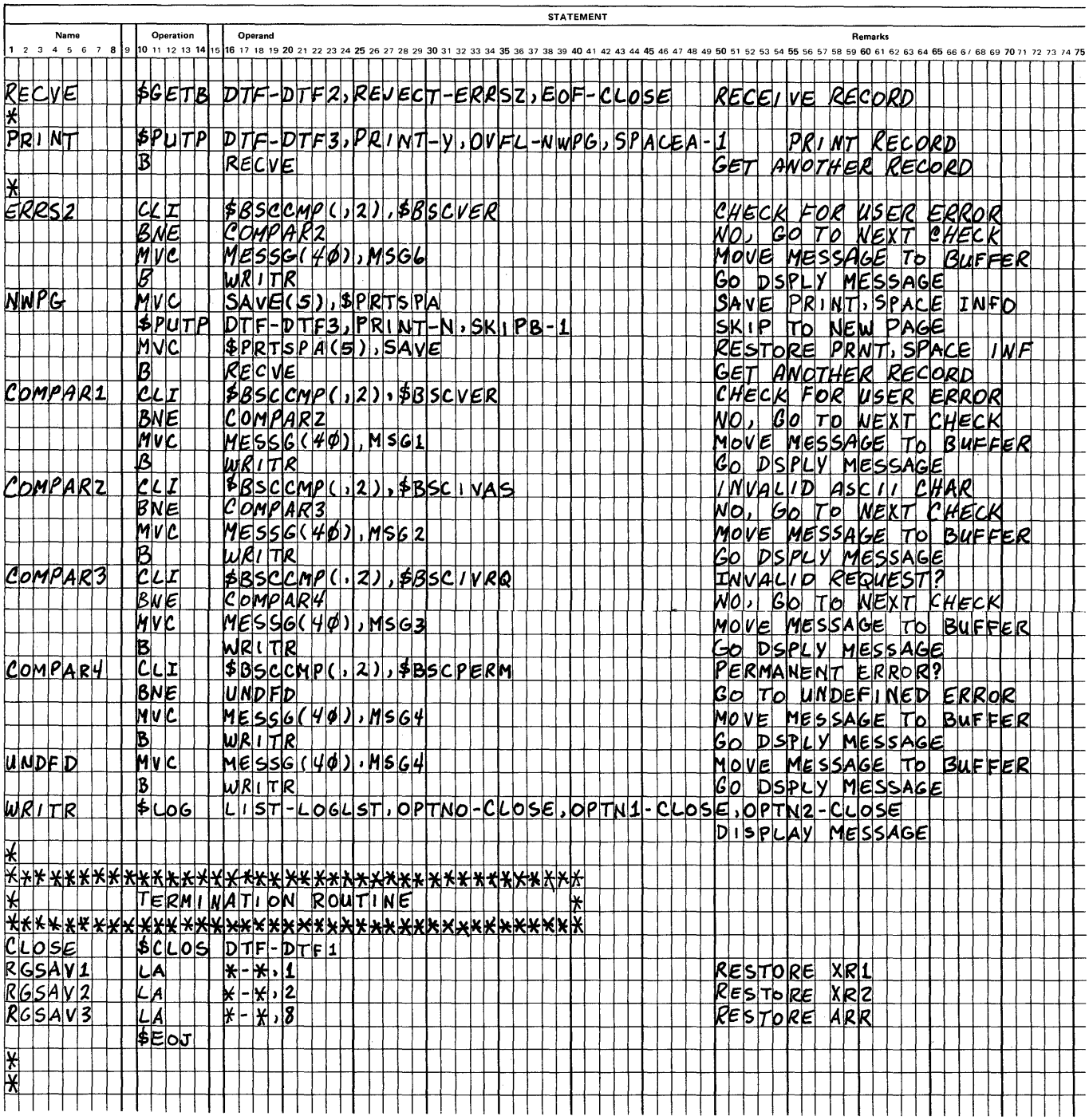
```
// LOAD #MPXDV
// FILE NAME-$SOURCE,RETAIN-S,BLOCKS-30
// FILE NAME-$ASMINPT,RETAIN-S,UNIT-F1,LABEL-$ASMINPT,BLOCKS-60
// FILE NAME-$WORK,RETAIN-S,BLOCKS-30
// FILE NAME-$WORK2,RETAIN-S,BLOCKS-20
// MEMBER PROGRAM1-MPASMSG1
// MEMBER PROGRAM2-MPASMSG2
// COMPILE SOURCE-INFILE
// RUN
```

SAMPLE PROGRAM 1

This sample program shows how macroinstructions can be used to write a program that includes binary sychronous communication, error logging, and printing:

```
         TITLE 'SAMPLE PROGRAM'          THIS STATEMENT APPEARS ON
*                                        EVERY PAGE
SAMPLE   START 0                         PROGRAM ASSEMBLY STARTS HERE
*
****************************************
*        EXTERNAL REF FOR D/M          *
****************************************
*
         EXTRN #$BSDB                    EXTERNAL REF FOR BSC D/M
*
         EXTRN #$BDMC                    EXTERNAL REF FOR PRINTER D/M
*
****************************************
*            COMMON EQUATES            *
****************************************
*
         $COMN                           GEN COMMON LABELS
*
         $DTFO ALL-Y,FIELD-Y             GEN ALL DEVICE LABELS & OFFSETS
*
         $LOGD                           GEN SYSLOG LABELS & OFFSETS
*
****************************************
*           SAVE ENVIRONMENT           *
****************************************
BEGIN    ST    RGSAV1+3,1                SAVE XR1
         ST    RGSAV2+3,2                SAVE XR2
         ST    RGSAV3+3,8                SAVE ARR
****************************************
*       ALLOCATE AND OPEN FILES        *
****************************************
         $ALOC DTF-DTF1
         $OPEN DTF-DTF1
****************************************
*      SEND,RECEIVE,PRINT ROUTINE      *
****************************************
SEND     $PUTB DTF-DTF1,REJECT-COMPAR1       SEND RECORD
         AZ    MYREC+25(4),ONE(4)            ADD 1 TO RECORD COUNT
         CLC   MYREC+25(4),MAX               CHECK FOR MAX TO BE SENT
         BL    SEND                          IF LOW SEND ANOTHER RECORD
*
```

Sample Program 1 (Part 1 of 3)



```
STATEMENT
Name | Operation | Operand | Remarks
RECVE    $GETB   DTF-DTF2,REJECT-ERRSZ,EOF-CLOSE   RECEIVE RECORD
*
PRINT    $PUTP   DTF-DTF3,PRINT-Y,OVFL-NWPG,SPACEA-1   PRINT RECORD
         B       RECVE                              GET ANOTHER RECORD
*
ERRS2    CLI     $BSCCMP(,2),$BSCVER                CHECK FOR USER ERROR
         BNE     COMPAR2                            NO, GO TO NEXT CHECK
         MVC     MESSG(40),MSG6                     MOVE MESSAGE TO BUFFER
         B       WRITR                              GO DSPLY MESSAGE
NWPG     MVC     SAVE(5),$PRTSPA                    SAVE PRINT,SPACE INFO
         $PUTP   DTF-DTF3,PRINT-N,SKIPB-1           SKIP TO NEW PAGE
         MVC     $PRTSPA(5),SAVE                    RESTORE PRNT,SPACE INF
         B       RECVE                              GET ANOTHER RECORD
COMPAR1  CLI     $BSCCMP(,2),$BSCVER                CHECK FOR USER ERROR
         BNE     COMPAR2                            NO, GO TO NEXT CHECK
         MVC     MESSG(40),MSG1                     MOVE MESSAGE TO BUFFER
         B       WRITR                              GO DSPLY MESSAGE
COMPAR2  CLI     $BSCCMP(,2),$BSCIVAS               INVALID ASCII CHAR
         BNE     COMPAR3                            NO, GO TO NEXT CHECK
         MVC     MESSG(40),MSG2                     MOVE MESSAGE TO BUFFER
         B       WRITR                              GO DSPLY MESSAGE
COMPAR3  CLI     $BSCCMP(,2),$BSCIVRQ               INVALID REQUEST?
         BNE     COMPAR4                            NO, GO TO NEXT CHECK
         MVC     MESSG(40),MSG3                     MOVE MESSAGE TO BUFFER
         B       WRITR                              GO DSPLY MESSAGE
COMPAR4  CLI     $BSCCMP(,2),$BSCPERM               PERMANENT ERROR?
         BNE     UNDFD                              GO TO UNDEFINED ERROR
         MVC     MESSG(40),MSG4                     MOVE MESSAGE TO BUFFER
         B       WRITR                              GO DSPLY MESSAGE
UNDFD    MVC     MESSG(40),MSG4                     MOVE MESSAGE TO BUFFER
         B       WRITR                              GO DSPLY MESSAGE
WRITR    $LOG    LIST-LOGLST,OPTNO-CLOSE,OPTN1-CLOSE,OPTN2-CLOSE
                                                    DISPLAY MESSAGE
*
**********************************
*        TERMINATION ROUTINE     *
**********************************
CLOSE    $CLOS   DTF-DTF1
RGSAV1   LA      *-*,1                              RESTORE XR1
RGSAV2   LA      *-*,2                              RESTORE XR2
RGSAV3   LA      *-*,8                              RESTORE ARR
         $EOJ
*
*
```

Sample Program 1 (Part 2 of 3)

STATEMENT			
Name (1–8)	Operation (10–14)	Operand (16–)	Remarks (55–71)

```
****************************************
*        DECLARATIVE CODE              *
****************************************
*
DTF1     $DTFB RECL-8Ø,BLKL-8Ø,RCAD-MYREC,FTYP-TSM,              DEFINE BSC FILE  X
               CHAIN-DTF2,TRANSP-Y,BUFNO-2                       FOR SEND
*
DTF2     $DTFB RECL-8Ø,BLKL-8Ø,RCAD-MYREC,FTYP-RCV,              DEFINE BSC FILE  X
               CHAIN-DTF3,TRANSP-Y,BUFNO-2                       FOR RECEIVE
*
DTF3     $DTFB RCAD-MYREC,IOAREA-PRTBUF,PRINT-Y,                 DEFINE PRINT     X
               RECL-8Ø,PAGE-66,OVFL-6Ø                           FILE
*
LOGLST   $LMSG TYPE-2,MSGLN-4Ø,MSGAD-MESAREA                     GEN SYSLOG PARM
*
MYREC    EQU   *
         DC    CL8Ø'   TEST DATA     RECORD ØØØ1'                LOGICAL REC BUF
*
MAX      DC    CL4'Ø2Ø1'
*
ONE      DC    CL4'ØØØ1'
*
MSG1     DC    CL4Ø'TEST - FILE NOT OPEN OR NOT A PUT FILE  '
*
MSG2     DC    CL4Ø' TEST - INVALID ASCII CHARACTER         '
*
MSG3     DC    CL4Ø' TEST - INVALID REQUEST                 '
*
MSG4     DC    CL4Ø' TEST - PERMANENT ERROR                 '
*
MSG5     DC    CL4Ø' TEST - UNDEFINED ERROR                 '
*
MSG6     DC    CL4Ø' TEST - FILE NOT OPEN OR NOT A GET FILE '
*
MESAREA  EQU   *
MESSG    DS    CL4Ø                                              MESSAGE BUFFER
*
PRTBUF   EQU   *
         DC    CL146'4Ø'                                         PRINTER I/O AREA
*
SAVE     DS    CL5                                               SAVE AREA - DTFP
*
         END   BEGIN
```

Sample Program 1 (Part 3 of 3)

SAMPLE PROGRAM 2

This sample program shows how macroinstructions can be used to write a subroutine to print a record:

```
***************************************
*        COMMON EQUATES               *
***************************************
*
         $COMN                        GEN COMMON LABELS
*
         $DTFO PRT-Y,FIELD-Y          GEN PRNTR LABELS & OFFSETS
*
***************************************
*   ALLOCATE AND OPEN PRINTER FILE    *
***************************************
*
         $ALOC DTF-DTFPT              ALLOCATE PRT FILE
         $OPEN DTF-DTFPT              OPEN PRT FILE
         B     PRTNOERR
*
***************************************
*        TERMINATION ROUTINE          *
***************************************
*
EOJ      $CLOS DTF-DTFPT              CLOSE PRT FILE
         $EOJ                         RETURN CONTROL TO SUPERVISOR
*
***************************************
*        PRINT SUBROUTINE             *
***************************************
*
PRTNOERR EQU   *
         ST    RTN1+3,$ARR            SAVE RETURN ADDRESS
         MVC   USINGREC+40(28),NOERRS MOVE DATA TO BUFFER
         $PUTP DTF-DTFPT,PRINT-Y,SPACEA-1
         MVC   USINGREC+131(132),USINGREC+132  CLEAR PRT AREA
RTN1     B     *-*                    RETURN
***************************************
*        DECLARATIVE CODE             *
***************************************
*
DTFPT    $DTFP RCAD-USINGREC,         DEFINE PRINTER FILE              X
               IOAREA-USINGIOB,                                        X
               SPACEA-1
USINGIOB EQU   *
         DS    XL146                  PHYSICAL BUFFER FOR PRINTER
USINGREC EQU   *
         DC    CL133' '               LOGICAL RECORD FOR PRINTER
NOERRS   DC    CL28'       NO ERRORS           '
```

Sample Program 2

SAMPLE PROGRAM 3

This sample program shows how macroinstructions can be used to write a subroutine to find a module in the library and load it into main storage:

```
****************************************
*           COMMON EQUATES             *
****************************************
         $COMN                         GENERATE COMMON LABELS
         $LOAD                         GENERATE LOADER OFFSETS/LABELS
         $LOGD                         GENERATE SYSLOG OFFSETS/LABELS
****************************************
*       LOADER SUBROUTINE              *
****************************************
SAMPLE   START X'8ØØ'
         LA    MESSG,2                 PUT PARM LIST ADR IN XR2
         $LOG  OPTNØ-MSGOPØ,OPTN1-MSGOP1,OPTN2-MSGOP2 DISPLAY MESSAGE
         B     FIND                    DO A FIND
MSGOPØ   EQU   *
         $LOG  LIST-MSGOPA             OPTION Ø MESSAGE ISSUED
         B     FIND                    DO A FIND
MSGOP1   EQU   *
         $LOG  LIST-MSGOPB             OPTION 1 MESSAGE ISSUED
         B     FIND                    DO A FIND
MSGOP2   EQU   *
         $LOG  LIST-MSGOPC             OPTION 2 MESSAGE ISSUED
FIND     EQU   *
         $FIND PLIST-PARM              FIND MODULE
         $LOAD TYPE-FETCH,PLIST-PARM,LOAD-ENDADR LOAD MODULE
         $EOJ                          CALL END OF JOB
****************************************
*          PARAMETER LISTS             *
****************************************
PARM     $FNDP TYPE-0,NAME-MIKE,V-DC   GENERATE LOADER PARM LIST
MESSG    $LMSG MEMBER-P,MINOR-W,MIC-1801,                              X
               OPTNØ-Y,OPTN1-Y,OPTN2-Y,OPTN3-Y
MSGOPA   $LMSG TYPE-2,SKIP-Y,SPACE-3,MSGAD-MSGØ,MSGLN-2Ø
MSGOPB   $LMSG TYPE-2,SKIP-Y,SPACE-2,MSGAD-MSG1,MSGLN-2Ø
MSGOPC   $LMSG TYPE-2,SKIP-N,SPACE-1,MSGAD-MSG2,MSGLN-2Ø
****************************************
*       DEFINE DATA CONSTANTS          *
****************************************
MSGØ     EQU   *
         DC    CL2Ø'OPTION Ø CODE CALLED'
MSG1     EQU   *
         DC    CL2Ø'OPTION 1 CODE CALLED'
MSG2     EQU   *
         DC    CL2Ø'OPTION 2 CODE CALLED'
ENDADR   EQU   *
         END
```

Sample Program 3

SAMPLE PROGRAM 4

This sample program shows how macroinstructions can be used to write a subroutine to update a disk file:

```
         $COMN
         EXTRN #$ISUP
         OPEN  DTF-DISK                OPEN DISK FILE
****************************************
* READ FILE AND CHECK REORDER IND      *
****************************************
AGAIN    $GETD ACCESS-ISU,                                                X
               DTF-DISK,                                                  X
               EOF-THRU
         L     $DTFWKB(,$XR2),$XR1     LOCATE RECORD
         CLI   RINDIC(,$XR1),C'X'      REORDER INDIC ON?
         BNE   AGAIN                   IF NO, READ ANOTHER RECORD
         MVC   RECUP+79,79(80,$XR1)    MOVE RECORD TO RECORD WORK AREA
         LA    RECUP,$XR1              XR1--> RECORD IN WORK AREA
         CLI   ORDIND(,$XR1),C'X'      IS PREVIOUS REORDER IND ON
         BE    DECRES                  YES-PROCESS REORDER QUANT
         MVI   ORDIND(,$XR1),C'X'      NO- PUT IND ON
         MVC   ORDERD(,$XR1),RODQTY(4,$XR1) MOVE REORD QTY TO PREV ORD
         AZ    ORDERD(4,$XR1),ORDQTY(4,$XR1) ADD ANY BACKORDER NEEDED
         MVC   ORDQTY(,$XR1),ZERO(4)   ZERO BACKORDER FIELD
         B     PUTBCK
****************************************
* PROCESS REORDER QUANTITY             *
****************************************
DECRES   CLC   ORDQTY(,$XR1),ZERO(4)   ANY MORE BACKORDERS ?
         BE    AGAIN                   NO--READ NEXT RECORD
         MVI   ORDIND(,$XR1),X'40'     YES--TURN PREV ORDER IND OFF
         MVC   ORDQTY(,$XR1),ZERO(4)   ZERO BACKORDER FIELD
PUTBACK  $PUTD ACCESS-ISU,DTF-DISK,UPDATE-Y
         B     AGAIN                   READ ANOTHER DISK RECORD
****************************************
* CLOSE FILES AND EOJ                  *
****************************************
THRU     $CLOS DTF-DISK
         $EOJ
****************************************
* DTF'S, EQUATES, AND STORAGE          *
****************************************
DISK     $DTFD ACCESS-ISU,RECL-80,     INVENTORY FILE DTF                 X
               NAME-INVENTRY,IOAREA-DISKIO,BLKL-512,                      X
               KEYL-6,RCAD-RECUP,KDISP-5
DISKIO   EQU   *                       DISK I/O AREA
         DS    512CL1
ZERO     DC    CL1'0'                  ZERO FIELD
RINDIC   EQU   34
RODQTY   EQU   42
ORDQTY   EQU   46
ORDERD   EQU   63
ORDIND   EQU   64
         $DTFO DISK-Y,FIELD-Y          DEFIN OFFSETS FOR DISK
```

Sample Program 4

Appendix A. Error Information

Any errors made in coding macroinstructions are flagged in the $ASMINPT file. When an error is found in a macroinstruction, an error code and an error message are placed immediately following the macroinstruction in the $ASMINPT file. The error code and message are then printed on your assembly listing when the source program is assembled.

The following listing shows the error codes that may be caused by errors in macroinstructions. Other error codes may be generated by the macroinstruction processor and are caused by errors in the macroinstruction definitions. These error codes are explained in the *IBM System/32 Basic Assembler and Macro Processor Reference Manual*, SC21-7673.

MIC	Message
2600	NO ACCESS–CONSECUTIVE INPUT ASSUMED
2601	OUTPUT RECORD BUFFER ADDRESS DEFAULTED
2602	KEY ADDRESS FIELD DEFAULTED TO ZEROS
2603	NO RECORD LENGTH SPECIFIED. ASSUMED 1
2604	NO I/O AREA ADDRESS SPECIFIED. ASSUMED 0
2605	NO BLOCK LENGTH SPECIFIED. ASSUMED 256
2606	NO FILE NAME SPECIFIED. ASSUMED FILENAME
2607	WRONG ACCESS METHOD FOR 'ORDLD'
2608	WRONG ACCESS METHOD FOR 'LIMIT'
2609	WRONG ACCESS METHOD FOR 'BUFNO'
2610	WRONG ACCESS METHOD FOR 'KEYADD'
2611	WRONG ACCESS METHOD FOR 'CURENT'
2612	WRONG ACCESS COMBINATION FOR 'HIGH'
2613	WRONG ACCESS METHOD FOR 'KEYL'
2614	WRONG ACCESS METHOD FOR 'KDISP'
2615	WRONG ACCESS METHOD FOR 'MSTNDX/MSTBYT'
2616	KEY/RECORD ADDRESS AREA DEFAULTED TO 0
2617	CURRENT/LAST KEY AREA DEFAULTED TO 0
2618	HIGH/LOW KEY HOLD AREA DEFAULTED TO 0
2619	KEY LENGTH DEFAULTED TO 1 FOR INDEX FILE
2620	KEY DISPLACEMENT DEFAULTED TO 0
2621	LOAD MUST BE BLANK OR 2 IF PLIST EQUAL 2
2622	PLIST & LOAD@ CAN'T BE EQUAL FOR A LOAD
2623	KEYWORD GIVEN FOR TYPE IS NOT VALID
2624	CONFLICTING PARAMETERS–BLKL,RECL
2625	CONFLICTING PARAMETERS–TRANSP,CODE
2626	CONFLICTING PARAMETERS–TERMAD,TYPE
2627	CONFLICTING PARAMETERS–RCVID,TYPE
2628	CONFLICTING PARAMETERS–RCVID/RCVCT,TYPE
2629	CONFLICTING PARAMETERS–SNDID/SNDCT,TYPE
2630	CONFLICTING PARAMETERS–SNDCT,TYPE
2631	CONFLICTING PARAMETERS–RVIMSK,RVIADR
2632	CONFLICTING PARAMETERS–TRANSP,ITB,FTYP
2633	STATION IDS RECOMMENDED ON SWITCHED LINE

MIC	Message
2634	MISSING REQUIRED OPERAND–FTYP
2635	MISSING REQUIRED OPERAND–RCAD
2636	MISSING REQUIRED OPERAND–TERMAD
2637	MISSING REQUIRED OPERAND–RCVCT
2638	MISSING REQUIRED OPERAND–RCVID
2639	MISSING REQUIRED OPERAND–SNDCT
2640	MISSING REQUIRED OPERAND–SNDID
2641	MISSING REQUIRED OPERAND–RECL
2642	MISSING REQUIRED OPERAND–BLKL
2643	INVALID OPERAND–CODE
2644	INVALID OPERAND–DLYCT
2645	INVALID OPERAND–BUFNO
2646	INVALID OPERAND–RCVCT
2647	INVALID OPERAND–SNDCT
2648	INVALID OPERAND–FUNC
2649	INVALID OPERAND–ACCESS
2650	INVALID OPERAND–CRDLD
2651	FORMAT MUST BE Y & AN OPTION SPECIFIED
2652	UPSI OPERAND NOT 8 DIGITS. DEFAULT 0's
2653	INVALID OPERAND–TYPE
2654	INVALID OPERAND–TERMAS
2655	INVALID OPERAND–TRANSP
2656	INVALID OPERAND–ITB
2657	INVALID OPERAND–LIMIT
2658	INVALID OPERAND–HUC
2659	INVALID OPERAND–PRINT
2680	INVALID OPERAND–OPC
2681	INVALID OPERAND–SHIFT
2682	INVALID OPERAND–WAIT
2683	INVALID OPERAND–MEMBER
2684	INVALID OPERAND–OPTN0
2685	INVALID OPERAND–OPTN1
2686	INVALID OPERAND–OPTN2
2687	INVALID OPERAND–OPTN3
2688	INVALID OPERAND–SKIP
2689	INVALID OPERAND–FORMAT
2690	INVALID OPERAND–HALT
2691	INVALID OPERAND–UPDATE
2692	INVALID OPERAND–SPACEB
2693	INVALID OPERAND–SPACEA
2694	MISSING REQUIRED OPERAND–IOAREA

Appendix B. Macroinstruction Summary Chart

[Name]	$ALOC	[DTF-address]
[Name]	$CLOS	[DTF-address]
	$COMN	
[Name]	$CSLD	[FUNC-function] [,ERR-address]
[name]	$DTFB	RECL-number,RCAD-address,BLKL-number, FTYP-RCV/TSM [,BUFNO-1/2] [,BUFST-address] [,IOBST-address] [,TYPE-PP/AA/MA/MP/MC] [,CODE-E/A] [,UPSI-mask] [,CHAIN-address] [,ITB-Y/N] [,TRANSP-Y/N] [,RVIADR-address] [,RVIMSK-code] [,DLYCT-number] [,RCVID-address] [,RCVCT-number] [,SNDID-address] [,SNDCT-number] [,TERMAD-number]
[Name]	$DTFD	ACCESS-code,RECL-number,NAME-filename, BLKL-number,IOAREA-address [,UPSI-mask] [BUFNO-1/2] [,LIMIT-Y/N] [,ORDLD-Y/N] [,CHAIN-address] [,RCAD-address] [,KEYL-number] [,KDISP-number] [,KEYADD-address] [,MSTNDX-address] [,MSTBYT-number] [,CURENT-address] [,HIGH-address]
[Name]	$DTFO	[DISK-Y/N] [,PRT-Y/N] [,BSC-Y/N] [,CRT-Y/N] [,ALL-Y/N] [,FIELD-Y/N]
[Name]	$DTFP	RCAD-address, IOAREA-address [,OVFL-number] [,PAGE-number] [,UPSI-mask] [,HUC-Y/N] [,CHAIN-address] [,PRINT-Y/N] [,RECL-number]
[Name]	$DTFS	[PUTDAT-address] [,PUTLOC-number] [,UPSI-mask] [,CHAIN-address] [,PUTLEN-number] [,OPC-code] [,GETDAT-address] [,GETLOC-number] [,GETLEN-number] [,FUNKEY-number] [,CMDKEY-number] [,SHIFT-A/N] [,CURSOR-number] [,SPACE-number] [,WAIT-Y/N] [,IOBST-address]
[Name]	$EOJ	
[Name]	$FIND	PLIST-name
[Name]	$FNDP	[NAME-module] [,V-DC/EQU/ALL] [,TYPE-O/P/R/S]
[Name]	$GETB	[DTF-address] [,REJECT-address] [,EOF-address]
[Name]	$GETD	ACCESS-code [,DTF-address] [,ERR-address] [,EOF-address] [,NRF-address]

[Name]	$GETS	[DTF-address] [,GETDAT-address] [,GETLEN-number] [,GETLOC-number] [,OPC-code] [,FUNKEY-number] [,CMDKEY-number] [,SHIFT-A/N] [,WAIT-Y/N] [,CURSOR-number] [,SPACE-number]
[Name]	$LMSG	[TYPE-1/2] [,MEMBER-code] [,MINOR-code] [,SUBID-code] [,FORMAT-Y/N] [,HALT-Y/N] [,MIC-number] [,OPTN0-Y/N] [,OPTN1-Y/N] [,OPTN2-Y/N] [,OPTN3-Y/N] [,SKIP-Y/N] [,SPACE-1/2/3] [,MSGLN-number] [,MSGAD-address]
[Name]	$LOAD	[PLIST-address] [,LOAD-address] [,TYPE-code]
[Name]	$LOG	[LIST-address] [,OPTN0-address] [,OPTN1-address] [,OPTN2-address]
	$LOGD	
[Name]	$OPEN	[DTF-address]
[Name]	$PGS	[DTF-address] [,OPC-code] [,PUTDAT-address] [,PUTLEN-number] [,PUTLOC-number] [,GETDAT-address] [,GETLEN-number] [,GETLOC-number] [,FUNKEY-number] [,CMDKEY-number] [,SHIFT-A/N] [,CURSOR-number] [,SPACE-number] [,WAIT-Y/N]
[Name]	$PUTB	[DTF-address] [,REJECT-address]
[Name]	$PUTD	ACCESS-code [,DTF-address] [,ERR-address] [,EOX-address] [,DUPREC-address] [,SEQERR-address] [,UPDATE-Y/N]
[Name]	$PUTP	[DTF-address] [,PRINT-Y/N] [,SKIPB-number] [,SPACEB-0/H/1/1H/2/2H/3/3H] [,SKIPA-number] [,SPACEA-0/H/1/1H/2/2H/3/3H] [,ERR-address] [,OVFL-address]
[Name]	$PUTS	[DTF-address] [,PUTDAT-address] [,PUTLOC-number] [,PUTLEN-number] [SPACE-number]
[Name]	$TRAN	[TRL-address]
[Name]	$TRL	TO-address,FROM-address,LEN-number,TRT-address
[Name]	$TRTB	[CODE-E/A] [,HEX-hex]

READER'S COMMENT FORM

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

Error in publication (typographical, illustration, and so on). **No reply.**

Page Number *Error*

Inaccurate or misleading information in this publication. Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

Page Number *Comment*

Note: All comments and suggestions become the property of IBM.

☐ Check if reply is requested.

Name _____

Address _____

_____

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

IBM System/32
System Control Programming
Macroinstructions
Reference Manual

GC21-5157-0

GC21-5157-0

Cut Along Line

Fold

Fold

FIRST CLASS
PERMIT NO. 387
ROCHESTER, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

IBM System/32 SCP Macroinstructions Reference Manual (File No. S32-36) Printed in U.S.A. GC21-5157-0

IBM

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
Atlanta, Georgia 30301
(USA Only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

GC21-5157-0

IBM System/32 SCP Macroinstructions Reference Manual (File No. S32-36) Printed in U.S.A. GC21-5157-0

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
Atlanta, Georgia 30301
(USA Only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)