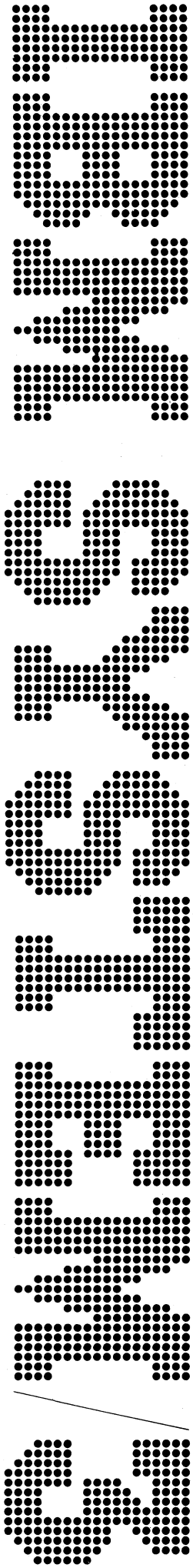


IBM System/3
Models 8, 10, and 12
System Control Programming Macros
Reference Manual

Program Numbers:
5702-SC1 (Models 8 and 10)
Feature Numbers: 6020/6021

5705-SC1 (Model 12)

GC21-7562-5
File No. S3-31

Sixth Edition (June 1978)

This is a major revision of, and obsoletes, GC21-7562-4. Miscellaneous technical changes and corrections have been made throughout the manual; changes to text and illustrations are indicated by a vertical line at the left of the change.

This edition applies to the following system control programs and to all subsequent versions and modifications until otherwise indicated in new editions or technical newsletters.

Version	Modification	Program Number	System/3 Model
15	00	5702-SC1	Models 8 and 10
04	00	5705-SC1	Model 12

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/3 Bibliography*, GC20-8080, for the editions that are applicable and current.

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

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

This publication could contain technical inaccuracies or typographical errors. Address your comments about this publication to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. Comments become the property of IBM.

This manual describes the macro instructions provided by the IBM System/3 Model 10 Disk System Macros Feature. 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 macro instructions and system programming for the IBM System/3 Model 10 Disk System.

This publication describes how to use the macro instructions provided through the Macros Feature. The following topics are discussed:

- Coding macro instructions
- Descriptions of the various macro instructions
- OCL necessary to call the macro processor
- Error conditions detected by the macro processor

A sample program shows how macro instructions are used.

The IBM System/3 Model 8 is supported by IBM System/3 Model 10 disk system control programming and program products. Although the Model 8 is not referenced in this manual, the facilities described in this manual for the Model 10 are also applicable to the Model 8. It should be noted that not all devices and features which are available on the Model 10 are available on the Model 8. Therefore, Model 8 users should be familiar with the contents of *IBM System/3 Model 8 Introduction,* GC21-5114.

Minimum System Requirements

The minimum system configuration and optional device support for the Macros Feature is shown in the *IBM System/3 Models 4, 6, 8, 10, and 12 System Generation Reference Manual,* GC21-5126.

System/3 Model 12

The facilities described in this publication for the Model 10 also apply to the Model 12 except where specifically noted throughout the manual. The following information concerns Model 12 macros only:

$GETD/$PUTD
The Model 12 does not support multivolume or indexed disk files in the simulation area; thus AC (access) codes in these macros reflect these differences.

$DTFD/$IOBD/$RDD/$WRTD
The Model 12 uses 3340 disk drives. The valid parameters for the DISK keyword are 5444, 5445, and 3340.

$DTFU
The Model 12 must specify a PIOB-address keyword with an address of a 23-byte printer IOB area.

5444/5445/3340 references
All references to 5444 relate to the simulation area, and all references to 5445 relate to the main data area.

Related Publications

The following publications contain information which further describes topics discussed in this manual:

- *IBM System/3 Basic Assembler Reference Manual,* SC21-7509
- *IBM System/3 Models 4, 6, 8, and 10 System Control Program Logic Manual*, SY21-0502
- *IBM System/3 Models 8 and 10 System Control Programming Reference Manual*, GC21-7512
- *IBM System/3 Model 8 Introduction*, GC21-5114
- *IBM System/3 Models 4, 6, 8, and 10 Data Management and Input/Output Supervisor Logic Manual*, SY21-0512
- *IBM System/3 Models 8, 10, 12, and 15 Components Reference Manual*, GA21-9236
- *IBM System/3 Model 12 Introduction,* GC21-5116
- *IBM System/3 Model 12 System Control Programming Reference Manual,* GC21-5130
- *IBM System/3 Model 12 System Control Program Logic Manual,* SY21-0046
- *IBM System/3 Model 12 User's Guide,* GC21-5142
- *IBM System/3 Multiline/Multipoint Binary Synchronous Communications Reference Manual*, GC21-7573
- *IBM System/3 Multiple Line Terminal Adapter RPQ Program Reference and Component Description Manual*, GC21-7560

Contents

Chapter 1. Introduction

A macro instruction is a source statement that causes generation of a predetermined set of assembler statements each time the macro instruction is used. The Macros Feature is a macro processor that provides macro instructions which perform both system services and input/output device support. By using these macro instructions, you can perform both system and input/output operations with less coding.

Figure 1 is an overview of the operation of the macro processor. The OCL statements used to call the macro processor are explained in *Chapter 3: OCL and Sample Program.*

WRITING MACRO INSTRUCTIONS

You code macro instructions as follows:

Starting Column 1	8	14	72
Name	Operation	Operands	Continuation
Symbol or blank	Macro name	No operands or one or more separated by commas	Any nonblank character if continuation is being used

The name field may contain any valid assembly 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.

[Name]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

The desired mnemonic operation code (macro instruction name) must appear as specified in the macro instruction description. The operation code must start in column 8.

[NAME]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

The operands specify available services and options. The operands must start in column 14 and are written as follows:

1. Each operand consists of a keyword followed by a dash and a parameter.

[NAME]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

2. No blanks should be left between operands.

[NAME]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

3. Commas precede all but the first operand.

[Name]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

4. The parameter part of the operand must immediately follow the dash.

[Name]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

5. The keyword part of each operand must correspond to one of the keywords in the macro instruction description.

[Name]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

6. Some operands are not required. These optional operands are indicated by enclosing the operand within brackets [KEYWORD-parameter].

[Name]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

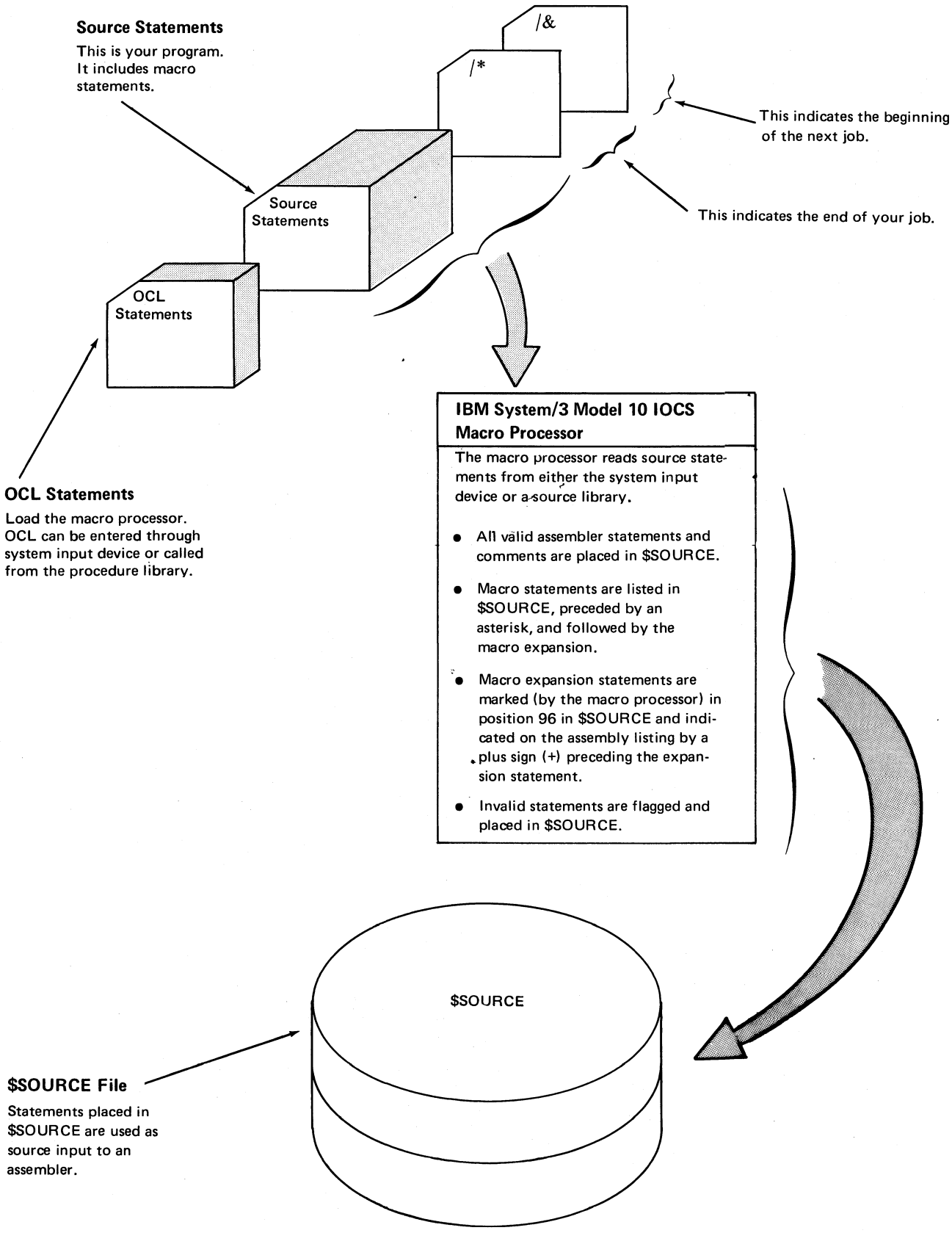


Figure 1. Macro Processor Overview

7. 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. These are the only valid options for the keyword parameter.

[Name]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

   When the options Y/N are given in a macro instruction, Y indicates a yes response, N indicates a no response.

8. The operands may be written in any order. If a keyword is not specified, the default value is used. A default value is selected for optional keywords that are omitted. The default value is indicated in the macro instruction description by a line under the default option. For example, [KEY-A/B/C] indicates the option A is the default value.

[Name]	$FIND	NAME-module name,FIND-label [,PACK-P/S]

9. No operands may be specified beyond column 71. If continuation is required, column 72 must contain a nonblank character and the last operand 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 14. For an example of continuation coding, see Figure 2.

10. 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 macro instruction. Figure 3 shows examples of comments used with macro instructions. On the assembly listing, all comments on the generated code are justified by the macro processor to begin in column 40. Any comments too long to be contained in columns 40 through 71 are truncated from the right.

```
NAME1  $MCRO OPERAND1,OPERAND2,OPERAND3,OPERAND4,OPERAND5,OPERAND6,    X
             OPERAND7,OPERAND8

NAME2  $MACR OPERAND1,OPERAND2,                                        X
             OPERAND3,                                                 X
             OPERAND4,                                                 X
             OPERAND5
```

Figure 2. Continuation Coding Examples

```
COMNT1 $MCRO OPERAND1,OPERAND2         THIS INST. HAS TWO OPERANDS

COMNT2 $MACR OPERAND1,                 THIS INSTRUCTION AND THIS COM-  X
             OPERAND2                  MENT ARE CONTINUED.
*                                      THIS COMMENT IS QUITE LENGTHY
*                                      AND IS ENTERED BEFORE THE IN-
*                                      STRUCTION. OTHERWISE,IT WOULD
*                                      FOLLOW THE MACRO EXPANSION IN
COMNT3 $ACRO OPERAND1                  THE LISTING
```

Figure 3. Comments on Macro Instructions

MACRO INSTRUCTIONS PROVIDED

The macro instructions provided through the macro processor and the functions they perform are shown in Figure 4.

All code for the macro instructions must be on the program pack or the system pack. The program pack is the disk pack from which the macro processor is loaded. The system pack is the disk pack from which initial program load (IPL) is performed.

You may want to delete some macro instructions from your library to reduce the amount of disk space required for the macro instructions. For instance, if your system does not include the 3410/3411 Magnetic Tape Subsystem, the tape macro instructions would be of no use to you, and you might wish to delete them to save space. You can delete macro instructions from your library by using the library maintenance utility program, $MAINT.

Group	Macro Instruction Name	Descriptive Name
System Services	$COMN	Common Equates
	$EOJ	End of job
	$FIND	Find a directory entry
	$FTCH	Load a module and pass control
	$LOAD	Load a module
	$SNAP	Snap dump main storage
	$SVC	Supervisor call
	$TRAN	Generate an interface to the translate routine
	$TRL	Generate a translate parameter list
	$TRTB	Generate a translate table
	$XCTL	Load a module and exchange control
General I/O Support	$ALOC	Allocate space
	$CHK	Check for I/O completion
	$CKL	Generate a check list
	$CLOS	Prepare a device for termination
	$OPEN	Prepare an I/O device
Unit Record Device Support	$DTFU	Define the file for unit record
	$DTOU	Unit record DTF offsets
	$GPU	GET or PUT for unit record
	$PKBU	Construct an interface to the printer-keyboard
	$PRNT	Print a message
Disk Device Support	$DTFD	Define the file for disk
	$DTOD	Disk DTF offsets
	$GETD	Construct a disk GET interface
	$IOBD	Input/output block for disk
	$IOED	Input/output block offsets
	$PUTD	Construct a disk PUT interface
	$RDD	Read from disk
	$WAIT	Wait for disk IOS completion
	$WRTD	Write to disk
Tape Device Support	$CTLT	Control command for tape
	$DTFT	Define the file for tape
	$DTOT	Tape DTF offsets
	$GETT	Construct a tape GET interface
	$PUTT	Construct a tape PUT interface
	$RDT	Read from tape
	$WRTT	Write to tape
	$WTT	Wait for tape I/O completion
3741 Device Support	$DTFK	Define the file for 3741
	$DTOD	3741 DTF offsets
	$GETK	Construct a 3741 GET interface
	$PUTK	Construct a 3741 PUT interface
CPU Com-mands	$CCP	Command CPU
	$LCP	Load CPU
	$SCP	Store CPU

Figure 4. Macro Instructions

Chapter 2. Macro Instruction Statements

You code macro instructions to generate a block of assembler statements to perform a certain function. Some functions may be the same each time they are used, others may be modified by specifying different operands. This chapter explains the System/3 macro instructions in detail.

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

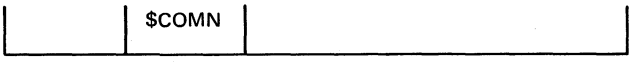
- Common equates
- System services
- Input/output support

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

Common Equates ($COMN)

This macro instruction provides equates for various labels and values used by other macro instructions.

The format of the $COMN macro instruction is:



You must generate equates using $COMN whenever any of the following macro instructions are used in your program:

$ALOC	$PRNT
$CLOS	$PUTD
$CTLT	$PUTK
$FIND	$PUTT
$FTCH	$RDD
$GETD	$RDT
$GETK	$WRTD
$GETT	$WRTT
$GPU	$WAIT
$LOAD	$WTT
$OPEN	$XCTL
$PKBU	

Programming Considerations

When you use the macro processor you should remember the following restrictions. First, the generated code for some macro instructions uses register 1, and the generated code for other macro instructions uses register 2. You should save the contents of the register used by the generated code before issuing the macro instruction. Otherwise, the contents are destroyed. These macro instructions use register 1:

$RDD
$TRAN
$WRTD
$WAIT

These macro instructions use register 2:

$ALOC	$OPEN
$CHK	$PKBU
$CLOS	$PRNT
$CTLT	$PUTK
$FIND	$PUTT
$FTCH	$PUTD
$GETD	$RDT
$GETK	$WRTT
$GETT	$WTT
$GPU	$XCTL
$LOAD	

The second consideration concerns the labels you use in your program. System routines and some generated code have labels beginning with the dollar sign ($). To avoid duplicate label errors, you should not use the dollar sign as the first character of labels in programs using the macro processor.

SYSTEM SERVICES

By using system services macro instructions, you can communicate with the System/3 system control program. These macro instructions can do the following:

- Read records from the system input device.
- Log and write error messages.
- Pass control to an inquiry program and receive control after the inquiry program has finished.
- Determine the location of an object module on disk.
- Obtain object modules from disk and load them into main storage.
- Pass control to modules in main storage.
- Build and use a translate list.
- Dump a part of main storage during execution of your program.
- Terminate the current job.

Supervisor Call ($SVC)

The supervisor call macro instruction branches to one of the following system routines:

- System input (SYSIN)
- Halt/syslog
- VTOC read
- Rollout/rollin

A detailed description of each of these routines is contained in the *IBM System/3 Models 4, 6, 8, and 10 System Control Program Logic Manual*, SY21-0502.

The format of the $SVC macro instruction is:

[Name]	$SVC	RIB-SYSIN/HALT/VTOC/ROLL

RIB-SYSIN, HALT, VTOC, or ROLL indicates the system routine you want to call. The following discussion explains how you can use these routines through the macro instruction.

System Input (SYSIN)

You read a record from the system input device by calling the system input routine through the $SVC macro instruction. The system input device may be any one of the following:

- 5471 printer keyboard. Only 96-byte, single-buffered input is allowed for this device. Double buffering is ignored.
- 1442 card reader. Single and double buffering are supported. Only 80 bytes of the 96-byte buffer are used as input; the remaining 16 bytes are cleared to blanks. If single buffering is not indicated, double buffering is assumed.
- 5424 multi-function card unit (MFCU). Both single and double buffering are supported. Support for both the primary (MFCU1) and secondary (MFCU2) hoppers is provided. All 96 bytes are used as input. If single buffering is not specified, double buffering is assumed.
- 3741 Data Station/Programmable Work Station, directly attached. Both single and double buffering are supported. If single buffering is not specified, double buffering is assumed. Only 96-byte records are supported.

To call the system input routine, you must do the following:

1. Construct a parameter list as input to the system input routine. For the required parameter list, see Part 6 of *IBM System/3 Models 4, 6, 8, and 10 System Control Program Logic Manual*, SY21-0502.
2. Put the address of the parameter list in register 2.
3. Issue the macro instruction:

   [Name] $SVC RIB-SYSIN
4. Analyze the return code provided by system input.

The macro processor generates the coding required to branch to the system input routine. The system input routine reads the input record and returns control to your program with a return code in the first byte of the parameter list. Because of this, you must reset the operation code before each call to system input. You must analyze the return code to determine the outcome of the operation.

Halt/Syslog

Specifying RIB-HALT in your $SVC macro instruction calls halt/syslog: a group of system output routines providing communication with the operator. You may want to use halt/syslog 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 halt/syslog. Instead, another halt is issued to the operator until a correct option is taken. When an immediate cancel (option 3) is selected, control is passed directly to the end-of-job (EOJ) routine by halt/syslog.

Two types of printed output are available through halt/syslog. Both are printed on the system log device.

- A log is a 4 or 6-character statement which identifies the type and source of an error.
- A message is a printed statement which may be used to indicate errors that have occurred or to issue instructions to the operator, such as requesting that a disk file be placed on a certain drive.

Both logs and messages may be issued with or without an accompanying halt.

Note: You cannot issue system halts through the $SVC macro instruction, but you may design your halts to indicate the same errors and accept the same responses as the system halts.

Three devices may be used as the system log device:

- 5203 line printer (left carriage only)
- 1403 line printer
- 5471 printer-keyboard

The device used is determined when you perform initial program load (IPL). You may change devices by entering a // LOG statement in your job stream.

To use halt/syslog you must do the following:

1. Build the appropriate parameter list as determined by the function you want to perform. The parameter list formats are described in Appendix B of *IBM System/3 Models 4, 6, 8, and 10 System Control Program Logic Manual*, SY21-0502.
2. Put the address of the parameter list in register 2.
3. Issue the macro instruction:

   [Name] $SVC RIB-HALT
4. Process the operator's reply in your program.

The generated code passes control to the halt/syslog routine. The halt/syslog routine performs the operation indicated by the parameter list. If a reply is to be returned by the operator, halt/syslog ensures that the reply is valid and returns it to your program in the parameter list. You must then process the reply. If the operator's reply is not valid, a halt is issued until a valid reply is given.

Note: When option 3 (immediate cancel) is selected by the operator, control is passed from halt/syslog directly to the end-of-job (EOJ) routine.

VTOC Read

You can perform input operations on various data areas on cylinder zero of disk files on the IBM 5444 Disk Storage Drive by using the volume table of contents (VTOC) read routine. This routine cannot be used for files on the IBM 5445 Disk Storage Drive.

The data areas you have access to are:

- Volume label
- Volume table of contents (VTOC) index
- Format-1 labels
- Configuration records

You call the VTOC read routine by specifying RIB-VTOC in the $SVC macro instruction.

Volume Label is one sector (256 bytes) containing the volume identification, VTOC location, and a system directory that shows the status and location of the source and object libraries. Volume label requests allow you to read information from the volume label. The format of the volume label is given in Figure 5.

VTOC Index consists of two sectors (512 bytes) containing one 10-byte entry for each file in the volume. Each 10-byte entry contains the name of the referenced file and the location of the format-1 label associated with the file. Figure 6 shows the format of the VTOC index.

Format-1 Label describes each file maintained on the disk pack. The format-1 request allows you to read a format-1 label. Figure 7 describes the format-1 label. Each label is 64 bytes long.

Configuration Record is one sector (256 bytes) providing device information to the system. The configuration record format is provided in Appendix A of *IBM System/3 Models 4, 6, 8, and 10 System Control Program Logic Manual*, SY21-0502.

Calling VTOC Read requires the following steps:

1. Build the parameter list describing the operation to be performed. The parameter list is described in Part 6 of *IBM System/3 Models 4, 6, 8, and 10 System Control Program Logic Manual*, SY21-0502.
2. Put the address of the parameter list in register 2.
3. Issue the macro instruction:

   [Name] $SVC RIB-VTOC
4. Check the return code to determine the outcome of the operation.

When you issue the $SVC macro instruction specifying RIB-VTOC, the generated code calls the VTOC read routine. VTOC read performs the operation and returns control to you with the address of the parameter list still in register 2.

Rollout

You use rollout to interrupt the current program so that another program can be executed. When the second program is finished, the first program is reinstated and continues executing. A program that calls rollout is rolled out and when the interrupting program is completed, the first program is rolled in.

The 5471 printer keyboard is required for a rollout request. Once rollout is initiated, the printer keyboard becomes the system input device.

When using rollout, you should follow these procedures:

1. Be certain the following restrictions are met:
   - A program calling rollout cannot run in program level 2.
   - A rollout-calling program must be so defined to the linkage editor.
   - Programs cannot run in program level 2 when rollout-calling program is in program level 1.
   - The same I/O devices are available to the interrupting program as were available to the original program.
   - Whenever an interrupting program shares the same disk files as a rolled-out program, only reading and updating are allowed by the two programs. Loading and additions are not allowed.
2. Determine whether rollout has been requested by testing the inquiry-request-pending bit in the system communication region. If the request is pending, call rollout. If it is not on, proceed with the current program.
3. Set on the rollout request bit in the system communication region.
4. Issue the macro instruction:

   [Name] $SVC RIB-ROLL
5. Set off the rollout request bit in the system communication region.

The coding generated by the $SVC macro instruction calls the rollout routine. Rollout performs the following steps:

1. Places the current program (program being executed) and the current contents of the scheduler work area on disk.
2. Allows a new program to be loaded in place of the current program and passes control to the new program.
3. Reloads the original program and previous contents of the scheduler work area and passes control to the point where the original program was interrupted.

Hexadecimal Displacement	Number of Bytes	Contents of Volume Label
0-2	3	VOL (Label identifier)
3-8	6	Volume identifier, 1-6 characters
9-A	2	VTOC index pointer (C/S)
		System Directory
		Source Library
B-C	2	Source directory pointer
D-E	2	Next available library sector
F-10	2	End of library
11-12	2	Number of directory sectors
13-14	2	Number of permanent library sectors
15-16	2	Number of active library sectors
17-18	2	Number of available library sectors
19-24	12	Reserved
		Object Library
25-26	2	Object directory pointer
27-28	2	End of directory
29-2A	2	Start of library
2B-2C	2	Allocated end of library
2D-2E	2	Extended end of library
2F-30	2	Number of available permanent directory entries
31-32	2	Number of available temporary directory entries
33-35	3	First temporary directory entry (C/S/D)
36-38	3	Next available temporary directory entry (C/S/D)
39-3A	2	Next available library sector for permanent library entries
3B-3C	2	Next available library sector for temporary library entries
3D-3E	2	Number of available library sectors for permanent library entries (after last PERM)
3F-40	2	Number of available library sectors for temporary library entries
41-42	2	Total number of active library sectors
43-44	2	Number of active O type permanent sectors
45-46	2	Number of active R type permanent sectors
		System Information
47	1	System indicator
48-49	2	Rollin/Rollout pointer
4A	1	Rollin/Rollout size (tracks)
4B-4C	2	SWA pointer
4D	1	SWA size (tracks)
4E-51	4	Start and end of library (C/S)
52-5B	10	Owner ID
5C-69	14	Device constants
6A-75	12	Alternate track assignments
76-A8	51	Available tracks (Format-5)
A9-B3	11	Save area for copypack, $COPY utility
B4	1	Run OXRF indicator
B5	1	Reserved
B6-B8	3	Reserved
B9-BA	2	Checkpoint/Restart
BB-C9	15	Unused
CA-CD	4	Reserved
CE-D7	10	Scientific system file indicator
D8-EF	24	Suspected defective track indicators
F0-FF	16	Reserved

Notes:

1. References in this table to C/S indicate the cylinder and sector location of the field. Cylinder and sector are given in hex numbers. References to C/S/D indicate the cylinder, sector, and displacement into the sector of the field. Again, all are given in hex numbers.
2. Fields in the volume label that tell the number or quantity of various entries on the disk are hex numbers.

Figure 5. Volume Label Format

A	B	C	D	E	F

Symbol	Number of Bytes	Contents
A	6	Not used
B	500	Index with up to 50 entries of the following format: 8 bytes—Filename left-justified (20 indicates duplicate name in the index 2 bytes—Sector number and displacement within that sector or file entry
Note: The remaining 6 bytes are as follows:		
C	2	S/D address of the first member on the S list
D	1	Number of Format-1's on the S list
E	2	S/D address of the last member on the S list
F	1	Number of free entries in the VTOC index (Only P and T formats are considered not free.)

Figure 6. Volume Table of Contents (VTOC) Index Format

Hex Disp	Number of Bytes	Description	Contents
0	1	Entry ID	Entry identification pointer to VTOC index
1-2	2	Chain address	Sector number or displacement into sector of next Format-1
3-A	8	Filename	Filename
B-10	6	Date	Date of file
11	1	Retain	Retain indicator for file (P, S, or T)
12-13	2	File type	*Byte* *Value* *Meaning* 1 X'00' Should be zero 2 X'80' Indexed file X'40' Consecutive file X'20' Direct file X'10' Multivolume file X'08' Last pack of a consecutive MVF only
14-15	2	Record length	The number of bytes within each record
16	1	Key length	The number of bytes within the record key
17-18	2	Key location	Sector number/displacement into the sector
19-1B	3	Last record	Last record displacement (C/S/D)
1C-1E	3	Last key	Last key displacement (C/S/D)
1F-20	2	Data start	Start of data address (C/S)
21-22	2	Data end	End of data address (C/S)
23-24	2	Index start	Start of index address (C/S)
25-26	2	Index end	End of index address (C/S)
27	1	Records or tracks	X'80'—number of tracks X'00'—number of records
28-29	2	Number of records or tracks	Number of record/tracks created (see previous byte)
2A	1	Volume sequence number	Volume sequence number for MVF
2B-2C	2	Back pointer	Back pointer for scratch files
2D-3F	19		Not returned by VTOC read

Note: Contents in this table given as C/S indicate a two-byte address given as cylinder number/sector number. C/S/D indicates a three-byte address giving cylinder number/sector number/displacement. The numbers are hex.

Figure 7. Format-1 Label Format

Find a Directory Entry ($FIND)

A load module must be in the object library. Specific information must be obtained from the module's object library directory entry before a load or fetch can be performed. There are two ways you can locate a load module and obtain the information:

- Issue a $FIND before issuing a $LOAD, Form II. The information obtained during the find is used during the load operation.
- Issue a load with find ($LOAD, Form I), a fetch ($FTCH) or a fetch to address ($XCTL). These functions perform the find operation as part of their normal functions.

The $FIND macro instruction searches the object library directory for the requested module name and returns the directory entry in the 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 $FIND macro instruction.

The format of the $FIND macro instruction is:

[Name]	$FIND	NAME-module[,FIND-address] [,PACK-P/S]

NAME-module provides the name of the module to be found. Only names of object modules (O modules) can be entered here.

FIND-address specifies the label that becomes the address of a 12-byte parameter list built by the generated code. Initially the parameter list contains input to the supervisor. After execution, it contains the directory entry of the module. The format and contents of the parameter list after execution are shown in Figure 8. If this operand is not specified, a macro label is generated.

PACK-P or S specifies the program disk pack (P) or the system disk pack (S) to be searched. If this operand is not specified, P is assumed.

Load a Module ($LOAD)

This macro instruction loads a module into storage at the address you specify. Control is returned after the module has been loaded. You may then pass control to the module at the specified address. 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 $LOAD macro instruction. Two forms of this macro can be used.

Load with Find (Form I)

The load with find macro instruction locates the module and loads it into main storage.

The format of this macro instruction is:

[Name]	$LOAD	NAME-module name[,FIND-address] [,LOAD-2/address] [,USE-R/NR] [,PLIST-address] [,PACK-P/S]

NAME-module name provides the name of the module to be loaded and is required. Only O modules can be specified.

FIND-address becomes the address of the parameter list passed to the find routine. The parameter list is generated by the macro processor. After execution of the load, this parameter list contains the modified entry for the module as shown in Figure 9.

LOAD-2 or address specifies the address where the module is to be loaded into main storage. The 2 indicates that the address is in register 2; the address is the symbolic address where the module is to be loaded. If this operand is not specified, 2 is assumed.

USE-R or NR indicates whether the code generated by the macro instruction is to be reusable (R) or nonreusable (NR). If the operand is not specified, NR is assumed.

You can reuse the generated code to load the same module more than one time, or to load different modules. If you wish to load different modules using the same generated code, you should also specify the PLIST operand.

Entry	Number of Bytes	Displace-ment	Description
Disk Address	2	1	Cylinder/sector address of the module.
No. of Text Sectors	1	2	Text sector length of the module in hexadecimal
Link Edit Address	2	4	Hexadecimal storage address at which the module was linkage edited.
Disp. of RLDs	1	5	Number of bytes, in hexadecimal, into the first sector containing RLDs, of the first relocation directory (RLD) entry of the module.
Entry Point Address	2	7	Hexadecimal storage address at which program execution begins (without RLDs).
Storage Size	1	8	Amount of storage (in sectors) required to execute the program.
Attributes	2	10	Byte 1: Bit 0 1 = Permanent entry 0 = Temporary entry 1 1 = Inquiry program 2 1 = Rollout-calling program 3 1 = Must run in dedicated environment 4 1 = Requires source information 5 1 = Deferred mounting allowed 6 1 = PTF applied 7 1 = Overlay object program Byte 2: Bit 0 1 = The system input device must be dedicated to this program 1 1 = Checkpoint/Restart program 2 1 = This program will access the source file directly 3 1 = Macro processor is allowed 4 Reserved 5 1 = This program requires that a new load address be calculated at load time to place it in main storage beyond its own program common region 6 Reserved 7 Reserved
Level	1	11	Release version of this entry.
Note: Determination of displacement into the parameter list begins with 0.			

Figure 8. Find Parameter List Description

Entry	Number of Bytes	Displace-ment	Description
Disk Address	2	1	Cylinder/sector address of the module in hexadecimal. See note.
No. of Text Sectors	1	2	Text sector length of the module, in hexadecimal.
Link Edited Address	2	4	Storage address at which the module was linkage edited.
Disp. of RLDs	1	5	Hexadecimal displacement, in bytes, into the first sector containing RLDs, of the first relocation directory (RLD) entry of the module.
Relocated Entry Point Address	2	7	Storage address at which program execution begins, after resolving RLDs.
Load Address	2	9	Address at which the requested module is loaded.
Note: If a directory entry was not found on a load with find, the first byte contains a character O. Determination of displacement into the parameter list begins with 0.			

Figure 9. Find Parameter List after Load Execution

PLIST-address is used only when the generated code is reusable. The address specified identifies the leftmost byte of a parameter list passed to the load routine. To load a different module using the same generated code, you must update the parameter list to indicate the desired module. Figure 10 shows the format and contents of the parameter list.

PACK-P or S specifies the program disk pack (P) or the system disk pack (S) containing the requested module.

Load Only (Form II)

The load-only macro instruction loads a module previously found by the $FIND macro instruction. The format of this macro instruction is:

[Name]	$LOAD	FIND-address[,LOAD-2/address] [,PACK-P/S]

FIND-address is the address used in the previous $FIND macro instruction. It identifies the directory entry of the module in main storage. After execution of the load, this address points to the directory entry of the module as shown in Figure 9.

LOAD-2 or address specifies the address where the module is to be loaded in main storage. The 2 indicates that the address is in register 2; the address is the symbolic address where the module is to be loaded. If this operand is not specified, 2 is assumed.

PACK-P or S specifies the program disk pack (P) or system disk pack (S) containing the requested module.

Load a Module and Pass Control ($FTCH)

The fetch macro instruction ($FTCH) finds a module in the directory, loads the module into main storage, and passes control to it. Your program does not regain control. When a module is fetched into main storage, the relocation factor is added, as necessary, to the module's link edit address. This determines the location in main storage where the module is loaded. The module receives control at its entry point.

The format of the $FTCH macro instruction is:

[Name]	$FTCH	NAME-module name[,PACK-P/S]

NAME-module name specifies the object module to be fetched into main storage. The name must be the same as the name in the directory entry.

PACK-P or S specifies the program disk pack (P) or the system disk pack (S) containing the requested module.

Entry	Number of Bytes	Displacement	Description
Module Type	1	0	Must contain O to indicate an object module.
Module Name	6	6	The name of the module to be loaded.
FE	1	7	X'FE'
Load Address	2	9	The address at which the module is to be loaded.

Figure 10. Load Parameter List Description

Load a Module and Exchange Control ($XCTL)

This macro instruction finds a module in the directory, loads the module into main storage at the address you specify, and passes control to it. Control is not returned to your program. As with the $FTCH macro instruction, relocation factors are resolved, and control is passed to the entry point of the program.

The format of the $XCTL macro instruction is:

[Name]	$XCTL	NAME-module name[,LOAD-2/address] [,PACK-P/S]

NAME-module name specifies the name of the module to be loaded and given control. The module must be an O module.

LOAD-2 or address specifies the address where the module is to be loaded in main storage. The 2 indicates that the address is in register 2; the address is the symbolic address where the module is to be loaded. If this operand is not specified, 2 is assumed.

PACK-P or S specifies the program disk pack (P) or the system disk pack (S) containing the requested module.

Generate a Translate Parameter List ($TRL)

This macro instruction generates a parameter list needed by the Model 10 Translate routine. This list is called via the $TRAN macro instruction. $TRL does not generate executable code. Figure 11 shows the format of the translate parameter list.

Translate Routine Operation

To use the Model 10 translate routine, you must provide a translate area. The format of the area is:

Byte	*Field Description*
0	Byte contents used to determine whether a character is to be translated.
1	Byte contents are substituted for characters that are not to be translated.
2-257	256-byte translate table.

The translate routine processes a field, specified by the $TRAN macro instruction, one byte at a time.

The translate table must be constructed so that the displacement (from the beginning of the table) of the translated representation of a character is equal to the hexadecimal representation of the untranslated character.

Note: If you are using the IBM System/3 Model 10 Disk System Multiple Line Terminal Adapter Input/Output Control System, Program Number 5799-WAU, the translate area is provided.

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.

If an unequal compare results, the hexadecimal value in the translate table is substituted for the original character.

The format of the $TRL macro instruction is:

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

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

FROM-address specifies the symbolic address of the first 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 FROM field.

TRT-address specifies the symbolic address of the first byte of the translate area.

All four operands are required.

Generate a Translate Table ($TRTB)

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

The format of the $TRTB macro instruction is:

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

name specifies the symbolic address of the first byte of the generated translate table.

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

Note: If you specify CODE-A, you may want to specify DC 128XL1'FF' after the $TRTB macro instruction to allow for invalid ASCII characters.

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 a blank.

Generate an Interface to the Translate Routine ($TRAN)

This macro instruction generates an interface to the Model 10 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. You should check the completion code to see if any characters that are not to be translated were encountered.

The format of the $TRAN macro instruction 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. See Figure 11 for a description of the parameter list.

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

Figure 11. Translate Parameter List

Snap Dump Main Storage ($SNAP)

This macro instruction provides an interface with the non-terminating system storage dump routine. You must specify a dump identifier and the limits of the area to be dumped. The contents of the specified main storage area are put on the system logging device; therefore, it is recommended that the logging device be a printer. Output from the dump routine consists of:

- The specified dump identifier
- The contents of registers 1 and 2
- The contents of the specified main storage area

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

The format of the $SNAP macro instruction is:

[Name]	$SNAP	ID-hex,START-address,END-address

ID-hex specifies a 2-byte hexadecimal number to be used as the dump identifier

START-address specifies the symbolic address of the low-storage limit of the area to be dumped.

END-address specifies the symbolic address of the high-storage limit of the area to be dumped.

All three operands are required.

End-of-Job ($EOJ)

The $EOJ macro instruction passes control to the end-of-job routine. This routine returns control to the supervisor for normal end-of-job.

The format of the macro instruction is:

[Name]	$EOJ	

INPUT/OUTPUT SUPPORT

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

- General—macro instructions used with all device types. The following macros are in this group:

  $ALOC
  $OPEN
  $CKL } 5471 Console and
  $CHK } Teleprocessing only
  $CLOS

- Unit Record—macro instructions that support unit-record devices. The following macros are in this group:

  $DTFU
  $DTOU
  $GPU
  $PKBU
  $PRNT

- Disk—macro instructions that support disk devices. The following macros are in this group:

  $DTFD $RDD
  $DTOD $PUTD
  $IOBD $WRTD
  $IOED $WAIT
  $GETD

- Tape—macro instructions that support tape devices. The following macros are in this group:

  $DTFT $PUTT
  $DTOT $WRTT
  $GETT $CTLT
  $RDT $WTT

- 3741—macro instructions that support the 3741. The following macros are in this group:

  $DTFK
  $GETK
  $PUTK

General I/O Support

The general I/O support macro instructions are used with both unit record and disk devices. The normal sequence for using these macro instructions is:

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

Allocate Space ($ALOC)

The routines called by the $ALOC macro instruction allocate input/output devices and space on disk devices. These routines check to ensure that:

- The system supports the requested device.
- The device requested is available to the requesting program.
- The DTF tables do not extend into the last 1K (1024) bytes of the calling program.
- The LOCATION parameter of the OCL FILE statement is valid.
- The correct disk pack is mounted and space is available to the calling program.
- No more than 40 DTFs (disk and tape) are present in the calling program.
- The correct tape input file is mounted, or the tape label is written on the output file, and that the tape is positioned at the beginning of the file.

An allocate request requires that pre-open DTFs be supplied as input to the routine. For a description of DTFs, see *Define the File for Unit Record ($DTFU), Define the File for Disk ($DTFD), Define the File for 3741 ($DTFK), and Define the File for Tape ($DTFT).* When the allocate request is for a disk or tape device, OCL file statements are also required. More than one DTF can be allocated at one time by chaining the DTFs. To chain DTFs, you must enter the address of the next DTF in the DTF you are building. The last DTF in a chain has X'FFFF' entered in place of the address. If your program operates as an interrupt handler, such as a Binary Synchronous Communications program or a Multiple Line Terminal Adapter program, all DTFs in the program should be chained together and allocated on one operation. When an error condition occurs, the allocate routine calls halt/syslog to display the proper halt code.

Note that 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 macro instruction.

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

- The contents of register 1 are restored.
- The format-1 labels and configuration record are updated.
- For a non-disk or non-tape DTF, bit 1 in the rightmost byte of the attribute bytes of the post-open DTF is set on to indicate device allocation.
- The address of the first DTF allocated is returned in register 2.

Note: If you are using telecommunications, $ALOC must not be issued while a telecommunications operation is in process.

The format of the $ALOC macro instruction is:

[Name]	$ALOC	[DTF-address]

DTF-address specifies the address of the high-order byte of the DTF being allocated. 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 macro instruction prepares an input/output file for data transfer. The file to be prepared (opened) must previously have been allocated by using the allocate macro instruction. Depending on the device, one or more of the following functions are performed for each file opened.

- The post-open DTF is formatted (see Figure 12).
- Pre-open DTF information is preserved in the format-1 label as required.
- Input/output buffers, index buffers, and IOBs are formatted.
- Buffers are initialized as required.
- Cards are positioned at the wait station for card output files.
- 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:
  1. The access method and the file organization are compatible.
  2. The volume and file are mounted on the correct disk or tape drive.

Note: More than one DTF can be opened at one time by chaining the DTFs. To chain DTFs, you must enter the address of the next DTF in the DTF you are building. The last DTF in a chain has X'FFFF' entered in place of the address. See *$DTFU, $DTFD, $DTFT* and *$DTFK*.

Input The pre-open DTF and format-1 label are input to the open routine. Before the open macro instruction is issued, you must be sure to have the device allocated by previously issuing the allocate macro instruction. 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 macro instruction. 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 in the other program level or for an inquiry program (see *Rollout).*
- The record length and key length must be specified correctly.

Pre-Open Conditions	Post-Open Conditions
1. Unformatted disk files are present for output files. 2. The I/O buffer is in the unformatted mode.	1. Formatted disk files are created. 2. I/O buffers, IOBs, and various work areas are formatted. 3. A bit is set on in the DTF attribute bytes to indicate an opened file.

Figure 12. Comparison of Pre-Open and Post-Open DTFs and Data Areas

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 restored.
- The format-1 labels are updated.
- Bit 7 in the rightmost attribute byte in the post-open DTF is set on to indicate the file has been opened.
- The device code (displacement 0 in the DTF) is altered to indicate the unit on which the file resides.
- The buffers are initialized.
- The address of the last DTF opened is returned in register 2.

The format of the $OPEN macro instruction is:

[Name]	$OPEN	[DTF-address]

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

Generate a Check List ($CKL)

This macro instruction creates an entry for a check list. It does not generate executable code. The check list identifies DTFs to be checked for I/O completion by the $CHK macro instruction. The following types of DTFs can be identified in the check list:

- 5471 Printer-Keyboard (console)
- Binary Synchronous Communications (BSC)
- Multiple Line Terminal Adapter (MLTA)

For a description of the check list entries, see Figure 13.

For a description of BSC, see *IBM System/3 Multiline/ Multipoint Binary Synchronous Communications Reference Manual*, GC21-7573; for a description of MLTA, see *IBM System/3 Multiple Line Terminal Adapter RPQ Program Reference and Component Description Manual*, GC21-7560.

All the check list entries that are to be tested by the same $CHK macro instruction must be issued consecutively. The same DTF may be in the list more than once. The check list entries that are generated are contiguous in main storage. You can then issue the $CHK macro instruction to test the entire list, by specifying the first entry in the list; or begin testing anywhere in the list, by specifying the label of one of the later entries.

If a console DTF is to be used for both the request key and data input functions at one time, you must specify two check list entries for that DTF (for one entry, specify REQK-Y; for the other entry, either omit the REQK operand or specify REQK-N).

Note: The address you specify in the $CHK macro instruction identifies the beginning of the check operation. Any entries occurring earlier in the list are ignored in that operation.

The format of the $CKL macro instruction is:

[Name]	$CKL	DTF-address [,SKIP-Y/N] [,REQK-Y/N] [,RTN-Y/N] [,LAST-Y/N]

DTF-address specifies the symbolic address of the leftmost byte of the DTF for which this check list entry is being created.

SKIP-Y or N specifies whether this entry should be skipped when the check list is scanned. If this operand is omitted, N (no) is assumed. If Y is specified, you must update the checklist entry before you can check the DTF specified in this macro instruction. You can access the skip indicator in the entry by using the name specified on the macro instruction.

REQK-Y or N specifies whether the check routine should determine whether the Request Key has been pressed on the 5471 Printer-Keyboard. If this operand is not specified, N (no) is assumed. You can change this entry during program execution.

Note: The keyword *CONS-Y or N* has been replaced by REQK (above); if you have coded CONS, however, the same function will be performed.

RTN-Y or N specifies whether you want control returned to your program even if no I/O operation is complete. This operand is valid only for the first entry in the check list. If this operand is not entered, N (no) is assumed.

LAST-Y or N specifies whether this is the last entry in the check list. LAST-Y (yes) must be specified for the last entry. If this operand is omitted, N (no) is assumed.

Disp	Field Description
0	Flag byte: X'80'—Skip this entry X'40'—Request key should be checked X'20'—This is the last entry in the check list X'10'—Return control to the user if no I/O completion is found (significant only in the first entry of a check list)
1-2	Address of the DTF for this entry

Figure 13. Check List Format

Check for I/O Completion ($CHK)

This macro instruction generates the linkage required to use the check routine. You must issue the $CHK macro instruction for each BSC or MLTA get, put, read, write, or online test request. For a description of BSC macro instructions, see *IBM System/3 Multiline/Multipoint Binary Synchronous Communications Reference Manual,* GC21-7573. For a description of MLTA macro instructions, see *IBM System/3 Multiple Line Terminal Adapter RPQ Program Reference and Component Description Manual*, GC21-7560.

You can also use the check routine to test for completion of console operations and to determine whether the request key on the console has been pressed. However, if your program does not use BSC or MLTA, you will use less main storage by using the wait function provided with the $PKBU macro instruction.

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 $CHK macro instruction.

Check Routine Operation: The check routine tests for completion of an I/O operation by examining the DTFs identified in the check list - see *Generate a Check List ($CKL)*. If an I/O operation is complete, the completion code is set in the DTF, and the address of the DTF is returned in register 2 to the calling program. No subsequent DTFs in the list are tested.

When a REQK-Y entry is encountered in the list, the check routine tests the inquiry request bit in the system communication region to determine whether the request key was pressed on the console. If it was, the completion code in the console DTF is set to X'50', the DTF address is put in register 2, and control is returned to your program.

If no I/O completion is found by the end of the check list, one of the following actions is taken.

1. Control is returned to your program with the address of the last DTF in the list register 2 if:
   - Each entry in the list is inactive, closed, or has the skip indicator on (X'57').
   - RTN-Y was specified in the $CKL macro instruction that created the first entry in the check list (X'56').
2. Control is not returned to your program. Instead, the check routine issues a halt ([ ] displayed on the console stick lights) and waits for an I/O operation to be completed. When the operation is complete, the completion code is set in the DTF and the address of the DTF is returned in register 2.

Note: If the only operations pending are on the console, you must issue the $CHK macro instruction again to reset the [ ] halt.

The format of the $CHK macro instruction is:

[Name]	$CHK	[CKL-address]

CKL-address specifies the symbolic address of the first entry in the check list. You may also begin at a subsequent point in the check list by specifying the symbolic address of a later entry. If this operand is omitted, the address is assumed to be in register 2.

Note: The address you specify identifies the beginning of the check operation. Any entries occurring earlier in the list are ignored in that operation.

Prepare a Device for Termination ($CLOS)

The close macro instruction prepares a device for job termination. The routine returns post-open DTFs to their pre-open state and updates file labels to reflect the current file status. For devices other than disk or tape, 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 macro instruction.

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

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

- The contents of register 1 is restored.
- The post-open DTFs are reininialized to the pre-open state.
- Any pending operations for unit record devices are performed.
- The format-1 label for disk is updated to indicate current file status.
- The buffer contents scheduled for disk or tape output and disk update operations are written.
- The data and index are written to disk, and an indicator is set if key sorting is required at end-of-job for output files and file additions.
- Tape trailer labels are read or written.

Note: More than one DTF can be closed at one time by chaining the DTFs. To chain DTFs, each DTF to be closed must contain the address of the next DTF in the chain. The last DTF in a chain has X'FFFF' entered in place of the address. See *$DTFU, $DTFD, $DTFT,* and *$DTFK.*

The format of the $CLOS macro instruction 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.

Unit Record Support

Five macro instructions are provided for performing I/O operations on unit record devices. They are:

- $DTFU—define the file (DTF) for unit record
- $DTOU—define the file (DTF) offsets
- $GPU—get or put for unit record
- $PKBU—construct an I/O interface to the printer-keyboard
- $PRNT—print a message on the halt/syslog device

The DTF and DTF-offsets macro instructions are used together. The DTF macro instruction generates a DTF control block and initializes it to values you specify. The DTF-offsets macro instruction generates equates to give unique labels to the offsets in the DTF.

The remaining macro instructions are used to perform the actual input or output operations.

Define the File for Unit Record ($DTFU)

Through the DTF, you provide information about a file to the allocate and open routines. The pre-open and post-open DTFs for the various unit record devices are explained in *Appendix B: Define the File Control Blocks.*

The format of the $DTFU macro instruction is:

[Name]	$DTFU	DEV-code, FTYP-code, PIOB-address [,UP-mask] [,DIO-Y/N] [,HUC-Y/N] [,RECL-number] [,CHN-address] [,RCAD-address] [,RDA1-address] [,RDA2-address] [,PUA1-address] [,PTA1-address] [,OVFL-number] [,PG-number] [,MSKP-number] [,REPLY-number] [,SPACE-number] [,REQK-Y/N] [,FILL-Y/N] [,CHK-Y/N] [,RTN-Y/N]

Note: PIOB required on Model 12 only.

DEV-code specifies the device. This is a required operand. One of the following codes must be entered; however, the same code cannot be specified in more than one $DTFU macro instruction in the same program.

Code	Meaning
MFCU or MFCU1	MFCU primary hopper
MFCU2	MFCU secondary hopper
PRNTR or PRNTR1	5203 printer (left carriage) or 1403 printer
PRNTR2	5203 printer (right carriage)
D1442	1442 card read/punch
CONSOL	5471 printer-keyboard

FTYP-code specifies the type of file. This is a required operand. One of the following codes must be entered.

Code	Meaning
I	Input: MFCU, 1442, or console read
P or P1	Output: MFCU or 1442 punch, printer, or console write
C or C1	Combined: MFCU or 1442 read and punch or console write to operator with reply
P2	Output: MFCU print
P3	Output: MFCU punch and print
C2	Combined: MFCU read and print
C3	Combined: MFCU read, punch, and print

PIOB-address specifies the address of the leftmost byte of the printer IOB. The size of the printer IOB is 23 bytes. This parameter is required on the Model 12 only and is not valid on the Model 10.

UP-mask specifies the mask for testing the external indicators set in the // SWITCH statement. For example, to set on bits 0, 3, 5, and 7, you would enter UP-10010101. If this operand is not entered, a mask of zero is assumed.

DIO-Y or N specifies whether two physical input/output buffers are supplied. If this operand is not entered, N (no) is assumed. Y (yes) can be specified only for card input files.

HUC-Y or N specifies whether a halt should be issued when an unprintable character is detected in the print record. If this operand is not entered, N (no) is assumed. This operand applies to line printer files only.

RECL-number is a decimal value specifying the logical record length for a 1442 punch file, a line printer file, or a console input/output file. If this operand is not entered, zero is assumed and the value must be updated in the DTF before the output operation may be performed.

CHN-address specifies the address of the next DTF in the forward chain. If this operand is not entered, the end-of-chain ID (X'FFFF') is assumed.

Note that DTFs may be chained to permit opening of more than one DTF with one open request. See *Prepare An I/O Device ($OPEN)* for more information on chaining.

RCAD-address specifies the address of the leftmost byte of:

- The logical record when using the output function for MFCU, 1442, or line printer files.
- The buffer area for all console operations. This is the only input/output area specified for console files.

This address must be different from the address specified for either PUA1 or PTA1. If this operand is not entered, X'FFFF' is assumed and you must update this value in the DTF before performing the output operation.

Output records for unit record devices are built or altered in the work area pointed to by RCAD. These records are called logical records. For devices other than the console, data management routines move the record to the physical buffer (PUA1 or PTA1) before performing the output operation.

RDA1-address specifies the address of the leftmost byte of the first physical input buffer. If this operand is not entered, X'FFFF' is assumed and you must update it in the DTF before issuing the input command. The address must be on a 128-byte boundary.

RDA2-address specifies the address of the leftmost byte of the second physical input buffer when dual buffers are used. If this operand is not entered, X'FFFF' is assumed and you must update this value before using dual buffering in an input operation. The address must be on a 128-byte boundary.

PUA1-address specifies the address of the leftmost byte of the physical punch buffer. If this operand is not entered, X'FFFF' is assumed and you must update it before performing a punch operation. The address must be on a 128-byte boundary.

PTA1-address specifies the address of the leftmost byte of the physical print buffer for print files. If this operand is not entered, X'FFFF' is assumed and you must update the value before an output operation to the printer is issued. This address must be of a 256-byte area aligned on a 256-byte boundary for MFCU print operations. For line-printer operations, the area must be aligned on a X'7C' boundary (the first 124-byte boundary after a 256-byte boundary).

OVFL-number is a decimal number specifying the overflow line for line-printer output files. If this operand is not entered, zero is assumed and 60 is inserted by the open routines.

PG-number is a decimal value specifying the total number of lines on the form used by the printer file. The number must not be greater than 112; if the operand is not entered, 66 is assumed.

MSKP-number specifies the maximum line number used in the skip-before or skip-after byte of the line-printer DTF. If this operand is not entered, zero is assumed.

Note: The generated code from the $GPU macro instruction does not perform carriage control for printer files. To prevent overprinting, you must set the space before, space after, skip before, skip after bytes in the printer DTF to perform the spacing you desire.

REPLY-number specifies the decimal length of the operator's reply for a write-to-operator-with-reply operation on the console. If this operand is not entered, zero is assumed and you must update the entry before the operator's reply can be received.

SPACE-number is specified only for console files. It is a two-digit decimal number that indicates the number of lines to be spaced before and after the console operation. The first digit specifies the space before; the second, the space after. If only one digit is specified, it is assumed to be the space-after value. If this operand is not entered, zero is assumed.

Note that at least one space-before is performed for all console operations.

REQK-Y or N specifies whether the request key must be pressed before an input record is accepted from the console. If this operand is not entered, N (no) is assumed.

FILL-Y or N specifies whether the operator must fill the input or reply record for console files. If this operand is not entered, N (no) is assumed.

CHK-Y or N specifies whether the console operation must be completed before control is returned to your program. N indicates a check for completed operation will be performed at the time of the operation; the console operation must be completed before control returns. Y indicates no check will be performed at the time of the operation; control returns directly after the console operation is started. See *Construct an Interface to the Printer-Keyboard ($PKBU)* and *Check for I/O Completion ($CHK)* for more information on console operations. If this operand is not specified, N is assumed.

RTN-Y or N is specified only if CHK-Y was specified for a console operation. This operand specifies the type of check operation to be performed. If Y is specified, the console operation is checked for completion and control is returned to your program with a completion code to indicate whether the operation was completed. If N is specified, the console operation is checked for completion and a wait is performed if the operation is not completed. Control is not returned to your program until completion occurs. If this operand is not specified, N is assumed.

If the entry specified in the CHEK parameter in the $DTFU or $PKBU macro instruction is not the same as the entry specified in the RTN parameter of the $CHK macro instruction, control does not return from a console operation until the operation has been completed. For more information on specifying the check operation for the console, see *Construct an Interface to the Printer-Keyboard ($PKBU).*

Unit Record DTF Offsets ($DTOU)

This macro instruction generates a list of equates to establish labels in the unit record DTFs. These labels are offsets from the beginning of the DTF and are used as displacements from the beginning of the DTF when you must access the DTF. The labels assigned by this macro instruction are shown with the respective DTFs in *Appendix B: Define the File Control Blocks.* You must not issue this macro instruction more than once for each unit record device.

The format of the $DTOU macro instruction is:

	$DTOU	[DEV-code]

Note: You do not assign a name to the $DTOU macro instruction.

DEV-code specifies the device described in the related DTF. If the operand is not entered, MFCU is assumed. The valid codes are:

Code	Meaning
MFCU	MFCU
PRNTR	Line printer
D1442	1442 Card Read/Punch
CONSOL	5471 Printer-Keyboard

Get or Put for Unit Record ($GPU)

You can use this macro instruction in one of three ways:

- Get data from an input file
- Put data to an output file
- Get data from and put data to a combined file

You may write your own routines to handle special conditions, such as:

- An end-of-file routine for input files
- An error handling routine
- An overflow routine for printer files

This macro instruction requires that you use the $DTOU macro instruction to establish the labels for the unit record 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 $GPU macro instruction.

Note: The generated code from the $GPU macro instruction does not perform carriage control for printer files. To prevent overprinting, you must set the space before, space after, skip before, skip after bytes in the printer DTF to perform the spacing you desire.

The format of the $GPU macro instruction is:

[Name]	$GPU	ERR-address,MODUL-label [,OPC-code] [,DEV-code] [,DTF-address] [,EOF-address] [,OVRTN-address] [,DEFER-Y/N]

ERR-address specifies the address of a routine you supply to handle permanent error conditions encountered during the input or output operation. This operand is required.

In your error handling routine, you must determine whether to cancel the program or continue processing with the next record. You may wish to determine the type of error encountered before canceling or continuing the program. The type of error is indicated in the DTF. (See *Appendix B: Define The File Control Blocks.*)

MODUL-label provides the name of the system data management routine to be used for the input/output operation. Figure 14 shows the names of the system routines and the functions they provide. You must identify the label used in this operand by specifying it as the operand of an EXTRN instruction in your program. This operand is required.

Note: In order to conserve main storage, you should determine whether you can use the same module for more than one function rather than a new module for each function. For example, if you want to read, punch, and print, specifying $$MFFF would use less main storage than using $$MFRD, $$MFPU, and $$MFPR.

OPC-code indicates the input/output operation to be performed. If this operand is not entered, READ is assumed. The following codes may be specified:

Code	Meaning
READ	Read from the MFCU or 1442 Card Read/Punch
RDPRT	Read and print on the MFCU
RDPCH	Read and punch on the MFCU
RDPP	Read, punch, and print on the MFCU
PUNNF	Punch, with no feed, on the 1442
PUNCH	Punch on the MFCU or 1442
PUPR	Punch and print on the MFCU
PRINT	Print on a line printer or MFCU

This operand is used with the MODUL operand to perform the operation.

Device	System Module	Functions
MFCU	$$MFRD	Reads cards from either hopper.
	$$MFPU	Punches cards from either hopper.
	$$MFPR	Prints on cards from either hopper.
	$$MFRU	Reads and/or punches cards from either hopper.
	$$MFRP	Reads from and/or prints on cards from either hopper.
	$$MFPP	Punches and/or prints cards from either hopper. The defer operation may also be used.
	$$MFFF	Supports the following functions: • Read • Punch • Print • Punch deferred • Print deferred
1442	$$ARFF	Feeds, reads, and punches cards.
5203/1403	$$LPRT	Performs printing, skipping, and spacing as requested through the DTF.
Console	$$COAM	Performs input and output operations on the printer-keyboard.

Note: These modules are described in more detail in *IBM System/3 Models 4, 6, 8, and 10 Data Management and Input/Output Supervisor Logic Manual*, SY21-0512.

Figure 14. System Unit Record Module Names and Functions

Note 1: When you issue a combined command involving a read, the output operations are performed on the card presently at the wait station and before the next card is read. Therefore, before issuing the first combined command, you must issue a READ command to get the first input record and advance it to the wait station.

Note 2: For any read operation, the logical record address (specified by RCAD in the DTF) is replaced by the current input buffer address. Before a combined file is used for output, you should ensure that the logical record address (offset $RDLRA for the MFCU or $FDLRA for the 1442) points to the logical record for the output operation. This can be done by using the input buffer as the logical record or by storing the logical record address at the offset before performing the output operation.

DEV-code specifies the device for the file. If this operand is not entered, MFCU is assumed. The following codes may be specified:

Code	Meaning
MFCU or MFCU1	MFCU, primary hopper
MFCU2	MFCU, secondary hopper
PRNTR or PRNTR1	5203 printer (left carriage) or 1403 printer
PRNTR2	5203 printer (right carriage)
D1442	1442 card read/punch

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

EOF-address is used only with the input function. This operand provides the address of a routine you have written that is to receive control when end-of-file is reached.

OVRTN-address is required for line printer output files. It specifies the address of your routine that handles the overflow condition.

DEFER-Y or N is used only with output operations to the MFCU. This operand enables you to print one record on a card and punch a different record in the same card. To do this, you first issue the $GPU macro instruction for either a print or a punch with DEFER-Y. You then modify the logical record as needed to a different format and issue another $GPU macro instruction for the remaining operation with DEFER-N. Both operations are then performed. If this operand is not specified, N (no) is assumed.

Construct an Interface to the Printer-Keyboard ($PKBU)

This macro provides access to the 5471 Printer-Keyboard (console). The following functions are provided:

- Get a record
- Put a record
- Write to operator with reply

To use this macro instruction, you must issue the $DTOU macro instruction to establish the labels for the unit record 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 $PKBU macro instruction.

The format of the $PKBU macro instruction is:

[Name]	$PKBU	[OPC-code] [,DTF-address] [,ERR-address] [,RECL-number] [,REPLY-number] [,SPACE-number] [,RCAD-address] [,REQK-Y/N] [,FILL-Y/N] [,EOF-address] [,CHK-Y/N] [,RTN-Y/N]

OPC-code specifies the operation to be performed. If this operand is not entered, SET is assumed. The operation codes are:

Code	Meaning
GET	Get a record from the console
PUT	Put a record to the console
WTOR	Put a record to the console and get a reply
SET	Use the operation previously defined for this DTF. If there is no previous operation established, PUT is assumed

DTF-address is the address of the high-order byte of the DTF to be used in this operation. If this operand is not specified, the address is assumed to be in register 2.

ERR-address specifies the address of your error routine which receives control whenever a permanent error is detected on the console. If this operand is not entered, no checks are performed for permanent error conditions.

This operand should not be used when CHK-Y is specified.

RECL-number specifies the decimal length of the logical record. If this operand is not entered, the record length previously established in the DTF is used.

REPLY-number specifies the decimal length of the reply to be received during a WTOR operation. If this operand is not entered, the reply length previously established in the DTF is used.

SPACE-number is a two-digit decimal number that indicates the number of lines to be spaced before and after the console operation. The first digit specifies the number of lines to space before; the second digit specifies the number of lines to space after the operation. If only one digit is specified, it is used as the space-after value. If this operand is not entered, the spacing values previously established in the DTF are used. In any case, the console always performs at least one space before for each operation.

RCAD-address specifies the address of the leftmost byte of the buffer area used. If this operand is not entered, the address previously established in the DTF is used.

REQK-Y or N specifies whether the request key must be pressed before an input record can be accepted from the console. If the operand is not entered, the entry previously established in the DTF is used.

Note 1: When REQK-Y is specified, the program must be identified as an inquiry program (a program that calls roll-out) when the program is linkage-edited.

Note 2: When FTYP-C is specified, in the $DTFU macro instruction, or when OPC-WTOR is specified in the $PKBU macro instruction, REQK-Y is ignored.

FILL-Y or N specifies whether the input or reply record from the operator must be the exact length requested. If this operand is not entered, the entry previously established in the DTF is used.

EOF-address is the address of the routine in your program that is to receive control when an end-of-file record is read from the printer keyboard. This operand should not be used when CHK-Y is specified.

CHK-Y or N specifies whether the console operation must be completed before control is returned to your program. N indicates a check for completed operation will be performed; the console operation must be completed before control returns. Y indicates no check. Control returns directly after the console operation is started, with a completion code of X'00' to indicate the operation has not been completed. The completion code is located in the printer-keyboard post-open DTF at label $CDCMP. If FTYP-C was specified in the $DTFU macro instruction or if OPC-WTOR was specified in the $PKBU macro instruction, control is not returned to your program until the output operation is completed. If this operand is not specified, the option established in the DTF is used. See *Note* in the discussion of RTN-Y or N.

RTN-Y or N is specified only if CHK-Y was specified for a console operation. This operand specifies the type of check operation to be performed. If Y is specified, the console operation is checked for completion, and control returns to your program. The completion code in the DTF remains X'00' until the operation is completed. You must check the completion code in your program to determine whether it has changed. If N is specified, the console operation is checked for completion, and a wait is performed if the operation is not completed. Control is not returned to your program until completion occurs. The completion code reflects the outcome of the operation. If this operand is not specified, the option established in the DTF is used.

If the entry specified in the CHEK parameter in the $DTFU or $PKBU macro instruction is not the same as the entry specified in the RTN parameter of the $CHK macro instruction, control does not return from a console operation until the operation has been completed.

Note: When CHK-Y is specified for a console operation, you must determine that the operation has been completed before you can initiate another console operation. You can do this in two ways:

1. Recall the operation, using the check function in the console macro instruction.

2. Issue the $CHK macro instruction, using the associated check list macro instruction ($CKL).

The operation must be tested at least once to determine that the operation was completed.

You can recall an operation by issuing the same macro instruction again or by issuing another $PKBU instruction with no operands specified. If no operands are specified, the address of the DTF must be in register 2. When the operation is recalled, the CHK option is ignored and the RTN option is used to determine the type of check requested by the recall.

For more information on using the check routine to test completion, see *Construct a Check List ($CKL)* and *Check for I/O Completion ($CHK)*

Print a Message ($PRNT)

This macro instruction prints a message on the halt/syslog device using the halt/syslog routines. This instruction provides a printed message with wait and no halt. 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 $PRNT macro instruction.

Note: If the halt/syslog device has been allocated to your program through another DTF, or if the length of the message is greater than 132, no message is printed. Control is returned directly to your program.

The format of the $PRNT macro instruction is:

[Name]	$PRNT	NAME-address,LEN-number

NAME-address specifies the address of the leftmost byte of the message you want printed. This operand is required. If the SYSLOG device is a line printer, the message must be aligned on a X'7C' boundary.

LEN-number is a decimal number specifying the length of the message to be printed. This operand is required.

Disk Device Support

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

- Build a pre-open DTF for disk and assign its offsets.
- Build an input/output block (IOB) for disk and assign its offsets.
- Build the interfaces required to get input records from a disk device via a get or a read.
- Build the interfaces required to put output records to a disk device via a put or a write.
- Build the routine to wait for disk completion.

The disk DTFs provide information to the disk data management, and the disk IOBs provide information to the input/output supervisor routines that perform the input or output operations. These operations are provided through the disk support macro instructions.

Define the File for Disk ($DTFD)

The DTF provides information needed to allocate and open a file on the disk device. This macro instruction generates the code that builds the disk DTF. See *Appendix B: Define the File Control Blocks* for a description of the pre-open and post-open disk DTFs.

The format of the $DTFD macro instruction is:

[Name]	$DTFD	AC-code, RECL-number, NAME-filename,IO-address, BLKL-number [,DISK-5444/5445] [,UP-mask] [,BUFNO-1/2] [,MVF-N/Y] [,LIM-N/Y] [,ORD-N/Y] [,BIN-N/Y] [,CHN-address] [,RCAD-address] [,ENT-number] [,MSTX-address] [,MVFN-number] [,KEYL-number] [,KEYD-number] [,KEYA-address] [,MVFT-address]

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

Code	Access Method
CA	Consecutive add
CG	Consecutive get
CO	Consecutive output
CU	Consecutive update
DG	Direct get
DO	Direct output
DU	Direct update
IA	Indexed add
IO	Indexed output
IR	Index random get
IRA	Index random add
IRU	Indexed random update
IRUA	Indexed random update and add
IS	Indexed sequential get
ISA	Indexed sequential add
ISU	Indexed sequential update
ISUA	Indexed sequential update and add

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

NAME-file name specifies the name of the file. The name may be eight characters or less in length. This operand must be specified.

IO-address provides the address of the leftmost byte of an area in main storage reserved for all buffers and IOBs for the access method. OPEN allocates buffers and builds the IOBs. This operand must be specified. The amount of main storage required is:

[(BLKL+X) times BUFNO]
plus (256 + X) if the access method is indexed;
plus (256 + X + one key length) if the access method is indexed multivolume;
plus (Y) if the file type is multivolume and the access method is indexed random.

where:
X = 22 bytes when using the 5444;
26 bytes when using the 5445
Y = 32 bytes for 5444 multivolume random input or update;
16 bytes for 5445 multivolume random input or update;
88 bytes for 5444 multivolume random add or update and add;
52 bytes for 5445 multivolume random add or update and add

BLKL-number specifies the number of bytes in the buffer. The minimum number can be determined as follows:

- If the record length is less than or equal to 256 and evenly divisible into 256, the buffer length is 256.
- If the record length is greater than 256 and a multiple of 256, the buffer length is equal to the record length.
- If the record length is not evenly divisible into 256 and not a multiple of 256, the buffer length is the multiple of 256, next higher than the record length plus 255.

Note: These buffer lengths are minimum lengths. Larger lengths may be specified, but they must be in multiples of 256.

The following access methods can always operate in a minimum of a 256-byte buffer:

Consecutive output
Consecutive add
Consecutive output multivolume
Consecutive add multivolume
Consecutive input
Consecutive input multivolume
Indexed output
Indexed output and add
Indexed output multivolume
Indexed sequential input
Indexed sequential input with limits
Indexed sequential input multivolume

DISK-5444 or 5445 specifies whether the disk device is the IBM 5444 Disk Storage Drive or the IBM 5445 Disk Storage. If this operand is not specified, 5444 is assumed.

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

BUFNO-1 or 2 allows you to specify either one or two buffers for the file. You can use two buffers only with consecutive access methods. All consecutive access methods allow dual buffering except the consecutive update and consecutive update multivolume. If this operand is omitted, one buffer is assumed.

MVF-N or Y specifies whether the access method is multivolume. If this operand is omitted, N (no) is assumed.

LIM-N or Y 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.

ORD-N or Y 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. ORD-Y must be specified for indexed multivolume output access methods. If this operand is not entered, N (no) is assumed.

BIN-N or Y is specified only with the direct output, direct get, and direct update access methods. Y (yes) indicates direct binary relative record numbers; N (no) indicates direct decimal relative record numbers. If this operand is omitted, N is assumed.

CHN-address indicates the address of the next DTF in the chain of DTFs. If there is no DTF chain, the operand is omitted and X'FFFF' is assumed.

RCAD-address specifies the address of the leftmost byte of the logical record. If this operand is not entered, X'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 label $DFWKB in the DTF. For information on the mode used by the different access methods, see Figure 16.

ENT-number specifies the number of entries in the master track index. This operand is specified only for indexed random or indexed add access methods. The number of entries in the master track index is one less than the number you specify in this operand for 5445 single volume files (two less for 5445 multivolume files), when the following access methods are used:

- Indexed add
- Indexed random input and add
- Indexed random input, update, and add

MSTX-address specifies the address of the leftmost byte of the master track index in main storage. This operand must be specified for indexed random and indexed add access methods. You must allocate space in main storage for the master track index. The length of the master track index is determined by the following formulas:

- For single volume random input or update access methods:
  Length = ENT (key length + 2)

- For single volume random add or update and add access methods, 5444/5445:
  Length = ENT (key length + 2)
  3340:
  Length = ENT (key length + 2) + 2 (key length)

- For multivolume random input or update access methods, 5444:
  Length = ENT (key length + 2)
  ENT must be equal to or greater than 4.
  5445/3340:
  Length = ENT (key length + 2)
  ENT must be equal to or greater than 2.

- For multivolume random add or update and add access methods, 5444:
  Length = ENT (key length + 2)
  ENT must be greater than or equal to 4.
  5445:
  Length = ENT (key length + 2)
  ENT must be greater than or equal to 2.
  3340:
  Length = ENT (key length + 2) + 2 (key length)

MVFN-number indicates the number (in hexadecimal) of volumes for a multivolume direct access method. This operand must be specified for these access methods.

KEYL-number specifies the length of the key field and must be used for all indexed access methods but no others. The key field length can be no more than 29 bytes.

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

KEYA-address specifies one of the following:

- Main storage address of the leftmost byte of the key field for indexed random access methods.

- Main storage address of the leftmost byte of the relative record number field for direct access methods.

- Main storage address of the leftmost byte of the save area for current and last keys for indexed sequential add access methods.

- Main storage address of the leftmost byte of the save area for high and low keys for indexed sequential with limits access methods (LIM-Y).

This operand is required for these access methods.

You must allocate the main storage space for the fields. The amount of space required is:

- The number of bytes in the key field for indexed random access methods.

- 23 bytes for direct access methods with decimal keys. The decimal key is located in the rightmost 15 bytes of the field.

- 8 bytes for direct access methods with binary keys. The binary key is located in the rightmost 3 bytes of the field.

- Two times the key length for indexed sequential add or indexed sequential with limits access methods. The low key is located in the left half of the field, the high key in the right half.

MVFT-address must be specified for all multivolume direct files, and only for the access methods used with these files. This operand specifies the address of the leftmost byte of the table of extents used for the access methods used with these files. You must allocate main storage space for the table. The number of bytes allocated must be equal to six times the number of volumes in the file for the 5444; seven times the number of volumes for the 5445.

Disk DTF Offsets ($DTOD)

This macro instruction generates a list of equates used to label the fields in the post-open disk DTF. The labels generated are provided with the disk post-open DTF in *Appendix B: Define the File Control Blocks.* The labels generated by the macro instruction are offsets from the beginning of the DTF and must be used as displacements from the DTF address when you access the DTF.

The format of the $DTOD macro instruction is:

Input/Output Block for Disk ($IOBD)

This macro instruction generates a disk input/output block (IOB) for use by the disk input/output supervisor. A 22-byte IOB is generated for 5444 disk devices, a 26-byte IOB is generated for 5445 disk devices. For a detailed description of the disk IOB, see *Appendix C: Disk Input/Output Block.*

The format of the $IOBD macro instruction is:

[Name]	$IOBD	[DISK-5444/5445] [,CYL-number] [,SCTR-number] [,HEAD-number] [,NUM-number] [,BUFF-address] [,Q-number] [,ERREC-IOS/USER] [,LOG-Y/N] [,VER-Y/N] [,CHN-address]

Disk-5444 or 5445 specifies whether the disk device being used is the IBM 5444 Disk Storage Drive Model 1 or the IBM 5445 Disk Storage.

CYL-number indicates the beginning cylinder to be accessed. You can specify the cylinder by a decimal number (1-199) or or a hex number (X'01'-X'C7') for the 5445, a decimal number (4-202) or a hex number (X'04'-X'CA') for the 5444, or a decimal number (1-166) or a hex number (X'01'-X'A6') for the 3340. If this operand is not entered, X'FF' is assumed. You must then insert the correct number into the IOB before performing the input/output operation. This can be done through the macro used to initiate the I/O operation.

SCTR-number specifies the first sector to be accessed. The number specified must be a decimal from 1 through 48 for the 5444 and 3340 disk drives or from 1 through 20 for the 5445 disk drive. If this operand is not entered, X'FF' is assumed. You must then insert the correct number before performing the input/output operation. You can specify the sector through the $RDD and $WRTD macro instructions.

HEAD-number is specified only for the 5445 disk storage drive. It specifies the head to be used with the cylinder and sector when an I/O operation is performed. The number specified may be decimal (0-19) or hexadecimal (X'00'-X'13'). If this operand is omitted, X'FF' is assumed and the value must be updated when the I/O operation is performed.

NUM-number specifies the number of sectors used. You may specify the number in either decimal or hexadecimal form. If this operand is not entered, X'00' is assumed. You must then update this number in the IOB before performing the input/output operation. This can be done through the macro instruction used to issue the input/output operation.

BUFF-address is the address of the leftmost byte of your data area. If this operand is omitted, X'FFFF' is assumed, and you must update the IOB before performing the input/output operation.

Q-number specifies the drive on which the record is located. You may specify the disk drive alone (F1,R1,F2, R2,D1,D2), or you may specify the hexadecimal Q-code in the form Q-X'nn', where nn is a valid hexadecimal Q-code. The valid Q-codes are shown in Figure 15. If you specify only the disk drive, you must set the read/write bits (the last four bits of the Q-code) before you can perform the I/O operation. This can be done through the $RDD or $WRTD macro instructions.

I/O Operation	Q-Byte Setting (Hex)	
	DRIVE 1	DRIVE 2
5444 Removable Disk		
Control	A0	B0
Read	A1	B1
Write	A2	B2
Scan	A3	B3
5444 Fixed Disk		
Control	A8	B8
Read	A9	B9
Write	AA	BA
Scan	AB	BB
5445 Disk		
Control	C0	C8
Read	C1	C9
Write	C2	CA
Scan	C3	CB

Figure 15. Q-Byte Hexadecimal Settings

ERREC-IOS or USER indicates whether the input/output supervisor is to handle error recovery. If you specify IOS, the supervisor handles error recovery and retries the operation when errors occur. If you specify USER, the supervisor does not retry the operation and returns control to you. If this operand is not specified, IOS is assumed.

LOG-Y or N indicates whether the I/O supervisor is to log errors that occur during the operation. If you specify Y (yes), error conditions are logged on the system pack. This information is used by IBM customer engineers. N (no) indicates no logging is to be done for this IOB. If this operand is not entered, Y is assumed.

VER-Y or N is used for output operations. Y (yes) indicates the written data should be verified; N (no) indicates it should not. If this operand is omitted, Y is assumed.

CHN-address specifies the address of the leftmost byte of the next IOB for the operation if more than one IOB is required.

Input/Output Block Offsets ($IOED)

This macro instruction generates equates to establish labels for the disk IOBs. These labels are offsets from the beginning of the IOB and are used as displacements from the beginning of the IOB when you wish to refer to one of the fields. The labels generated by this macro instruction are given with the fields of the IOB in *Appendix C: Disk Input/Output Block.*

The format of the $IOED macro instruction is:

	$IOED	

Construct a Disk Get Interface ($GETD)

The $GETD macro instruction generates the interface needed to communicate with disk data management when a record is being read from a disk file. To use this macro instruction, construct a disk DTF for the file and use the $DTOD macro instruction 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. The names of the data management modules and the functions of the modules are shown in Figure 16. 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 macro instruction.

The code generated by this macro instruction 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 macro instruction is:

[Name]	$GETD	AC-code[,DTF-address] [,ERR-address] [,EOF-address] [,NRF-address] [,LSTV-address] [,NOKY-address]

$GETD	$PUTD	AC-Code 5444	AC-Code 5445	Move/ Locate Mode	System Module 5444	System Module 5445	Access Method
	✓	CA	CA5	M	$$CSOP	$$CFOP	Consecutive Add
	✓	CAM	CAM5	M	$$CSOM	$$CFOM	Consecutive Add Multivolume
✓		CG	CG5	M	$$CSIP	$$CFIP	Consecutive Get
✓		CGM	CGM5	M	$$CSIM	$$CFIM	Consecutive Get MVF
	✓	CO	CO5	M	$$CSOP	$$CFOP	Consecutive Output
	✓	COM	COM5	M	$$CSOM	$$CFOM	Consecutive Output MVF
✓	✓	CU	CU5	L	$$CSUP	$$CFUP	Consecutive Update
✓	✓	CUM	CUM5	L	$$CSUM	$$CFUM	Consecutive Update MVF
✓		DG	DG5	L	$$DAID	$$DFID	Direct Get
✓		DGA	DGA5	L	$$DAIB	$$DFIB	Direct Get (Binary Keys)
✓		DGAM	DGAM5	L	$$DAIT	$$DFIT	Direct Get (Binary Keys) MVF
✓		DGM	DGM5	L	$$DAIM	$$DFIM	Direct Get MVF
✓	✓	DO	DO5	L	$$DAUD	$$DFUD	Direct Output
✓	✓	DOA	DOA5	L	$$DAUB	$$DFUB	Direct Output (Binary Keys)
✓	✓	DOAM	DOAM5	L	$$DAUT	$$DFUT	Direct Output (Binary Keys) MVF
✓	✓	DOM	DOM5	L	$$DAUM	$$DFUM	Direct Output MVF
✓	✓	DU	DU5	L	$$DAUD	$$DFUD	Direct Update
✓	✓	DUA	DUA5	L	$$DAUB	$$DFUB	Direct Update (Binary Keys)
✓	✓	DUAM	DUAM5	L	$$DAUT	$$DFUT	Direct Update (Binary Keys) MVF
✓	✓	DUM	DUM5	L	$$DAUM	$$DFUM	Direct Update MVF
	✓	IA	IA5	M	$$IOAD	$$IFAD	Indexed Add
	✓	IAM	IAM5	M	$$IOAM	$$IFAM	Indexed Add MVF
	✓	IO	IO5	M	$$IOUT	$$IFUT	Indexed Output
	✓	IOM	IOM5	M	$$IOUM	$$IFUM	Indexed Output MVF
✓		IR	IR5	L	$$IRIP	$$IGIP	Indexed Random Input
✓	✓	IRA	IRA5	L	$$IRAD	$$IGAD	Indexed Random Add
✓	✓	IRAM	IRAM5	L	$$IRAM	$$IGAM	Indexed Random Add MVF
✓	✓	IRBM	IRBM5	L	$$IRBM	$$IGBM	Indexed Random Update & Add MVF
✓		IRM	IRM5	L	$$IRIM	$$IGIM	Indexed Random Input MVF
✓	✓	IRU	IRU5	L	$$IRUP	$$IGUP	Indexed Random Update
✓	✓	IRUA	IRUA5	L	$$IRUA	$$IGUA	Indexed Random Update & Add
✓	✓	IRUM	IRUM5		$$IRUM	$$IGUM	Indexed Random Update MVF
✓		IS	IS5	M	$$ISIP	$$IHIP	Indexed Sequential Input
✓	✓	ISA	ISA5	M	$$ISAD	$$IHAD	Indexed Sequential Add
✓	✓	ISAM	ISAM5	M	$$ISAM	$$IHAM	Indexed Sequential Add MVF
✓	✓	ISBM	ISBM5	M	$$ISBM	$$IHBM	Indexed Sequential Update & Add MVF
✓		ISL	ISL5	M	$$ISIL	$$IHIL	Indexed Sequential Input Within Limits
✓		ISM	ISM5	M	$$ISIM	$$IHIM	Indexed Sequential Input MVF
✓	✓	ISU	ISU5	L	$$ISUP	$$IHUP	Indexed Sequential Update
✓	✓	ISUL	ISUL5	L	$$ISUL	$$IHUL	Indexed Sequential Update Within Limits
✓	✓	ISUM	ISUM5	L	$$ISUM	$$IHUM	Indexed Sequential Update MVF
✓	✓	ISUA	ISUA5	M	$$ISUA	$$IHUA	Indexed Sequential Update & Add

The Model 12 does not support multivolume or indexed disk files in the simulation area; thus AC (access) codes in these macros reflect these differences.

Figure 16. Disk Data Management Modules

AC-code specifies the appropriate access method. One of the codes from Figure 16 must be used.

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 should be passed in the event of a permanent I/O error. If this operand is not specified, no permanent I/O error checking code is generated.

EOF-address specifies the address in your program that receives control when the end of file is detected. 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.

LSTV-address is used when processing a random, offline, multivolume file. This operand supplies the address in your program which receives control when the requested key is too high for the final volume in a multivolume file.

NOKY-address supplies the address in your program that is to receive control under either of the following conditions:

- The requested key is too low for the current volume when processing an indexed random offline multivolume file.
- The requested key is too high for any volume when processing an indexed random online multivolume file.

This operand is not used with other access methods.

Read From Disk ($RDD)

This macro instruction generates an interface to the disk input/output supervisor that is to read from the disk device. When using this macro instruction, you must:

- Provide an IOB and use the $IOED macro instruction to establish the offsets in the IOB.
- Wait for the completion of the input operation.
- Check for end of data when the record is received.

If both reading and writing are to be performed (using the same IOB) for a program, the bits of the Q-byte will be altered to cause an invalid operation. In this case, you must set off the bits of the Q-byte for all but the first read (or write) operation in the program.

If you will need to use the data in register 1 at a later time, you should save the contents of that register before issuing the $RDD macro instruction.

The format of the $RDD macro instruction is:

[Name]	$RDD	IOB-address,CS-address,NSECT-number [,DISK-5444/5445]

IOB-address provides the address of the leftmost byte of the IOB which you created through your $IOBD macro instruction. The label provided must be the same as the name specified on your $IOBD macro instruction.

CS-address is the address of the rightmost byte of the main storage area containing the disk cylinder/sector address of the area you want to read. The cylinder/sector address for use with the 5444 is a two-byte, hexadecimal number. The first byte specifies the cylinder; the second specifies the sector. For use with the 5445, a three-byte hexadecimal disk address is provided through this entry. The first byte specifies the cylinder; the second, the head number; the third, the sector.

NSECT-number indicates the hexadecimal number of sectors, minus one, to be read in this operation.

DISK-5444 or 5445 specifies whether the operation is on a 5444 disk drive or a 5445 disk drive. If this operand is omitted, 5444 is assumed.

Construct a Disk Put Interface ($PUTD)

The $PUTD macro instruction generates the interface needed to communicate with disk data management when putting a record to disk or updating a previously retrieved record. You must provide a DTF for the file and use the $DTOD macro instruction 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 output operation. (See Figure 16). 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 macro instruction.

The code generated by this macro instruction 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 data management used for direct output is really for direct update; therefore, you must use the update method of processing.

The format of the $PUTD macro instruction is:

[Name]	$PUTD	AC-code[,DTF-address] [,ERR-address] [,EOX-address] [,DUP-address] [,SERR-address] [,KERR-address] [,UPD-Y/N] [,LSTV-address] [,NOKY-address] [,HKER-address]

AC-code specifies the access method being used. One of the codes from Figure 16 must be used.

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

ERR-address is the address of the area in your program where control should be passed when a permanent I/O error occurs. If this operand is not specified, no permanent I/O error checking code is generated.

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 entered only when creating a consecutive or indexed file or when records are to be added to the file.

DUP-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 add access method.

SERR-address is the address in your program where control is passed in the event of a sequence error while loading an indexed file.

KERR-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.

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

LSTV-address specifies the address in your program that receives control when a requested key is too high for the last specified volume. This operand is used only when processing an indexed, random, offline, multivolume file.

NOKY-address supplies the address in your program that is to receive control under either of the following conditions:

- The requested key is too low for the current volume when processing an indexed, random, offline, multivolume file.
- The requested key is too high for any volume when processing an indexed random online multivolume file.

This operand is not used with other access methods.

HKER-address specifies the address in your program that is to receive control when an indexed sequential add multivolume is attempted and the requested key is higher than any key presently in the file, but lower than the highest permissible key.

Write to Disk ($WRTD)

This macro instruction generates an interface to the disk input/output supervisor needed to write records to disk. When you use this macro instruction, you must:

- Provide an IOB, and use the $IOED macro instruction to establish the offsets in the IOB.
- Wait for the completion of the output operation.

If both reading and writing are to be performed (using the same IOB) for a program, the bits of the Q-byte will be altered to cause an invalid operation. In this case, you must set off the bits of the Q-byte for all but the first read (or write) operation in the program.

If you will need to use the data in register 1 at a later time, you should save the contents of that register before issuing the macro instruction.

The format of the $WRTD macro instruction is:

[Name]	$WRTD	IOB-address,CS-address,NSECT-number [,DISK-5444/5445]

IOB-address provides the address of the disk IOB for this operation. The address is the name specified on the related $IOBD macro instruction.

CS-address is the address of the rightmost byte of the main storage area containing the disk cylinder/sector address of the area to which you want to write. The cylinder/sector address for use with the 5444 is a two-byte hexadecimal number. The first byte specifies the cylinder; the second specifies the sector. For use with the 5445, a three-byte hexadecimal disk address is provided through this entry. The first byte specifies the cylinder; the second, the head number; the third, the sector.

NSECT-number specifies the number of disk sectors, minus one, to be written.

DISK-5444 or 5445 specifies whether the operation is on a 5444 disk drive or a 5445 disk drive. If this operand is omitted, 5444 is assumed.

Wait for Disk IOS Completion ($WAIT)

This macro instruction is used with the $RDD and $WRTD macro instructions. It generates the code which allows you to wait for completion of the disk IOS operation. You provide the label of the associated IOB (whose offsets are established through the $IOED macro instruction) and an address to receive control in the event of an error. If you will need to use the data in register 1 at a later time, you should save the contents of that register before issuing the $WAIT macro instruction.

The format of the $WAIT macro instruction is:

[Name]	$WAIT	[IOB-label] [,ERR-address]

IOB-label is the name assigned to the IOB in the $IOBD macro instruction. This same IOB must have previously been specified in either a $RDD or $WRTD macro instruction. If this operand is not entered, the address is assumed to be in register 1.

ERR-address specifies the address of the routine in your program that handles errors detected in the operation. If this operand is not entered, no error checking is performed.

Tape Device Support

This section describes the macro instructions that support the IBM 3410/3411 Magnetic Tape Subsystem. The following functions are provided:

- Build a pre-open DTF for tape and assign its offsets.
- Build the interfaces required to read input records from a tape device via a get or a read.
- Build the interfaces required to write output records to a tape device via a put or a write.
- Build the interface required to issue tape control commands.
- Wait for completion of read, write, or tape control operations.

The tape DTFs provide information to the tape data management routines that perform the input/output operations. These operations are provided through the tape support macro instructions.

Define the File for Tape ($DTFT)

The DTF provides information needed to allocate and open a tape device. This macro instruction generates the code that builds the tape DTF. See *Appendix B: Define the File Control Blocks* for a description of the pre-open and post-open DTFs.

The format of the $DTFT macro instruction is:

[Name]	$DTFT	NAME-filename,IO-address, AC-IN/OUT,BLKL-number, RECL-number [,UP-mask] [,CHN-address] [,BASIC-Y/N] [,MODE-LOCATE/MOVE] [,MBUFF-Y/N] [,RCAD-address] [,RECFM-code] [,LIOA-number] [,SPAN-Y/N] [,CODE-A/E] [,OSET-B/number] [,END-code]

NAME-filename is a required operand specifying the name of the tape file. The filename can be up to eight characters in length and must be the same as the name on the // FILE statement.

IO-address specifies the address of the leftmost byte of the main storage area used to contain all buffers and IOBs. This operand is required. The length of the area specified by this address is specified in the LIOA operand.

Note: If basic data management routines are used to process the file, this operand should point to a 22-byte area to contain the tape IOB.

AC-IN or OUT specifies the type of file. IN specifies an input file; OUT, an output file. This operand is required.

BLKL-number is a required operand that specifies the decimal block length for the file. The minimum block length allowed is 18 bytes. If a shorter length is specified, 18 is assumed. For files with fixed-length records, the block length must be a multiple of the record length; for files with variable-length records, the block length must equal, the length of the longest record plus eight.

Note: If basic tape data management is used, the block length in the DTF ($TDBKL) must be updated after the file is opened and before any read or write operation is performed. The field must also be updated before any subsequent read or write if the length used is different than the previous read or write.

RECL-number is a decimal value specifying the length of a logical record in the file. If variable-length records are used for the file, the record length specified must be equal to the longest record plus four. The minimum record length when variable-length records are used is four, which results in zero-length records. The minimum record length for files using fixed-length records is 18. This operand is required.

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

CHN-address indicates the address of the next DTF in the chain of DTFs. If there is no DTF chain, the operand is omitted and X'FFFF' is assumed.

BASIC-Y or N specifies whether this DTF uses the basic access method. If this operand is not entered, N (no) is assumed.

Note 1: BASIC-Y must be specified if any of the following macro instructions are used to process the file: $RDT, $WRTT, $CTLT or $WTT.

Note 2: If you process ASCII files using the basic access method, you must translate the characters in your program.

Note 3: Multivolume files are supported with the basic access method; the EXTRN that is used for this method must be for $$BTMM. $$BTTM and $$BTAM cannot be used in the same program. $$BTMM supports both single and multivolume files.

Note 4: Deferred open is not allowed with the basic access method.

MODE-LOCATE or MOVE indicates whether the locate mode or move mode is used. If this operand is not specified, MOVE is assumed. When locate mode is specified, the record address (RCAD-address) is set to the address of the record in the buffer. When move mode is used, records are moved from the buffer to the location specified by the record address.

Locate mode is valid only for input files.

MBUFF-Y or N indicates whether more than one buffer is used. If this operand is not specified, N (no) is assumed. The number of buffers is determined by the length of the I/O area, specified by the LIOA operand.

RCAD-address specifies the symbolic record area address when move mode is used. If this operand is not specified, X'0000' is assumed and the address must be supplied when the operation is requested.

Note: When basic tape data management routines are used to process the file, this operand must point to an area of at least 80 bytes for use by the open routine. After the file is opened, another area can be used as the buffer. To do this, you must update the buffer address at location $TDWKB in the tape DTF.

RECFM-code specifies the record format used for the file. The codes and their meanings are:

Code	*Record Format*
F	Fixed, EBCDIC or ASCII
FB	Fixed blocked, EBCDIC or ASCII
V	Variable, EBCDIC
VB	Variable blocked, EBCDIC
D	Variable, ASCII
DB	Variable blocked, ASCII

If this operand is not specified, F is assumed.

LIOA-number is the total decimal length of the I/O area. If multiple buffers are used, the area must be large enough to contain the IOBs and buffers for the number of buffers used. The following formula can be used to determine the length of the buffer area:

LIOA = (22 + block length) (number of buffers).

The minimum length of 102 bytes is assumed if this operand is not specified.

SPAN-Y or N specifies whether spanned records are used. If spanned records are used, BASIC-Y must also be specified. If this operand is omitted, N (no) is assumed. Specifying SPAN-Y causes the spanned record bit in the tape label to be set on. When you use SPAN-Y, you must span the records from block to block.

CODE-A or E specifies whether the file is an EBCDIC file or ASCII file. If the file is an EBCDIC file, specify CODE-E. If the file is an ASCII file or can be either ASCII or EBCDIC, specify CODE-A. If this operand is not entered, E is assumed.

Note: If CODE-A is specified in $DTFT, the CODE operand in the $GETT or $PUTT macro instruction must also be A. If CODE-E is specified in $DTFT, the CODE operand in the $GETT or $PUTT macro instruction can be either A or E.

OSET-B or number specifies the buffer offset of an ASCII block. B indicates that the first four bytes of the block contain the decimal block length and no buffer offset is present. B is valid only when RECFM-D or RECFM-DB is also specified. Only OSET-B or OSET-00 are valid for output files. OSET-number specifies, in decimal, the length of the buffer offset for the ASCII block. This buffer offset is skipped over when the record is supplied to your program. The maximum valid specification is OSET-99. If this operand is not specified, zero is assumed.

END-code specifies the tape control actions to be taken when the file is closed. The valid codes and their meanings are:

Code	*Action*
REWIND	Rewind the tape
UNLOAD	Rewind and unload the tape
LEAVE	No action taken

If this operand is not entered, REWIND is assumed.

Tape DTF Offsets ($DTOT)

This macro instruction generates a list of equates used to label the fields in the post-open tape DTF. The labels created are provided with the tape post-open DTF in *Appendix B: Define The File Control Blocks.* The labels generated by the macro instruction are offsets from the beginning of the DTF and must be used as displacements from the DTF address when you access the DTF.

The format of the macro instruction is:

	$DTOT	

Construct a Tape Get Interface ($GETT)

The $GETT macro instruction generates the interface required to communicate with tape data management when a record is being read from a tape file. To use this instruction, you must construct a tape DTF for the file and use the $DTOT macro instruction 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 $GETT macro instruction. You must also provide the labels for the necessary data management routines through EXTRN statements in your program. The names and functions of the data management routines are shown in Figure 17.

Module Name	Type of File Being Processed
$$CSIT	EBCDIC fixed input
$$CSOT	EBCDIC fixed output
$$CSIA	EBCDIC or ASCII fixed input
$$CSOA	EBCDIC or ASCII fixed output
$$CSTI	EBCDIC fixed or variable input
$$CSTO	EBCDIC fixed or variable output
$$CSAI	EBCDIC or ASCII fixed or variable input
$$CSAO	EBCDIC or ASCII fixed or variable output

Figure 17. Tape Data Management Modules

The code generated by this macro instruction gives control to the data management routine; the routine completes execution and returns control to the generated code. If the ERR or EOF operand is specified, the generated code tests the completion code returned by data management and branches to your routine. If reading variable length records, tape data management returns the length of the record at label $TDCRL in the DTF.

The format of the $GETT macro instruction is:

[Name]	$GETT	[DTF-address] [,CODE-A/E] [,RECFM-F/V] [,ERR-address] [,EOF-address]

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.

CODE-A or E specifies whether any ASCII files are used in this program. This determines whether the data management modules used to process the files must be capable of processing both EBCDIC and ASCII files. A indicates ASCII files are used in your program; E indicates only EBCDIC files are used. If this operand is omitted, E is assumed.

Note: This operand determines the data management module that will process the file. One set of data management modules processes only EBCDIC files, another set processes both EBCDIC and ASCII files. If you have only EBCDIC files in your program, less main storage is required if you specify CODE-E or omit this operand. If you have both types of files in your program, less main storage is required if you specify CODE-A, even though you are reading an EBCDIC file.

The entry for this operand must correspond with the data management module name provided in the EXTRN in your program.

RECFM-F or V specifies whether the record to be read is fixed-length or variable-length. If this operand is not specified, F is assumed.

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.

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

Read from Tape ($RDT)

This macro instruction generates an interface to basic tape data management to read from a tape device. When using this macro instruction, you must:

- Provide a tape DTF and use $DTOT to establish the offsets in the DTF.
- Wait for completion of the input operation and check for end-of-file by using the $WTT macro instruction.
- Provide EXTRN statements in your program for the basic tape data management module ($$BTAM or $$BTMM) and for the entry point to the read routine in that module (DMBTRW).

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 $RDT macro instruction. The generated code for this macro instruction uses register 2.

The code generated by this macro instruction branches to basic tape data management to begin the read operation.

The format of the $RDT macro instruction is:

[Name]	$RDT	[DTF-address] [,DIRECT-FORW/BACK]

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

DIRECT-FORW or BACK specifies the direction of the read. If this operand is not entered, forward (FORW) is assumed.

Construct a Tape Put Interface ($PUTT)

This macro instruction generates the interface needed to communicate with tape data management when writing a record to tape. You must provide a DTF for the file and use the $DTOT macro instruction to establish the offsets in the DTF. You must also provide, through EXTRN statements in your program, the labels of the tape data management modules necessary to perform the output operation (see Figure 17).

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 $PUTT macro instruction.

The code generated by this macro instruction 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 $PUTT macro instruction is:

[Name]	$PUTT	[DTF-address] [,CODE-A/E] [,RECFM-F/V] [,ERR-address]

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

CODE-A or E specifies whether any ASCII files are used by your program. This determines whether the data management modules used to process the file must be capable of processing both EBCDIC and ASCII files. A indicates ASCII files are used in your program, E indicates only EBCDIC files are used. If this operand is omitted, E is assumed.

Note: This operand determines the data management module that will be used to process the file being defined. One set of data management modules processes only EBCDIC files, another set processes both EBCDIC and ASCII files. If you have only EBCDIC files in your program, less main storage is required if you specify CODE-E or omit this operand. If you have both types of files in your program, less main storage is required if you specify CODE-A, even though you are defining an EBCDIC file. The entry for this operand must correspond with the data management module name in the EXTRN in your program.

RECFM-F or V specifies whether the record is fixed-length or variable-length. If this operand is not specified, F is assumed.

ERR-address specifies the address in your program where control should be passed if a permanent I/O error occurs. If this operand is not entered, no permanent I/O error checking code is generated and you should check the return code in your program to determine the outcome of the operation.

Write to Tape ($WRTT)

This macro instruction generates the interface to basic tape data management needed to write records to tape. When you use this macro instruction, you must:

- Provide a DTF for the file and use the $DTOT macro instruction to establish the offsets in the DTF.
- Wait for the completion of the I/O operation by using the $WTT macro instruction.
- Provide EXTRN statements in your program for the basic tape data management module ($$BTAM or $$BTMM) and for the entry point to the write routine in that module (DMBTRW).

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 $WRT macro instruction, because the generated code for that macro instruction uses register 2.

The code generated by this macro instruction branches to basic tape data management to start the operation.

The format of the $WRT macro instruction is:

[Name]	$WRTT	[DTF-address]

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

Control Command for Tape ($CTLT)

This macro instruction generates the interface to basic tape data management to issue control commands to the tape device. It is not used to get records from or put records out on a tape file. To use this macro instruction, you must:

- Provide a DTF for the file on the tape device and use the $DTOT macro instruction to establish the offsets in the DTF.
- Wait for completion of the operation by issuing the $WTT macro instruction.
- Provide EXTRN statements in your program for the basic tape data management module ($$BTAM or $$BTMM) and for the entry point to the control routine in that module (DMBTPS).

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 $CTLT macro instruction.

The code generated by this macro instruction branches to the basic tape data management to initiate the operation.

The format of the macro instruction is:

[Name]	$CTLT	[DTF-address] [,OPC-code]

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

OPC-code specifies the control operation to be performed. The valid codes and their meanings are:

Code	*Operation*
FSF	Forward space file
FSB	Forward space block
BSF	Backspace file
BSB	Backspace block
REW	Rewind tape
RUN	Rewind and unload tape
WTM	Write tape mark

If this operand is not specified, rewind tape, REW, is assumed.

Wait For Tape I/O Completion ($WTT)

This macro instruction is used with the $RDT, $WRTT, and $CTLT macro instructions. It generates the linkage to basic tape data management in order for the tape data management to wait for the completion of operations that have been initiated. You must provide the address of the tape DTF for the file and use the $DTOT macro instruction to establish the offsets for that DTF. You must also provide EXTRN statements in your program for the basic tape data management module ($$BTAM or $$BTMM) and for the entry point in the wait routine in the module (DMBTWT). You may also provide addresses where control is to be returned in the event of a permanent I/O error, end-of-file condition, or end-of-tape condition.

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 $WTT macro instruction.

The generated code from this macro instruction checks the completion code in the DTF to determine the outcome of the operation. When an abnormal completion is detected, control is passed to the appropriate address in your program, if you have specified ERR, EOJ, or EOT, or to the next instruction in your program.

The format of the $WTT macro instruction is:

[Name]	$WTT	[DTF-address] [,ERR-address] [,EOF-address] [,EOT-address]

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

ERR-address is the address of the routine in your program that receives control when a controlled cancel is indicated in the completion code. If this operand is not entered, the controlled cancel is ignored and control returns to the next instruction in your program.

EOF-address specifies the address of your routine that receives control when end-of-file occurs. If this operand is omitted, the end-of-file condition is ignored and control returns to the next instruction in your program.

EOT-address is the address of the routine in your program that receives control when end-of-tape is detected. If this operand is not specified, the condition is ignored and control returns to the next instruction in your program.

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

3741 Support

This section describes the macro instructions that support the IBM 3741 Data Station/Programmable Work Station. The following functions are provided through the use of these macro instructions:

- Build a preopen DTF and assign its offsets.
- Build the interface required to read input records from the 3741 via a get.
- Build the interface required to write output records to a 3741 via a put.

The 3741 DTFs provide information to the data management routines that perform the input/output operations. These operations are provided through the 3741 macro instructions.

Define the File for 3741 ($DTFK)

This DTF provides information needed to allocate, open, and access a file on the 3741. This macro instruction generates the code that builds the 3741 DTF.

The format of the $DTFK macro instruction is:

[Name]	$DTFK	NAME-filename,RECL-number, IO-address [,AC-I/O] [,RCAD-address] [,BUFNO-1/2] [,CHN-address] [,UP-mask]

NAME-filename is a required operand specifying the name of the 3741 file. The name may not exceed eight characters in length.

RECL-number is a decimal value specifying the length of a logical record. The decimal value may be from 1 to 128.

IO-address specifies the address of the leftmost byte of the main storage area that is used to contain all buffers and IOBs. The length of the area specified by this address must be the record length plus 26 times the BUFNO.

AC-I/O specifies the type of DTF, input or output. If the operand is not entered, an input DTF is assumed.

RCAD-address specifies the address of the leftmost byte of the logical record. If this operand is not entered, X'FFFF' is assumed.

BUFNO-1 or 2 allows you to use one or two buffers. If this operand is omitted, one buffer is assumed.

CHN-address indicates the address of the next DTF in the chain of DTFs. If there is no DTF chain, the operand is omitted and X'FFFF' is assumed.

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

Construct a 3741 GET Interface ($GETK)

When a record is being read from a 3741 file, the $GETK macro instruction allows the 3741 data management to communicate with the 3741 file. To use this instruction, you must construct a 3741 DTF for the file and use the $DTOD macro instruction to establish the offsets in the DTF. You must also provide an EXTRN statement with the label $$CPIP to use this macro. If you need to use the data in register 2 at a later time, you should save the contents of that register before issuing the $GETK macro instruction.

The code generated by the $GETK macro instruction gives control to the data management routine; the routine completes execution and returns control to the generated code.

The format of the $GETK macro instruction is:

[Name]	$GETK	[DTF-address] [,ERR-address] ,EOF-address

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 If the controlled cancel option is taken in response to a permanent I/O error, the ERR-address supplies the address in your program to which control is passed. 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 to which control is passed when the end-of-file is detected. This operand must be specified.

Construct a 3741 PUT Interface ($PUTK)

When writing a record to the 3741, the $PUTK macro instruction allows the 3741 data management to communicate with the 3741 file. You must provide a DTF for the file and use the $DTOD macro instruction to establish the offsets in the DTF. You must also provide an EXTRN with the label $$CPOP to use this macro instruction. 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 $PUTK macro instruction. The routine completes execution and returns control to the generated code.

The format of the $PUTK macro instruction is:

[Name]	$PUTK	[DTF-address] [,ERR-address]

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

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

Use macro

$DTOK

for 3741

offset equates

(Not IBM original!)

Command CPU—Generate the CCP Assembler Instruction ($CCP)

The format of the $CCP macro instruction is:

[Name]	$CCP	QBYTE-hex,RBYTE-hex

QBYTE-hex is a required operand specifying the Q-code for the CCP instruction.

RBYTE-hex is a required operand specifying the R-code for the CCP instruction.

Note: For a complete description of the codes for the Q-byte and the R-byte that can be entered in the operands, see *IBM System/3 Models 8, 10, 12, and 15 Components Reference Manual*, GA21-9236.

Load CPU—Generate the LCP Assembler Instruction ($LCP)

The format of the $LCP macro instruction is:

[Name]	$LCP	QBYTE-hex[,ADDR-address] [,REG-1/2] [,DISP-hex]

QBYTE-hex is a required operand specifying the Q-code for the LCP instruction.

ADDR-address specifies the address of the data. This operand is required unless REG and DISP operands are specified.

REG-1/2 specifies the base register for the displacement of the data. This operand is required unless ADDR operand is specified.

DISP-hex specifies the displacement of the data from the address pointed to by the base register. This operand is required unless ADDR operand is specified.

Note: For a complete description of the codes for the Q-byte that can be entered in the operand, see *IBM System/3 Models 8, 10, 12, and 15 Components Reference Manual*, GA21-9236.

Store CPU—Generate the SCP Assembler Instruction ($SCP)

The format of the $SCP macro instruction is:

[Name]	$SCP	QBYTE-hex[,ADDR-address] [,REG-1/2] [,DISP-hex]

QBYTE-hex is a required operand specifying the Q-code for the SCP instruction.

ADDR-address specifies the address of the data. This operand is required unless REG and DISP operands are specified.

REG-1/2 specifies the base register for the displacement of the data. This operand is required unless ADDR operand is specified.

DISP-hex specifies the displacement of the data from the address pointed to by the base register. This operand is required unless ADDR operand is specified.

Note: For a complete description of the codes for the Q-byte that can be entered in the operand, see *IBM System/3 Models 8, 10, 12, and 15 Components Reference Manual*, GA21-9236.

OCL FOR MACRO PROCESSOR

OCL statements used to load and run the macro processor can be entered through the system input device or can be called from the procedure library. The OCL statements necessary to load and run the macro processor are shown in Figure 18. The COMPILE statement shown in Figure 18 is necessary only when input is in a source library.

SAMPLE PROGRAM

This sample program uses the macro processor and the IBM System/3 Basic Assembler Program, program number 5702-AS1. The coding shown in Figure 19 produces an object program. To use the program, you must link edit the object program and execute it. The macro processor can be used with any valid assembler on the IBM System/3 Model 10 Disk System and is not limited to use with the program product 5702-AS1.

Purpose of the Sample Program

The sample program in Figure 19 is used to print input records entered from the system input device. It reads data records from the system input device and prints them on a line printer. Each printed line reproduces one input record.

Termination of the Sample Program

The sample program terminates in one of two ways:

1. After successful completion of the program, EJ is displayed on the console display unit.

2. When an error occurs during processing, one of the following halts is displayed on the console display unit:

   A1—if an error is returned from the sysin routine.

   A2—if an error is returned from the printer routine.

   You respond to these halts by pressing the start key (or the halt/reset key on systems with the dual programming feature). EJ is then displayed on the console display unit.

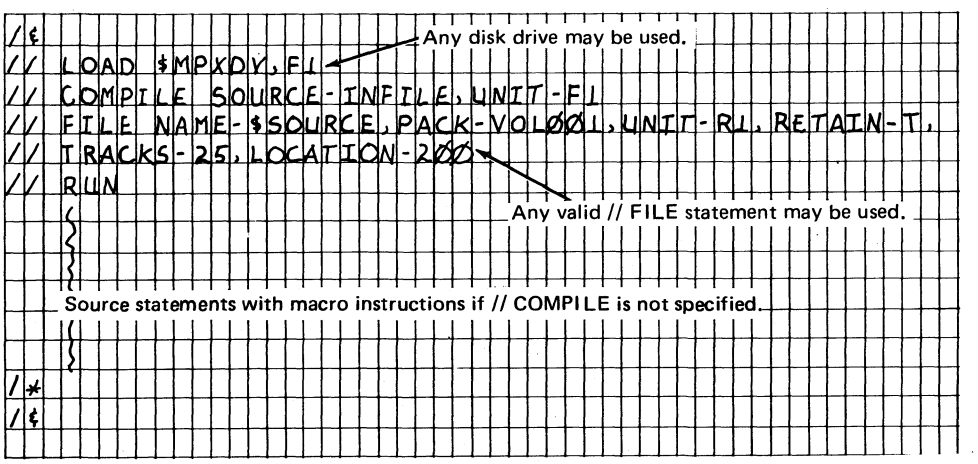
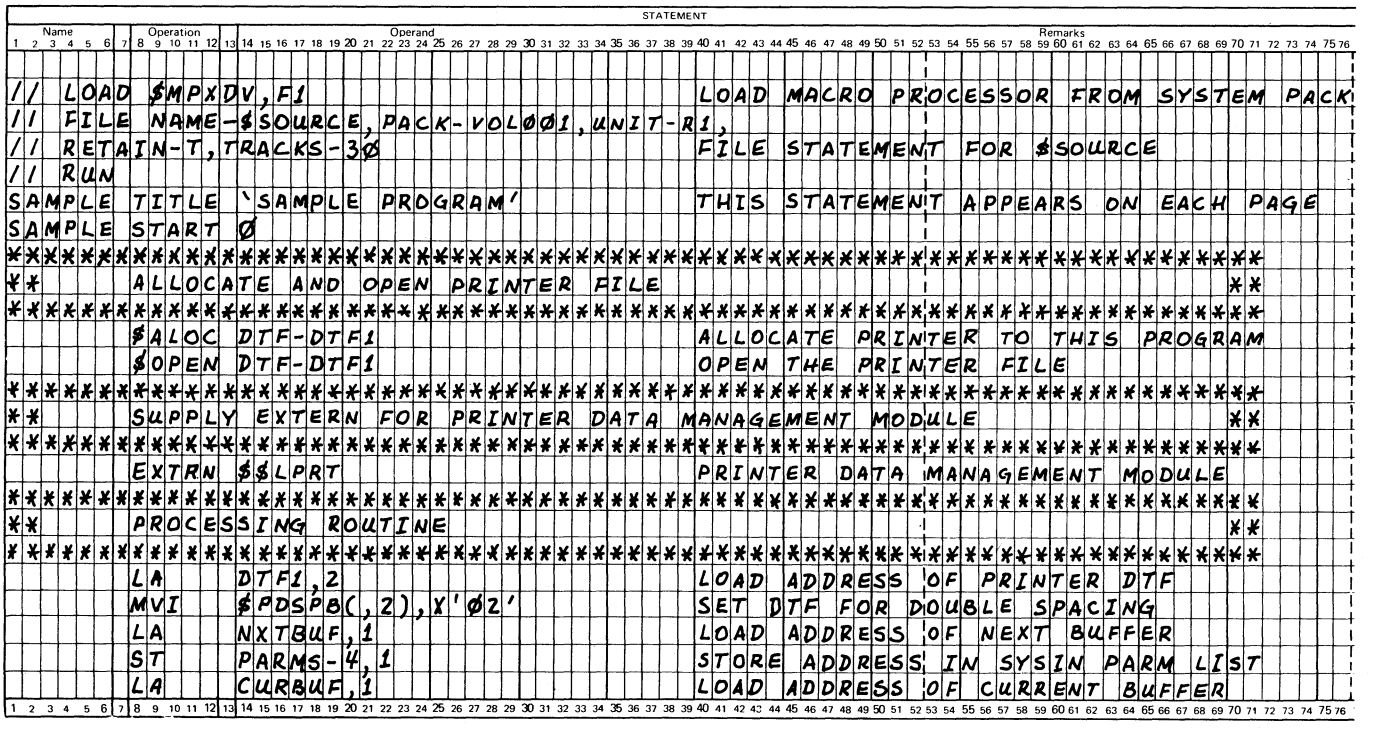
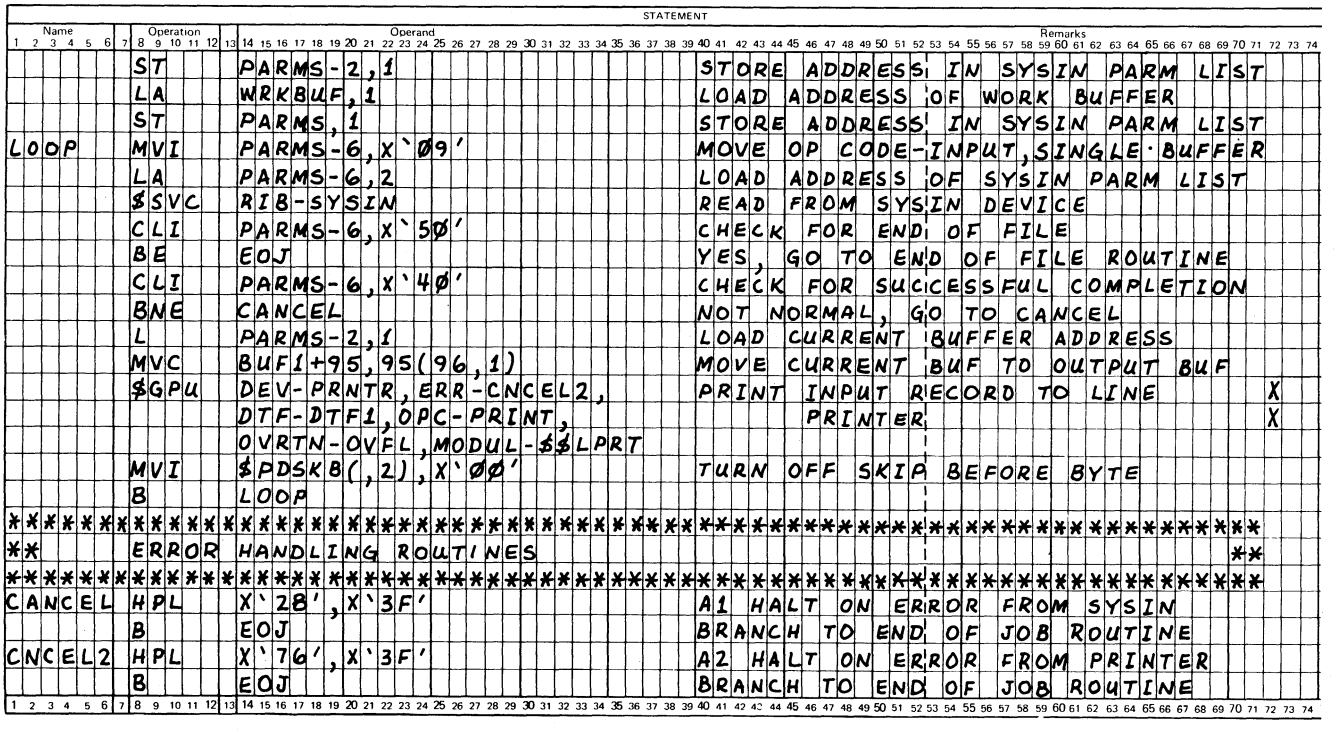


Figure 18. OCL Statements for Using the Macro Processor



```
//  LOAD  $MPXDV,F1                          LOAD MACRO PROCESSOR FROM SYSTEM PACK
//  FILE  NAME-$SOURCE,PACK-VOLØØ1,UNIT-R1,
//  RETAIN-T,TRACKS-3Ø                       FILE STATEMENT FOR $SOURCE
//  RUN
SAMPLE TITLE 'SAMPLE PROGRAM'                THIS STATEMENT APPEARS ON EACH PAGE
SAMPLE START Ø
**********************************************************************
**        ALLOCATE AND OPEN PRINTER FILE                            **
**********************************************************************
        $ALOC DTF-DTF1                       ALLOCATE PRINTER TO THIS PROGRAM
        $OPEN DTF-DTF1                       OPEN THE PRINTER FILE
**********************************************************************
**      SUPPLY EXTERN FOR PRINTER DATA MANAGEMENT MODULE            **
**********************************************************************
        EXTRN $$LPRT                         PRINTER DATA MANAGEMENT MODULE
**********************************************************************
**      PROCESSING ROUTINE                                          **
**********************************************************************
        LA    DTF1,2                         LOAD ADDRESS OF PRINTER DTF
        MVI   $PDSPB(,2),X'ØZ'               SET DTF FOR DOUBLE SPACING
        LA    NXTBUF,1                       LOAD ADDRESS OF NEXT BUFFER
        ST    PARMS-4,1                      STORE ADDRESS IN SYSIN PARM LIST
        LA    CURBUF,1                       LOAD ADDRESS OF CURRENT BUFFER
```

Figure 19 (Part 1 of 4). Sample Program



```
        ST    PARMS-2,1                      STORE ADDRESS IN SYSIN PARM LIST
        LA    WRKBUF,1                       LOAD ADDRESS OF WORK BUFFER
        ST    PARMS,1                        STORE ADDRESS IN SYSIN PARM LIST
LOOP    MVI   PARMS-6,X'Ø9'                  MOVE OP CODE-INPUT,SINGLE BUFFER
        LA    PARMS-6,2                      LOAD ADDRESS OF SYSIN PARM LIST
        $SVC  RIB-SYSIN                      READ FROM SYSIN DEVICE
        CLI   PARMS-6,X'5Ø'                  CHECK FOR END OF FILE
        BE    EOJ                            YES, GO TO END OF FILE ROUTINE
        CLI   PARMS-6,X'4Ø'                  CHECK FOR SUCCESSFUL COMPLETION
        BNE   CANCEL                         NOT NORMAL, GO TO CANCEL
        L     PARMS-2,1                      LOAD CURRENT BUFFER ADDRESS
        MVC   BUF1+95,95(96,1)               MOVE CURRENT BUF TO OUTPUT BUF
        $GPU  DEV-PRNTR,ERR-CNCEL2,          PRINT INPUT RECORD TO LINE      X
                                             PRINTER                         X
              DTF-DTF1,OPC-PRINT,
              OVRTN-OVFL,MODUL-$$LPRT
        MVI   $PDSKB(,2),X'ØØ'               TURN OFF SKIP BEFORE BYTE
        B     LOOP
**********************************************************************
**      ERROR HANDLING ROUTINES                                     **
**********************************************************************
CANCEL  HPL   X'28',X'3F'                    A1 HALT ON ERROR FROM SYSIN
        B     EOJ                            BRANCH TO END OF JOB ROUTINE
CNCEL2  HPL   X'76',X'3F'                    A2 HALT ON ERROR FROM PRINTER
        B     EOJ                            BRANCH TO END OF JOB ROUTINE
```

Figure 19 (Part 2 of 4). Sample Program

```
***********************************************************************
**     OVERFLOW ROUTINE                                              **
***********************************************************************
OVFL   LA    DTF1,2                    LOAD ADDRESS OF PRINTER DTF
       MVI   $PDSKB(,2),X'01'          TURN ON SKIP BEFORE BYTE
       B     LOOP                      BRANCH TO READ NEXT RECORD
***********************************************************************
**     END OF JOB ROUTINE                                            **
***********************************************************************
EOJ    $CLOS DTF-DTF1                  CLOSE PRINTER FILE
       $EOJ                            RETURN CONTROL TO SUPERVISOR
***********************************************************************
**     CONSTANTS AND WORK AREA                                       **
***********************************************************************
DTF1   $DTFU DEV-PRNTR,FTYP-P,         PRINTER FILE DEFINITION, OUTPUT X
               RECL-96,RCAD-BUF1,         FILE, 96 CHAR RECORDS, LOGICALX
               PTA1-BUF2,OVFL-60          RECORD AT BUF1, PRINT BUFFER
*                                         AT BUF2, OVERFLOW ON LINE 60
       $DTOU DEV-PRNTR                 DEFINE OFFSETS IN PRINTER DTF
BUF1   EQU   *                         LOGICAL RECORD FOR PRINTER
       DC    CL96' '
       ORG   *,256,X'7C'               ALIGN ON 124-BYTE BOUNDARY
BUF2   EQU   *                         PRINT PHYSICAL BUFFER
       DC    CL96' '
```

Figure 19 (Part 3 of 4). Sample Program

```
       ORG   *,256,X'00'               ALIGN ON 256-BYTE BOUNDARY
NXTBUF EQU   *                         NEXT BUFFER FOR SYSIN
       DC    CL128' '
CURBUF EQU   *                         CURRENT BUFFER FOR SYSIN
       DC    CL128' '
WRKBUF EQU   *                         WORK BUFFER FOR SYSIN
       DC    CL128' '
PARMS  DC    XL7'00'                   SYSIN PARAMETER LIST
       $COMN                           PROVIDE EQUATES FOR MACRO INST'S
       END   SAMPLE                    END OF SAMPLE PROGRAM
/*

************************************************************************
***    OCL TO ASSEMBLE THE PROGRAM                                   ***
************************************************************************

// LOAD $ASSEM,F1                      LOAD ASSEMBLER FROM DRIVE F1
// SWITCH 1XXXXXXX                     INDICATE INPUT IS ON $SOURCE
// FILE NAME-$SOURCE,PACK-VOL001,UNIT-R1,
// RETAIN-S,TRACKS-30                  FILE STATEMENT FOR $SOURCE
// FILE NAME-$WORK,PACK-VOL010,UNIT-R2,
// RETAIN-S,TRACKS-10                  FILE STATEMENT FOR $WORK
// RUN                                 ASSEMBLE THE JOB.
/&
```

Figure 19 (Part 4 of 4). Sample Program

Macro Instructions Used in the Sample Program

Eight macro instructions are used in this sample program. The macro instructions and their functions are:

Macro Instruction	Function
$ALOC	Allocates the printer file to this program
$OPEN	Opens the file after allocation
$SVC	Reads input records from the system input device
$GPU	Prints output records on the printer
$CLOS	Closes the output file
$EOJ	Calls the end-of-job routine
$DTFU	Constructs the DTF for the printer
$DTOU	Establishes the offsets for the DTF

Appendix A. Error Information

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

Figure 20 shows the error codes that may be caused by errors in macro instructions. Other error codes may be generated by the macro processor and are caused by errors in the macro definitions. These error codes are explained in Appendix B in *IBM System/3 Models 4, 6, 8, and 10 System Control Program Logic Manual,* SY21-0502.

Error Code	Error Description
BX	A keyword response has resulted in an invalid decimal digit or a boundary exceeded condition. (See note)
CE	An error in continuation exists in this macro instruction. Nonblank characters were found in columns 1-13 of the continued line. All remaining lines of this macro instruction will be flagged with the error code 'OC'.
CL	A keyword response resulted in a character string that exceeds the maximum length. (See note)
IC	An error in continuation exists in the previous macro instruction. Column 72 is blank. All remaining lines of this macro instruction will be flagged with the error code 'OC'.
ID	A delimiter is missing or invalid in the operand of the previous macro instruction.
IK	A keyword in the macro instruction being processed is not valid.
IR	An invalid parameter has been found in one of the operands of the previous macro instruction.
NF	The macro instruction being processed contains a mnemonic operation code not contained in the source library of the program pack.
OC	The mnemonic operation code of the previous macro instruction is invalid. (See codes 'CE' and 'IC' for a possible cause for this error code.)
ST	A keyword response has resulted in an invalid substring term. (See note)
SY	A keyword response has resulted in a substring syntax error. (See note)
TF	The variable symbol table is full. Recode your program using fewer macro instructions.

Note: These errors may be the result of any macro instruction or combination of macro instructions that precede the error code.

Figure 20. Macro Instruction Error Codes

GS'1 Halt

The maintenance personnel can locate the 2-byte save area at the label AERSAV within $SGENB. See Figure B-14.

Program	Description	Code	Error Description
$SGENB	System Generation, Phase One	BC	An invalid completion code was received from the SWA Read/ Write routine.
		PEM	The // END statement is missing from the procedure on the distribution disk cartridge.

Figure B-14. Halt Codes for System Generation–Phase One ($SGENB) GS'1 Halt

Macro Processing Error Messages

If an error occurs during macro processing, a 2-byte error code and message is written into $SOURCE:

ERROR *** cc code *** PROCESSING ABOVE MACRO

A description of Macro Processor error codes is contained in Figure B-15.

SYSTEM HALTS AND THE MODULES THAT CAN ISSUE THEM

Figure B-16 is provided to help determine which modules issue which halts. The figure contains a list of halts, the modules that initiate the halts listed, and a brief description of the reason for the halt being issued. The halts listed are only those halts which can be issued by the modules discussed in this manual.

Note: The halts shown are converted by the Model 6 Halt/ Syslog Transient ($$STOK) to display the Model 6 halt. See *Part 6. Transients and Scheduler Support* for a description of $$STOK.

B

Error Code	Error Description	Routine
AI	An AGO or AGOB record has an invalid sequence symbol.	$MPMN3
AT	A variable symbol table entry has an invalid attribute.	$MPATH
BE	A value compared in the operand of an AIF or AIFB record is more than 50 bytes long or has an invalid format (only symbolic parameters, set symbols, character strings, count functions, and type attributes are valid for comparison).	$MPSB1
	A model record is more than 71 bytes long.	$MPSUB
BI	A position in a binary self-defining term is other than 0 or 1.	$MPATH
BX	Arithmetic term exceeds bounds of − 8,388,608 to + 8,388,607.	$MPATH
	Value of symbolic parameter or decimal self-defining term exceeds maximum value of 65,535.	$MPCDX
CE	A macro instruction continuation statement has a non-blank entry in positions 1-13.	$MPXDV
CI	The count function is being used with other than symbolic parameters.	$MPATH, $MPSB1
CL	A character expression length is greater than 50 bytes.	$MPCEX
EI	An invalid operand or operator is used in an arithmetic expression. Valid operands are binary, character decimal, and hexadecimal self-defining terms; variable symbols; and count functions. Valid operators are addition (+), subtraction (-), multiplication (*), and division (/).	$MPATH
EM	A MEND record was found immediately following a TABLE record.	$MPMN2
ER	Consecutive operators have been detected within an arithmetic expression.	$MPATH
ET	An arithmetic expression has been ended with an operator.	$MPATH
HI	A hexadecimal self-defining term contains an invalid hexadecimal digit.	$MPATH
IA	An error has been detected in the format of an AIF or AIFB record.	$MPAIF
IC	The format of a macro instruction is for a continuation record to follow but continuation is not indicated.	$MPOPR
ID	An invalid delimiter occurred following a keyword parameter on a macro instruction.	$MPOPR
IG	A format error occurred in an operand of a GBLA, GBLB, GBLC, LCLA, LCLB, or LCLC record.	$MPGBL
IK	An invalid keyword was found on a macro instruction.	$MPOPR
IM	A sequence symbol is missing or misspelled.	$MPRED
IP	A prototype record has one of the following:	
	• Format error in an operand field	$MPOPR
	• Invalid entry in a name field	$MPROT
	• Operation field name incorrect	$MPSTM

Figure B-15 (Part 1 of 3). Error Codes for Macro Processor

Error Code	Error Description	Routine
IR	An invalid response to a keyword parameter was found on a macro instruction.	$MPOPR
IS	The length of the sequence symbol on an AGO or AGOB record was not less than 6.	$MPAGO
	An invalid variable symbol was found.	$MPGSY
IT	An error has been encountered in the placement of control records prior to the TEXT record within a macro definition.	$MPTBL
IU	A set symbol identified on a global or local record is also identified on a prototype or TABLE record within the same macro definition.	$MPVST
IV	An invalid value exists on the record being processed: • Null value when not permitted • Value exceeds 50 bytes when decoded • Value exceeds the limits of the record on which it appears	$MPCSB, $MPNCB
LP	Improper placement of left parenthesis or more than 3 levels of nested parenthesis within an arithmetic expression.	$MPATH
MM	The macro definition records are not in the expected sequence.	$MPMN3
MN	Invalid format on an MNOTE record.	$MPNTT
MS	One of the fixed format fields of a model record has exceeded its defined limits. An entry in field 1 must begin in position 1.	$MPMST
ND	A TABDF record does not follow a TABLE record.	$MPMN2
NE	No operator exists for the remaining operand in final step of arithmetic expression evaluation.	$MPATH
NF	The macro definition was not found in source library of program or system pack.	$MPMN1
NM	An error has been encountered in the placement of control records following the text record within a macro definition	$MPSTM
NO	Consecutive operands have been detected within an arithmetic expression.	$MPATH
NP	An invalid combination of operators was specified in an arithmetic expression.	$MPATH
OC	The mnemonic operation code of the record being processed is not a valid System/3 assembler operation code.	$MPXDV
OP	An invalid operator has been encountered within an arithmetic expression.	$MPATH
RP	Invalid placement of a right parenthesis within an arithmetic expression has occurred.	$MPATH
SA	An error exists in the format of a variable symbol required in the name field of a SETA record, or the operand is blank.	$MPSTA
SB	An error exists in the format of a variable symbol required in the name field of a SETB record, or the operand is not 0 or 1.	$MPSET

Figure B-15 (Part 2 of 3). Error Codes for Macro Processor

Error Code	Error Description	Routine
SC	An error exists in the format of a variable symbol required in the name field of a SETC record, or the operand is not enclosed with quotes and delimited by a blank.	$MPSTC
SD	A null value for a character self defining term exists within an arithmetic expression.	$MPATH
SM	Reference to an undefined variable symbol.	$MPVST
SS	A set symbol identified on a global record has been identified as another type of set symbol within a previous macro definition.	$MPGBL
	The attribute of a set symbol referenced in the name field of a SETA, SETB, or SETC record does not match its assigned attribute.	$MPSET, $MPSTA, $MPSTC
ST	When evaluating a character expression, the value of either term of substring is negative or the substring start term is 0.	$MPCEX
SY	Syntax error in use of substring or character expression exceeds the limits of the input record.	$MPCEX
TF	The variable symbol table is full. (The user should split his job into smaller requests.)	$MPGVA, $MPSTC, $MPSTM, $MPVST
TV	A table-definition record is invalid: • The value does not start in position 14 • The argument is not left-justified starting in position 1 • The argument exceeds the limits defined for the record • The mnemonic operation code (TABDF) is missing	$MPDEF

Figure B-15 (Part 3 of 3). Error Codes for Macro Processor

IO DISK IO ERROR.

EX OUT OF EXTENT.

DE HARDWARE ERROR.

Appendix B. Define the File Control Blocks

The DTF provides information to the data management routines about files you use. You must provide one DTF for each file you use in a program. Certain fields serve the same purpose in all pre-open DTFs. (Pre-open DTFs are reformatted to post-open when they are opened by using the allocate and open macro instructions.) Figure 21 describes the fields common to all pre-open DTFs.

The figures in this appendix describe both the pre-open and post-open DTFs for unit record and disk devices.

Figure	DTF Described
22	MFCU pre-open
23	MFCU post-open
24	1442 pre-open
25	1442 post-open
26	Line printer pre-open (Model 10)
27	Line printer pre-open (Model 12)
28	Line printer post-open (Model 10)
29	Line printer post-open (Model 12)
30	Printer-keyboard pre-open
31	Printer-keyboard post-open
32	Disk pre-open
33	5444 disk post-open
34	5445 disk post-open
35	Tape pre-open
36	Tape post-open
37	3741 pre-open
38	3741 post-open

The labels given to the fields in these figures are the labels generated by the offsets macro instructions, $DTOU, $DTOD, and $DTOT. Displacements refer to the right-most byte of the field. Addresses in the DTFs point to the leftmost byte of the referenced area.

Displacement	Length in Bytes	Field Description	Field Contents
0	1	Device Address	*Address* — *Device* X'A0' — R1 (5444 removable disk pack one) X'A8' — F1 (5444 fixed disk pack one) X'B0' — R2 (5444 removable disk pack two) X'B8' — F2 (5444 fixed disk pack two) X'C0' — D1 (5445 drive one) X'C8' — D2 (5445 drive two) X'F0' — MFCU1 (primary hopper) X'F8' — MFCU2 (secondary hopper) X'E0' — 5203 printer (left carriage) or 1403 printer X'E8' — 5203 printer (right carriage) X'10' — 5471 printer-keyboard (console) X'50' — 1442 card read/punch X'40' — 3741 Data Station/Programmable Work Station
1	1	External Indicators	Opening of a file may be conditioned by a SWITCH statement. Any external indicator bit set on is compared to the corresponding UPSI bit in the communication region, and if that bit is on, the file is opened.
3	2	File Attributes	Essential information about the file organization and the method by which it is to be processed is represented by these bits. (See Figures 22-38)
5	2	Record Length	The hexadecimal length of one logical record for the file.
7	2	Address of Next DTF	The address of the next DTF in the forward chain. This is used by the open and close routines to find the next DTF.

Figure 21. General Pre-Open DTF

Displacement	Length in Bytes	Contents
0	1	Device address: X'F0' MFCU1 X'F8' MFCU2
1	1	External indicator
2	1	Attribute byte 1: *Bits On* — *Use* 0 — Input 1 — Output 0 & 1 — Combined 4 — Print
3	1	Attribute byte 2: *Bit On* — *Use* 4 — Dual I/O areas (used only for input)
5	2	Record length in hexadecimal
7	2	Address of next pre-open DTF
13	6	Reserved (used by MFCU data management)
15	2	Address of second read IOB (supplied by macro processor)
17	2	Address of second read I/O area
22	5	Reserved (used by MFCU data management)
24	2	Pointer to input/output supervisor error recovery procedure
26	2	Address of first read IOB (supplied by macro processor)
28	2	Address of first read I/O area
30	2	Address of first punch IOB (supplied by macro processor)
32	2	Address of first punch I/O area
34	2	Address of first print I/O area
61	27	Reserved (used by MFCU data management)

Figure 22. MFCU Pre-Open DTF

Label	Displacement	Length in Bytes	Contents
$RDDEV	0	1	Device address: X'F0' MFCU1 X'F8' MFCU2
$RDUPS	1	1	External indicator
$RDAT1	2	1	Attribute byte 1: *Bits On* — *Use* 0 — Input 1 — Output 0 & 1 — Combined 4 — Print
$RDAT2	3	1	Attribute byte 2: *Bits On* — *Use* 0 — End of file on last read 1 — File allocated 4 — Dual I/O areas 5 — Device used as system input 6 — /& read on last input operation 7 — File opened
$RDCHA	5	2	Address of next DTF in backward chain
$RDCHB	7	2	Address of next DTF in forward chain
$RDARR	9	2	Address recall register save area (return address)
$RDXR1	11	2	Register 1 save area (contents of calling program register 1)
$RDLRA	13	2	Logical record address
$RDCMP	14	1	Completion code: *Code* — *Meaning* X'40' — Normal completion X'41' — Abnormal condition X'42' — End of file indicator
$RDOPR	15	1	Operation byte: *Bits On* — *Meaning* 0 — Read 1 — Print 2 — Punch 3 — Move (deferred operation) 4 - 7 — Must be zero
$RDSTS	16	1	Stacker select/print: *Bits* — *Setting* — *Meaning* 2 — 1 — Print four lines 5 — 1 — Stacker select command given 6 - 7 — 01 — Select stacker 1 10 — Select stacker 2 11 — Select stacker 3 00 — Select stacker 4
$RDQ	17	1	Q-byte (device address)
$RDR	18	1	R-byte
$RDSTA	19	1	IOS/ERP status information
$RDSNS	21	2	Sense area
$RDWKA	22	1	Work area
$RDSVA	24	2	IOS/ERP permanent save area address
$RDERP	26	2	Disk address of ERP
$RDRIO	28	2	Read IOB address
$RDUIO	30	2	Punch IOB address
$RDPUB	32	2	Punch I/O area address
$RDPTB	34	2	Print I/O area address
$RDPTL	35	1	Print record length in hexadecimal
$RDPUL	36	1	Punch record length in hexadecimal

Figure 23. MFCU Post-Open DTF

Displacement	Length in Bytes	Contents
0	1	Device Address, X'50'
1	1	External indicator
2	1	Attribute byte 1: *Bits On* — *Meaning* 0 — Input 1 — Output 0 & 1 — Combined
3	1	Attribute byte 2: *Bit On* — *Meaning* 4 — Dual I/O area
5	2	Record length
7	2	Address of next DTF in forward chain.
13	6	Reserved (used by 1442 data management)
15	2	Address of second read IOB (supplied by macro processor)
17	2	Address of second read I/O area
22	5	Unused
24	2	Pointer to error recovery work area (10 bytes)
26	2	Address of first read IOB (supplied by macro processor)
28	2	Address of first read I/O area
30	2	Address of first punch IOB (supplied by macro processor)
32	2	Address of first punch I/O area
61	29	Reserved (used by 1442 data management)

Figure 24. 1442 Pre-Open DTF

Label	Displace-ment	Length in Bytes	Contents
$FDDEV	0	1	Device address, X'50'
$FDUPS	1	1	External indicator
$FDAT1	2	1	Attribute byte 1: *Bits On* — *Meaning* *0* — Input 1 — Output 0 & 1 — Combined
$FDAT2	3	1	Attribute byte 2: *Bits On* — *Meaning* 1 — Device allocated 3 — Dual I/O area 5 — Device used as system input 6 — /& read on last input operation 7 — File is opened
$FDCHA	5	2	Address of next DTF in backward chain
$FDCHB	7	2	Address of next DTF in forward chain
$FDARR	9	2	ARR save area (return address of calling program)
$FDXR1	11	2	Register 1 save area (contents of calling program register 1)
$FDLRA	13	2	Logical record address
$FDCMP	14	1	Completion code: *Code* — *Meaning* X'40' — Normal completion X'41' — Abnormal completion X'42' — End-of-file indicator
$FDOPR	15	1	Operation byte: *Bits On* — *Meaning* 0 — Read 2 — Punch 2 & 3 — Punch, no feed
$FDSTS	16	1	Stacker select: Bit 5 on, stacker select Bit 6 off, 7 on; select stacker 1 Bit 6 on, 7 off; select stacker 2
$FDQ	17	1	Q-byte (device address)
$FDR	18	1	R-byte
$FDSTA	19	1	IOS/ERP status information
$FDSNS	21	2	Sense area
$FDWKA	22	1	Work area
$FDSVA	24	2	IOS/ERP permanent save area address
$FDERP	26	2	Disk address of ERP
$FDRIO	28	2	Read IOB address
$FDUIO	30	2	Punch IOB address
$FDPUB	32	2	Current processing data area address
$FDPRV	34	2	Previous operation bytes
$FDPUL	36	2	Punch record length

Figure 25. 1442 Post-Open DTF

Displacement	Length in Bytes	Contents
0	1	Device address: X'E0' left carriage, X'E8' right carriage
1	1	External indicator
2	1	Attribute byte 1; bit 1 on indicates output
3	1	Attribute byte 2; bit 6 on indicates halt on unprintable characters
5	2	Record length
7	2	Address of next pre-open DTF
27	20	Reserved (used by printer data management)
29	2	Address of IOB (supplied by the macro processor)
31	2	Address of output buffer
32	1	Reserved (used by printer data management)
33	1	Overflow line
34	1	Form length
52	18	Reserved (used by printer data management)

Figure 26. Line Printer Pre-Open DTF (Model 10)

Displacement	Length in Bytes	Contents
0	1	Device address: X'E0' left carriage, X'E8' right carriage
1	1	External indicator
2	1	Attribute byte 1; bit 1 on indicates output
3	1	Attribute byte 2; bit 6 on indicates halt on unprintable characters
5	2	Record length
7	2	Address of next pre-open DTF
11	4	Register save area
13	2	Logical record address
15	2	Reserved (used by printer data management)
21	6	Reserved (used by printer data management)
23	2	Address of print IOB
24	1	Total number of lines per page
25	1	Reserved (used by printer data management)
26	1	Overflow line
27	1	Maximum skip value
28	1	Reserved (used by printer data management)
30	2	Address of print buffer
32	2	Reserved (used by printer data management)

Figure 27. Line Printer Pre-Open DTF (Model 12)

Label	Displacement	Length in Bytes	Contents
$PDDEV	0	1	Device address: X'E0' left carriage X'E8' right carriage
$PDUPS	1	1	External indicator
$PDAT1	2	1	Attribute byte 1; bit 1 on indicates output
$PDAT2	3	1	Attribute byte 2: *Bits On* — *Meaning* 1 — Device allocated 4 — Dual I/O areas 6 — Halt on unprintable characters 7 — File opened
$PDCHA	5	2	Address of next opened DTF in backward chain
$PDCHB	7	2	Address of next DTF in forward chain
$PDARR	9	2	Address recall register save area (return address)
$PDXR1	11	2	Register 1 save area (contents of calling program register 1)
$PDLRA	13	2	Logical record address
$PDCMP	14	1	Completion code: *Code* — *Meaning* X'40' — Normal completion X'41' — Abnormal completion X'48' — Overflow
$PDOPR	15	1	Operation code; X'40' indicates print
$PDSKB	16	1	Skip-before value (line number)
$PDSPB	17	1	Space-before value (number of lines)
$PDSKA	18	1	Skip-after value (line number)
$PDSPA	19	1	Space-after value (number of lines)
$PDQ	20	1	Q-byte (device address)
$PDR	21	1	R-byte
$PDSTA	22	1	IOS/ERP status information: *Bits On* — *Meaning* 2 — Wait 4 — Overflow 5 — Halt for unprintable characters 6 — Unprintable character detected 7 — Abnormal condition
$PDSVA	24	2	IOS/ERP permanent save area
$PDXLC	25	1	Work area
$PDSNS	27	2	Sense area
$PDERP	29	2	Disk address of ERP
$PDIOB	31	2	Address of buffer-associated IOB
$PDPRA	33	2	Address of current I/O area
$PDLRL	34	1	Logical record length
$PDOFL	35	1	Overflow line
$PDDCT	36	1	Position counter

Figure 28. Line Printer Post-Open DTF (Model 10)

Label	Displace-ment	Length in Bytes	Contents
$PDDEV	0	1	Device address: X'E0' left carriage X'E8' right carriage
$PDUPS	1	1	External indicator
$PDAT1	2	1	Attribute byte 1; bit 1 on indicates output
$PDAT2	3	1	Attribute byte 2: *Bits On* — *Meaning* 1 — Device allocated 4 — Dual I/O areas 6 — Halt on unprintable characters 7 — File opened
$PDCHA	5	2	Address of next opened DTF in backward chain
$PDCHB	7	2	Address of next DTF in forward chain
$PDARR	9	2	Address recall register save area (return address)
$PDXR1	11	2	Register 1 save area (contents of calling program register 1)
$PDLRA	13	2	Logical record address
$PDCMP	14	1	Completion code: *Code* — *Meaning* X'40' — Normal completion X'41' — Abnormal completion X'48' — Overflow
$PDOPR	15	1	Operation code; X'40' indicates print
$PDSKB	16	1	Skip-before value (line number)
$PDSPB	17	1	Space-before value (number of lines)
$PDSKA	18	1	Skip-after value (line number)
$PDSPA	19	1	Space-after value (number of lines)
$PDQ	20	1	Q-byte (device address)
$PDR	21	1	R-byte
$PDIOB	23	2	Address of current IOB
$PDLP	24	1	Lines per page
$PDPCT	25	1	Position counter
$PDOFL	26	1	Overflow line counter
$PDMSK	27	1	Maximum skip value
$PDPGS	28	1	Page size save area
$PDPRA	30	2	Address of current I/O area
$PDRCL	31	1	Record length
$PDRES	32	1	Reserved

Figure 29. Line Printer Post-Open DTF (Model 12)

Displacement	Length in Bytes	Contents
0	1	Device address; X'10'
1	1	External indicator
2	1	Attribute byte 1: *Bits On* — *Meaning* 0 — Input 1 — Output 0 & 1 — Both input and output
3	1	Attribute byte 2: *Bit On* — *Meaning* 6 — Halt on unprintable characters
5	2	Record length
7	2	Address of next pre-open DTF
9	2	Return address register save area
11	2	Register 1 save area
13	2	Reserved
14	1	Completion code
15	1	Operation code
16	1	Logical record length
17	1	Length of operator reply
18	1	Space before/space after byte
20	2	Logical record address

Figure 30. Printer-Keyboard Pre-Open DTF

Label	Displace-ment	Length in Bytes	Contents
$CDDEV	0	1	Device address; X'10'
$CDUPS	1	1	External indicator
$CDAT1	2	1	Attribute byte 1: *Bits On* — *Meaning* 0 — Input 1 — Output 0 & 1 — Both input and output
$CDAT2	3	1	Attribute byte 2: *Bits On* — *Meaning* 1 — Device allocated 6 — Halt on unprintable characters 7 — File opened
$CDCHA	5	2	Address of next DTF in backward chain
$CDCHB	7	2	Address of next DTF in forward chain
$CDARR	9	2	Address recall register save area (return address)
$CDXR1	11	2	Register 1 save area (contents of calling program register 1)
$CDLRA	13	2	Logical record address
$CDCMP	14	1	Completion code: *Code* — *Meaning* X'00' — Operation has been initiated but no wait for completion has been executed, or CHEK-Y is specified and the operation is not completed X'40' — Normal completion X'41' — Abnormal completion X'42' — End-of-file indicator
$CDOPR	15	1	Operation byte: *Bits On* — *Meaning* 0 — Input 1 — Output 0 & 1 — Write/read (WTOR) 3 — Issue input request only when the request key is depressed 6 — Operator must key the exact number of characters
$CDCT1	16	1	Count of bytes in the first area
$CDCT2	17	1	Count of bytes in the second area
$CDSPC	18	1	Space command: *Bits* — *Contents* 0 - 3 — Number of lines to space before print 4 - 7 — Number of lines to space after print
$CDIO2	20	2	Address of input buffer

Figure 31. Printer-Keyboard Post-Open DTF

Displacement	Bytes	Notes	Contents
0	1	1	Device address (any valid disk device)
1	1	1	External indicators (UPSI)
3	2	1	File attributes
5	2	1	Record length
7	2	1	Address of next DTF
11	4		Reserved for post-open DTF
13	2	2	Logical record address (move mode)
15	2		Reserved for post-open DTF
17	2	3	Input/output area address (address of the area for IOBs and I/O buffers)
19	2		Reserved for post-open DTF
21	2	3	Block length (length of a physical block of records; used to determine the size of the data I/O buffers)
25	4		Reserved for post-open DTF
27	2		Address of MVF extent tabel (direct multivolume)
29	2		Number of MVF table extents (direct multivolume)
30	1		Reserved for post-open DTF
38	8	3	Filename (used to identify a disk file)
43	5		Reserved for post-open DTF
48	5		Reserved for post-open DTF (as required)
50	2	4	Address of requested key (indexed random) or address of record address area (direct)
50	(2)	4	Address of current key (indexed sequential add)
50	(2)	4	Address of high key (processing within limits)
54	4		Reserved for post-open DTF (as required)
56	2		Key length (indexed)
58	2		Reserved for post-open DTF (as required)
60	2		Key displacement in record (indexed)
62	2	4	Address of master track index (indexed random)
62	(2)	4	Address of last key (indexed sequential add)
62	(2)	4	Address of low key (processing within limits)
64	2		Number of bytes in master track index (indexed random)
160	96		Reserved for post-open DTF (as required)

Notes:

1. DTF fields common to all pre-open DTFs (see Figure 21).
2. Work buffer address not required for the Allocate Initiator or Open, but will be kept for post-open DTF.
3. DTF fields common to all pre-open disk DTFs.
4. Use of these fields varies with the type of access method used.

Figure 32. Disk Pre-Open DTF

Field Name	Disp.	Length	Contents
$DFDEV	0	1	Device address X'A0' = R1 X'A8' = F1 X'B0' = R2 X'B8' = F2
$DFUPS	1	1	External indicator
$DFATR	3	2	File attributes *Byte 1:* *Bit On* — *Meaning* 0 Indexed 1 Consecutive 2 Direct 3 Multivolume 4 Input 5 Output 6 Update 7 Add *Byte 2:* *Bit On* — *Meaning* 0 Binary 1 Ordered load 2 Random 3 Limits 4 Double buffers/shared I/O-move mode 5 Closed by end of volume 6 End of volume/limits 7 Opened
$DFCHA	5	2	DTF chain pointer A (backward)
$DFCHB	7	2	DTF chain pointer B (forward)
$DFARR	9	2	ARR save area (return address)
$DFXRS	11	2	XR1 save area (contents of object program XR1)
$DFWKB	13	2	Address of logical record (chared I/O-address of logical input record)
$DFCMP	14	1	Completion code *Code* — *Meaning* X'40' Normal completion X'41' Controlled cancel taken on permanent I/O error X'42' End of file (input) X'44' No record found (out of extent for direct files) X'48' Overflow (printer) X'50' Key field does not match key in update record X'60' Duplicate load or add attempted X'62' Out of sequence (load or add attempted) X'70' End of extent (output) Completion codes other than X'40' are returned before the data management function is actually completed.

Figure 33 (Part 1 of 3). 5444 Disk Post-Open DTF

		Field Name	Disp.	Length	Contents
		$DFOPC	15	1	Operation byte: *Bit on* — *Meaning* 0 — Get 1 — Put/add or put/load 2 — Put/update 3-7 — Must be zero
End of Basic DTF					
		$DFIOB	17	2	Address of current I/O IOB
		$DFPRB	19	2	Address of current process IOB (dual I/O only; shared I/O-address of logical output record)
		$DFBKL	21	2	Block length (length of data buffer)
		$DFRCL	23	2	Logical record length
		$DFPTR	25	2	Data block index (address of next record)
		$DFXTA	27	2	Data start extent
		$DFMVF	27	(2)	Address of direct MVF extent table
		$DFXTB	29	2	Data end extent (disk address)
		$DFNUM	29	(2)	Number of extents (direct MVF)
		$DFSWA	30	1	Scheduler work area format-1 label sequence number
		$DFWAA	31	1	Work area A
		$DFWAB	32	1	Work area B
		$DFWAC	33	1	Work area C
		$DFWAD	34	1	Work area D
		$DFRMA	36	2	Work area, length of first part of overlap record
		$DFRMB	38	2	Work area, length of second part of overlap record
		$DFIND	39	1	Indicator bits
End of DTF for:	$$CSOP	$DFNXR	43	4	Disk address of current record (CSDD)
	$$CSIP				
End of DTF for:	$$CSUP	$DFEOF	46	3	Disk address of logical end of file (CSD) or for direct files, maximum number (in binary) of records in the file
	$$DAIB				
	$$DAID				
	$$DAIM				
	$$DAIT				
	$$DAUB				
	$$DAUD	$DFNXK	46	(3)	Disk address of logical end of index (CSD)
	$$DAUM	$DFKPR	48	2	Pointer within index (pointer to next buffer entry)
End of DTF for:	$$DAUT	$DFKAD	50	2	Address of key in core (last Get or Put)
		$DFCUR	50	(2)	Address of current key (ISAD, ISUA)
		$DFHI	50	(2)	Address of high key (limit)
		$DFKXA	52	2	Start extent of index (disk address of first track)
		$DFKBF	54	2	Address of index IOB
	$$IOUT	$DFKL	56	2	Key length
	$$ISIP	$DFKXB	58	2	End extent of index
End of DTF for:	$$ISUP	$DFKD	60	2	Displacement of key in record

Figure 33 (Part 2 of 3). 5444 Disk Post-Open DTF

		Field Name	Disp.	Length	Contents
		$DFLST	62	2	Address of last key (ISAD, ISUA)
	$$ISIL	$DFMIX	62	(2)	Address of master track index
End of DTF for:	$$ISUL	$DFLOW	62	(2)	Address of low key (limit)
	$$IOAD				
	$$IRIP				
	$$IRUP				
	$$IRAD	$DFBYT	64	2	Number of bytes in master index
End of DTF for:	$$IRUA	$DFKXP	65	(2)+1	Logical start of index overflow (CSD)
End of DTF for:	$$ISAD	$DFSNP	67	2	Save next index pointer (ISAD, ISUA)
		$DFSLA	69	2	Save last address (CS) (ISUA)
End of DTF for:	$$ISUA	$DFSLP	71	2	Save last index pointer (ISUA)
		$DFSEQ	72	1	Logical sequence number of current SWA F1
		$DFNXT	73	1	Actual sequence number of current volumes
		$DFF1S	74	1	First byte of saved SWA F1
	$$ISIM	$DFF1	137	63	Last byte of saved SWA F1
	$$ISUM	$DFAR1	139	2	ARR save area (return address for Disk Data
	$$CSIM				Management when going to End of Volume)
	$$CSOM	$DFXR1	141	2	XR1 save area (contents of Disk Data
End of DTF for:	$$CSUM				Management XR1 when going to End of Volume)
	$$IOUM				
	$$ISAM				
End of DTF for:	$$ISBM	$DFKEY	143	2	Address of volume information table
	$$IRIM				
	$$IRUM	$DFTAB	145	2	Address of indexed MVF extent table
	$$IRAM	$DFENT	146	1	Number of track index entries in the volume
	$$IOAM				information table
End of DTF for:	$$IRBM	$DFVOL	147	1	Number of on-line indexed MVF volumes

Figure 33 (Part 3 of 3). 5444 Disk Post-Open DTF

Length of DTF for Various Modules

Field Name	Disp.	Length	Contents
$DFDEV	0	1	Device address X'C0' D1 X'C8' D2
$DFUPS	1	1	External indicator
$DFATR	3	2	File attributes *Byte 1:* *Bit On* — *Meaning* 0 Indexed 1 Consecutive 2 Direct 3 Multivolume 4 Input 5 Output 6 Update 7 Add *Byte 2:* *Bit On* — *Meaning* 0 Binary 1 Ordered load 2 Random 3 Limits 4 Double buffers/shared I/O-move mode 5 Closed by end of volume 6 End of volume/limits 7 Opened
$DFCHA	5	2	DTF backward chain pointer
$DFCHB	7	2	DTF forward chain pointer
$DFARR	9	2	ARR save area (return address)
$DFXRS	11	2	XR1 save area
$DFWKB	13	2	Address of logical record
$DFCMP	14	1	Completion code *Code* — *Meaning* X'40' Normal completion X'41' Controlled cancel taken on permanent I/O error X'42' End of file (input) X'44' No record found (out of extent for direct files) X'48' Overflow (printer) X'50' Key field does not match key in update record X'60' Duplicate load or add attempted X'62' Out of sequence (load or add attempted) X'70' End of extent (output) Completion codes other than X'40' are returned before the data management function is actually completed.

Figure 34 (Part 1 of 3). 5445 Disk Post-Open DTF

Length of DTF for Various Modules		Field Name	Disp.	Length	Contents
		$DFOPC	15	1	Operation byte: *Bit On* — *Meaning* 0 — Get 1 — Put/add or put/load 2 — Put/update 3-7 — Must be zero
End of Basic DTF (16 bytes)					
		$DFIOB	17	2	Address of current IOB
		$DFPRB	19	2	Address of current process IOB (dual I/O only)
		$DFBKL	21	2	Block length (length of data buffer)
		$DFRCL	23	2	Logical record length
		$DFPTR	25	2	Data block index (address of next record)
		$DFXTA	27	2	Data start extent
		$DFMVF	27	(2)	Address of direct MVF extent table
		$DFXTB	29	2	Data end extent (disk address)
		$DFNUM	29	(2)	Number of extents (direct MVF)
		$DFSWA	30	1	Scheduler work area format-1 label sequence number
		$DFWAA	31	1	Work area A
		$DFWAB	32	1	Work area B
		$DFWAC	33	1	Work area C
		$DFWAD	34	1	Work area D
		$DXRMA	37	3	Work area (length of first part of overlap record)
		$DXRMB	40	3	Work area (length of second part of overlap record)
		$DXIND	41	1	Indicator bits
End of DTF for:	$$CFOP	$DXNXR	46	5	Disk address of current record
	(48 bytes)	$DXSPC	47	1	Number of tracks in split cylinder file
	$$CFIP	$DXIOA	50	3	Save area for disk address from IOB
End of DTF for:	$$CFUP	$DXDAT	52	2	Save area for buffer pointer from IOB
	(57 bytes)	$DXEOF	56	4	Disk address of logical end of file or for direct files, maximum number (in binary) of records in the file
	$$DFIB				
	$$DFID				
	$$DFIM				
	$$DFIT				
	$$DFUB				
	$$DFUD	$DXNXK	56	(4)	Disk address of logical end of index
	$$DFUM	$DXKPR	58	2	Pointer within index (to next buffer entry)
End of DTF for:	$$DFUT				
	(61 bytes)	$DKKAD	60	2	Address of key in core (last get or put)
		$DXCUR	60	(2)	Address of current key
		$DXHI	60	(2)	Address of high key (limit)
		$DXKXA	62	2	Start extent of index (disk address of first track)
	$$IFUT	$DXKBF	64	2	Address of index IOB
	$$IHIP	$DXKL	66	2	Key length
End of DTF for:	$$IHUP	$DXKXB	69	3	End extent of index
	(72 bytes)	$DXKD	71	2	Displacement of key in record

Figure 34 (Part 2 of 3). 5445 Disk Post-Open DTF

Length of DTF for Various Modules	Field Name	Disp.	Length	Contents
	$DXLST	73	2	Address of last key
$$IHUL	$DXMIX	73	(2)	Address of master track index
End of DTF for: $$IHIL (74 bytes)	$DXLOW	73	(2)	Address of low key (limit)
$$IGIP				
End of DTF for: $$IGUP (76 bytes)	$DXBYT	75	2	Reserved
$$IFAD				
$$IGAD	$DXKXP	77	(4)	Logical start of index overflow
End of DTF for: $$IGUA (78 bytes)				
End of DTF for: $$IHAD (80 bytes)	$DXSNP	79	2	Save next index pointer
	$DXSLA	82	3	Save last address
End of DTF for: $$IHUA (85 bytes)	$DXSLP	84	2	Save last index pointer
	$DXSEQ	85	1	Logical sequence number of current SWA F1
	$DXNXT	86	1	Actual sequence number of current volumes
	$DXF1S	87	1	First byte of saved SWA F1
	$DXF1	150	63	Last byte of saved SWA F1
	$DXAR1	152	2	ARR save area (return address for disk data management when going to end of volume)
$$IHIM				
$$IHUM				
$$CFIM	$DXXR1	154	2	XR1 save area (contents of disk data management SR1 when going to end of volume)
$$CFOM				
End of DTF for: $$CFUM (155 bytes)				
$$IFUM				
$$IHAM				
End of DTF for: $$IHBM (157 bytes)	$DXKEY	156	2	Address of volume information table
$$IGIM	$DXTAB	158	2	Address of index MVF extent table
$$IGUM				
$$IGAM	$DXENT	159	1	Number of track index entries in volume information table
$$IFAM				
End of DTF for: $$IGBM (161 bytes)	$DXVOL	160	1	Number of online indexed MVF volumes

Figure 34 (Part 3 of 3). 5445 Disk Post-Open DTF

Displacement	Length	Contents
0	1	Device address
1	1	External indicator (see note)
2	1	Attributes
3	1	Attributes
5	2	Record length
7	2	Address of next DTF
11	4	Not used
13	2	Logical record area
15	2	Not used
17	2	Address of I/O area
19	2	Length of I/O area (see note)
21	2	Block length
27	6	Not used
28	1	Attributes
30	2	Not used
38	8	File name
39	1	Buffer offset (ASCII only)
Note: These positions are not used in the Pre-Open Basic DTF		

Figure 35. Tape Pre-Open DTF

Label	Displacement	Length	Contents
$TDDEV	0	1	Device address (X'60')
$TDERP	1	1	External indicator (See Note)
$TDATR	2	2	Attribute byte 0 *Bit* — *Meaning* 1 — Consecutive (Always on) 3 — Multivolume file 4 — Input 5 — Output 6 — Basic access method Attribute byte 1 *Bit* — *Meaning* 0 — Unload } Rewind 1 — Leave } 2 — Standard labeled file 3 — Locate mode 4 — Multiple buffering 5 — Deferred open 6 — Force EOV call to close 7 — Opened
$TDCHA	4	2	DTF backward chain pointer
$TDCHB	6	2	DTF forward chain pointer
$TDARR	8	2	ARR save area (return address)
$TDXRS	A	2	XR1 save area (object program's XR1)
$TDWKR	C	2	Address of the logical record
$TDCMP	E	1	Completion code: *Code* — *Meaning* X'40' — Normal completion X'41' — Controlled cancel taken on permanent I/O error X'42' — End-of-file (input) X'70' — End-of-volume (output) X'90' — Incorrect length on input operation
$TDOPC	F	1	Operation byte: *Code* — *Meaning* X'40' — Put X'80' — Get
$TDIOB	10	2	Address of current I/O IOB
$TDPRB	12	2	Address of current process IOB (See Note)
$TDBKL	14	2	Block length (Length of data buffer)
$TDRLC	16	2	Maximum record length (See Note)
$TDPTR	18	2	Block index (See Note)
$TDCRL	1A	2	Current record length
$TDAT2	1C	1	Attribute byte 2 *Bit* — *Meaning* 0 — Fixed 1 — Variable 2 — Unblocked 3 — Blocked 4 — Not used 5 — ASCII Format D 6 — ASCII file being processed 7 — Closed (Multivolume files only)

Figure 36 (Part 1 of 2). Tape Post-Open DTF

Label	Displacement	Length	Contents
$TDHTC	1D	1	Not used
$TDNUM	1E	1	SWA Format 1 number
$TDWAA	1F	2	Work area A (Block length counter for variable length records) (See Note)
$TDWAB	21	2	Work area B (Buffer offset for ASCII) (See Note)
$TDWAC	23	2	Work area C (Block count); used by
$TDWAD	25	2	Tape Close to write or compare block count. Basic users must update this count. (See Note) Work area D (Block count save area) (See Note)
$TDIND	27	1	Indicator bits *Bit* — *Meaning* 0 — CPR IOB has not been waited on 1 — Truncated block 2 — Empty variable length block 3 — EOV call to close 4 — $$CDVE encountered error reading trailer label
Note: These positions are not used in the Post-Open Basic DTF			

Figure 36 (Part 2 of 2). Tape Post-Open DTF

Displacement	Bytes	Notes	Contents
0	1	1	Device address (X'40')
1	1	1	External indicators (UPSI)
3	2	1	File attributes
5	2	1	Record length
7	2	1	Address of next DTF
11	4		Reserved for post-open DTF
13	2	2	Logical record address (move mode)
15	2		Reserved for post-open DTF
17	2		Input/output area address (address of the area for IOBs and I/O buffers)
19	2		Reserved for post-open DTF
21	2		Block length (length of a physical block of records; used to determine the size of the data I/O buffers)
25	4		Reserved for post-open DTF
27	2		Address of MVF extent tabel (direct multivolume)
29	2		Number of MVF table extents (direct multivolume)
30	1		Reserved for post-open DTF
38	8		Filename (used to identify a file)

Notes:
1. DTF fields common to all pre-open DTFs (see Figure 21).
2. Work buffer address not required for the Allocate Initiator or Open, but will be kept for post-open DTF.

Figure 37. 3741 Pre-Open DTF

Field Name	Disp.	Length	Contents
$DFDEV	0	1	Device address (X'40')
$DFUPS	1	1	External indicator
$DFATR	3	2	File attributes

Byte 1:

Bit On	*Meaning*
0	Unused
1	Consecutive
2	Unused
3	Unused
4	Input
5	Output
6	Unused
7	Unused

Byte 2:

Bit On	*Meaning*
0	Unused
1	
2	
3	
4	Double buffers
5	Unused
6	
7	Opened

$DFCHA	5	2	DTF chain pointer A (backward)
$DFCHB	7	2	DTF chain pointer B (forward)
$DFARR	9	2	ARR save area (return address)
$DFXRS	11	2	XR1 save area (contents of object program XR1)
$DFWKB	13	2	Address of logical record (shared I/O-address of logical input record)
$DFCMP	14	1	Completion code

Code	*Meaning*
X'40'	Normal completion
X'41'	Controlled cancel taken on permanent I/O error
X'42'	End of file (input)

Completion codes other than X'40' are returned before the data management function is actually completed.

Figure 38 (Part 1 of 2). 3741 Post-Open DTF

	Field Name	Disp.	Length	Contents
	$DFOPC	15	1	Operation byte: *Bit on* — *Meaning* 0 — Get 1 — Put/add or put/load 2 — Put 3-7 — Must be zero
End of Basic DTF				
	$DFIOB	17	2	Address of current I/O IOB
	$DFPRB	19	2	Address of current process IOB (dual I/O only; shared I/O-address of logical output record)
	$DFBKL	21	2	Block length (length of data buffer)
	$DFRCL	23	2	Logical record length
	$DFPTR	25	2	Data block index (address of next record)
	$DFXTA	27	2	Unused
	$DFMVF	27	(2)	
	$DFXTB	29	2	
	$DFNUM	29	(2)	
	$DFSWA	30	1	Scheduler work area format-1 label sequence number
	$DFWAA	31	1	Unused
	$DFWAB	32	1	
	$DFWAC	33	1	
	$DFWAD	34	1	
	$DFRMA	36	2	
	$DFRMB	38	2	

Figure 38 (Part 2 of 2). 3741 Post-Open DTF

Appendix C. Disk Input/Output Block

You build the disk IOB by issuing the $IOBD macro instruction. If you use $RDD, $WRTD, or $WAIT in your program, you must use the $IOED macro instruction to assign the offset in the IOB. The format of the IOB and the labels assigned to the fields are shown in Figure 39. IOBs for the IBM 5444 Disk Storage Drive are 22 bytes long; for the IBM 5445 Disk Storage Drive, the IOBs are 26 bytes long.

Label	Displace-ment	Length in Bytes	Contents
$DICHN	1	2	Address of the next IOB in the chain. IOBs are chained only when the file requires more than one IOB. This area is always present, even when chaining is not used. When the operation specified by this IOB is complete, this area contains the disk address last used (cylinder/sector for the 5444; head/record for the 5445).
$DICMP	2	1	A 1-byte completion code indicating the status of the operation just performed. You should check this byte before assuming that the data transfer has occurred. Before the wait routine is called, each bit in this byte has the following meaning: *Bit On* — *Meaning* 0 — Seek has been started on the operation requested by the IOB. 1 — The operation requested is complete. 2 — Data transfer is pending on this operation. 3 — Data transfer has been started on this operation. 4 — A wait will occur for this IOB. 5 — If bit 7 is also on, there is an error on an associated IOB; if bit 7 is off, a scan equal has been found. 6 — The scan is not satisfied. 7 — A permanent error has occurred on this IOB or an associated IOB. After the wait routine has finished, the code in this byte has the following meaning: *Code* — *Meaning* X'40' — Successful completion. X'41' — Permanent I/O error. X'42' — Scan not satisfied. X'44' — Scan equal found. X'45' — Permanent error on an associated IOB.
$DIQB	3	1	The Q-byte of the start I/O (SIO) command. You set this byte through the $IOBD macro instruction.
$DIRB	4	1	The R-byte of the start I/O command. It further defines the operation requested. Figure 40 shows the possible R-byte settings for the SIO command.
$DICB (5444 only)	5	1	The hexadecimal value of the cylinder address where the operation is to begin. You set this byte through your $IOBD macro instruction.
$DIFL2 (5445 only)	5	(1)	Flag byte for use with the 5445. The meanings of the bits are: *Bit On* — *Meaning* 0 — End of cylinder has been reached. (Used by IOS only.) 1 — End of cylinder has been reached. You may test this bit when control is returned to your program. 2-5 — Not used. 6 — Seek not used on this operation. 7 — Reserved; must be 0.
$DISB (5444 only)	6	1	The hexadecimal value of the beginning sector address of the operation. You set this byte through the $IOBD macro instruction.
$DINB (5444 only)	7	1	The number of sectors minus one, in hexadecimal, involved in the data transfer. You set this byte through the $IOBD macro instruction.
$DIDAD (5445 only)	7	(2)	Address of the leftmost byte of the 5445 disk address.
$DIDAT	9	2	Address of the leftmost byte of your data area. You provide this address through the $IOBD macro instruction.

Figure 39 (Part 1 of 2). Disk IOB Format

Label	Displacement	Length in Bytes	Contents
$DISNS	11	2	The area used by the input/output supervisor to contain device status sense information. The contents of this area is described in Figure 41.
$DIERR	12	1	The area used by disk IOS to count the number of retries required to complete the I/O request.
$DIFLG	13	1	Indicates special handling required for I/O operations through the various bit settings. You set bits 0 and 4 through the $IOBD macro instruction. If no special handling is required, the byte must be set to X'00'. The bit settings are: *Bit On* — *Meaning* 0 — No recovery is to be attempted if a data check, missing address mark, no record found, or track condition check error condition occurs. 1 — No verification is to be done on write operations. 2 — No error logging is to be done if any disk I/O error should occur. Control is to be returned to the calling routine. 3 — Disk IOS should not use the C and S bytes in the IOB, but should pick up the F, C, and S bytes at the end of the IOB for use in this operation. This bit should be used only by the system control program. 4 — The calling routine is not using disk data management; therefore, this IOB is not associated with a DTF. 5 — There is no load I/O of the disk file data register (DFDR). 6 — Error logging is in progress. 7 — A data transfer operation involving an alternate track is in progress.
$DIARR	15	2	The save area for the address recall register.
$DIXR2	17	2	The save area for register 2.
$DIDCH	19	2	The disk data management chain pointer. It contains the address of the second of the two IOBs used for double buffering.
$DIDTF	21	2	The address of the DTF associated with this IOB.
$DICC (5445 only)	22	1	The hexadecimal value of the cylinder address where the operation is to begin. You set this byte through the $IOBD macro instruction.
$DIHH (5445 only)	23	1	The hexadecimal value of the head address where the operation is to begin. You set this byte through the $IOBD macro instruction.
$DIR (5445 only)	24	1	The hexadecimal value of the record address where the operation is to begin. You set this byte through your $IOBD macro instruction.
$DIN (5445 only)	25	1	The number of records minus one, in hexadecimal, involved in the data transfer. You set this byte through the $IOBD macro instruction.

Figure 39 (Part 2 of 2). Disk IOB Format

I/O Operation	SIO R-Byte Settings (Hex)	Interpretation
5444 Settings		
Control	00	Seek
Read	00 01 02 03	Data Identifier Diagnostic Verify
Write	00 01	Data Identifier
Scan	00 01 02	Equal Low or equal High or equal
5445 Settings		
Control	00 01	Seek Recalibrate
Read	00 03	Key-data Verify key-data
Write	00	Key-data
Scan	00 01 02	Key-data equal Key-data low or equal Key-data high or equal

Figure 40. R-Byte Settings

Device	Byte	Bit On	Indication
5444	0	0	I/O no-op (single only)
		1	Intervention required
		2	Missing address mark
		3	Equipment check
		4	Data check
		5	No record found
		6	Track condition check
		7	Seek check
	1	0	Scan equal
		1	Access arm at cylinder 0
		2	End of cylinder
		3	Seek busy
		4	Hundred cylinders
		5	Device overrun
		6	Status address A*
		7	Status address B*
5445		0	Format error
		1	Intervention required
		2	Missing address mark
		3	Equipment check
		4	Data check
		5	No record found
		6	No-op
		7	Overrun
		0	Disk drive error
		1	Unsafe
		2	Seek 1 complete
		3	Seek 2 complete
		4	Data operation complete
		5	End of cylinder has been reached
		6	Scan equal
		7	Disk drive ID**

* Indicates which drive on the 5444 had last data transfer:
Bits 6-7 00—Drive 1
01—Drive 2

** Bit 7 0—Drive 1
1—Drive 2

Figure 41. Device Status Sense Information

Appendix D. Macro Instruction Summary Chart

Figure 42 is a summary chart containing all valid macro processor instructions. The macro instructions are listed in alphabetic order. Four items are given for each macro instruction:

- Name
- Format of the instruction with all valid operands
- Function of the macro instruction
- Maximum number of statements generated

For more detailed information on any of the macro instructions, see *Chapter 2. Macro Instruction Statements.*

Figure 42 (Part 1 of 4). Macro Instruction Summary Chart

Name	Macro Instruction			Function	Maximum Number of Statements Generated
$ALOC	[Name]	$ALOC	[DTF-address]	Assigns the file indicated by the DTF to your program.	5
$CCP	[Name]	$CCP	QBYTE-hex,RBYTE-hex	Generates the CCP assembler instruction.	4
$CHK	[Name]	$CHK	[CKL-address]	Tests for I/O operation completion in the check list.	4
$CKL	[Name]	$CKL	DTF-address[,SKIP-Y/N] [,REQK-Y/N] [,RTN-Y/N] [,LAST-Y/N]	Generates an entry for the check list to be used by the check routine.	6
$CLOS	[Name]	$CLOS	[DTF-address]	Prepares the device for job termination.	5
$COMN		$COMN		Provides equates used by various other macro instructions.	14
$CTLT	[Name]	$CTLT	[DTF-address] [,OPC-code]	Issues control commands to the tape device.	5
$DTFD	[Name]	$DTFD	AC-code,RECL-number,NAME-filename,IO-address, BLKL-number[,DISK-5444/5445] [,UP-mask] [,BUFNO-1/2] [,MVF-N/Y] [,LIM-N/Y] [,ORD-N/Y] [,BIN-N/Y] [,CHN-address] [,RCAD-address] [,ENT-number] [,MVFN-number] [,KEYL-number] [,KEYD-number] [,KEYA-address] [,MVFT-address] [,MSTX-address]	Builds a DTF for a disk file.	43
$DTFK	[Name]	$DTFK	NAME-filename,RECL-number, IO-address [,AC-I/O] [,RCAD-address] [,BUFNO-1/2] [,CHN-address] [,UP-mask]	Builds a DTF for 3741 file.	15
$DTFT	[Name]	$DTFT	NAME-filename,IO-address,AC-IN/OUT,BLKL-number, RECL-number[,UP-mask] [,CHN-address] [,BASIC-Y/N] [,MODE-LOCATE/MOVE] [,MBUFF-Y/N] [,RCAD-address] [,RECFM-code] [,LIOA-number] [,SPAN-Y/N] [,CODE-A/E] [,OSET-B/number] [,END-code]	Builds a DTF for a tape file.	23

Name	Macro Instruction			Function	Maximum Number of Statements Generated
$DTFU	[Name]	$DTFU	DEV-code,FTYP-code,PIOB-address [,UP-mask] [,DIO-Y/N] [,HUC-Y/N] [,RECL-number] [,CHN-address] [,RCAD-address] [,RDA1-address] [,RDA2-address] [,PUA1-address] [,PTA1-address] [,OVFL-number] [,PG-number] [,MSKP-number] [,SPACE-number] [,REQK-Y/N] [,FILL-Y/N] [,CHK-Y/N] [,RTN-Y/N] [REPLY-number] *Note:* PIOB required on Model 12 only.	Builds a DTF for a unit record file.	25
$DTOD		$DTOD		Establishes labels for fields on post-open disk DTFs.	126
$DTOT		$DTOT		Establishes labels for fields on post-open tape DTFs.	72
$DTOU		$DTOU	[DEV-code]	Establishes lables for fields on post-open unit record DTFs.	36
$EOJ	[Name]	$EOJ		Terminates the job and returns control to the supervisor.	4
$FIND	[Name]	$FIND	NAME-module[,FIND-label] [,PACK-P/S]	Provides the disk address of a module in the O. library.	13
$FTCH	[Name]	$FTCH	Name-module name[,PACK-P/S]	Finds and loads an O. module and passes control to it.	10
$GETD	[Name]	$GETD	AC-code[,DTF-address] [,ERR-address] [,EOF-address] [,NRF-address] [,LSTV-address] [,NOKY-address]	Gets a record from a disk file via disk data management.	13
$GETK	[Name]	$GETK	[DTF-address] [,ERR-address] ,EOF-address	Gets a record from 3741 file via 3741 data management.	8
$GETT	[Name]	$GETT	[DTF-address] [,CODE-A/E] [,ERR-address] [,EOF-address] [RECFM-F/V]	Gets a record from a tape file via tape data management.	9
$GPU	[Name]	$GPU	ERR-address,MODUL-label[,OPC-code] [,DEV-code] [,DTF-address] [,EOF-address] [,OVRTN-address] [,DEFER-Y/N]	Gets a record from or puts a record to a unit record file via data management.	10
$IOBD	[Name]	$IOBD	[DISK-5444/5445] [,CYL-number] [,SCTR-number] [,HEAD-number] [,NUM-number] [,BUFF-address] [,Q-number] [,ERREC-IOS/USER] [,LOG-Y/N] [,VER-Y/N] [,CHN-address]	Builds an IOB for a disk file.	20

Figure 42 (Part 2 of 4). Macro Instruction Summary Chart

Figure 42 (Part 3 of 4). Macro Instruction Summary Chart

Name	Macro Instruction			Function	Maximum Number of Statements Generated
$IOED		$IOED		Establishes labels for fields in the disk IOB.	23
$LCP	[Name]	$LCP	QBYTE-hex[,ADDR-address] [,REG-1/2] [,DISP-hex]	Generates the LCP assembler instruction.	4
$LOAD	[Name]	$LOAD	NAME-module name [,FIND-address] [,LOAD-2/address] [,USE-R/NR] [,PLIST-address] [,PACK-P/S]	Finds and loads or loads a previously found O. module into main storage.	15
	[Name]	$LOAD	FIND-label [,LOAD-2/address] [,PACK-P/S]		
$OPEN	[Name]	$OPEN	[DTF-address]	Prepares a previously allocated file for data transfer.	5
$PKBU	[Name]	$PKBU	[OPC-code] [,DTF-address] [,ERR-address] [,RECL-number] [,REPLY-number] [,SPACE-number] [,RCAD-address] [,REQK-Y/N] [,FILL-Y/N] [,EOF-address] [,CHK-Y/N] [,RTN-Y/N]	Performs input/output operations on the printer keyboard.	35
$PRNT	[Name]	$PRNT	NAME-address,LEN-number	Prints a message on the halt/syslog device.	9
$PUTD	[name]	$PUTD	AC-code [,DTF-address] [,ERR-address] [,EOX-address] [,DUP-address] [,SERR-address] [,KERR-address] [,UPD-Y/N] [,LSTV-address] [,HKER-address] [,NOKY-address]	Writes a record on a disk file via disk data management.	19
$PUTK	[Name]	$PUTK	[DTF-address] [,ERR-address]	Writes a record on a 3741 file via 3741 data management	6
$PUTT	[Name]	$PUTT	[DTF-address] [,CODE-A/E] [,ERR-address] [RECFM-F/V]	Writes a record on a tape file via tape data management.	7
$RDD	[Name]	$RDD	IOB-address,CS-address,NSECT-number [,DISK-5444/5445]	Reads a record from a disk file via the input/output supervisor.	7
$RDT	[Name]	$RDT	[DTF-address] [,DIRECT-FORW/BACK]	Reads a record from a tape file via basic tape data management.	5
$SCP	[Name]	$SCP	QBYTE-hex[,ADDR-address] [,REG-1/2] [,DISP-hex]	Generates the SCP assembler instruction.	4
$SNAP	[Name]	$SNAP	ID-hex,START-address,END-address	Prints the specified area of main storage on the system logging device.	11
$SVC	[Name]	$SVC	RIB-SYSIN/HALT/VTOC/ROLL	Branches to one of the system routines.	5
$TRAN	[Name]	$TRAN	[TRL-address]	Translates a record using the system translate routine.	8

Name	Macro Instruction			Function	Maximum Number of Statements Generated
$TRL	[Name]	$TRL	TO-address,FROM-address,LEN-number, TRT-address	Builds a parameter list to pass information to the system translate routine.	13
$TRTB	[Name]	$TRTB	[CODE-E/A] [,HEX-hex]	Generates an EBCDIC to ASCII or an ASCII to EBCDIC translate table.	20
$WRTD	[Name]	$WRTD	IOB-address,CS-address,NSECT-number [,DISK-5444/5445]	Writes a record on a disk file via the input/output supervisor.	7
$WAIT	[Name]	$WAIT	[IOB-label] [,ERR-address]	Waits for completion of a disk input/output operation.	6
$WRTT	[Name]	$WRTT	[DTF-address]	Writes a record to a tape file via the basic tape data management.	5
$WTT	[Name]	$WTT	[DTF-address] [,ERR-address] [,EOF-address] [,EOT-address]	Waits for completion of a basic tape data management I/O operation.	10
$XCTL	[Name]	$XCTL	NAME-module name[,LOAD-2/address] [,PACK-P/S]	Finds and loads a module at a specified address and passes control to it.	10

Figure 42 (Part 4 of 4). Macro Instruction Summary Chart

Index

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

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

IBM System/3 Models 8, 10, and 12 SCP Macros Reference (File No. S3-31) Printed in U.S.A. GC21-7562-5