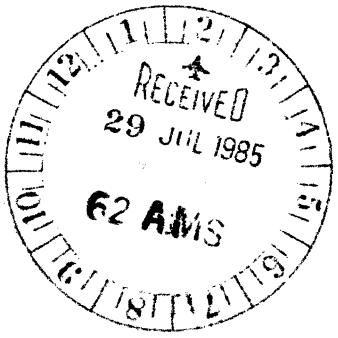
MPX-32

Release 1.5B

Reference Manual

Volume II



September 1982

Publication Order Number: 323-003662-000

GOULD
Electronics & Electrical Products

This manual is supplied without representation or warranty of any kind. Gould Inc., S.E.L. Computer Systems Division therefore assumes no responsibility and shall have no liability of any kind arising from the supply or use of this publication or any material contained herein.

LIMITED RIGHTS LEGEND

for

PROPRIETARY INFORMATION

The information contained herein is proprietary to Gould S.E.L. and/or its vendors, and its use, disclosure or duplication is subject to the restrictions stated in the Gould S.E.L. license agreement Form No. 620-06(1/82) or the appropriate third-party sublicense agreement. The information is provided to government customers with limited rights as described in DAR 7-104.9A.

HISTORY

The MPX-32 Release 1.0 Reference Manual, Publication Order Number 323-001012-000, was printed June, 1979.

Publication Order Number 323-001012-100 (Revision 1, Release 1.3) was printed February, 1980.

Publication Order Number 323-001012-200 (Revision 2, Release 1.4) was printed July, 1980.

Publication Order Number 323-003662-000 (Revision 3, Release 1.5B) was printed September, 1982. The updated manual contains the following pages:

Title page
Copyright page
iii/iv through xvii/xviii
1-1 through 1-10
2-1 through 2-56
3-1 through 3-18
4-1 through 4-63/4-64
5-1 through 5-58
6-1 through 6-33/6-34
7-1 through 7-7/7-8
8-1 through 8-9/8-10
9-1 through 9-14
10-1 through 10-24
11-1 through 11-29/11-30
A-1 through A-7/A-8
B-1 through B-21/B-22
C-1 through C-39/C-40
D-1 and D-2
E-1 and E-2
F-1 through F-3/F-4
G-1 through G-3/G-4
GL-1 through GL-11/GL-12
IN-1 through IN-17/IN-18

CONTENTS

3. TASK STRUCTURE AND OPERATION

4. THE DEBUGGER (DEBUG)

5. INTERACTIVE PROCESSING

6. THE FILE MANAGER (FILEMGR)

10. MEDIA CONVERSION UTILITY (MEDIA)

11. SOURCE UPDATE UTILITY (UPDATE)

ILLUSTRATIONS

TABLES

Documentation Conventions

Notation conventions used in command syntax and message examples throughout this manual are described below.

lowercase letters

In command syntax, lowercase letters identify a generic element that must be replaced with a value. For example,

```
!ACTIVATE taskname
```

means replace taskname with the name of a task, e.g.,

```
!ACTIVATE DOCCONV
```

In messages, lowercase letters identify a variable element. For example,

```
**BREAK** ON:taskname
```

means a break occurred on the specified task.

UPPERCASE LETTERS

In command syntax, uppercase letters specify a keyword must be entered as shown for input, and will be printed as shown in output. For example,

```
SAVE filename
```

means enter SAVE followed by a filename, e.g.,

```
SAVE DOCCONV
```

In messages, uppercase letters specify status or information. For example,

```
taskname,taskno ABORTED

*YOUR TASK IS IN HOLD. ENTER CONTINUE TO RESUME IT
```

Braces { }

Elements placed one under the other inside braces specify a required choice. You must enter one of the arguments from the specified group. For example,

```
{counter  }
{startbyte}
```

means enter the value for either counter or startbyte.

Brackets [ ]

An element inside brackets is optional. For example,

[CURR]

means the term CURR is optional.

Items placed one under the other within brackets specify you may optionally enter one of the group of options or none at all. For example,

[base name]
[progname]

means enter the base name or the program name or neither.

Items in brackets within encompassing brackets specify one item is required only when the other item is used. For example,

TRACE [lower address [upper address]]

means both the lower address and the upper address are optional, and the lower address may be used alone. However, if the upper address is used, the lower address must also be used.

Commas between multiple brackets within an encompassing set of brackets are semi-optional; that is, they are not required unless subsequent elements are selected. For example,

M.DFCB fcb,lfc[,[a],[b],[c],[d],[e]]

could be coded as

M.DFCB FCB12,IN

or

M.DFCB FCB12,IN,,ERRAD

or

M.DFCB FCB13,OUT,,ERAD,,PCK

Horizontal Ellipsis . . .

The horizontal ellipsis indicates the previous element may be repeated. For example,

name ,...,name

means you may enter one or more name values separated by commas.

Vertical Ellipsis

The vertical ellipsis specifies commands, parameters, or instructions have been omitted. For example,

```
COLLECT 1
  .
  .
  .
LIST
```

means one or more commands have been omitted between the COLLECT and LIST commands.

Numbers and Special Characters

In a syntax statement, any number, symbol, or special character must be entered as shown. For example,

(value)

means enter the proper value enclosed in parentheses; e.g., (234).

Underscore

In syntax statements, underscoring specifies the letters, numbers or characters that may be typed by the user as an abbreviation. For example,

ACTIVATE taskname

means spell out the command verb ACTIVATE or abbreviate it to ACTI.

RESET

means type either RESET or RST.

In examples, all terminal input is underscored; terminal output is not. For example,

TSM >EDIT

means TSM was written to the terminal; EDIT is typed by the user.

Subscript Delta

A subscript delta specifies a required space. For example,

EDT >STO TSSPGM

means a space is required between O and T.

1. THE ASSEMBLER (ASSEMBLE)

The Macro Assembler (or Assembler) translates source code mnemonics into binary-equivalent machine instructions for the 32/7x CPU and interprets Assembler directives. Assembler directives provide the ability to use symbolic addresses and storage areas, equate symbols, define references to external object modules, control listed output characteristics, etc.

Within source provided to the Assembler, the user can access system services. This is done either by setting up appropriate registers and using SVC or CALM instructions, and/or using macro calls which are expanded into assembly-level code by the Assembler.

The Assembler uses the MPX-32 System Macro Library (M.MPXMAC) for MPX-32 SVC-related services by default. The RTM System Macro Library (M.MACLIB) can also be assigned for assembly to access RTM CALM equivalent services.

On a CONCEPT/32 computer, a new SVC type 15 replaces CALM instructions. During reassembly of a program, the Assembler automatically converts CALM instructions to their equivalent SVC 15,X'nn' number if OPTION 20 is set.

Also, an address exception trap will be generated when a doubleword operation code is used with an incorrectly bounded operand; therefore coding changes will be required when a trap occurs.

1.1 General Description

In developing Assembly language source code, four separate, non-MPX-32 documents are key:

- The Macro Assembler Reference Manual, publication number 323-001220, which documents Assembler directives.
- The SYSTEMS 32/70 Computer Reference Manual, publication number 301-000140, which documents the SYSTEMS 32/70 CPU instruction set and mnemonics.
- The SYSTEMS 32/27 Single Slot Central Processing Unit, publication number 301-000400, which documents the SYSTEMS 32/27 CPU instruction set and mnemonics.
- The CONCEPT 32/87 Reference Manual, publication number 301-000810, which documents the SYSTEMS 32/87 CPU instruction set and mnemonics.

After a successful assembly, the user has an object module which can be output to a subroutine library file, output to a permanent file or device medium, or cataloged immediately into a task suitable for execution on MPX-32. Object modules can be linked together into a single task by assembling and cataloging them in the same job (an SGO file is the default output for assembly and default input for the Cataloger), by accessing the Subroutine Library during a separate Cataloger run, or by using $SELECT job control statements prior to cataloging (batch only).

The Assembler dynamically establishes both a macro storage table and a symbol table in memory before it starts assembly. As a default option, all available memory is allocated for the symbol table and zero for the macro storage table. The ratio of available space allocated for macro storage can be changed. From 0 to 80 percent of available memory can be allocated to macro storage. The percentage is specified with an OPTION statement as described in Section 1.3.

1.2 Files and File Assignments

This section describes the input and output files used by the Assembler.

1.2.1 Source Code (PRE and SI)

Source code is assigned to logical file codes (lfc's) PRE and SI. Source is input first from PRE and then from SI. User program source should be assigned to SI while source consisting of non-executable assembler directives (such as SET directives) can be assigned to PRE. The user can input source code from any device or file. The default assignment for SI is to SYC and for PRE is to NU (the null device).

1.2.2 Macro Libraries (MAC, MA2)

The System Macro Library provides a collection of macro definitions which can be used by source programs. The user can add macros to the system library or create his own macro library using the macro library Editor (MACLIBR) as described in Chapter 9 of this volume. Rules and conventions for using macros residing in a macro library are the same as for macros defined and used only within one program, and are described in the Macro Assembler Reference Manual. In general, a macro is accessed in a source program by using its name in the opcode/instruction field of a source statement and supplying any required or optional parameters in the operand field.

The System Macro Library, M.MPXMAC, is assigned by default for assembly to lfc MAC. A different macro library (e.g., M.MACLIB for RTM-compatible macros) can be assigned to MAC if desired.

The Assembler also supports another macro library, MA2. This library is searched by the Assembler for every name found in the opcode/instruction field. It is searched before the permanent symbol table and may be useful to override an existing opcode or Assembler directive.

1.2.3 Listing (LO)

The Assembler produces a listing that pairs hexadecimal representation of object code with the corresponding source statements. The listing lfc is LO. An SLO file is assigned to LO by default. Listings are further described in the Macro Assembler Reference Manual.

1.2.4 Object Code (BO and GO)

Object code is output on the file or device assigned to lfc BO as well as to lfc GO. The default assignment for BO is to a system SBO file which is output to the system device defined as POD at SYSGEN or via the OPCOM SYSASSIGN command. The default assignment for lfc GO is to an SGO file. Other utilities such as LIBED and CATALOG will access an SGO file for the job automatically; however, SGO is temporary and will be lost if not used in the same job.

1.2.5 Compressed Source (CS)

The Assembler will optionally accept source program input or produce source output in compressed format. To output compressed source, assign a file or device to lfc CS. Compressed source as input is specified by assigning the name of the file or device to lfc SI.

If both BO and CS are assigned to SBO files, the binary output is output prior to the compressed source output.

1.2.6 Temporary Files (UT1, UT2)

UT1 is a temporary file used to hold the source text for processing on pass 2 of the Assembler. On pass 1, the Assembler writes the source text, along with the expanded macro text, to UT1 and on pass 2 reads UT1.

UT2 is a temporary file used for the cross reference and symbol table during the assembly.

Table 1-1
Assembler File Assignments

Input/Output Description	Logical File Code	Assignments for Assembly	Previous Processor Assignment	How Specified for Assembly	Comment
Source Code	PRE	Default: ASSIGN1 PRE=MPXPRE Options: ASSIGNn PRE={filename / devmnc}	Permanent file built using EDIT or MEDIA	ASSIGN statements	If the user specifies an assignment to PRE, source is read from PRE until an EOF is reached, then source is read from SI.
	SI	Default: ASSIGN2 SI=SYC	Work file built using EDIT. Permanent file built using EDIT or MEDIA	EDT > BATCH EDT > BATCH jobfile or	For further description see "Accessing the Assembler"
			Cards	??BATCH {D,devmnc / F,jobfile}	
			Other device medium e.g., magnetic tape, where file was copied from cards of a file via MEDIA.	Same route shown for cards.	The file or device can contain compressed source.
			Interactively. See "Accessing the Assembler"		
		Options: ASSIGNn SI= {filename / devmnc} or $SELECT			$SELECT can only be used in batch.
Macro Library	MAC	Default: ASSIGN1 MAC = M.MPXMAC,,U Options: ASSIGN1 MAC={M.MACLIB,,U / userlib,,U / M.RTMMAC,,U}	Macro libraries are maintained or created via the MACLIBR utility.	Accessed automatically for macro calls included in the source code.	MACLIB contains RTM - compatible macros. RTMMAC contains RTM macros for use with the RTM Catologer (see Section 2.10).
	MA2	None			

Table 1-1 (Cont'd)
Assembler File Assignments

Input/Output Description	Logical File Code	Assignments for Assembly	Previous Processor Assignment	How Specified for Assembly	Comment
Binary Object Code	BO and GO	Defaults: ASSIGN2 BO = SBO GO = SGO Options: ASSIGNn BO= {filename / devnmc} ASSIGNn GO= {filename / devnmc}	If a user file is assigned to BO or GO, it must be pre-established via the FILEMGR utility.	ASSIGN statements.	If an SBO file fills up, the Assembler automatically allocates an additional 84 sectors (500 cards). If other than SBO, the utility aborts. SBO output (the default assignment for lfc BO) is routed to the device specified as the Punched Output Device (POD) during SYSGEN (see Volume 3) or reassigned via the OPCOM SYSASSIGN command. (See Volume 1, Section 4.) SGO output is routed to a temp file accumulated for a job so that the object code can be accessed by utilities such as LIBED and the Cataloger. Thus, to enter the object module(s) directly in a library, run LIBED. To catalog the object module(s), immediately run the Cataloger. Note that if you want to retain output and you are not going to catalog or enter the object module(s) into a library during the same job, make a permanent copy on the file or device assigned either to BO or GO.

Table 1-1 (Cont'd)
Assembler File Assignments

Input/Output Description	Logical File Code	Assignments for Assembly	Previous Processor Assignment	How Specified for Assembly	Comment
Compressed Output	CS	No default. ASSIGNn CS= {filename / devnmc}			Output is generated in compressed form. See the Macro Assembler Reference Manual.
Listed Output-Source, Object, errors, if any.	LO	Default: LO = SLO Options: LO= {filename / devnmc}	N/A		If an SLO file fills up (comes to EOF) the Assembler allocates an extra 2000 lines. If the allocation fails or if LO is assigned to other than SLO, the Assembler aborts in this situation.
Temporary Source	UT1	ASSIGN3 UT1=DC,100			Temporary file for source text on pass 2.
Temporary Source Table	UT2	ASSIGN3 UT2=DC,200,U			

1.3 Options

The options used by the Assembler include control options and macro percentage options. The default output control options are a listing, an object file, and a cross-reference. The additional options are the compressed source output and the object output to an SGO file. The macro percentage option is defaulted to 0 percent. Options 10-18 inclusive indicate a percentage of from 0-80. If more than one percentage is specified, the lowest percentage is used.

Option	Description
1	No listed Output (Source Listing)
2	No Punched Output (Binary Object)
3	List Internally Generated Symbols with Cross Reference
4	No Symbol Cross-Reference
5	Binary Output Directed to SGO File
7	Compressed Source Output
8	SI Not Blocked
9	LO, BO, and CS Not Blocked
10	Allot 0 Percent to Macro Storage
11	Allot 10 Percent to Macro Storage
12	Allot 20 Percent to Macro Storage
13	Allot 30 Percent to Macro Storage
14	Allot 40 Percent to Macro Storage
15	Allot 50 Percent to Macro Storage
16	Allot 60 Percent to Macro Storage
17	Allot 70 Percent to Macro Storage
18	Allot 80 Percent to Macro Storage
19	Outputs Symbolic Information to the Cataloger for use by the Symbolic Debugger
20	Generates Replacement 15,X'nn' Instructions for Call Monitor Instructions

1.4 Accessing the Assembler

To access the Assembler as part of a batch job, create a job file using the EDITOR, punch cards, or other media. The job file can be read to SYC and the job activated in several ways:

from the OPCOM console:

```
"<Attention>"
??BATCH { F,jobfile }
        { D,devmnc  }
```

from the OPCOM program:

```
TSM>OPCOM

??BATCH { F,jobfile }
        { D,devmnc  }
```

from the EDITOR:

```
TSM>EDIT
      .
      .
      .
EDT>BATCH [jobfile]
```

If the jobfile is the current EDITOR work file, issue just the BATCH command.

To activate the Assembler and run online, use the TSM ASSIGN commands to make Assembler assignments equivalent to those preceding the EXECUTE ASSEMBLE command on a jobfile, then proceed to issue Assembler directives. (SELECT and OBJECT statements are not available when running the Assembler online.)

```
TSM>ASSEMBLE
ASS>
```

At the Assembler prompt, enter Assembler directives and source code.

1.5 Assembler Directives

See the Macro Assembler Reference Manual.

1.6 Listings

The Assembler produces a listing of source code, object code equivalents, symbol cross references, and error diagnostics.

Typical Assembler output is shown and described in Chapter 7 of the Macro Assembler Reference Manual.

1.7 Errors and Aborts

Errors are detected during both passes of Assembler processing. They are described in Appendix H of the Macro Assembler Reference Manual. Abort codes are described in Appendix C of this volume.

1.7.1 Aborts

When LO is assigned to an SLO file and end-of-file is detected, an additional allocation of 2000 lines will be attempted. If the attempt is unsuccessful or if LO is not assigned to an SLO file, the assembly will be aborted with abort code AS03.

When BO is assigned to an SBO file and end-of-file is detected, an additional allocation of 500 cards (84 sectors) will be attempted. If the attempt is unsuccessful or BO is not assigned to an SBO file, the assembly will be aborted with abort code AS02.

A macro library is not required by the Assembler, but if one is provided, it must be in the proper format if an attempt is made to read it. If the format is invalid, the assembly is aborted. If a macro prototype from a macro library exceeds the remaining size of the macro storage table, the following message is printed on the listing just preceding the macro call, and the macro call is flagged by the Assembler:

```
THE FOLLOWING MACRO CAUSED A TABLE OVERFLOW
```

In some cases, a cross-reference is not generated because there is not enough memory available to sort the cross-reference information. If this occurs, the following message will be printed on LO:

```
**XREF COULD NOT BE PERFORMED**
```

There is not enough memory to store required macros and the symbol table. See Sections 1.1 and 1.3.

If the number of symbols in a program exceeds the maximum number of symbols that the symbol table can hold, the following message is printed on the file or device assigned to LO:

```
SYMBOL TABLE OVERFLOW
```

If the macro table size is exceeded due to too many bytes of in-line macros, in-line FORM skeletons, repeated code, or macro-call argument data, the following message is printed on the file or device assigned to LO:

```
MACRO TABLE OVERFLOW
```

1.8 Examples

Example 1 - In the sequence, the user assembles source code from a file name SJ.MEDIA, outputs object code to a file name OJ.MEDIA, then catalogs the object into a load module file named MEDIA. SLO output for the job is directed to a file named MOUT.

```
$JOB SJ.MEDIA OWNER SLOF=MOUT
$OPTION 17                      (Allocates 70% of Assembler address
                                 space for macros)
$ALLOCATE 16000                 (Allocates 16,000 bytes beyond minimum
                                 Assembler requirement)
$ASSIGN3 UT1=DC,400             (Temporary file for Assembly)
$ASSIGN3 UT2=DC,800,U           (Temporary file for Assembly)
$ASSIGN1 MAC=M.MACLIB,,U        (Uses RTM-compatible macro library)
$A1 BO=OJ.MEDIA
$A1 SI=SJ.MEDIA
$EXECUTE ASSEMBLE
$A1 SGO=OJ.MEDIA                (Object code is now assigned for
                                 cataloging)
$EXECUTE CATALOG
FILES 64                        (Up to 64 files can be allocated
                                 dynamically)
BUFFERS 16                      (Up to 16 buffers can be allocated
                                 dynamically)
ENVIRONMENT MULTICOPY           (More than one copy of MEDIA can be
                                 active at the same time)
ASSIGN2 *IN=SYC                 (Default assignment for MEDIA)
ASSIGN2 *OT=SLO,500             (Default assignment for MEDIA)
OPTION PROMPT                   (Provides MED  prompt automatically in
                                 interactive environment. User does not
                                 have to use the TSM OPTION PROMPT
                                 command)
CATALOG MEDIA
$EOJ
$$
```

2.6 Cataloger Directives

Cataloger directives are summarized below and described in detail on subsequent pages.

For recommended organization of Cataloger directives and job control statements see Section 2.4.1.1 for a nonsegmented task and Section 2.4.2.1 for a segmented task.

Most Cataloger directives can be abbreviated to four characters. ASSIGN statements following the CATALOG directive may be abbreviated to A1, A2, A3, or A4. PROGRAM and PROGRAMX must be completely spelled out.

If a directive or parameter can be abbreviated, the abbreviation is indicated in syntax statements by underlining.

Legal delimiters are commas and blanks. Commas need be used only where shown.

Directive	Function
ABSOLUTE	Specifies an absolute origin for the DSECT.
ALLOCATE	Allocates additional memory for main load module.
ASSIGN1	Equates permanent disc files (optionally unblocked) with lfc's used in task to be cataloged.
ASSIGN2	Equates system SBO, SLO, SYC, or SGO with lfc used in task to be cataloged.
ASSIGN3	Equates device (optionally unblocked) with lfc used in task to be cataloged. Default for tapes and discs is blocked. Assigns a temporary disc file and its size (see Appendix A). Option for unblocking applies only to these units.
ASSIGN4	Equates lfc in task to be cataloged with existing lfc. Equates the assignment for this lfc to existing lfc's device assignment.
BUFFERS	Establishes number of blocking buffers required for dynamic assignments in non-shared tasks. In a shared task, establishes total number of blocking buffers required.
CATALOG	Identifies and describes the load module(s) to be cataloged.
ENVIRONMENT	Describes memory class, residency, map size, and sharing or multicopying requirements of task.
EXCLUDE	Specifies global names in library object modules not to include in load module even though referenced in object modules being cataloged.
EXIT	Terminates Cataloger directive input.
FILES	Establishes number of dynamic disc file assignments in non-shared task. In a shared task, establishes total disc file assignments.

2.6.1 The ABSOLUTE Directive

The ABSOLUTE directive allows the user to build an absolute load module (one that requires no relocation by MPX-32 at load time). The user is responsible for insuring the base address specified is higher than MPX-32 and the TSA. If the base address results in an overlap between the task and MPX-32 or the task's TSA, the task will not load. Memory between the end of the TSA and the start of the task is still allocated to the task and is available for use by the task.

Syntax:

ABSOLUTE [base]

where:

base is a hexadecimal logical address that is to be the base address of the task. This address is rounded up to the nearest 512 word boundary. If no base is supplied, a value of $40000_{(16)}$ is used.

Note: The CSECT origin is not effected by this directive. The transient area option on the CATALOG directive (TRA=X) has no effect when the ABSOLUTE directive is used. Multiple ABSOLUTE directives are not allowed.

2.6.2 The ALLOCATE Directive

A task is always allocated enough memory to accommodate a cataloged load module. ALLOCATE is used to increase the memory allocation for a task at execution time. Other means of allocating more memory are the $ALLOCATE or TSM ALLOCATE commands, which are runtime-specific for a particular task and the M.GE service, used within a task, to obtain memory dynamically.

The ALLOCATE directive used when cataloging a task gets additional memory every time the task is run, i.e., it is static. The allocation cannot be reduced at runtime or by dynamic service calls.

For further description of memory allocation in the logical address space of a task, see Volume 1.

Syntax:

ALLOCATE bytes

where:

bytes specifies the number of additional bytes (in hex) to allocate to the task.

Note: If the size of the operating system plus the size of the task plus the size of the allocate equals more than 128KW, the task cannot be loaded and an abort condition will occur.

2. THE CATALOGER (CATALOG)

The Cataloger produces load modules that are ready to activate in one of three task operating environments: real time, interactive, or batch.

2.1 General Description

To produce the desired load module, the user creates a job stream of job control commands and Cataloger directives. A careful distinction must be made between job control commands for the job that executes the Cataloger, and directives which will reside in the cataloged load module.

The Cataloger creates a load module which contains the transfer address of the task, a Resource Requirement Summary Table, and relocation matrices, in addition to the program code.

The load module resides in a permanent system file of the same name specified in the CATALOG directive.

Assignment statements occurring between the $EXECUTE CATALOG command and the CATALOG directive cause the Cataloger to create entries in the Resource Requirement Summary Table (see the MPX-32 Technical Manual, Chapter 2) located in the load module file (see the MPX-32 Technical Manual, Chapter 6). Directives and assignment statements to the Cataloger are coded without the dollar sign ($). All directives must begin in column one of their respective line.

A sample job stream follows:

```
$JOB TESTCAT DALE      }
$ALLOCATE 28000        }   Job Control Commands
$EXECUTE CATALOG       }
A2 INP=SYC             }
A2 OUT=SLO,100         }
ALLOCATE 18000         }   Directives which will be cataloged
OPTION PROMPT          }   in load module TEST1.
CATALOG TEST1          }
$EOJ
$$
```

A simple load module is illustrated in Figure 2-1.

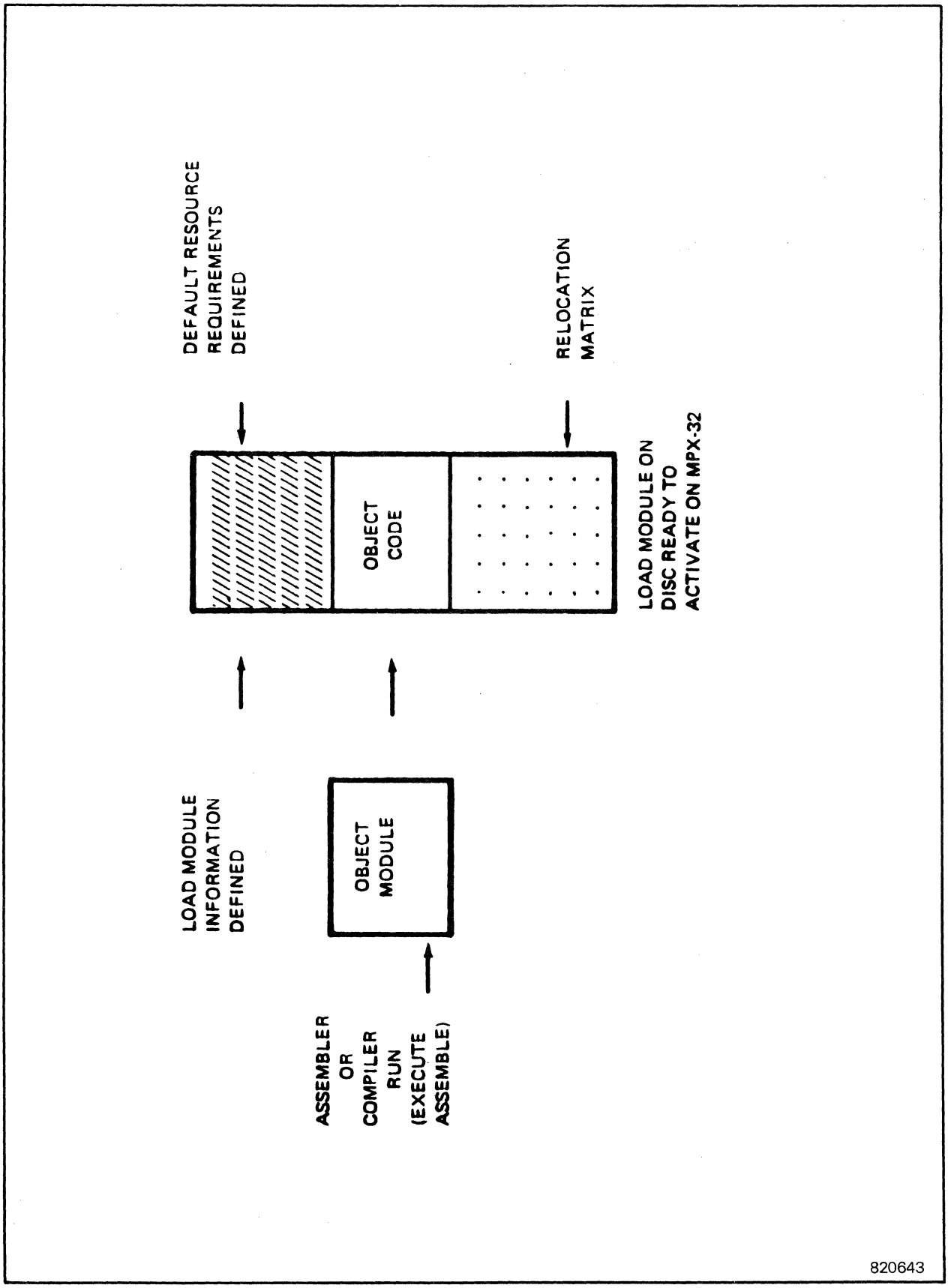


Figure 2-1. Cataloging a Load Module

2.1.1 Load Module Information

The Cataloger ENVIRONMENT and CATALOG directives establish the following special characteristics for a task:

Residency - a task defined as resident either remains in physical memory unless it aborts (RTM-compatible) or remains resident until it exits or aborts (MPX). In either case, it is not a candidate for swap to disc. Default is nonresident, i.e., a task is swappable.

The task needs to execute in a special class of physical memory. E=requires Class E memory; must execute within the first 128KW of memory. H=requires high-speed memory; must execute in H or E. S=the task can execute in slow memory or in any other class of memory that is available; this is the default definition.

Multicopying - the task can be active concurrently in several logical address spaces. The entire task is copied to physical memory each time it is activated.

Sectioned Sharing - the task can be active concurrently in several logical address spaces. The CSECT area of the task is copied into physical memory once and a new DSECT area is established in physical memory each time the task is activated. DSECT areas are deallocated as sharers exit. CSECT remains allocated until all sharers exit.

No Sharing - the task is unique. Only one copy of the load module can be active in one logical address space at a time (the default).

Privilege - any task which accesses a privileged system service must be cataloged as privileged in order for the service to be executed. A privileged task is allowed to write into any area of memory in its logical address space, including the system area, and to execute the 32/75 privileged instruction set. Default is unprivileged.

Base Priority - the priority (1-64) at which the task is executed if activated in a real time environment (by the OPCOM ACTIVATE or ESTABLISH command, another task, a timer, or an interrupt). If activated via TSM or in the batchstream, this priority will be overridden.

Unless defined otherwise with the ENVIRONMENT directive, a task is:

nonresident,

unique,

executable in any memory class available (S, H, or E).

If not specified with the CATALOG directive, the base priority of a task is 60 and its status is unprivileged.

(For further description of multicopying and sharing, see Section 2.1.3.)

The information on residency, priority, etc., is output by the Cataloger at the beginning of the main load module for a task so that it is available for the MPX-32 allocator and execution scheduler immediately upon activation.

2.1.2 Resource Requirements

The resource requirements for a task include all files and devices used by the task:

default assignments

runtime assignments that override the defaults

runtime assignments for required or optional files or devices that do not have default assignments

dynamic assignments

A task's default resource requirements, if any, are established by ASSIGN directives used when the main load module is cataloged. Runtime resources (required, optional, or overriding) are established by the user with ASSIGN directives when the task is activated (at runtime).

Another type of resource requirement is for files or devices that are allocated dynamically by the task via M.ALOC service calls. See Volume 1, Chapter 7.

A prerequisite for any blocked I/O used by a task is a blocking buffer, which the allocator establishes in the Task Service Area (TSA). (Files on disc and magnetic tape are assumed to be blocked unless you specify otherwise when using an ASSIGN directive or M.ALOC service call.) Files also require table entries in the TSA.

The Cataloger preserves resource information on the default files and devices used by a task, including the number of blocking buffers and table entries that are required. At activation, runtime-assigned files and devices are allocated as specified (overriding file and device assignments are merged into the defaults), so that the appropriate memory is allocated for table space and buffers. However, if files and/or devices are allocated dynamically by the task, you must indicate the number of additional file table entries and buffers required.

Cataloger FILES and BUFFERS directives are used to account for dynamic assignments. The FILES directive specifies the number of files and devices allocated dynamically (and thus the number of table entries to leave room for) and the BUFFERS directive specifies the number of blocking buffers required for blocked files or device media accessed dynamically.

Resource requirements for shared tasks require special treatment because several concurrent sharers of the task can use varying runtime assignments that imply varying allocation of blocking buffers and file space. FILES and BUFFERS directives for cataloging shared tasks must reflect the maximum number of files and devices that can be assigned: default (or override), required, optional, and dynamic. This information is required by the Cataloger in order to ensure that the TSA (Task Service Area) for each sharer is the same size and that the DSECT section of the shared task begins at the same location in each sharer's logical address space.

2.1.3 Absolute Load Modules

The Cataloger provides the capability for a user to build an absolute load module. An absolute load module requires no relocation by the loader thereby reducing the allocation time for the task.

The ABSOLUTE directive instructs the Cataloger to resolve all relocatable addresses relative to the base address supplied by the user in the directive. The user is responsible for selecting a base address large enough to be beyond the TSA (task service area) for the task. The TSA is allocated after the end of MPX-32 and varies in size based on the number of files and buffers required in the task.

Tasks that are cataloged as absolute may require recataloging if the size of MPX-32 changes. If there is an overlap between MPX-32 or the task's TSA and the absolute task itself, the task will be aborted during the loading phase.

2.1.4 Sectioned versus Nonsectioned Tasks

The Cataloger supports both sectioned and nonsectioned tasks. Nonsectioned tasks are allocated in a contiguous area in a user's logical address space (in effect, they are comprised of one large DSECT). You can catalog nonsectioned tasks as multicopied, and they will be copied into physical memory to support multiple concurrent activations; or nonsectioned tasks can be cataloged unique, so that only one activation - exit can occur at a time. Nonsectioned tasks cannot, however, be shared in the sense that sectioned tasks can be.

Sectioned tasks use Assembler CSECT and DSECT directives to define pure code and data (CSECT) and impure data (user dependent) sections of the task. The Cataloger merges all CSECT's into a write-protected allocation in upper memory and all DSECT's into a writable allocation in lower memory just above the task's TSA (Task Service Area). Sectioned tasks can take advantage of the CSECT/DSECT sectioning to write-protect pure code and data, but the primary purpose of CSECT/DSECT is to support sharing. If shared, the CSECT of the task is copied into memory once and only the DSECT is recopied with subsequent activations.

A sectioned task can be defined as shared, multicopied, or unique via an ENVIRONMENT directive. A nonsectioned task can be defined as multicopied or unique only. The default for any task is unique as described previously.

Footnote: A task can be developed with CSECT and DSECT directives that are NOP'd for assembly, so that if use or size increases to a point where it is more efficient to use the 8KW or 2KW CSECT than to multicopy, the user can remove the NOP's from the task and recatalog to have the Cataloger build the CSECT and DSECT areas. If using CSECT/DSECT to protect pure code and data, the same memory allocation described for sharing is made by the Cataloger.

There are facets of memory allocation that should be considered in implementing CSECT/DSECT. The minimum allocation for a CSECT area is 8KW on a 32/7x and 2KW on a CONCEPT/32; DSECT is allocated in a separate map block along with the TSA. This means that the minimum space used for the task's DSECT is 8KW or 2KW, including TSA size. This allocation is required by the 8KW map block granularity of the 32/7x or the 2KW map block granularity of the CONCEPT/32. If a task is less than 8KW or 2KW total, and would thus require only one 8KW or 2KW DSECT, multicopying and nonsectioning may allow more efficient use of memory.

2.1.5 Segmented versus Nonsegmented Tasks

Two types of load module can be part of one task. There is one main load module. The name supplied with the CATALOG directive for this module is the name used to activate the task, determine its status, etc. There can be any number of overlay load modules associated with a task, each constructed with a separate CATALOG directive. The main and overlay load modules reside in separate disc files and are linked to each other via system service calls within the object code.

If a task is comprised of a main load module and overlay modules, it is segmented, or overlayed. If it does not use overlays, it is "nonsegmented".

2.1.6 Object Modules and Load Modules

Each load module is composed of one or more assembled or compiled 'modules' of object code. For purpose of this discussion, an object module is the product of assembling or compiling 'n' lines of source code terminated by an END directive or equivalent, and the source module is the source code that forms the object module.

Object modules are normally named. The object module that contains a starting address for the task is defined as such by providing a transfer address that indicates where to begin execution.

It is generated in the Assembler by providing the transfer address with an END directive. If more than one object module has a transfer address, the Cataloger takes the transfer address for the last object module cataloged as the transfer address for the module.

Since an object module produced by an assembly or compilation is identical in format to any other object module, source modules for a task can be written in different languages. The object modules produced by assembly or compilation can be interspersed when they are cataloged into a task.

Object modules are normally output to SGO by the Assembler or compiler. They can be accessed automatically for cataloging or they can be routed to a file and incorporated in a subroutine library. Object modules are retrieved by the Cataloger as described in Section 2.1.7.

Note: The load module files created by CATALOG are the proper size and are file type CA.

2.1.7 Password-Protected Load Modules

Read only (RO) password protection can be supplied for a load module file via the Cataloger PASSWORD directive. Or RO password protection may be supplied for a load module file by using the FILEMGR utility. If a load module file is RO protected, you must use a Cataloger PASSWORD directive when you catalog or recatalog the load module.

If a task uses overlay load modules, each module may have a unique password.

Password only (PO) protection should not be supplied for a load module file. The task will not activate.

2.1.8 The Cataloging Process

The Cataloger makes two passes through SGO, the user library (if assigned), and the system subroutine library.

2.1.8.1 First Pass

On the first pass, the Cataloger searches through the file or device assigned to SGO - normally the temporary system file SGO. (See File Assignments.) It builds a table that includes all REF's and DEF's found in SGO object modules. A REF is output by the assembler or compiler when it encounters an EXT directive in the source. A DEF is output by the assembler or compiler when it encounters a DEF directive in the source.

If the Cataloger finds a REF on SGO with no corresponding DEF, it goes to the user library. It adds DEF's it finds that match the REF's on SGO. It also adds any REF's it finds within the object modules that contain the DEF's it was looking for, to the REF's in the table. If it finds a REF in the user library with no corresponding DEF, or cannot find a SGO REF on the user library, it searches the system subroutine library. It adds DEF's it finds there that match the list of REF's it has built. It adds any REF's it finds within the object modules that contain the DEF's it was looking for, to the REF's in the table.

The Cataloger now has a table that contains the names of all DEF's and REF's that were found in the order they were found (SGO, user library, system subroutine library).

2.1.8.2 Second Pass

The Cataloger retrieves an object module for the first occurrance of each DEF. (SGO, user library, or system subroutine library.) It resolves all matching REF's to these DEF's. If it has found two DEF's with the same name, it takes the first object module that contains the DEF. (It notes duplicate DEF's on the listing output at the end of the Cataloger run.) If any REF's cannot be resolved to DEF's, they are also noted on the listing. Object modules are retrieved from the SGO in the order they are found.

2.1.8.3 Common References

The Cataloger follows the same procedure in the first pass for common block definitions and references.

In the second pass, uninitialized common is allocated based on the largest area defined for a given block. Initialized common is allocated based on the size required by the first function code that initializes the block.

Note that this 'common' is not Global (GLOBAL00-GLOBAL99) or DATAPOOL. Global and DATAPOOL areas are allocated separately in memory. This is common allocated within the task itself, e.g., within a FORTRAN BLOCKDATA subprogram and/or by COMMON op codes in the source.

2.1.8.4 Included or Excluded Object Modules

The INCLUDE directive can be used to specify object modules from a library to include in a load module, even though they are not referenced on SGO. These are added to the first pass REF's in the table built by the Cataloger. There is also an EXCLUDE directive, which specifies DEF's in a library to exclude, even though they are referenced on SGO. These are kept on the table and honored by not adding references to them in the first pass.

2.1.8.5 Selective Retrieval from SGO

The PROGRAM directive can be used to specify names of object modules to include from SGO. Since if no PROGRAM (or PROGRAMX) directive is used, all object modules are taken, this is a means of retrieving SGO object modules selectively for a load module. PROGRAMX excludes all object modules on SGO from a load module.

2.1.8.6 Symbol Tables (SYMTAB's)

A symbol table is built for each load module that is cataloged. It is the table described previously with all references as resolved from the second pass. The symbol table is used if a segmented task is being cataloged. Any external references not resolved for one load module can be resolved when all SYMTAB's are present. SYMTAB's must be saved and restored when a segmented task is cataloged in stages.

Symbols in the symbol table include all external references, all global symbols, and all program names.

2.1.9 Allocation and Use of Global Common and DATAPOOL Partitions

Global common partitions (GLOBAL00-GLOBAL99) named in the object modules are resolved directly to memory locations in system common defined via SYSGEN or the FILEMGR. How DATAPOOL is structured and resolved depends upon a DATAPOOL dictionary that the user creates with the Datapool Editor utility (DPEDIT).

Labeled common blocks are identified as Global by the name "Global" and "Globaldd" where dd is two decimal digits from 00 through 99. When a common block with one of these names is encountered by the Cataloger, space is not allocated for it in the module's area. Instead, all references to the common block are linked to the core partition of the same name. Therefore, a Global common memory partition must be created before a program that references it can be cataloged. If the definition of the partition changes, the programs that reference it must be recataloged.

DATAPOOL references in an object module are included in the table of external references built by the Cataloger and they are resolved to locations in the DATAPOOL area of system common according to the DATAPOOL dictionary supplied by the user (see File Assignments).

The memory allocation unit on the 32/7x is 8KW and on the CONCEPT/32 is 2KW (one mapblock). Global and DATAPOOL are memory partitions which can be defined at SYSGEN or dynamically via the File Manager. In the latter case (dynamically), partitions must be allocated in 16 page (8KW or 2KW) increments. In SYSGEN, protection granule allocation is possible, providing the means to define multiple partitions within a mapblock; however the allocation unit for the task remains 1 mapblock. The unused partitions in a mapblock are write protected and are not included in the task's logical address space.

For further description of Global common and DATAPOOL, see Volume 1. See also the DATAPOOL editor, Volume 2, Chapter 3.

2.2 Files and File Assignments

Figure 2-2 provides an overview of Cataloger input sources and output routes. The path taken by default if no special assignments are made for cataloging is shown with arrows. It should be noted that the assignments covered in this section are external to the task being cataloged and apply only to the cataloging process itself. Default assignments used by the task which is being cataloged (internal assignments) are made via Cataloger ASSIGN directives within the task as described in Section 2.6.

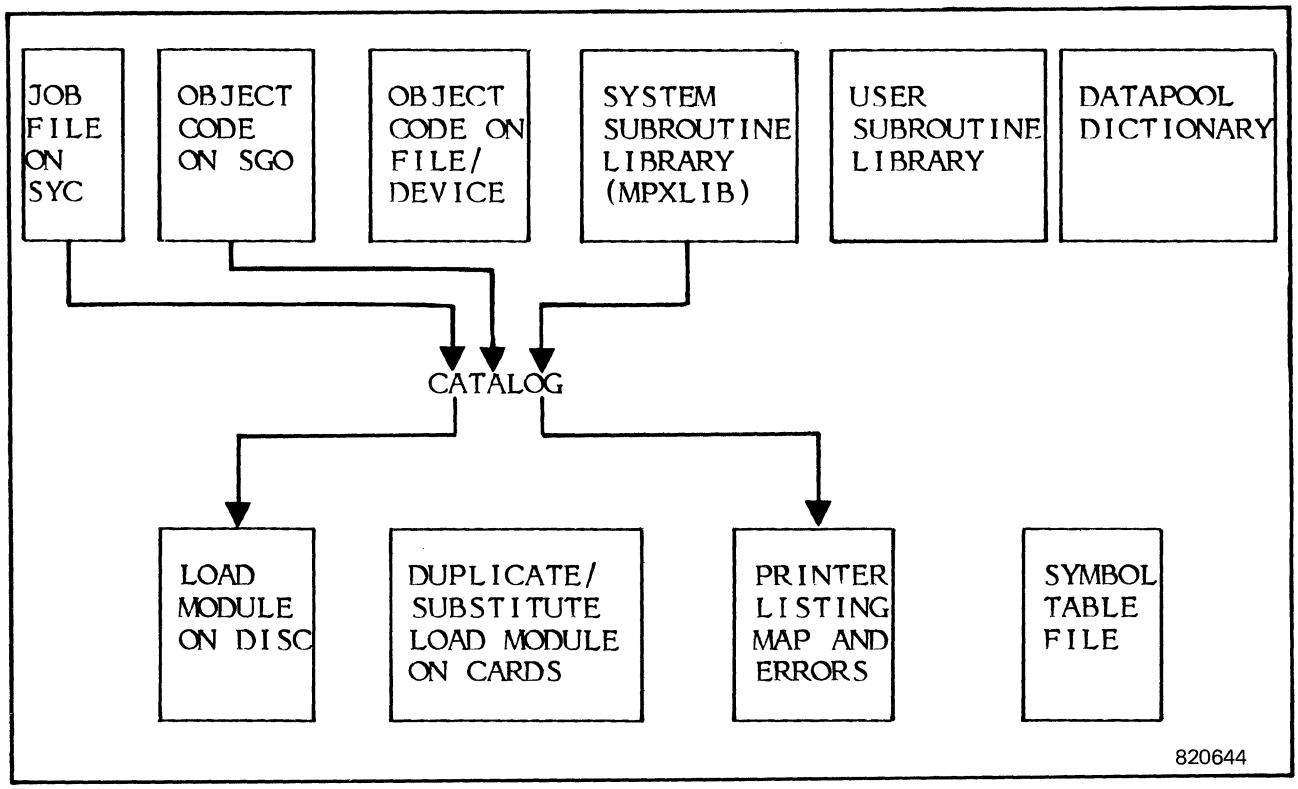


Figure 2-2. I/O Overview

Job File - Contains Job Control Commands, including ASSIGN's, SELECT's, etc., and Cataloger directives for the job. Cataloging can be one of several parts of a job (including for example, compilation or assembly), or a single job using code stored on a file, device, or library. For the required sequence of Cataloger directives, see USING THE CATALOGER, Section 2.4. For sample job files used in cataloging, see Section 2.8.

The alternative routes for reading the file to SYC (interactive and batch) are described in the File Assignment Table, Section 2.2.1, and Activation, Section 2.5.

Object Modules - The result of a compilation or assembly. Object modules can come from SGO (same job as a compilation or assembly), from a permanent file or device medium produced during a compilation or assembly, from a system library file (MPXLIB), or from one or two user-built subroutine library file(s).

System Subroutine Library - A file named MPXLIB. If the SYSTEMS Scientific Runtime Library has been purchased, it is delivered on magnetic tape and output to disc in the file named MPXLIB. MPXLIB contains FORTRAN math subroutines and I/O formatting routines. These 'external' object modules can be accessed by object modules written in various languages, including Assembler.

The user can add object modules (subroutines) to the Scientific Runtime Library or modify the library via the LIBED utility program. The Cataloger searches MPXLIB by default (LIS=MPXLIB). (The directory for MPXLIB is on a file named MPXDIR.) The user can develop a library of user object modules via LIBED and assign it instead of MPXLIB. (See next description.)

User Subroutine Library - A library of subroutines or programs (object modules) built by the user after compilation or assembly by using the LIBED utility. The user can create as many subroutine libraries and associated directories as he needs. The number of these libraries and directories that can be assigned and accessed during a particular CATALOG session is controlled by a variable set when the CATALOG program was assembled. The maximum number of user libraries available is six, therefore, the maximum number of subroutine libraries and associated directories that can be accessed is seven, (six user libraries and the system library). If assigned to lfc LIS (and LID), a seventh user library is searched instead of the library named MPXLIB. The lfcs for the user libraries are Lnn (and Dnn), starting with L01, etc. The subroutine libraries are searched in the lfc order LIB, L01, L02,...,L05, and LIS. The library associated with lfc LIB (and DIR) has the distinct characteristic of always being searched before the other user libraries (Lnn) and the system subroutine libraries. This provides the capability of establishing an installation-wide subroutine library as an extension to the system subroutine library.

> Note: Users who have acquired the source for the CATALOG program may wish to adjust the maximum number of user libraries for their particular requirements. The equated variable MAXULIB located at the beginning of the program contains the specified number. The CATALOG program is nominally distributed with a maximum of six user libraries.
>
> Users who have a universal subroutine library may wish to recatalog the CATALOG program to set the lfc LIB (and DIR) to default to their subroutine library.

DATAPOOL Dictionary - A file built via the DPEDIT utility that contains names and locations of DATAPOOL variables. Allows the Cataloger to find DATAPOOL locations in common memory when DATAPOOL variable names are used in the task being cataloged. More than one dictionary can be built with DPEDIT. It is up to the user to assign the appropriate dictionary for his task when it is cataloged.

Load Module - A cataloged task that is ready for execution. The load module is normally output to disc as a system file; however, output can be suppressed or directed to cards or to a non-system file or device via Cataloger options. (See OPTIONS, Section 2.3.)

Printer Listing; Load Module Map and Errors - These are described in the LISTINGS section. Can be suppressed. See OPTIONS.

Symbol Table (SYMTAB) - This is the mechanism for resolving external references. Used when cataloging a task with overlays in separate Cataloger runs.

2.2.1 File Assignments Chart

Table 2-1, columns 1-3, describes input files used by the Cataloger, their associated file codes, if any, and default assignments, if any. Columns 4-6 relate the Cataloger input files to previous use of other processors as applicable. Where it is feasible to override a default assignment, or to supply more files than the defaults accommodate, columns 3-6 describe options. Output files are also included.

Table 2-1

Cataloger File Assignments

Input/Output Description	Cataloger Logical File Code	Default and Optional Assignments for Cataloging	How Built (Previous Processor Assignment)	How Specified for Cataloging	Comment
Input					
Object modules from compilation or assembly	SGO	Default: SGO=SGO	In assembly or compilation, default assignments for object modules are: GO=SGO (temp file for job) BO=SBO (temp file output to card punch)	Cataloger uses the SGO file associated with the job for object code by default. See "device" for SBO.	Compilation or Assembly: SGO output is temporary. If you want to retain output and you are not going to catalog or enter the object module(s) into a library during the same job, make a permanent copy. See options next. To enter the object module(s) directly in a library, run LIBED. Default input assignment is from SGO (LGO=SGO).
		Option: SGO=filename	In assembly or compilation, change GO or BO assignment disc to a file, e.g., $ASSIGN1 GO=MYFILE	Change SGO assignment, e.g., $ASSIGN1 SGO=MYFILE or Use $SELECTF and $OBJECT, e.g., $OBJECT $SELECTF MYFILE	The object module(s) from the file are read onto the SYC file with the job (via $SELECTF) and automatically transferred by job control to SGO because of the $OBJECT directive.
		Option: SGO=device	Change Go assignment, e.g., ASSIGN3 GO=MT or Use default BO assignment to card punch.	Change or add assignment, e.g., $ASSIGN3 SGO=MT or Use $SELECTD and $OBJECT, e.g., $OBJECT $SELECTD MT	The Cataloger accesses the specified device for object modules The object module(s) from the device are read onto the SYC file and transferred to SGO as in SELECTF above.

Table 2-1 (Cont'd)

Cataloger File Assignments

Input/Output Description	Cataloger Logical File Code	Default and Optional Assignments for Cataloging	How Built (Previous Processor Assignment)	How Specified for Cataloging	Comment
Object modules from System Subroutine Library and Library Directory or User Library and related directory	LIS LID	LIS=MPXLIB [password],U LID=MPXDIR [password],U Option: user library as described below.	Via LIBED utility, where default output assignments for library object modules are: LIB=MPXLIB DIR=MPXDIR Original SUBLIB is optionally contained on FORTRAN installation tape as Math Subroutine Library	MPXLIB/MPXDIR is searched automatically during Catalog, whether another assignment to LIB and DIR is made or not. (See below.)	Object modules (subroutines) for MPXLIB can come from Math Subroutine Library, optionally with added user object modules; or a user can create and name his own library of object modules when he used LIBED and assign it.
Object modules from user library and related directory.	Lnn LIB Dnn DIR	Option: Lnn=user library, [password],U LIB=user library, [password],U Dnn=user directory, [password],U DIR=user directory, [password],U	Via LIBED utility, where user supplies his own LIB and DIR output assignments.	The specified user library and directory will be searched during Cataloging in addition to the library and directory assigned to LIS and LID.	
DATAPOOL variables used in object modules	DPD	No default. If DATAPOOL variables used in object modules, use: DPD=dictionary	The DATAPOOL dictionary is built via the DPEDIT utility, where the DPD directive is used to assign a file for the dictionary. A device is not acceptable.	ASSIGN1 DPD=dictionary when the main load module is cataloged.	The name of the dictionary that corresponds to variables referenced in the cataloged task must be supplied in the main load module of the task.
Global Common Areas	N/A	N/A	Global common memory partitions are defined via SYSGEN or the FILEMGR, with internal structure (position of data within a common area) defined entirely by the user.	N/A	The Cataloger determines the location of a GLOBAL name and resolves references to its location in common memory.

Table 2-1 (Cont'd)

Cataloger File Assignments

Input/Output Description	Cataloger Logical File Code	Default and Optional Assignments for Cataloging	How Built (Previous Processor Assignment)	How Specified for Cataloging	Comment
Symbol Table as Input	N/A	N/A	Previous run of the Cataloger. See below.	Use the SYMTAB directive and specify the symbol table file with SELECT, e.g., SYMTAB $SELECTF SYMFILE	
Symbol Table as Output	SYM	No default. SYM = {filename / devmnc}	By Cataloger. SYM assignment must be made before using SYM option to build table.	Use the SYM option of the CATALOG directive and assign a file or device to SYM, e.g., ASSIGN1 SYM=SYMFILE	
Job File	SYC	SYC=SYC	Work file built using EDITOR. Permanent file built using EDIT or MEDIA. Cards. Other device medium e.g., magnetic tape, where jobfile was copied from cards or a file via MEDIA. Interactively. See "Accessing the Cataloger".	EDITOR>BATCH EDITOR>BATCH jobfile or ??BATCH {D,devmnc / F,jobfile} Same route shown for cards.	For further description see "Accessing the Cataloger".
Temporary Symbolic Debug File	#SY	ASSIGN3 #SY=DC,200	Internal file for Cataloger	ASSIGN3 #SY=DC,200	Used to generate Symbolic Debug information during load module construction

Table 2-1 (Cont'd)

Cataloger File Assignments

Input/Output Description	Cataloger Logical File Code	Default and Optional Assignments for Cataloging	How Built (Previous Processor Assignment)	How Specified for Cataloging	Comment
Output					
Load Module File	N/A	N/A	By Cataloger. Can be main load module, or in segmented task, an overlay load module.	In CATALOG directive, user specifies name of the load module. This is also the name of the file on which the Cataloger builds the load module.	Output to a file can be suppressed by the NOP option on the CATALOG directive.
Duplicate or Substitute Load Module in Card Format	SBM	Default of SBM=CP is specified via the CAR option on the CATALOG directive. Option: Assign a file or device to SBM, e.g., SBM=filename	By Cataloger.	By using the CAR option on the CATALOG command. Card output can be redirected to a file or device other than the card punch by assigning a different device to the lfc SBM, e.g., ASSIGN1 SBM=filename	For duplicate or substitute output, CAR option must be used.
Listing: Module Map and Errors	SLO	SLO=SLO,100 Option: ASSIGN1 SLO=filename ASSIGN3 SLO=devmnc	By Cataloger. Outputs a map which outlines structure of load module and defines number of records and errors, if any. For further description, see Listings and Errors sections.	Default output is to 100 record SLO file, which is output to the device assigned as system LOD. Output can be redirected to a file or device via ASSIGN1 or ASSIGN3 SLO=statement. Output of the module map can be suppressed by using the NOP option on The CATALOG directive.	

2.3 Options

OPTION 1 is used by the Cataloger to suppress the automatic subroutine library search for external references. Therefore, all necessary object modules must be explicitly specified via INCLUDE directives.

OPTION 19 is used by the Cataloger to include symbolic debug information which is placed at the end of the load module (Note - this does not affect memory requirements, it only increases disc usage).

Other options for cataloging a load module are specified as parameters of the CATALOG directive. Their directives are:

NOM suppress printing the load module map.

NOP suppress load module output to the permanent system file named in the CATALOG command.

CAR output the load module in punch card format to the file or device specified by the logical file code SBM. The file will be blocked. No EOF's are written until the end of the load module.

Options for the task being cataloged can be specified with the OPTION directive (see Section 2.6.16).

2.4 Using the Cataloger

This discussion is broken into two major areas, one which describes cataloging concerns pertaining to a nonsegmented task (one load module, no overlays) and a second major area that describes the more complex concerns when a task is segmented.

2.4.1 Cataloging a Nonsegmented Task

For a description of how the Cataloger resolves external references and allocates common blocks, see Section 2.1.8.

2.4.1.1 Job Organization

The following organization of Cataloger and job control directives reflects all possible directives pertaining to a nonsegmented task. It flags directives that are optional by enclosing them in brackets. For detail descriptions, see individual commands in Section 2.6.

The only Cataloger directive that is not optional is CATALOG. Directives shown between $EXECUTE CATALOG and CATALOG can be in any order, but they must precede the CATALOG directive for the main load module. EXCLUDE and/or INCLUDE should precede a PROGRAM or PROGRAMX directive.

Directives in Appropriate Order	Function
[$ASSIGNn]	Supply override, optional, or additional assignments for cataloging. (See File Assignments Chart.)
[$OBJECT $SELECTF filename $SELECTD devmnc]	Can be used to get $OBJECT modules from a permanent file or from a device medium. Use only if creating a job file to run in batch. If interactive, use INCLUDE for this function.
$EXECUTE CATALOG	Activates the Cataloger. Required.
ALLOCATE	Allocates additional memory for task at run time.
ABSOLUTE	Specifies an absolute origin for the DSECT.
PASSWORD	Required to establish or confirm password protection for the load module file to be cataloged.
USERNAME	Required to establish that files used in default and dynamic assignments for the task being cataloged are located in a particular user directory, i.e., that they are not system files.
FILES BUFFERS	Specifies number of dynamically assigned files and blocked files or devices used by the task. (See Section 2.1.2.)
ASSIGNn	Supplies default assignments for task being cataloged.
ENVIRONMENT	Defines task residency, sharability, multicopy, map size, or special memory class for task.
OPTION	Specifies default options for task (0-31).

CATALOG loadmod [privilege] [priority] [options]

	Supplies load module name. This is the name of both the task and the file on which the load module is output by the cataloger. Specifies if the task is privileged or not, and establishes its base priority (1-64). Can also establish output options.
[EXCLUDE] [INCLUDE]	Used for special treatment of object modules on user library and system subroutine library. INCLUDE must be used if PROGRAMX is used. See command descriptions.
[PROGRAM] [PROGRAMX]	Used for selective retrieval of object modules from SGO or to bypass SGO completely.

2.4.1.2 Recataloging the Load Module

When a load module is recataloged and the cataloging process is successful, the old file is deleted, and a new file is created with the same name, on the same disc as the old file.

2.4.2 Cataloging a Segmented Task

Overlays provide a means of segmenting tasks for more efficient memory utilization. When it is impractical to have a large task in memory in its entirety, the task can be divided into a main load module and one or more overlay load modules. A segmented (or overlayed) task is brought into execution by activating the main load module.

The programmer must allocate sufficient space for the worst case memory utilization of the overlays in his program when cataloging the main load module. This is done by summing the memory requirements for the largest overlay at each level and issuing an ALLOCATE directive for the proper amount.

2.4.2.1 Job Organization

The main load module in a segmented task is organized similarly to a nonsegmented task, with the exception that if a symbol table (SYMTAB) is required, a SYMTAB directive followed by $SELECT is used to retrieve it. A CATALOG directive is used for each overlay load module that is cataloged. The order of the CATALOG directives is significant. The CATALOG directive for the main load module must appear first. Low level overlay load modules are cataloged immediately after the main load module and all overlay load modules of a particular level are cataloged sequentially. The association of an overlay load module at one level with an overlay load module at a lower level is established by using the LINKBACK directive. These technicalities are described in more detail in subsequent sections.

Directives in Appropriate Order	Function
[$OBJECT {$SELECT filename {$SELECTD devmnc}]	Can be used to get object modules from a permanent file or from a device medium to SGO. Use only if creating a job file to run in batch. If interactive, use INCLUDE for this function. Use here only if cataloging an overlay load module in a separate Cataloger job.
[$EXECUTE CATALOG]	Activates the Cataloger. Required only if cataloging an overlay load module in a separate Cataloger job.
[PASSWORD]	Required to establish or confirm password protection for the overlay load module file named with the following CATALOG directive.
[ORIGIN]	Establishes a new overlay origin for all load modules which follow up to next ORIGIN or LORIGIN directive. Does not establish new overlay level.
[LORIGIN]	Establishes new overlay level for all load modules which follow up to the next LORIGIN directive. Not required for lowest overlay level.
CATALOG loadmod O	Supplies load module name. This is the name of the file on which the overlay load module is output by the Cataloger. If it follows main module, it is taken as a low level overlay. All subsequent modules up to LORIGIN directive are at same level.
[LINKBACK]	Specifies associated overlay load module at lower level.
[EXCLUDE] [INCLUDE]	Used for special treatment of object modules on user library and system subroutine library. INCLUDE must be used if PROGRAMX is used. See command descriptions.
[PROGRAM [PROGRAMX]	Used for selective retrieval of object modules from SGO or to bypass SGO completely.

2.4.2.2 Overlay Levels

Overlay load modules are accessed by the main load module and access each other via system service calls. An overlay level consists of one or more overlay load modules that do not reference each other internally and can thus be loaded into the same logical memory locations within the task.

Low level overlays usually represent the overlays a main load module calls in after it is loaded. Higher level overlays which follow are associated with one of the lower level overlays.

The simplest overlay structure consists of a single overlay level. In this case, the overlay modules share a single transient area. Each overlay, as it is accessed via a system service such as M.OLAY replaces the previous overlay in memory.

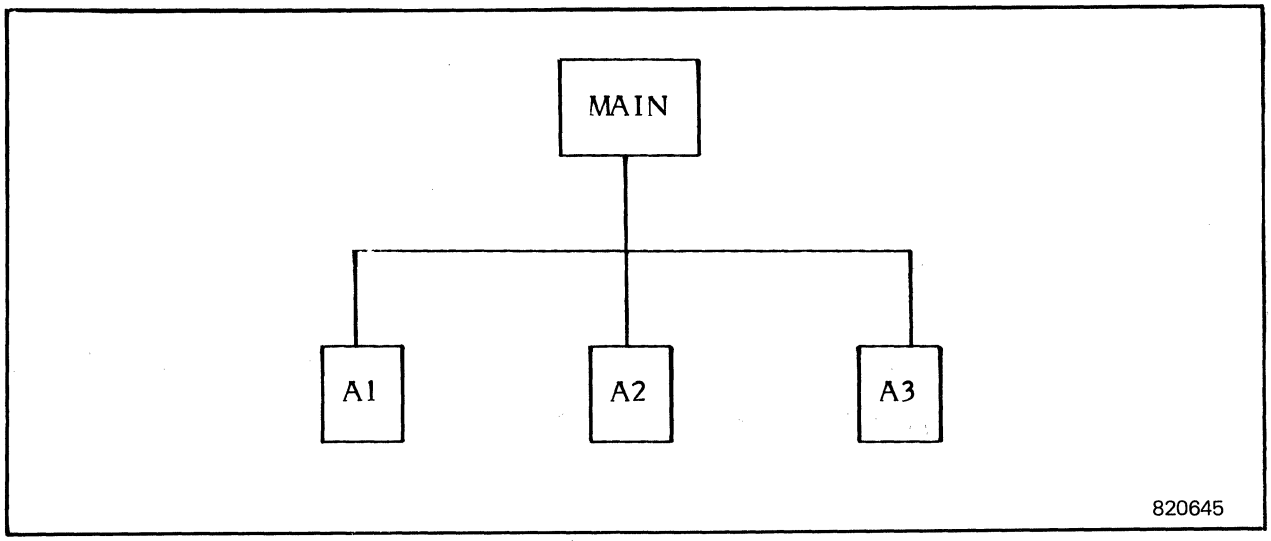


Figure 2-3. Simple Overlay Structure

An example of the logical structure of a task with more overlays and overlay levels is presented in Figure 2-4. This task consists of a main load module and seven overlay load modules. The overlay load modules are grouped into two levels: A and B. Level A overlays are low level. Level B overlays are higher level.

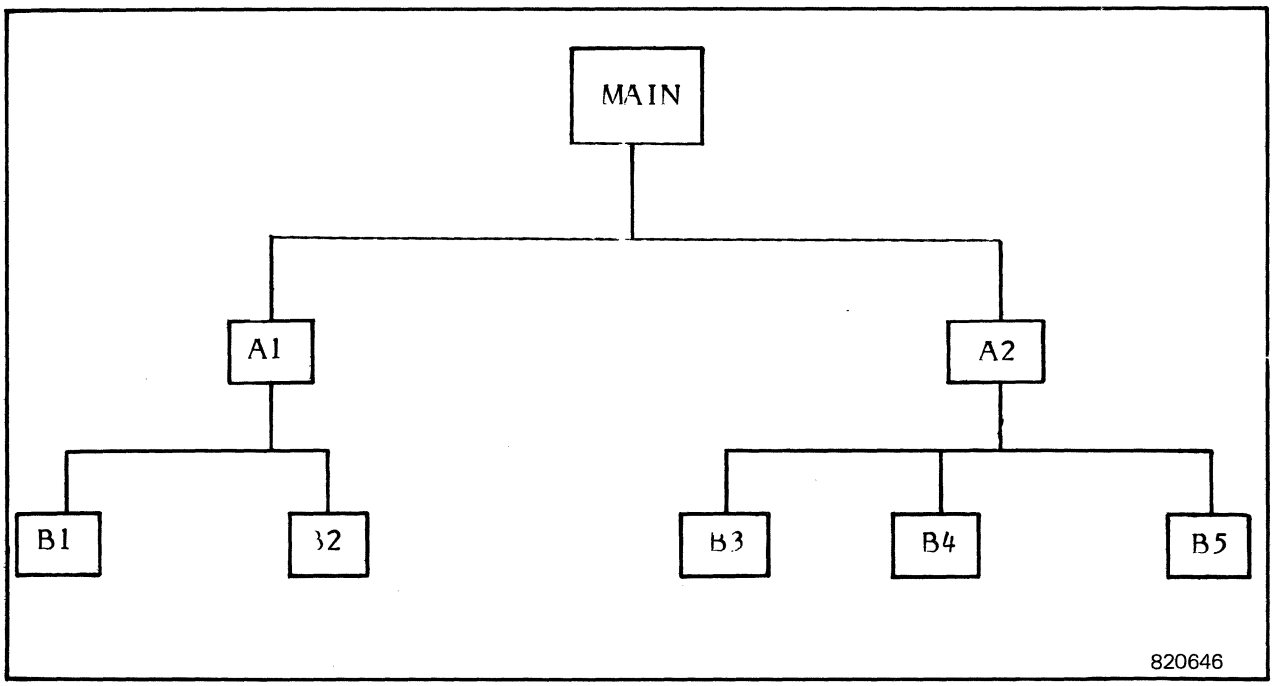


Figure 2-4. More Complex Overlay Structure

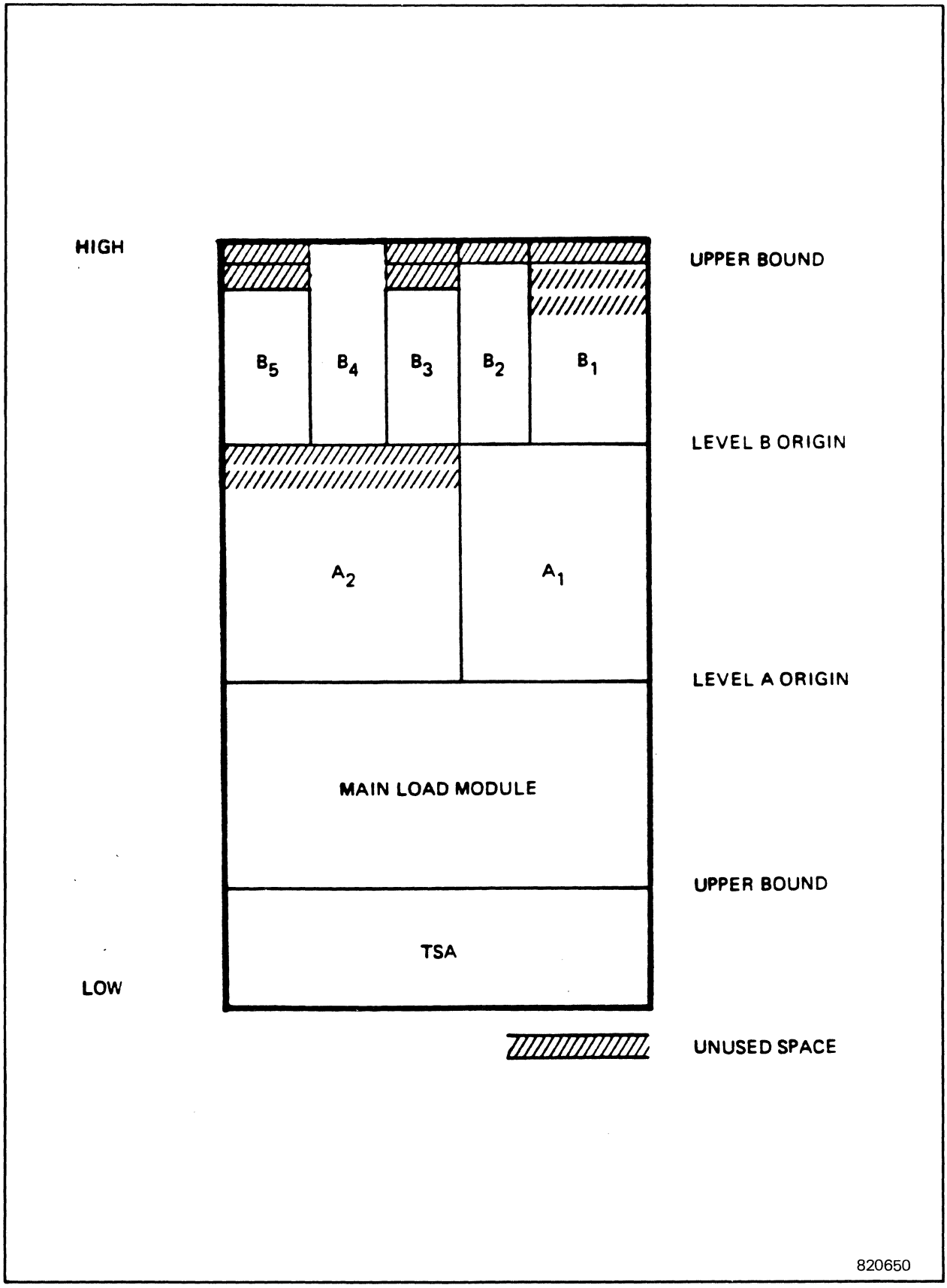


Figure 2-5. Default Memory Allocation for Overlays

The Cataloger ORIGIN (or LORIGIN) directives can be used to modify the overlay structure described previously. For example, a different origin can be set up for higher level load modules associated with A2 (B3, B4, and B5), so that space not being used when A2 is in memory can be used. The total program memory requirements are thus reduced and the programmer can lower the allocation amount in cataloging the main module. Figure 2-6 illustrates how the overlay area is modified.

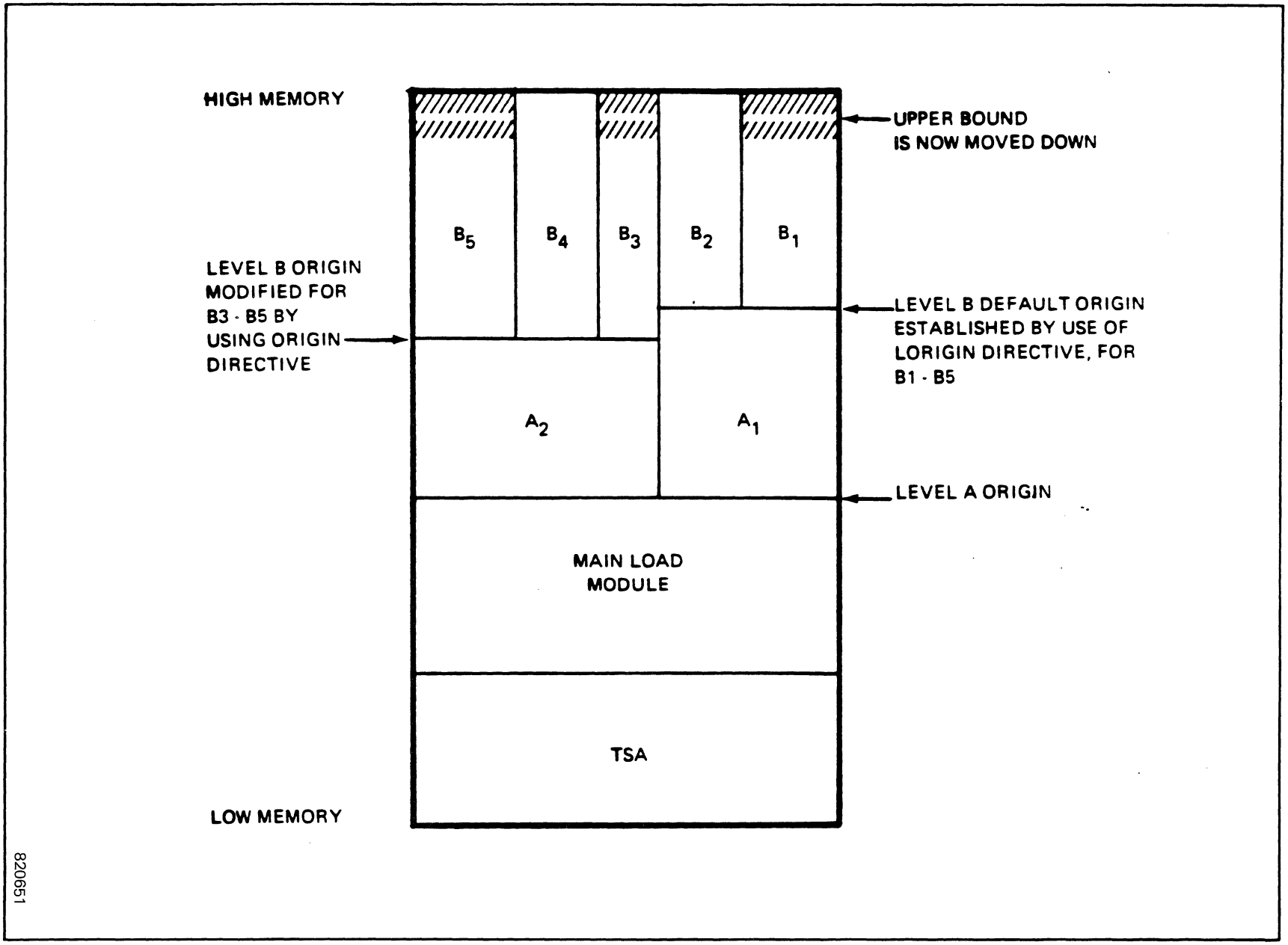


Figure 2-6. Modified Memory Allocation For Overlays

2.4.2.3 The Overlay Transient Area

If the programmer wants his overlays to be in low memory, he can use the overlay transient area by specifying TRA=xxx on the BUILD or CATALOG directive. He must then use the ORIGIN and LORIGIN directives to set the origins for his overlays.

2.4.2.4 Resolution of External References in Segmented Tasks

This section is based on the description of the cataloging process in Section 2.1.8. The resolution of DEF's, REF's, and COMMON definitions and references in a task with overlays is basically the same as described in that section, with several additions to the order in which the table is constructed in the first pass. The order of search for external references in segmented tasks is described below. For the main load module:

restored SYMTAB's, if any

other object modules in the main load module, if any

user library, if any (unless suppressed via $OPTION 1)

the system subroutine library (unless suppressed via $OPTION 1)

overlay load modules, beginning at the lowest level

If the external reference is contained in an overlay load module:

restored SYMTAB's

main load module

lower level overlay load modules associated with the current overlay, via a LINKBACK directive, beginning at the lowest level

other object modules within the current overlay load module

user library, if any (unless suppressed via $OPTION 1)

system subroutine library (unless suppressed via $OPTION 1)

object modules in higher level overlay load modules associated with the current overlay load module

Space for common blocks defined in the main load module is allocated with the main load module. The amount of space for each uninitialized block is the largest amount required in any load module that references it (main or overlay).

If a common block is defined outside the main load module, space for it is allocated with the lowest level overlay load module that defines it.

If a common block is defined in two overlay modules at the same level, space for it is allocated in both overlay load modules and references to it in higher level overlays are resolved to one load module or the other, as applicable.

If a common block is initialized with data, the size of the block is determined by the first occurrance of a definition that initializes the data, regardless of whether the same block is initialized with a larger value in any subsequent object modules or load modules. An overlay load module cannot initialize a common block that is defined in the main load module or an associated overlay load module. The overlay load module is only allowed to initialize common blocks it defines.

2.4.2.5 Cataloging a Segmented Task in Stages

The main load module can be cataloged in one session, with or without overlay load modules. Overlay modules can be cataloged in subsequent sessions. If the transient area size is not declared on the CATALOG directive for the main load module, a transient area is reserved by the Cataloger that is large enough to accommodate any overlay modules that are cataloged in the same run as the main load module. If overlay modules cataloged separately from the main load module require more space, an adequate transient area size must be specified when the main load module is cataloged.

The mechanism used to resolve external references when load modules are cataloged in separate stages is the SYMTAB. The SYMTAB contains the definitions of all common blocks and all DEF's from the previous cataloging session. All REF's must be resolved when the SYMTAB is built. SYMTAB's created during the current session can be added to the SYMTAB file, if desired, so that SYMTAB's can be restored in subsequent runs.

SYMTAB's are saved by assigning a file or device to lfc SYM and specifying the SYM option on the first load module being cataloged in the current session. They are restored by using the SYMTAB directive followed by $SELECT, to retrieve the above file or device.

Common blocks which are defined in cataloged load modules are not reallocated when new load modules are cataloged. Common block sizes are not expanded as a result of definitions contained in new load modules being cataloged.

References to Global Common and DATAPOOL are not affected, as these areas are allocated in a separate area of memory from the task.

2.4.2.6 Recataloging with Overlays

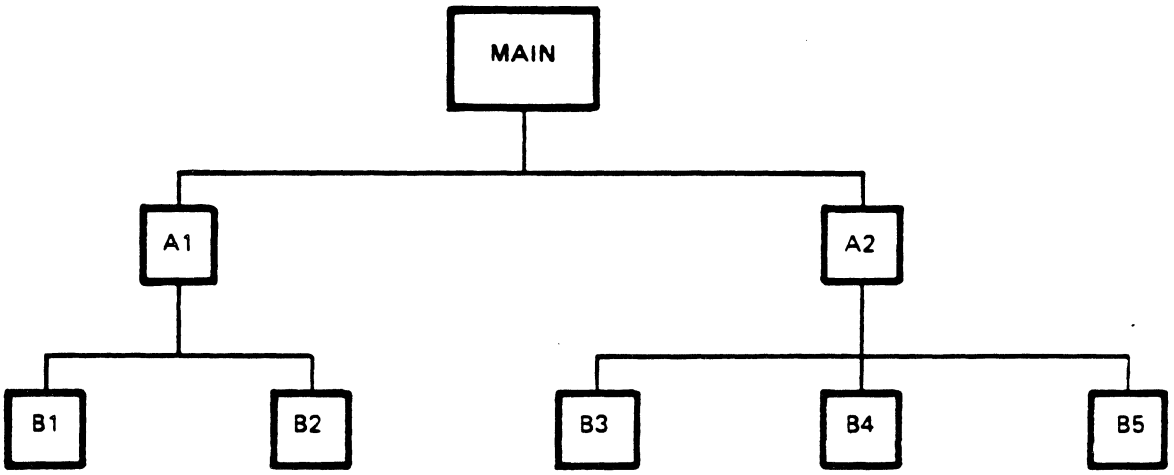
Care is required in recataloging some load modules and not others. Load modules whose sizes increase will end up with allocations that overlap the address spaces of load modules that are not being recataloged. In addition, resolution of external references and common blocks within the task can be affected.

Overlap can be detected by examining the addresses of each load module, which are printed in the module's map (see Listings, Section 2.9). Overlap is indicated when an overlay's end address is greater than the beginning address of a higher level overlay or is greater than the beginning address of the main load module.

Changing the size of the transient area changes the location of the main module in relation to the overlay modules. If the size of the transient area is changed, all previously cataloged overlay modules that reference the main load module must be recataloged.

When a load module is recataloged, the resolution of addresses for DEF's in object modules and common blocks defined within the task may also change. As a result, references to the object modules and common blocks by other load modules are incorrect unless they are recataloged. Assume inter-module referencing for the task as illustrated in Figure 2-7.

In the table at the bottom of Figure 2-7, if any load module(s) are recataloged, all other load modules which correspond to X's in the vertical column beneath the load module must also be recataloged, i.e., if the main load module is recataloged, A1 and A2 must be recataloged. If A1 and A2 are recataloged, all load modules must be recataloged.



Load Module Referenced

	Main	A1	A2	B1	B2	B3	B4	B5
Main		X	X					
A1	X			X	X			
A2	X					X	X	X
B1		X						
B2		X						
B3			X					
B4			X					
B5			X					

820652

Figure 2-7. Recataloging Illustration

2.5 Accessing the Cataloger

To access the Cataloger as part of a batch job, create a job file using the EDITOR, punch cards, or other media as described in Table 2-1. The job file can be read to SYC and the job activated in several ways:

from the OPCOM console:

" < Attention > "

??BATCH {F,jobfile / D,devmnc}

from the OPCOM program:

TSM > OPCOM

??BATCH {F,jobfile / D,devmnc}

from the EDITOR:

TSM > EDIT

EDT > BATCH [jobfile]

If the jobfile is the current EDITOR work file, issue just the BATCH command.

To activate the Cataloger and run on-line, use the TSM ASSIGN commands to make Cataloger assignments equivalent to those preceding the EXECUTE CATALOG command on a jobfile, then proceed to issue Cataloger directives. (SELECT and OBJECT statements are not available when running the Cataloger on-line.)

```
TSM >CATALOG
CAT >CATALOG loadmod privilege priority options
CAT > etc.
```

If there are no Cataloger commands involved in the cataloging task other than CATALOG, the command line and parameters shown entered above at the CAT prompt can be issued directly at the TSM prompt.

2.6.3 The ASSIGN1 Directive

The ASSIGN1 directive is used to supply default file assignments for logical file codes used by the task being cataloged. Assignments for a task must be cataloged with the main load module. For a description of techniques used to set up logical file codes see Volume 1, Chapter 7.

Syntax:

$$\underline{\text{ASSIGN1}}\ \text{lfc=filename}\left[\begin{matrix}\text{,password}\\ \text{,password,U}\\ \text{,,U}\end{matrix}\right]\ [\text{lfc=...}]$$

where:

lfc — is a logical file code used in the task to denote a generic input or output source.

filename — is an 8-character maximum name of a permanent disc file to assign to the lfc.

Any one of the optional parameters following the file name may be entered in the order shown in the syntax statement. Commas separate options. If an option is missing, the comma must be supplied, as in:

filename,,U

password — is an 8-character maximum password for the disc file if it has been password-protected.

If RO protected, the password is required to write to the file. If PO, the password is required to read or write to the file.

U — the file is optionally unblocked. Default: blocked.

Examples:

```
ASSIGN1 LIB=LIBRARY,,U DIR=DIRECTORY,,U

ASSIGN1 OT=OUTFILE IN=INFILE,MYPASS
```

2.6.4 The ASSIGN2 Directive

The ASSIGN2 directive is used to supply default system file assignments to logical file codes. At runtime, an lfc assignment to a system file results in IOCS creating one of the types of files described below for use by the task:

SBO — System Binary Output. A type of temporary file created and used by IOCS for buffering output to the device defined at SYSGEN or via the OPCOM SYSASSIGN command as POD (Punched Output Device). Output from the user task directed to the lfc associated with SBO will be buffered and routed by IOCS to the POD.

SLO — System Listed Output. A type of temporary file created and used by IOCS for buffering output to the device defined at SYSGEN or via the OPCOM SYSASSIGN command as LOD (Listed Output Device). Output from the user task directed to the lfc associated with SLO will be buffered and routed by IOCS to the LOD.

SYC — System Control. A temporary system file associated only with jobs processed in the batchstream. (One SYC per job.) SYC is used for buffering input from the device defined at SYSGEN or via the OPCOM SYSASSIGN command as SIO (System Input Device). Tasks that are not designed to run solely in the batchstream should not make assignments to SYC. Batch tasks can use SYC to input data records.

SGO — System General Object. A system file associated only with jobs processed in the batchstream. SGO is a permanent file used by Job Control to accumulate object code. The SGO file exists until a job is complete, at which time it is deleted. User tasks designed to run only in batch can do I/O to the SGO file as described in Section 2.1.8.

For further description of all the above system files, see Volume 1.

Syntax:

$$\underline{\text{ASSIGN2}}\ \text{lfc=}\left\{\begin{matrix}\text{SBO,cards}\\ \text{SLO,printlines}\\ \text{SYC}\\ \text{SGO}\end{matrix}\right\}\ [\text{lfc=...}]$$

where:

lfc	is a logical file code used in the task
SBO	System Binary Output file
cards	is the number of cards you expect to output as an object deck. Determines size of SBO temporary file required.
SLO	System Listed Output file
printlines	number of printlines required for listed output. Determines size of SLO temporary file required.
SYC	System Control file. Use only if task runs solely in the batchstream.
SGO	System General Object file. Use only if task runs solely in the batchstream.

2.6.5 The ASSIGN3 Directive

The ASSIGN3 directive is used to supply default device assignments for logical file codes used by the task being cataloged. It also assigns a temporary disc file (see Appendix A).

Syntax:

$$\underline{\text{ASSIGN3}}\ \text{lfc=devmnc}\begin{Bmatrix}\text{,blocks}\\ \text{,reel [,vol]}\end{Bmatrix}\begin{Bmatrix}\ \\ \text{[,U]}\end{Bmatrix}\ \text{[lfc=...]}$$

where:

lfc — is a logical file code used in the task

devmnc — is a device mnemonic of a configured peripheral device. See Appendix A.

blocks — number of disc blocks (192 words) to be allocated for this file.

reel — specifies a 1-4 character identifier for the reel. This parameter is required in batch. This parameter is not required in TSM and if not specified, the default is SCRA (scratch).

vol — if multivolume tape, indicates volume number. Default: 0 (not multivolume)

U — specifies the tape or disc is optionally unblocked. Default: Blocked

Note: There must be no embedded blanks within an lfc assignment. Commas must be inserted for all nonspecified options (see Examples). One or more blanks are the legal separator between one lfc assignment and the next.

Examples:

Tape: A3 IN=M91000,SRCE,,U OT=PT

Disc: A3 IN=DC,20

2.6.6 The ASSIGN4 Directive

The ASSIGN4 directive is used to associate one or more logical file codes used by the task being cataloged with an existing lfc assignment. This assignment will remain for the associated file or device even if the original assignment is deallocated.

Syntax:

ASSIGN4 lfc=lfc [lfc=lfc]

where:

lfc=lfc is a pair of logical file codes, where the first lfc is the new assignment and the second is the lfc already associated with a file or device in any previous ASSIGN directive (including ASSIGN4).

Any number of lfc to lfc associations can be established.

2.6.7 The BUFFERS Directive

The BUFFERS directive is used to specify the number of blocking buffers required for dynamic assignments (with M.ALOC) used in a task.

If the task is shared, specify the total number of blocking buffers it requires. (See Section 2.1.2.)

Syntax:

BUFFERS buffers

where:

buffers is the number of dynamic assignments requiring blocking buffers, or if a shared task, total blocking buffers required.

If OPTION 19 is set, the number of buffers supplied is added to the 3 buffers required by the Debugger.

2.6.8 The CATALOG Directive

The CATALOG directive is used to supply a load module file name. When cataloging the main module of a task, specifies the task's privilege, priority, and optionally, selects various output alternatives for the Cataloger. The name supplied for the main module is the name used to activate the task, determine its status, etc. There can be any number of overlay load modules associated with a task, each constructed with a CATALOG directive. The modules reside on separate disc files. The optional parameters can be specified in any order within the syntax statement.

Syntax:

CATALOG loadmod [,P / ,U / ,O] [,TRA=size] [,priority] [,NOM] [,NOP] [,CAR] [,SYM]

where:

loadmod — is the name of a permanent disc file where the main or overlay load module is to be stored.

P,U,O — for the main module only, specifies P for a privileged task, U for an unprivileged task. If an overlay module, specifies O. Overlays assume the privileged or unprivileged status of the main load module. Default: unprivileged, main module.

TRA=size — used with main load module to specify number of bytes (in hex) to allocate for overlay transient area. Default is an area large enough to accommodate all overlay load modules cataloged in the same run as the main load module.

priority — for main load module only, specifies base priority (1-64). Default: 60. Overlay load modules assume the priority of the related main load module. If an overlay module, do not specify priority.

The priority at which the task is executed depends on how the task is activated (online, batch, or real time). If in real time, the task maintains its base priority as cataloged. If activated via TSM or in the batchstream, its priority changes to the SYSGENed priorities of either TSM or Batch.

NOM — optionally inhibits printing a main or overlay load module map.

NOP — optionally inhibits output of a main or overlay load module to the file specified as the load module file.

CAR — optionally outputs the main or overlay load module on punch cards. Card output can be redirected to a different medium by assigning the file code SBM to the desired medium. (See Table 2-1.)

SYM — saves the symbol table for a main or overlay load module on a device or file. This option is used if cataloging load modules of a segmented task in different runs of the CATALOG program.

Note:

RTM parameters RT and BP are ignored, without reporting an error, thus RTM CATALOG directives will still work.

Files whose names begin with the letters SYSG are loaded with a TSA address of X'38000'. This facilitates SYSGEN's remapping between host and target systems.

2.6.9 The ENVIRONMENT Directive

The ENVIRONMENT directive is used to establish residency, execution in a special class of physical memory (E or H) and/or sharing characteristics for a task. As described in Section 2.1.3, the entries with this directive supply information for the load module information area in the main load module.

Unless defined otherwise with this directive, a task is:

nonresident

unique, i.e., not sharable, not multicopied

executable in any memory class available (S, H, or E)

Syntax:

ENVIRONMENT RESIDENT [,E / ,H / ,S] [,SHARED / ,MULTI] [,MAP2048 / ,MAP8192]

where:

RESIDENT makes the task resident in memory (locked in core); it cannot be swapped.

E execute in Class E memory only. If unavailable, delay execution until available.

H execute in Class H or faster memory. If both Class H and E memory are unavailable, delay execution until one or the other is available.

Note: if a 32/75 has no memory installed of the class requested, the first lower speed memory available is allocated to the task.

S Default: task is executed in any class memory available (H, S, or E).

SHARED copies the CSECT area of a sectioned task into physical memory once and copies DSECT as needed for sharing. Use only with a sectioned task.

MULTI multicopies the entire load module into physical memory as needed for concurrent activations. Can be a sectioned or nonsectioned task.

Default: the task is not available for multiple concurrent activations. One copy of the load module can be active at one time in the system.

MAP2048 indicates map size of target system is 2KW.

MAP8192 indicates map size of target system is 8KW. Default.

2.6.10 The EXCLUDE Directive

The EXCLUDE directive is used to exclude object modules in a library (system or user) from the load module being cataloged even though they are referenced in the object modules coming from SGO.

Object modules included from a library during cataloging may also reference the excluded object modules. The references will be ignored and the object modules will remain excluded.

All global symbols in an object module that are referenced by the program must be excluded for the object module to be excluded.

For further description of object modules and the cataloging process, see Section 2.1.

Syntax:

EXCLUDE name [name] ...

where:

name is the name of a global symbol in the object module.

2.6.11 The EXIT Directive

The EXIT directive is used to terminate Cataloger directive input.

Syntax:

EXIT

2.6.12 The FILES Directive

The FILES directive is used to specify the number of files required for dynamic assignments (with M.ALOC) used in a task.

If the task is shared, specify the total number of files required. (See Section 2.1.2.)

Syntax:

FILES number

where:

number — is an ASCII number of dynamic assignments or if a shared task, total logical file codes assigned.

If OPTION 19 is set, the number of files supplied is added to the 5 files required by the Debugger.

2.6.13 The INCLUDE Directive

The INCLUDE directive is used to include object modules from a library (system or user) in a load module being cataloged even though they are not referenced in the object modules on SGO. If PROGRAMX is used to ignore SGO as an input source, INCLUDE must be used to retrieve object modules from a library.

Syntax:

INCLUDE name [name] ...

where:

name — is the name of a global symbol in the object module.

2.6.14 The LINKBACK Directive

The LINKBACK directive specifies overlay load module(s) at lower level(s) for backward links when cataloging an overlay load module. (Forward links from lower to higher level overlay load modules are established automatically by the Cataloger.) Resolves references to object modules and common in the current load module with references to object modules and common blocks in the lower level overlay. (For further description, see Section 2.4.2.4.)

Syntax:

LINKBACK loadmod [loadmod]

where:

loadmod — is the name of an overlay load module at a lower level. User can supply more than one name.

2.6.15 The LORIGIN Directive

The LORIGIN directive is used to establish a new overlay level. Can also establish an origin for this level. Default origin is above the largest overlay load module at the preceding level. LORIGIN need not be used for the lowest level of overlays, but must be used for all higher levels.

Syntax:

LORIGIN { X bytes / loadmod }

where:

X bytes — overrides the default origin of the modules at this level with specific offset from beginning of overlay transient area. Specified by 'X', one or more blanks, and the number of bytes in hexadecimal.

loadmod — specifies the override origin at the end of a specific overlay load module at the previous level. Does not have to be largest overlay at that level.

2.6.16 The OPTION Directive

The OPTION directive specifies up to 32 default options for the task being cataloged. Options 1-32 set corresponding bits (0-31) in the option word in the Task Service Area (TSA) of the task.

When the task is activated, the task can use the M.PGOW service to return the contents of the TSA option word, check the bit settings, and take action as required.

Options 1-32 can also be specified before a task is run interactively or in batch. The TSM or Job Control OPTION commands will override cataloged options 1-20.

Syntax:

OPTION n [,n] ,...

where:

n — is a number from 1-32 which sets the corresponding bit in the TSA status word.

or

can be any of the following keywords:

PROMPT	Set prompt option
DUMP	Set dump option
LOWER	Set lower case input option
IPUBIAS	Set IPU bias option
CPUONLY	Set CPU only option

2.6.17 The ORIGIN Directive

The ORIGIN directive establishes a new origin for overlay load modules which follow. Can be used to override the default origin for a set of overlays. (Default origin is above the largest overlay load module at the preceding level.)

Syntax:

ORIGIN { X bytes | loadmod }

where:

X bytes — overrides the default origin of the modules at this level with specific offset from beginning of overlay transient area. Specified by 'X', one or more blanks, and the number of bytes in hexadecimal.

loadmod — specifies the override origin at the end of a specific overlay load module at the previous level. Does not have to be largest overlay at that level.

2.6.18 The PASSWORD Directive

The PASSWORD directive supplies the password required to write to a load module file that already exists and is password protected. (See Section 2.1.7.)

If a load module file is being created for the first time, can be used to supply a password for it. The file will be RO protected.

The PASSWORD directive remains in effect only for the current load module.

Syntax:

PASSWORD password

where:

password — is the one to eight character password associated with the load module file (if any); if not password protected, can be used to supply a password.

If no password is supplied, cancels the password previously associated with the load module file.

2.6.19 The PROGRAM Directive

The PROGRAM directive is used to specify object modules to include from SGO in a main or overlay load module. If omitted, all object modules on the file or device assigned to SGO are cataloged. (See also PROGRAMX, which is used to exclude all object modules on SGO from a load module.)

Syntax:

PROGRAM objmod [objmod]

where:

objmod is the name of the object module to include. More than one name can be specified.

2.6.20 The PROGRAMX Directive

The PROGRAMX directive is used to ignore the contents of the file or device assigned to lfc SGO in cataloging a load module. An INCLUDE directive is required to get object modules from a library if PROGRAMX is used. (See INCLUDE.)

Syntax:

```
PROGRAMX
```

2.6.21 The SYMTAB Directive

The SYMTAB directive is used when cataloging a segmented task in phases or when recataloging a segmented task. Following SYMTAB, a $SELECT job control statement is used to specify the name of a file or device assigned to lfc SYM in a previous run. (The SYM option must also have been used with the CATALOG directive at the previous session.) On the SYMTAB, the Cataloger has collected the names of all common blocks, DEF's, and REF's used previously.

For further description of SYMTAB use, see Section 2.4.2.5.

Syntax:

```
SYMTAB
$SELECTF filename
$SELECTD devmnc
```

2.6.22 The USERNAME Directive

The USERNAME directive establishes usernames for default files or dynamically assigned files used by the task being cataloged. If not used, files are expected to be system files.

Syntax:

```
USERNAME username [key]
```

where:

username is the one to eight character username establishing the directory in which files are located. Username is normally the same as the owner name used to logon to MPX, or can be any other owner name/user name from the M.KEY file.

key — if a user key is required to logon, it is also established in M.KEY. Supply the valid key for the above user name/owner name.

2.7 Errors

CT01 — Physical end-of-file encountered on subroutine library. The lfc of the library in question is displayed. This results from the library being updated by another user while it is allocated by the Cataloger.

CT02 — Load module file specified with CATALOG cannot be allocated.

CT03 — Unrecoverable I/O error encountered on the DATAPOOL dictionary file assigned to DPD.

CT04 — Listed output space is depleted and additional SLO space cannot be allocated.

CT05 — Unrecoverable I/O error on file or device assigned to SBM for symtab output.

CT06 — An error occurred during the cataloging process and the reason is described in the SLO output.

Below are the error messages output to SLO prior to the CT06 abort.

UNABLE TO DELETE LOAD MODULE, M.DELETE ERROR STATUS IS xx.

See the MPX-32 Reference Manual Volume 1 M.DELETE section for further details.

UNABLE TO CREATE LOAD MODULE, M.CREATE ERROR STATUS IS xx.

See the MPX-32 Reference Manual Volume 1 M.CREATE section for further details.

SYMBOL TABLE OVERFLOW

Allocate more memory for CATALOG to execute in.

UNDEFINED EXTERNAL "exname" REFERENCED IN "modname"

The program element (modname) references an external symbol (exname) that cannot be found in the SGO file or any of the subroutine libraries.

NO DATAPOOL CORE PARTITION DEFINED

A datapool partition must be defined in order to use datapool.

VALID DATAPOOL DICTIONARY FILE NOT ASSIGNED

Assign the datapool dictionary to lfc DPD.

UNDEFINED DATAPOOL "8-char name"

Datapool item could not be found in the datapool dictionary.

PROGRAM "8-char name", OBJECT RECORD X'xxxx' - OUT OF SEQUENCE

Absolute origins are not supported in MPX-32.

PROGRAM "8-char name", OBJECT RECORD X'xxxx' - CHECKSUM ERROR

Absolute origins are not supported in MPX-32.

PROGRAM "8-char name", OBJECT RECORD X'xxxx' - ABSOLUTE ORIGIN

Absolute origins are not supported in MPX-32.

PROGRAM "8-char name", OBJECT RECORD X'xxxx" - BOUND ERROR

Bounding value must be between 0 and 32.

PROGRAM "8-char name", OBJECT RECORD X'xxxx' - UNASSIGNED FUNCTION CODE

PROGRAM "8-char name", OBJECT RECORD X'xxxx' - ILLEGAL COMMON ORIGIN

PROGRAM "8-char name", OBJECT RECORD X'xxxx' - REFERENCE TO UNDEFINED COMMON BLOCK

PROGRAM "8-char name", OBJECT RECORD X'xxxx' - GLOBAL COMMON INITIALIZE

PROGRAM "8-char name", OBJECT RECORD X'xxxx' - PREMATURE END-OF-FILE

PROGRAM "8-char name", OBJECT RECORD X'xxxx' - DATAPOOL REFERENCE OUT OF RANGE

MULTIPLE TRANSFER ADDRESS IN MODULE "8-char name"

MULTIPLY DEFINED EXTERNAL "8-char name"

ERROR IN FIELD x: ILLEGAL DIRECTIVE

ERROR IN FIELD x: ILLEGAL BLANK FIELD

ERROR IN FIELD x: ILLEGAL ENTRY

ERROR IN FIELD x: EXCESSIVE ASSIGNMENTS

ERROR IN FIELD x: MISSING DIRECTIVE

ERROR IN FIELD x: ILLEGAL FILE NAME

2.8 Examples

Example 1 - Cataloging a nonsegmented task. Note that although the CATALOG directive BP parameter from RTM is kept, it will be ignored by MPX-32. The ability of a task to run in a particular environment is not a function of cataloging. See Volume 1.

```
$JOB CAT1 FADEN
$OBJECT
(Object)                    (Object modules to be cataloged.)
$EXECUTE CATALOG
CATALOG MODULE BP U         (Task is unprivileged; entire contents of the
                             SGO file are cataloged.)
$EOJ
```

Example 2 - Cataloging a segmented (overlaid) task. This sample produces the load modules in Figure 2-6.

```
$JOB CAT2 JAN
$OBJECT
(Object)                    (Object modules to be cataloged.)
$EXECUTE CATALOG
ASSIGN2 A=SLO,250
ASSIGN2 B=SYC
ASSIGN3 C=DC,500
CATALOG MAIN
PROGRAM PROGA
CATALOG A1 OV
PROGRAM PROGB
CATALOG A2 O
PROGRAM PROGC
LORIGIN A1                  (Establishes new overlay level and an origin at
                             the end of overlay A1)
CATALOG B1 O
LINKBACK A1                 (Links overlay B1 to lower level overlay A1)
PROGRAM PROGD
CATALOG B2 O
LINKBACK A1
PROGRAM PROGE
ORIGIN  A2                  (Establishes overlay origin at the end of
                             Overlay A2; does not change overlay level)
CATALOG B3 O
LINKBACK A2
PROGRAM PROGF
CATALOG B4 O
LINKBACK A2
PROGRAM PROGG
CATALOG B5 O
LINKBACK A2
PROGRAM PROGH
$EOJ
```

Example 3 - Cataloging a main load module with no linkage to overlay load modules.

```
$JOB CAT3 TERI
$OPTION 5                    (Routes load module to an SGO file)
$EXECUTE FORTRAN
(Source)                     (Programs to be cataloged)
$EXECUTE CATALOG
ASSIGN2 AB=SLO,100 CD=SBO,50
ALLOCATE 1000                (Allocates 1000 additional hexadecimal bytes
                              of memory for task)
ASSIGN1 XY=AFILE
CATALOG MODULE2 TRA=500 P 61
                             (Overlay transient area is 500 hexadecimal
                              bytes)
$EOJ
```

Example 4 - Cataloging overlay load modules with no link to main load module

```
$JOB CAT4 BATMAN
$OBJECT
(Object)                     (Object modules to be cataloged)
$OPTION 5                    (Routes output to SGO File)
$EXECUTE ASSEMBLE
(Source)                     (Produces object modules to be cataloged on
                              SGO)
$EXECUTE CATALOG
CATALOG OVERLAY1 O
PROGRAM PROGA
CATALOG OVERLAY2 O
PROGRAM PROGB
$EOJ
```

Example 5 - Cataloging with a user library

```
$JOB CAT6 ROBIN
$OBJECT
(Object)
$ASSIGN1 DIR=ULIBDIR,,U LIB=ULIB,,U
$ASSIGN1 SYM=SYMFILE         (File for SYMTAB output)
$EXECUTE CATALOG
ASSIGN2 1=SYC 2=SLO,1000
                             (Default assignment for MAINSEG)
CATA MAINSEG TRA=1520 ,,,,,SYM
                             (Note that commas are used to get default
                              parameters for priority, NOM etc.)
EXCLUDE OV1SUB               (OV2SUB is referenced by object modules in
                              MAINSEG but is to be included in OV1.)
INCLUDE MAINSUB              (MAINSUB, referenced by object modules in
                              OV1 but not by object modules in MAINSEG; is
                              to be included in MAINSEG.)
CATALOG OV1 O
INCLUDE OV1SUB
PROGRAMX                     (OV1 consists only of OV1SUB)
$EOJ
```

Example 6 - Cataloging overlay load modules linked to main segment

```
$JOB CAT7 OWNER
$OBJECT
(Object)
$EXECUTE CATALOG
SYMTAB                    (Restores SYMTAB Saved in Example 5 for
                           Linkage to MAINSEG to OV2 and OV3.)
$SELECTF SYMFILE
CATALOG OV2 O
PROGRAM OV2MAIN OV2SUB
CATALOG OV3 O
PROGRAM OV3MAIN
$EOJ
```

2.9 Listings

Sample load module map. Not supplied.

2.10 Creating RTM Tasks on the MPX-32 System

The RTM Cataloger is available on MPX-32 systems for users who want to take advantage of the program development capabilities of MPX-32 to produce programs for systems running under an RTM system.

The name of the alternate Cataloger is RTMCATL.

2.10.1 Assembling RTM Object Modules

MPX-32 will accept most Call Monitor (CALM) instructions in a form that is syntactically and functionally equivalent to RTM CALM's. Thus, source code that uses CALM's can be built to run on RTM or MPX-32. If the code is to run on MPX-32, several CALM's require modification. If the code is to run on RTM systems, make no modification. See the RTM Reference Manual for RTM CALM descriptions and the MPX-32 Reference Manual, Volume 1, Chapter 8 for exceptions related to MPX-32.

Also note that although much of the source code for RTM is compatible with MPX-32, the Communications Region (C.'s) and Task Service Area (T.'s) are constructed differently in the two systems. Thus, some coding sequences will not work on both systems correctly and others will. Two different versions of the source code may be required, one to run on each system. In some cases, the same source assembled against the RTM macro library (M.RTMMAC) for RTM systems and against the RTM-MPX compatible library (M.MACLIB) for MPX-32 systems will however, work.

The MPX-32 Assembler allows users to expand RTM macro calls by using the RTM Macro Library as follows:

```
$ASSIGN1 MAC = M.RTMMAC,,U
```

This is the same file called M.MACLIB on an RTM system.

2.10.2 Running RTMCATL

See the RTM Cataloger description in the RTM Reference Manual, for logical file codes and assignments which apply to using RTMCATL.

FORTRAN programs require an appropriate runtime library for cataloging. Subroutine libraries and directories normally used on an RTM system for cataloging (SUBLIB and SUBLIBD) must be available to run RTMCATL successfully. They are dynamically assigned to logical file codes LIS and LID by the RTM Cataloger.

RTMCATL is accessed just like CATALOG (see Section 2.5). Where the CATALOG directive is shown with MPX-32 parameters, use the CATALOG directive and the RTM syntax shown in the RTM Reference Manual.

2.10.3 Semantic Differences

Note that in RTM documentation, the term program applies to both the separate object modules produced in an assembly or compilation and the output of the Cataloger (an accumulation of one or more object modules).

In MPX-32 documentation, the word program seldom appears, and is replaced by the terms object module, load module, and task. For clarification of how MPX-32 documentation uses these terms, see Section 2.1.5.

MPX-32 also uses the terms segmented and nonsegmented to differentiate between tasks with overlays (segmented) and tasks without overlays (nonsegmented). See Section 2.1.4 for clarification of these terms.

2.10.4 Transporting the Cataloged Task to an RTM System

The MPX-32 File Manager (see Section 6 of this volume) can be used to copy the cataloged load module file to magnetic tape and the RTM File Manager (see the RTM Reference Manual) can be used to copy the load module file to an RTM system.

2.10.5 RTMCATL Load Modules Cannot be Used on MPX-32 Systems

Tasks produced with the special Cataloger will not run correctly on the MPX-32 system. They do not have the same load format as MPX-32 tasks.

3. THE DATAPOOL EDITOR (DPEDIT)

DATAPOOL is a memory partition defined either at SYSGEN or via the File Manager utility (FILEMGR). The DATAPOOL partition is structured via DATAPOOL dictionaries that are built and maintained via the DATAPOOL Editor (DPEDIT). DPEDIT provides the ability to add, change, delete, and equate variables in an existing dictionary or build a new dictionary.

3.1 General Description

With most common partitions (Global Common 00-99, for example) a task must define all locations of the common partition to use any one location. The size defined for each location must also be consistent across tasks which access the memory partition. Thus to change any location in a common partition (other than DATAPOOL), the source for all tasks which access the partition must be modified to reflect the new sequence and/or size of all variables when one changes. (Such tasks must then be reassembled and recataloged.)

DATAPOOL and DATAPOOL dictionaries provide the ability to reference memory locations symbolically by name and to define only the locations actually used by task.

With DATAPOOL, if a variable is changed, it is changed once in a dictionary and all tasks which reference the partition are simply recataloged with the modified dictionary. (If multiple dictionaries are used, their modification depends on whether they reference DATAPOOL locations whose offset would be affected by the change. The user can, if desired, group variables into different offsets from the beginning of the DATAPOOL partition so that tasks which are not related need not be concerned with a redefined location.)

3.1.1 Multiple Dictionaries

Having multiple DATAPOOL dictionaries for a single DATAPOOL partition provides the ability to let the DATAPOOL dictionary act as a translator. For example, if one dictionary defines the variable A as a 1 word offset from the beginning of DATAPOOL partition and a second dictionary defines the variable D as the same offset, A and D become equivalent values for the tasks which use the dictionaries. Multiple dictionaries also allow the user to selectively access locations by communicating tasks. For example, Task A provides 1 byte of status, Task B provides a second byte, Task C provides two more bytes, and Task D's dictionary allows it to pick up all four bytes of status. By providing a separate dictionary for each task, the user ensures that a task cannot modify a location not defined in it's dictionary.

In summary, via DATAPOOL dictionaries, the user has the ability to structure and access DATAPOOL in a number of ways, depending on the needs of tasks which communicate with each other. The reader is also referred to Volume 1, Chapter 2 for a description of various intertask communication features available with MPX-32, including run requests and messages.

3.1.2 Static versus Dynamic DATAPOOL

SYSGEN can be used to permanently allocate memory specified for the DATAPOOL partition in protection granule increments. SYSGEN marks the allocated protection granules as unavailable for outswap and creates an entry defining the partition in the System Master Directory (SMD).

Alternatively, the File Manager CREATEM directive can be used to create a DATAPOOL memory partition. A DATAPOOL partition defined via CREATEM is allocated dynamically when required by a task. Whereas a DATAPOOL partition created via SYSGEN is defined in protection granules, a DATAPOOL partition created via the FILEMGR is 8KW minimum on a 32/7x and 2KW on a CONCEPT/32. DATAPOOL cannot be created via both utilities. If SYSGEN is used, CREATEM cannot be used for DATAPOOL, and vice versa.

For dynamic allocation and deallocation, MPX-32 has the ability to generate multiple DATAPOOL map block(s) into more than one logical address space. If created in the FILEMGR, there can be more than one physical copy of DATAPOOL in memory at a time, depending on the association of tasks that access it simultaneously. Physical space is not taken up permanently (as it is with a SYSGEN-created DATAPOOL partition), thus it is reasonable to have multiple DATAPOOL partitions. Each task structures and shares a given DATAPOOL partition via a DATAPOOL dictionary. All tasks which access the same 'DATAPOOL' do so by specifying the same dictionary during cataloging and by using M.SHARE and M.INCL.

For further description of the use of system common areas such as DATAPOOL, see Volume 1, Chapter 2.

3.1.3 DPEDIT Directives

Directive	Function
/DPD	Assigns the DATAPOOL directory a new permanent file name.
/ENTER	Precedes data records. Data records are the mechanism for adding, deleting, or changing symbols in the DATAPOOL.
/LOG	Provides audit trail listing of all elements in the DATAPOOL directory.
/REMAP	Reuses the DATAPOOL partition by rebuilding from the /SAVE directory entries and hashing them into the DATAPOOL directory.
/SAVE	Preserves binary contents of each active entry in the DATAPOOL directory.
/VERIFY	Verifies DATAPOOL elements in the directory. Assures proper bounding, checks for duplicate entries, corrects improper relative addresses, and provides error flags.

3.1.4 Input Data Format

Data records are the means of structuring a DATAPOOL dictionary. They are built in 80-byte card image format and are used to add, delete, or change DATAPOOL symbols.

The structure of a data record is shown in Figure 3-1 and described in this section.

All fields of the data record except the SOURCE and DESCRIPTION fields must be left-justified and contain no embedded blanks. The VARIABLE SYMBOL field is used to contain the one- to eight-character (ASCII) name of the symbol to be added, deleted, or changed. The function to be performed is specified by the U field.

The U field specifies add by a blank, delete by a minus sign, or change by an asterisk.

The add function must include the fields up to and including the BASE SYMBOL field. The remaining fields are optional. A symbol can be added to the dictionary if it has not been previously defined in the dictionary and if its address is within the range of the DATAPOOL memory partition. If the PRECISION option is specified, address bounding is verified before adding the symbol to the dictionary.

The delete function utilizes only the VARIABLE SYMBOL and U fields. The remaining fields are ignored. A symbol can be deleted only if it is not used as a base. If the symbol to be deleted references a base, the responsibility count for the base symbol is decremented. Responsibility count is the number of times the symbol is used as a base for other symbols.

The change function must include the VARIABLE SYMBOL and U fields. The remaining fields are optional. All fields of a symbol can be changed if the symbol is not being used as a base. If the symbol being changed is used as a base, no changes can be made in the BASE SYMBOL or DISPLACEMENT fields.

Each column on the data record which is blank results in no change to the corresponding column of the original specification; a column which contains a number sign (#) causes the corresponding column of the original specification to be blanked; and a column which contains any other character results in a replacement of the corresponding column of the original specification.

Note that the change function is column oriented. When an entire field is to be replaced, the high-order columns of the field should be padded with number signs (#) in order to blank out unwanted characters from the original specification. For example, the BASE SYMBOL field should be padded with number signs (#) when it is to be entirely replaced by a new symbol which has fewer characters than in the original BASE SYMBOL field.

The E field, which equates symbols with base symbols, must contain an 'EQU'. Any other character string is invalid.

The BASE SYMBOL field is used in conjunction with the VARIABLE SYMBOL field and the E field. The base symbol referenced must have been previously defined by the DATAPOOL dictionary. This field may optionally contain a dollar sign ($) which indicates location 0 of the dictionary.

The DISPLACEMENT field modifies the base symbol location if a plus sign (+) is inserted in column 22. Absence of the plus sign (+) in column 22 causes the displacement to be ignored.

The purpose of the T field is for user documentation of symbol type, but if used, must contain either E, F, I, or L.

If the P field contains L, B, H, W, or D, the specified boundary will be verified against the actual symbol address to ensure proper bounding.

The purpose of the D field is for user documentation of array dimensions, but if used, must contain decimal integer(s).

The SOURCE and DESCRIPTION fields provide for user documentation. The SOURCE field provides a User Descriptor Area to identify the originator of the symbol. An asterisk in the first column of the DESCRIPTION field will cause a page eject during a LOG ALPHA. An asterisk in the second column of the DESCRIPTION field causes a page eject during a LOG relative function. The remaining columns of the DESCRIPTION field can be used for comments.

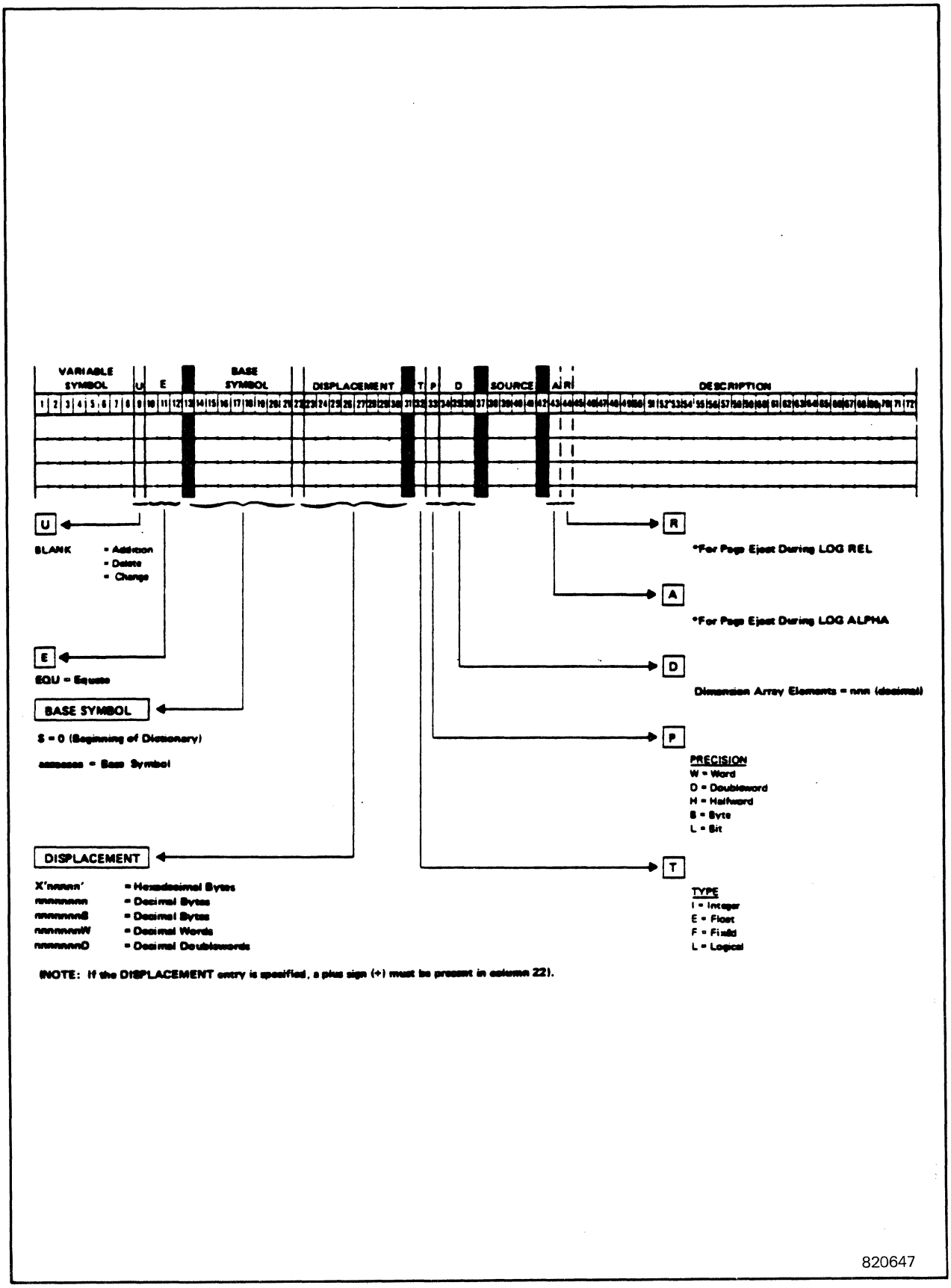


Figure 3-1. Datapool Editor Input Data Format

3.1.5 Dictionary Records

Figure 3-2 shows the format for a DATAPOOL dictionary entry built by the DATAPOOL Editor. The dictionary entries are used by the Cataloger and loader to resolve references to DATAPOOL symbols within a task s logical address space. When a task is cataloged, the user specifies which DATAPOOL dictionary to use by assigning the dictionary file to the logical file code DPD.

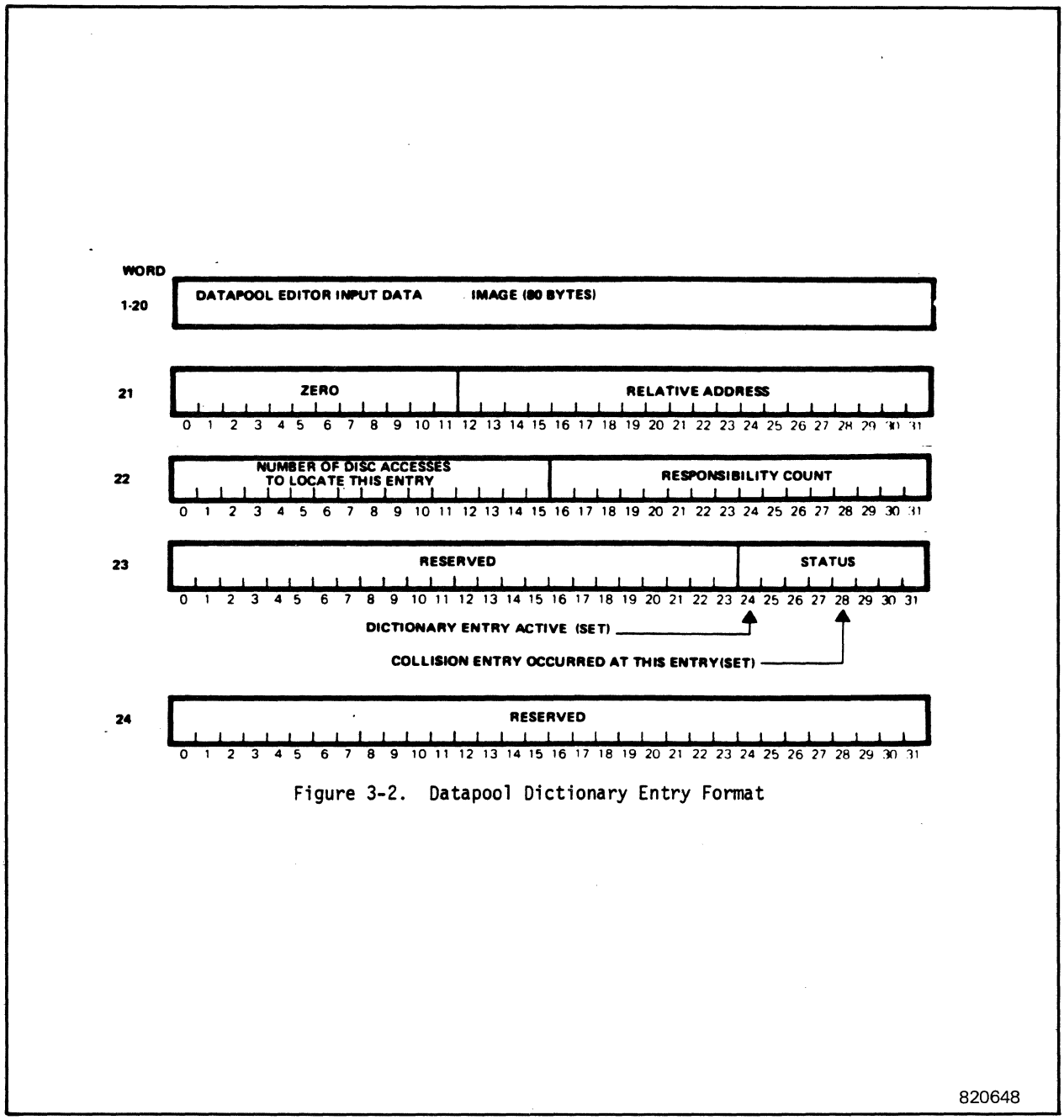


Figure 3-2. Datapool Dictionary Entry Format

Figure 3-2. Datapool Dictionary Entry Format

3.2 Files and File Assignments

Files required by DPEDIT are described in this section.

3.2.1 The Input File (SYC)

The input file includes both DPEDIT directives and the data cards described in the previous section. Data cards follow the DPEDIT ENTER directive. The logical file code for input is SYC. The default assignment is ASSIGN2 SYC = SYC.

3.2.2 The DATAPOOL Dictionary (DPD)

The user must create a permanent file space for the DATAPOOL dictionary via the FILEMGR before running DPEDIT. When the dictionary file is created by the user, its contents are initialized to zeros. The size of the file created should be sufficient to contain twice the number of symbols which will be defined in the dictionary. The blocks are constructed with eight records per block. To determine the size of the file to be created, double the number of entries and divide by eight. The minimum allowable size is five blocks.

The dictionary file for output is associated with the logical file code DPD with an ASSIGN1 unless you need to produce more than one version of the dictionary, in which case, the DPD directive is used to switch from the assigned file to subsequent files. See Section 3.2.4.

3.2.3 Audit Trail and Error Listings (LO and ER)

As DPEDIT processes directives, it produces one line of listed output for each operation it performs. Any operations that produce errors are listed to a separate file or device. The lfc for the audit trail is LO. The lfc for the error lines is ER.

Defaults:

LO=SLO
ER=SLO

All listed output can be produced on one file or device by using an ASSIGN4 to equate the two file codes.

3.2.4 Save and Remap Files (OT and IN)

The REMAP directive can be used to restructure an existing DATAPOOL dictionary that has been saved (via the /SAVE directive) from a previous DPEDIT run or in the current DPEDIT run.

The file or device to use for /SAVE is assigned to lfc OT. The file or device to use when this file is remapped is assigned to lfc IN. Before a remap, use the DPD directive to assign a different file or device for DATAPOOL dictionary output. Or, the name of the file can be specified with /REMAP. If the assignment is not changed, the existing dictionary is overwritten.

3.2.5 Scratch Files (UI and XUI)

A temporary file for sort resulting from a LOG directive (lfc's UI and XUI) is assigned by DPEDIT by default to a disc file, 100 blocks. The temporary file is unblocked.

Table 3 - 1
DPEDIT File Assignments

Input/Output Description	Logical File Code	Assignments for DPEDIT	Previous Processor Assignment	How Specified for DPEDIT	Comment
Directives and Data Records	SYC	Default: ASSIGN2 SYC = SYC	Work file built using EDIT. Permanment file built using EDIT or MEDIA.	EDT > BATCH EDT > BATCH jobfile or ??BATCH {D, devnmc / F, jobfile}	For further description see "Accessing DPEDIT."
			Cards.		
			Other device medium e.g., magnetic tape, where jobfile was copied from cards or a file via MEDIA.	Same route shown for cards.	
			Interactively. See "Accessing DPEDIT."		
			Source data records following an /ENTER directive are the primary means of input to DPEDIT. See Section 3.1 for detailed description.	Source data records may be accessed by a $SELECT Statement in batch.	
Existing Dictionary for REMAP Input	IN	No default. ASSIGNn IN = {filename / devnmc}	File space must be pre-established via the FILEMGR utility.	By assignment and use of the /REMAP or /DPD directive.	
Output Dictionary to use in Subsequent Remap	OT	No default. OT = {filename / devnmc}	Same as above.	By assignment and use of the /SAVE directive.	

Table 3 - 1 (Cont'd)
DPEDIT File Assignments

Input/Output Description	Logical File Code	Assignments for DPEDIT	Previous Processor Assignment	How Specified for DPEDIT	Comment
DATAPOOL Dictionary	DPD	No default. ASSIGNn DPD = {filename / devmnc}	Same as above.	By assignment. A different file or device can be accessed by using the DPD directive.	Unblocked
Audit Trail	LO	Default: ASSIGN2 LO = SLO Options: LO = {filename / devmnc}	N/A, unless using a disc file (see above)	An ASSIGN4 can be used to equate LO to ER so that listings are provided on the same SLO file.	
Separate Error Listing	ER	Default: ASSIGN2 ER = SLO Options: ASSIGN2 ER = {filename / devmnc}	See above.		
Temporary disc file	U1	Default: ASSIGN3 U1=DC,100,U			
Temporary disc file	XU1	Default: ASSIGN4 XU1=U1			

3.3 Options

None.

3.4 Using DPEDIT

For further description of the use and allocation of DATAPOOL (and GLOBAL) system common areas, particularly in context of a task's logical address space, see Volume 1, Chapter 2.

3.5 Accessing DPEDIT

To access DPEDIT as part of a batch job, create a job file using the EDITOR, punch cards, or other media as described in Table 3-1. The job file contains DPEDIT directives and data records preceded by Job Control ASSIGN's, etc. A job file can be read to SYC and the job activated in several ways:

from the OPCOM console:

```
" <Attention> "

??BATCH    {F,jobfile}
           {D,devmnc}
```

from the OPCOM program:

```
TSM > OPCOM

??BATCH    {F,jobfile}
           {D,devmnc}
```

from the EDITOR:

```
TSM > EDIT
        .
        .
        .
EDT > BATCH  [jobfile]
```

If the jobfile is the current EDITOR work file, issue just the BATCH command.

To activate DPEDIT and run online, use the TSM ASSIGN commands to make DPEDIT assignments equivalent to those preceding the EXECUTE DPEDIT command on a jobfile, then proceed to issue DPEDIT directives.

```
TSM > ASSIGN1 DPD=filename,,U
TSM > DPEDIT
DPE > /directive
```

3.6 DPEDIT Directives

DPEDIT directives are summarized in Section 3.1.3. They are described in detail in subsequent pages.

A comma between parameters is the legal delimiter. Blanks embedded after the directive in a DPEDIT command line are ignored.

3.6.1 /DPD Directive

The /DPD directive assigns a different permanent file to the DPD logical file code. It is used to maintain multiple dictionary files during a single edit run.

Syntax:

/DPD filename

where:

filename is the name of a permanent file containing the dictionary to assign to DPD. The file is dynamically allocated using RTM services and is thus automatically unblocked. Only one blank is allowed between the "/DPD" portion of the directive and the filename.

3.6.2 /ENTER Directive

The /ENTER directive indicates that data cards are to be processed by the Datapool Editor. These data cards are used to add symbols to the datapool dictionary, delete symbols from the dictionary, and change parameters defining a symbol in the dictionary.

The data cards that are to be processed as a result of the /ENTER directive must follow the directive. Processing of data cards continues until a directive or an end-of-file indicator is encountered. Multiple /ENTER directives may be used.

A symbol can be added to the dictionary if it has not been previously defined in the dictionary and if its address is within the range of the datapool memory partition. If the Precision option is specified, address bounding will be verified before adding the symbol to the dictionary.

A symbol can be deleted only if it is not used as a base. If the variable symbol to be deleted references a base, the responsibility count for the base symbol is decremented. The responsibility count is the number of times the symbol is used as a base for other symbols.

All fields of a variable symbol can be changed if the variable symbol is not being used as a base. If the variable symbol being changed is used as a base, changes cannot be made in the Base Symbol or Displacement fields.

Syntax

/ENTER

Example(s)

```
/ENTER
LIMA   EQU $
A      EQU $   +   100W   F   W   10
B      EQU A   +   10W                    **
```

In this example, the data cards containing the data cards to be processed follow the Enter directive.

3.6.3 /LOG Directive

The /LOG directive provides a listed output audit trail of all symbols defined in the DATAPOOL dictionary, the total number of entries in the dictionary, and the number of active entries.

Syntax:

/LOG [type]

where:

type — specifies the type of output desired. ALPHA specifies that the listed output will be ordered alphabetically. REL specifies that listed output will be produced in the sequence in which the DATAPOOL items reside in the DATAPOOL memory partition. If no type is specified, both types of output will be generated.

Example(s)

```
/LOG ALPHA
/LOG
/LOG REL
```

3.6.4 /REMAP Directive

The /REMAP directive is used to expand or rebuild a DATAPOOL dictionary without having to recreate dictionary entries through the /ENTER directive data record sequence.

/REMAP rebuilds a dictionary from the dictionary specified with a /SAVE directive or built during a previous run and assigned to lfc IN. Each entry is remapped through the hash coding scheme and written to the dictionary assigned to lfc DPD via the /DPD directive. Or the dictionary to be used for output can be 'assigned' to DPD via the optional file parameter on the /REMAP directive.

Note that the dictionary output file is initially destroyed by the /REMAP function, i.e., if you have built one dictionary and do not make a reassignment through /DPD or the file parameter, the dictionary you built will be lost.

Syntax:

/REMAP [file] , [R]

where:

file — is an optional field which contains the name of a permanent file to assign to DPD. Or the /DPD directive can be used for the same function.

R — if specified in this field, the file assigned to the logical file code IN will be rewound before processing the dictionary entry records.

3.6.5 /SAVE Directive

The /SAVE directive preserves the contents of each active entry in the DATAPOOL dictionary in dictionary entry records on the file assigned to the logical file code OT. An end-of-file is written to OT when the function is complete. The dictionary entry record is a binary record containing the entire dictionary entry. (See Figure 3-2.) A checksum and a sequence number are included in the record.

When a directive is issued, 'DPD' and 'OT' should not be assigned to the same file.

Syntax:

```
/SAVE
```

3.6.6 /VERIFY Directive

The /VERIFY directive checks each active entry in the datapool dictionary for proper placement in the dictionary, for precision to assure proper bounding, and for relative address within the range of the DATAPOOL to ensure that the computed value at entry time is correct. Any discrepancies detected in the dictionary are noted on a listed output file.

Improperly mapped entries are corrected and no error flags are generated. If an improperly mapped entry is encountered, and an entry of the same name is already in the dictionary, the current entry being verified is deleted and an error flag is generated.

Incorrect relative addresses are corrected and an error flag is generated. Invalid entries, that is, entries with no base symbol in the dictionary, or entries whose data record is invalid, are deleted and error flags are generated.

Range and precision errors generate flags.

Syntax:

```
/VERIFY
```

Example(s)

```
$ASSIGN1 DPD=DPD1
.
.
.
$EXECUTE DPEDIT
/REMAP ,R
/VERIFY
```

In this example, Datapool Editor will verify DPD1 after remapping it.

3.7 Listings

The DATAPOOL Editor has two listed output files, the audit trail (accessed through the logical file code LO) and the error audit trail (accessed through the logical file code ER). If either of these files overflows and is assigned to a System Listed Output (SLO) file, the old SLO is dynamically deallocated (released for system output on job termination) and a new SLO file is allocated with the same size requirements as the original. Figure 3-3 describes audit trail format.

The audit trail contains a definition of the operations performed, the source records (Figure 3-1), the relative address within the DATAPOOL of the symbol defined by the dictionary entry, the number of disc accesses required to get the entry, the number of times this symbol is used as a base, and when applicable, an error code defining why the requested operation was not performed.

AUDIT TRAIL FORMAT

PAGE HEADING:

CURRENT DATA FILE: Name (1)

ERROR CODE (EC)	FUNCTION	DATA CARD	RELATIVE ADDRESS (RA)	RESPONSIBILITY COUNT (RC)	COLLISION MAPPING (CM)

FORMAT EXPLANATION

PRINT COLUMNS	HEADING	DESCRIPTION
1-4	ERROR CODE	REFER TO ERROR CODES DESCRIPTION
9-14	FUNCTION	ADD, DELETE, LOG, OR CHANGE
17-96	DATA CARD	
101-108	RELATIVE ADDRES	Hexadecimal address assigned to this variable symbol relative to the beginning of the datapool core partition.
112-115	RESPONSIBILITY COUNT	Decimal number of times that symbol is used as a base.
117-119	COLLISION MAPPING	Decimal number of disc accesses required to locate this entry.

(1) name= 'MAIN' if specified by $ASSIGN1; otherwise, name=File specified by DPD or REMAP directive.

820649

Figure 3-3. DATAPOOL Editor Audit Trail Format

3.8 Errors

The following console messages are issued by the DATAPOOL Editor.

DPEDIT devmnc CKSM

The DATAPOOL Editor has encountered a checksum error on the input (IN) file to the REMAP function.

DPEDIT devmnc SQER

The DATAPOOL Editor has encountered a sequence error on the input (IN) file to the REMAP function.

These messages are only output if IN is assigned to a card device. The "devmnc" specification gives the device mnemonic, including the device address of the device to which IN is currently assigned. After the message is issued, the editor enters a program hold. To retry the read, reposition the deck in the reader and enter the operator command CONTINUE DPEDIT. If no retry is desired, enter the operator command ABORT DPEDIT.

For a description of abort codes, see Appendix C. Error messages EC11 through EC25 and ERnn are described in Appendix C. They report diagnostic error conditions which could cause an abort.

3.9 Examples

Example 1 - Saving Several Dictionaries

```
$JOB DPEDIT1 MEYERS
$ASSIGN1 DPD=DPD1,,U
$ASSIGN3 OT=MT,DPDS
$EXECUTE DPEDIT
/SAVE                    Save Dictionary DPD1
/DPD DPD2                Assign DPD to Dictionary DPD2
/SAVE                    Save DPD2
/DPD DPD3                Assign DPD to DPD3
/SAVE                    Save DPD3
$EOJ
$$
```

Example 2 - Remapping a Dictionary

```
$JOB DPEDIT2 MEYERS
$ASSIGN1 DPD=DPD1,,U
$ASSIGN3 IN=MT,DPDS
$EXECUTE DPEDIT
/REMAP ,R                    Rewind IN and Remap DPD1
/VERIFY                      Verify DPD1
/REMAP DPD2
/REMAP DPD3
/ENTER
A EQU $                      (See Figure 3-1 for column placement)
$EOJ
$$
```

Example 3 - Expanding, Saving, and Remapping a Dictionary

```
$JOB DPEDIT3 MEYERS
$EXECUTE FILEMGR
EXPAND DPD1,100              Expand Dictionary Size
$ASSIGN3 OT=MT,DPDT
$ASSIGN4 IN=OT
$EXECUTE DPEDIT
/SAVE                        Save DPD1
/REMAP DPD1,R                Rewind IN and Remap DPD1
$EOJ
$$
```

Example 4 - Saving a Dictionary on Cards

```
$JOB DPEDIT4 MEYERS
$ASSIGN1 DPD=DPD1,,U
$ASSIGN3 OT=CP
$EXECUTE DPEDIT
/SAVE
$EOJ
$$
```

4. THE DEBUGGER (DEBUG)

The MPX-32 Debugger is used to debug a single, cataloged user task. It can be accessed with a DEBUG command in TSM, with a $DEBUG statement in batch, by coding a M.DEBUG service call within the cataloged task, or by using the Break key after a task has been activated via TSM, in which case TSM provides the option of calling M.DEBUG.

If a command or job control statement is used, the user's task is activated and DEBUG gains control just before the system would transfer control to the user task's transfer address. In cases where the user task is already running, the context of the task (general purpose registers and PSD) just prior to the M.DEBUG call is retained and control is transferred to DEBUG to start debugging at that point in the user's task.

When DEBUG gains control, it prompts the user for a DEBUG command. DEBUG commands allow the user to:

trace task execution

set debugging traps within the task

display and/or alter contents of the task's logical address space, general purpose registers, etc.

watch for privileged task entry into the operating system or other areas of memory not usually accessed even by a privileged task

perform other operations that facilitate task debugging

This chapter concentrates on interactive (online) functioning of DEBUG. Batch functions are described in terms of differences between batch and online operation in Section 4.6, Batch Considerations.

4.1 General Description

DEBUG terms are summarized below.

Absolute Expression	An input expression whose value is determined solely by the terms and operators specified; i.e., an expression which is not relative. See "Relative Expression".
Base	A type of expression term representing any 32-bit number, usually a memory address.
Base Table	Internal DEBUG storage containing the definitions of all special bases and user bases. Maintained by the BASE and CLEAR commands. Displayed by the SHOW command.

Count	A special expression term equal to the number of occurrences of the most recently-occurring trap since that trap was set by the SET command.
Deferred Command	A command whose execution is deferred until the occurrence of a trap. Deferred commands are added to the trap list currently being built rather than being executed immediately. See "Immediate Command".
Immediate Command	A command which is executed immediately rather than being added to a trap list; not a deferred command.
Log File	A circular (wrap-around) temporary disc file on which DEBUG maintains a record of the last 100 (approximate) screens of terminal I/O.
Relative Expression	An input expression assumed by DEBUG to represent a displacement from a base address. The base is automatically added to the value of the expression. See "Absolute Expression".
Special Base	Any of the following bases, which are automatically defined by DEBUG: $ - Current Program Counter $PSD - Program Status Doubleword $TSA - Task Service Area $DSS - DSECT Start $DSE - DSECT End $PCH - Patch Area $CSS - CSECT Start $CSE - CSECT end
Status Report	An analysis of the user task's context, showing the user PSD and registers for each currently active task interrupt level (e.g., I/O end action receiver active).
Trap or Trap Instruction	An SVC 1,X'66' (H.MONS,29 call) instruction used by DEBUG to replace a user instruction in setting a trap in the user task; the control transfer caused by the execution of this instruction (to DEBUG's Entry Point 3).
Trap Address	The address in the user task where a trap instruction has been placed by the SET command.
Trap List	The sequence of DEBUG commands which is executed upon the occurrence of a trap.

Trap List Terminator

Any command which directs control away from a trap list. A trap list terminator must be the last command of a trap list. The following commands are trap list terminators:

BREAK
END
EXIT
FILE
GO
TRACE
TRACK
WATCH

Trap Table

Internal DEBUG storage containing the definitions of all currently set traps, including their trap addresses, COUNT's, and trap lists. Maintained by the SET, DELETE, and CLEAR commands. Displayed by the LIST command.

User Base

Any base other than the special bases; defined by the BASE command.

User Context

The user Program Status Doubleword (PSD) and user registers, collectively.

User PSD

The PSD maintained by DEBUG to indicate the PSD in effect for the user task. On entry to DEBUG, the user PSD is the last PSD in effect for the user task as of the moment of the control transfer; on entry to the user task, it is the PSD to be in effect as the user task gains control; while DEBUG has control, the user PSD may be modified by the following commands:

BREAK
CC
GO
TRACE
TRACK
WATCH

User Registers

The eight words of memory used by DEBUG to contain the user registers in effect for the user task. When DEBUG gains control, the user registers are as reported in the Task Service Area (TSA) in T.CONTXT; on entry to the user task, the user registers contain the register contents to be in effect as the user task gains control; while DEBUG has control, the user registers may be changed by the CR command.

User Task

The task being debugged.

4.1.1 Attaching DEBUG to a User Task

DEBUG functions essentially as an unsolicited overlay of the user task being debugged; i.e., the user neither catalogs DEBUG as an overlay nor identifies any overlay transient area for DEBUG when the user task is cataloged.

The DEBUG overlay receives special handling by MPX-32. The M.DEBUG system service (H.MONS,29, SVC 1,X'63') attaches DEBUG to the calling task as follows:

1. DEBUG is loaded at the beginning of the map block below the user task's pure code and data section (CSECT) and/or common areas. The lower address of the user's CSECT, if any, is thus decreased by the size of DEBUG (8KW). (Refer to the DEBUG memory map in Figure 4-1.)

2. The area "T.CONTXT" is initialized in the calling task's TSA. T.CONTXT contains eight words for the user's register contents at the point of call and two words for the user's PSD at the point of call. It is used by DEBUG to determine the last known context of the user task upon entry to any DEBUG entry point.

3. Control is passed to DEBUG's Entry Point 1 (startup entry point). Any task interrupt levels active at this point remain in effect. They are analyzed by DEBUG and displayed in a status report.

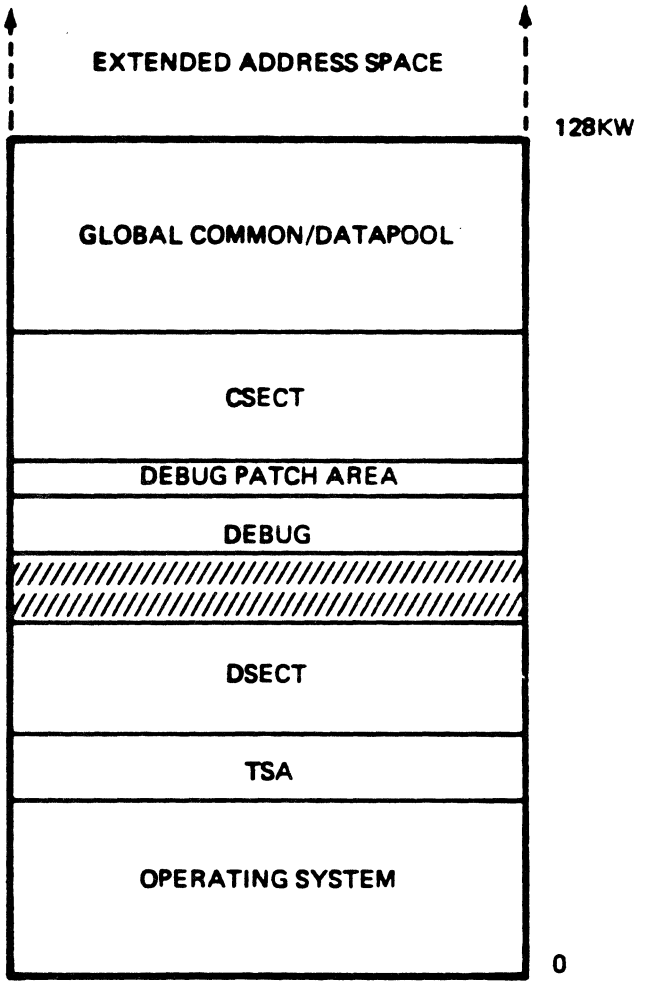


Figure 4-1. Debug Memory Map

In response to the TSM 'DEBUG' command, MPX-32 calls M.DEBUG on behalf of the user task as the last stage of task activation. In this situation, T.CONTXT is initialized with all registers zeroed and the user PSD indicates the user task's cataloged transfer addresses. Control is passed to DEBUG's Entry Point 1 instead of the user task's cataloged transfer address. The combination of DEBUG and the user task is still a single task, with a single TSA and a single dispatch queue entry.

Also, if a task is activated from TSM (TSM > RUN loadmod) and the user depresses the Break key while the task is processing, TSM provides the alternative of attaching DEBUG. In this case, the task context is saved as described previously in step 2.

4.1.2 I/O

4.1.2.1 Terminal I/O

When DEBUG is attached to a task, it obtains the screen size from the Unit Definition Table (UDT) for the terminal device assigned to the logical file code UT (User Terminal). (This is the screen size defined at SYSGEN.)

The number of lines per screen (any non-negative 32-bit number) is used by DEBUG. If the user has defined a value of 0 lines per screen, DEBUG disables the full-screen logic (described below). The minimum allowable screen width (number of characters per line) for debugging is 72 characters, and the maximum is 132 characters, the width of a System Listed Output (SLO) file.

The screen size detected by DEBUG is used to allocate a log file (a temporary disc file) large enough to contain approximately 100 full screens. The log file is manipulated by the LOG and REVIEW commands. It contains a record of the most recent screens of I/O to the user's terminal, providing a complete audit trail of the debugging session. The user is warned 10 screens before the end of the log file space is reached and the oldest records begin to be overwritten by the most recent (a circular file).

The screen width detected by DEBUG is also used to calculate how many words per line will fit into displays such as SNAP's. The format of a screen record is illustrated in Section 4.9.

The screen height is used to enable DEBUG to pause when a full screen of lines has been written to the terminal (terminal write operations by the user task are counted) with no intervening terminal input. This prevents long displays (e.g., SNAP's) from running off the top of the screen before they can be read by the user. A SYSGEN'd height of zero lines signifies that the terminal is a hard-copy device and disables the full-screen logic. A consequence of this disabling is that long SNAP's, for example, cannot be terminated prematurely.

When at the end of a full screen of consecutive output, DEBUG displays the message "CR FOR MORE:", the possible responses and their effects are as follows.

Response	Effect
Carriage Return	The display continues.
Anything Else	Terminates the current command; the next command is read from the terminal.

Terminal input and output are labelled with prefix characters (prompts) that indicate, both on the terminal and on the log file, who said what and when. The prompt for an input command from the terminal is one or two periods: "." or "..". A single prompt signifies that the command is an immediate command; a double prompt signifies a request for a deferred command. In all other cases, the prefix characters are pseudo-prompts in that they are only labels for terminal output lines. Table 4-1 identifies the various combinations of prompt characters which may arise and the significance of each.

Prompt	Significance
.	The user must input an immediate command on logical file code (lfc) #IN.
..	The user must input a deferred command on logical file code (lfc) #IN.
	DEBUG Pseudo-prompts (used to label DEBUG output on lfc #OT):
>	Immediate command from lfc #03 (FILE command)
>>	Deferred command from lfc #03 (FILE command)
!	Immediate command from a trap list.
!!	Deferred command from a trap list.
:	Follows any of the above prompts and pseudo-prompts; labels output resulting from a command.

Table 4-1. DEBUG Prompts and Labels

4.1.2.2 Command Files

A command file is any permanent disc file which contains DEBUG commands. The commands are in the form of 80-byte card images. The MPX-32 Text Editor can be used to create a DEBUG command file. The STORE (not SAVE) command should be used to write a command file. A command file is accessed via the FILE command.

4.1.2.3 SLO Files

DEBUG automatically creates SLO files when the LOG and DUMP commands are used.

4.1.3 Control Transfers

During a debugging session, control can pass back and forth between DEBUG and the user task any number of times. Since, in fact, DEBUG and the user task are parts of a single task, and since it is important for the scheduler to know which part is executing at any given time, all such control transfers take place through scheduler (H.EXEC) service calls.

Each time DEBUG gains control, T.CONTXT in the TSA contains the user task's context as of its last executed instruction, and T.REGS and T.REGP indicate the current task interrupt push-down level in effect for the user task (i.e., the stack is not pushed an additional level upon entry to DEBUG). DEBUG analyzes the TSA and DQE of the task in a status report on the terminal, detailing the user context (PSD and registers) for each active task interrupt level.

When DEBUG gains control, it runs privileged regardless of the privilege state of the user task. When DEBUG passes control to the user task, the scheduler restores the user task's privilege state.

The following is a summary of the control transfers which take place between DEBUG and the user task. DEBUG always gains control as a result of M.DEBUG, whether it is called by the task activation service, by the task itself, or by TSM at the request of the terminal user after the task is running.

The user task gains control:

1. when DEBUG executes a GO command;
2. when DEBUG executes a BREAK command;
3. for the execution of a user instruction during a TRACE, TRACK, or WATCH command;
4. when DEBUG executes a DETACH command.

DEBUG regains control from the user task:

1. when the user task executes a DEBUG trap instruction;
2. when IOCS recognizes a break from the user's terminal;
3. when the user task calls the M.BRKXIT service, after DEBUG executes a BREAK command;
4. after the execution of a user instruction during TRACE, TRACK, and WATCH;
5. when the user task would normally be aborted by MPX-32;
6. when the user task executes an Exit system service.

Note that if DEBUG gains control upon a trap instruction (SVC1,X'66') which was coded by the user (as opposed to one which was planted by the SET command), DEBUG will interpret it as a break from the terminal instead of a trap. (See Section 4.1.4.)

If for any reason during a debugging session it is useful to clear all active user task interrupt levels, the RESTART command may be used for this purpose.

4.1.4 Break Handling

A break occurs when:

the terminal user depresses the Break key

any task uses the M.INT service to simulate an interrupt and enter a break receiver

DEBUG acknowledges breaks by analyzing the user context in a status report when it receives control at its break-handling entry point. It prompts the terminal user for the next immediate command. If a command file is being used, command file processing terminates.

While DEBUG is attached to a user task, that task's break receiver (if any) can be accessed only by using the BREAK command.

DEBUG recognizes breaks only when:

it has executed a GO command and has not yet prompted for the next command, i.e., when the user task has control;

it is executing a WATCH command.

At all other times, breaks for the task are ignored.

If executing GO or WATCH as described above, the execution of a trap instruction that was not set by the SET command appears to DEBUG as a break which occurred between the execution of the trap instruction and the next user instruction.

4.2 Files and File Assignments

DEBUG has no static file allocations. When it gains control at its startup entry point, it dynamically allocates terminal input and terminal output file codes to the terminal and provides a log file with enough blocks for approximately 100 screens of terminal I/O.

During the debugging session DEBUG allocates SLO files for LOG and DUMP commands (enough for the log or dump being printed) and assigns command input from a file specified in the FILE command.

4.2.1 File Assignments Chart

Table 4-2, columns 1-3, describes input and output files used by the Debugger, their associated logical file codes, and default assignments. Column 4 is not applicable to the Debugger. The only file assignments feasible to override for debugging are the input and output assignments for batch processing as described in column 3.

Table 4-2

Debugger File Assignments

Input/Output Description	Logical File Code	Default and Optional Assignments	How Built (Previous Processor Assignment)	How Specified	Comment
Terminal Input	#IN	ASSIGN4 #IN=UT	N/A	N/A - Do not specify.	
Batch Input (Job File)		ASSIGN2 #IN=SYC User can override with Job Control ASSIGN's.)	N/A	BATCH command in EDITOR or OPCOM.	See Section 4.4
Terminal Output	#OT	ASSIGN4 #OT=UT	N/A	N/A - Do not specify.	
Batch Output		ASSIGN2 #OT=SLO,1000 (User can override with Job Control ASSIGN's.)	N/A	N/A unless you want to override.	
Log File (temporary disc file)	#01	ASSIGN3 #01=DC,n	N/A	N/A. DEBUG allocates #01 as a temporary file.	The log file is large enough to hold approximately 100 screens of terminal I/O or batch equivalent.
Output Files	#02	ASSIGN2 #02=SLO,n	N/A	N/A. Do not specify.	In batch, file assigned to #OT is used for DUMP output. A LOG command is treated as a comment.
Command File	#03	ASSIGN1 #03=filename	N/A	N/A. DEBUG makes this assignment automatically when the FILE command is used.	

4.3 Using the Debugger

This section describes the use of expressions, how to set traps and trap lists, and the use of relative and absolute addressing within the Debugger.

4.3.1 Expressions

An expression is an input character string which is composed of a term or a sequence of terms separated by operators. Each term represents a 32-bit, signed, binary number. Expressions are used to specify memory addresses, memory contents, logical masks, character strings, or numbers.

Terms that are legal for use in expressions are defined and described in the following paragraphs, as are operators that specify arithmetic and logical operations to be performed on two terms or expressions. All operators are binary (requiring two arguments) and have no hierarchy of precedence. They are executed left-to-right, one at a time, except where the user defines precedence by parentheses (exactly as in FORTRAN expressions containing operators of equal precedence).

The following general rules apply:

1. Any term is an expression.

2. If e and f are expressions and * is any operator,

   e*f

   is an expression whose value is the result of performing the "*" operation on the values of e and f.

3. If e, f, and g are expressions and * and # are any operators,

   e*f#g

   is evaluated as

   (e*f)#g;

   i.e., the value of e*f is calculated first.

4. Parentheses may be used to override the normal left-to-right execution of operators during the evaluation of an expression. During evaluation the subexpressions in innermost parentheses are each evaluated left-to-right and their values replace the parenthetical subexpressions. This process is continued through any number of levels of nested parentheses until no parentheses remain. Then the resulting expression is evaluated in the normal left-to-right manner.

5. All operators act on and result in 32-bit values.

DEBUG recognizes five types of terms in expressions: constants, register content references, memory content references, bases, and COUNT.

4.3.1.1 Constants

There are five types of constants, as follows:

Hexadecimal Constant - A string of 1 to 8 hexadecimal digits enclosed in apostrophes and preceded by the letter X (e.g., X'1EC'). If a "FORMAT N" command (which assumes decimal format) is not in effect, the letter X and the apostrophes may be omitted.

Decimal Constant - A string of 1 to 10 decimal digits enclosed in apostrophes and preceded by the letter N (e.g., N'193'). If a "FORMAT N" command is in effect, the letter N and the apostrophes may be omitted. The resulting value is truncated on the left to produce a 32-bit value.

C Constant - A string of 1 to 4 ASCII characters enclosed in apostrophes and preceded by the letter C (e.g., C'A1?'). If fewer than 4 characters are entered, trailing blanks are added to produce a 32-bit value.

G Constant - A string of 1 to 4 ASCII characters enclosed in apostrophes and preceded by the letter G (e.g., G'A1?'). If fewer than 4 characters are entered, leading binary zeroes are added to produce a 32-bit value.

Binary Constant - A string of 1 to 32 ASCII 1's and 0's enclosed in apostrophes and preceded by the letter B (e.g., B'101011'). If fewer than 32 bits are entered, leading binary zeroes are added to produce a 32-bit value.

4.3.1.2 Register Content References

These are references to general purpose registers in the form Rn, where n is 0-7. DEBUG uses 32-bit contents of the specified user register.

4.3.1.3 Memory Content (Indirect) References

Valid references are:

C(base)
C(base+hex)
C(base-hex)
C(base+dec)
C(base-dec)
C(hex)
C(dec)

where:

base is a base (see next section)

hex is a hexadecimal number

dec is a decimal number

These expressions specify the contents of the 32-bit word whose address is the expression inside the parentheses. Bits 30 and 31 of the expression value are zeroed to determine the word address.

4.3.1.4 Bases

Bases are symbolic terms whose names begin with $. The special bases automatically defined by DEBUG are as follows:

Name	Specifies
$	Bits 13-31 of user PSD
$PSD	Bits 0-31 of user PSD
$TSA	Address of TSA
$DSS	Address of first word of DSECT after the TSA
$DSE	Address of first word following the end of DSECT
$PCH	Address of first word of 256W DEBUG patch area
$CSS	Address of first word of user CSECT (=$CSE if no user CSECT)
$CSE	Address of first GLOBAL/DATAPOOL (=128KW if no GLOBAL or DATAPOOL)

Figure 4-2 shows the relative positions of the last six bases named above on a memory map of a user task which uses all possible memory areas (CSECT, DSECT, Global Common, and extended address space).

User bases can be defined by the BASE command. Their names consist of $ followed by one to eight alphanumeric characters, the first of which must be alphabetic. User base names must not match any of the special base names used by DEBUG.

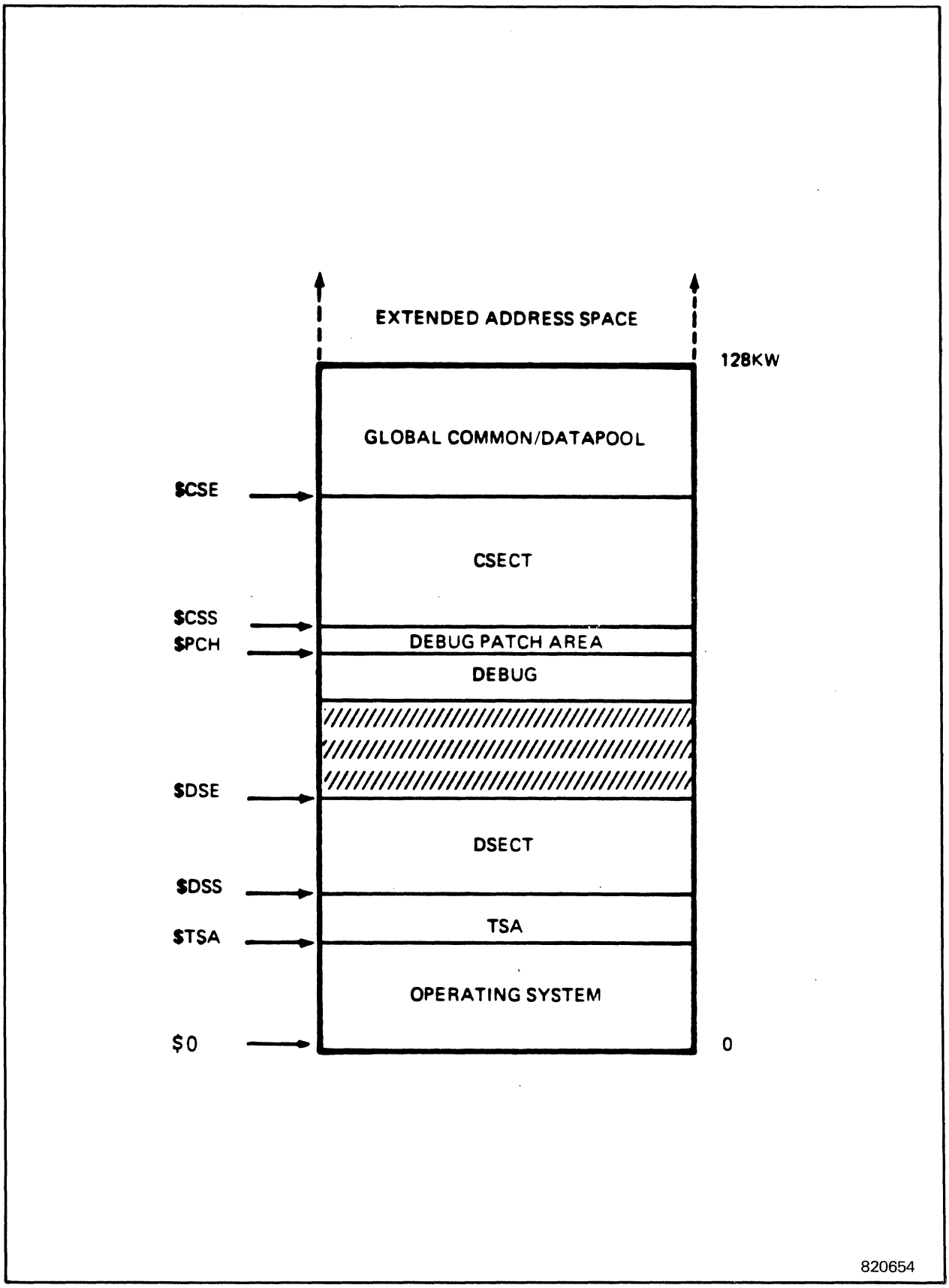


Figure 4-2. Debug Base Names

4.3.1.5 COUNT

The value of the special term COUNT is always the number of occurrences of the most recently-occurring trap since that trap was last set by the SET command. If no trap has occurred since DEBUG initialization, the value of COUNT is 0.

4.3.1.6 Operators

If a and b are expressions, then a#b is an expression where # is any of the DEBUG operators defined below.

Arithmetic Operators

x+y	the sum of x and y; overflow ignored.
x-y	y subtracted from x; overflow ignored.
x*y	x multiplied by y; overflow ignored.
x/y	x divided by y; remainder ignored.

Logical Operators

x ∧ y	x is logically shifted by y bits. The shift is to the left if y is positive and is to the right if y is negative.
x&y	x logically anded with y.
x!y	x inclusively ored with y.
x@y	x exclusively ored with y.

Relational Operators

The six relational operators yield a value of 1 if the specified relation is true, and a value of 0 if the relation is false. Comparisons are arithmetic; i.e., the 32-bit values being compared are assumed to be signed numbers.

x=y	x equals y
x > y	x is greater than y
x < y	x is less than y
x < > y	x is not equal to y
x > =y	x is greater than or equal to y
x < =y	x is less than or equal to y

4.3.2 Relative versus Absolute Expression Evaluation

An input expression in a DEBUG command can be used to represent an address as indicated in the command syntax. In these cases, an expression is evaluated as relative (i.e., the value of $DSS, the first word of DSECT, is automatically added to its value) unless it contains any base name outside of a memory content reference. (See Sections 4.3.1.3 and 4.3.1.4.)

All other input expressions are evaluated as absolute (i.e., no bias is added).

The same logic also applies to the expression within parentheses inside a memory content reference (i.e., such expressions represent addresses by virtue of their context).

Examples:

SET	1C00	1C00 evaluated as $DSS+1C00
N	1C00	1C00 evaluated as 7168
SET	$B+1C00	$B+1C00 evaluated as $B+1C00

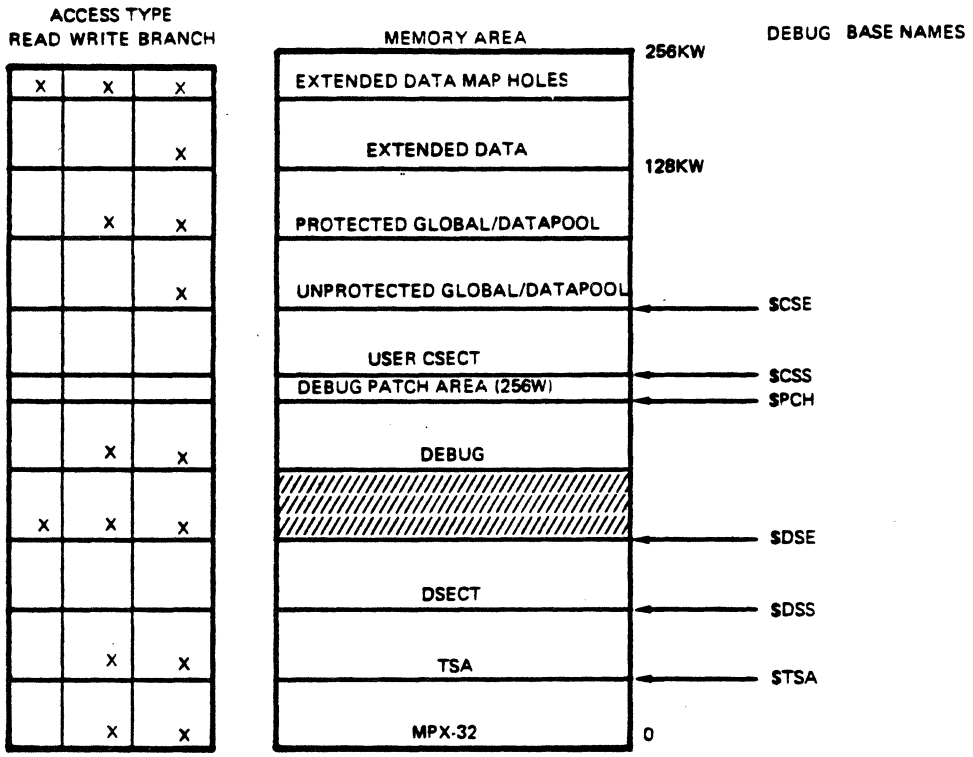
RELATIVE and ABSOLUTE commands may be used to override this expression evaluation logic. For example, if you use ABSOLUTE before SET 1C00, 1C00 would be the address 1C00, not $DSS+1C00.

4.3.3 Address Displays and References

When DEBUG displays a value which, by virtue of its context, represents a memory address (e.g., the address portion of the user PSD in a status report) it displays the value either as a five-digit hexadecimal memory address or as a five-digit hexadecimal displacement from some base address value, depending on the most recent ABSOLUTE or RELATIVE command.

4.3.4 Address Restrictions

When DEBUG encounters a value which, by virtue of its context, represents a memory address (e.g., the first argument of a DUMP command) it subjects the address to certain restrictions, as described in Figure 4-3. Any address violating the criteria of Figure 4-3 causes the current command to be terminated and a self-explanatory description of the violation to be displayed.



NOTATION:

X = PROHIBITED

ACCESS TYPES

READ: C( ) IN ANY COMMAND
DUMP
SCAN
SNAP

WRITE: CM

BRANCH: GO (BOTH ARGUMENTS)
KILL
SET
TRACE, TRACK, WATCH } BOTH ARGUMENTS AND ALL INSTRUCTION ADDRESSES

820655

Figure 4-3. DEBUG Command Address Restrictions

4.3.5 Traps and Trap Lists

The user sets traps in a task with the SET command. The user instruction at each trap address is replaced by DEBUG with an SVC 1,X'66' (trap) instruction, and the replaced instruction is saved by DEBUG. When the trap instruction is executed, DEBUG gains control and executes a trap list.

DEBUG locates the trap table entry which defines the trap and increments the trap's COUNT variable by 1. Then the stored commands comprising the trap list are executed (refer to the IF command description for a discussion of conditional execution of trap list commands). Any number of commands can be in the trap list. The commands are executed in the order they were added to the trap list. Any DEBUG commands can be used in a trap list, except LOG and REVIEW.

4.3.5.1 Setting a Trap

When the user issues an immediate SET command, DEBUG defers the execution of subsequent commands, storing them in the trap list for the trap that is being set.

Each deferred command is checked for validity before it is stored. A diagnostic message informs the user of any error in the command; the user can re-enter the command at his option. Note that expressions in deferred commands can contain references to user bases which have not yet been defined. This is the only 'error' allowed in a deferred command.

When the user enters a trap list terminator command which corresponds to the immediate SET command, DEBUG adds it to the trap list and stops building the trap list. Subsequent commands are immediate (not deferred).

Valid trap list terminators are BREAK, END, EXIT, FILE, GO, TRACE, TRACK and WATCH.

Sample Trap Lists:

```
. SET 100
..SNAP
..DUMP
..GO $
. SET 200
..SNAP
..DELETE $
. GO $
. GO 16
```

The user sets two traps, one at location 100 and the second at location 200, then uses GO to start execution at location 16. If the trap at location 100 is encountered in execution, control is returned to DEBUG. DEBUG displays a status report on the terminal and performs a snap and a dump. It executes the instruction replaced by the trap instruction and continues execution.

If the trap at location 200 is encountered, DEBUG displays a status report, performs a snap, and is terminated. The task continues to execute.

4.3.5.2 Nesting Trap Lists

Trap lists can be nested within a trap list. The user can set a second trap if the first trap is encountered, and so on, for as many trap lists as the user wants to nest.

DEBUG matches SET commands and trap list terminators as shown in Figure 4-4 to allow the nesting.

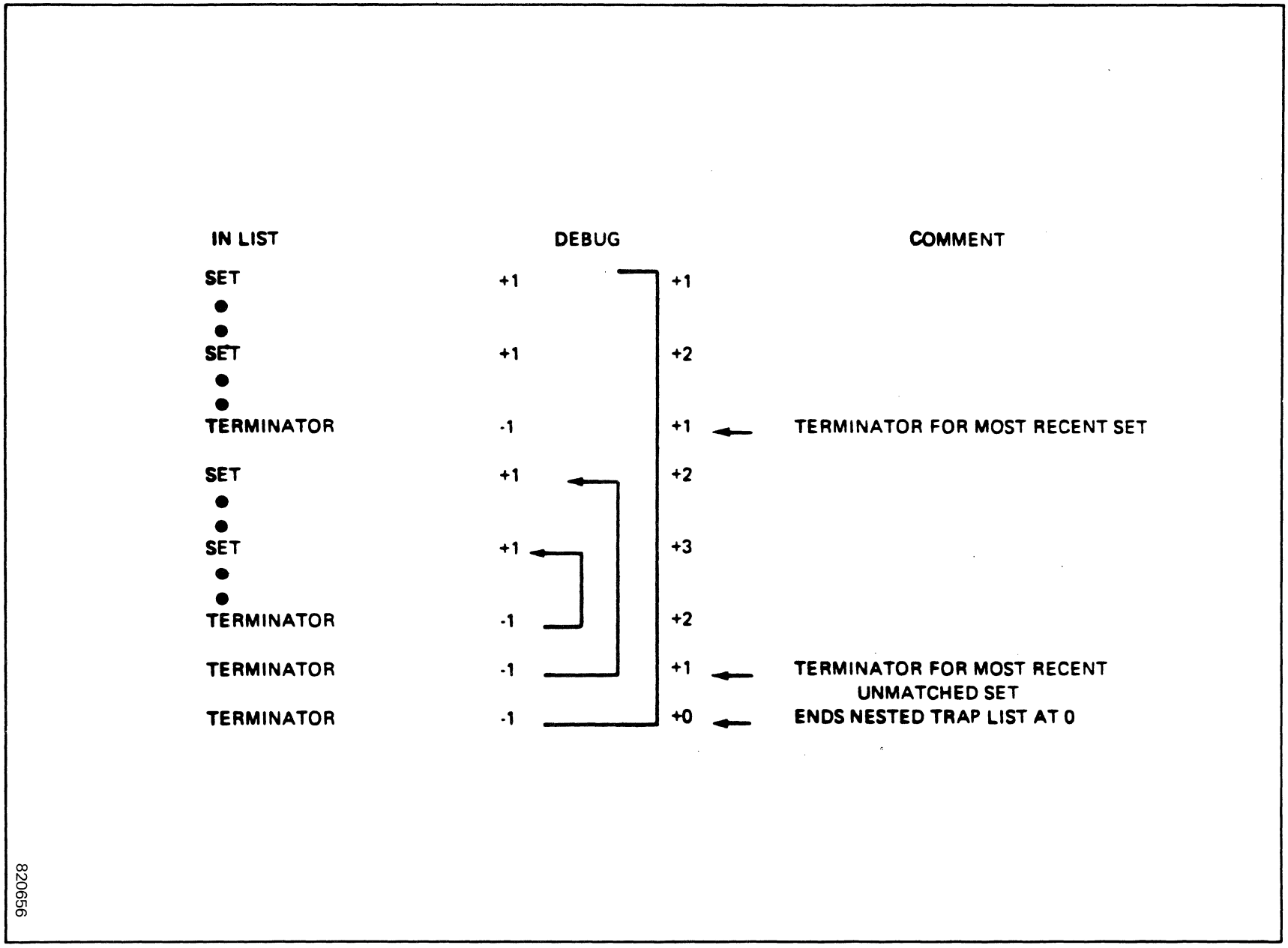


Figure 4-4. Nested Trap Lists

DEBUG increments a counter for each SET. Each terminator encountered matches the most recent unmatched SET and decrements the counter. When all SET's are matched, the trap list ends.

In the example which follows, the last trap list terminator command (TRACE) corresponds to the first trap set in the nested list, the second-to-last terminator (END) corresponds to the second trap set. An immediate prompt (.) is not issued until the user has supplied a terminator for each trap in the list.

Sample Nested Trap List:

```
. SET 100
..SNAP
..DUMP
..SET 200
..SNAP
..DUMP
..END
..TRACE
. GO $
```

If DEBUG encounters the trap at location 100, it displays a status report, performs a snap, a dump, sets the trap at location 200, and starts a trace. If it encounters the trap at location 200, it displays a status report, performs a snap, a dump, and transfers control back to the user at the terminal.

4.4 Accessing the Debugger

To access the Debugger from TSM, use:

TSM> DEBUG loadmod

The task is activated with DEBUG attached as an overlay. Control is passed to the Debugger instead of to the task's transfer address when activation is complete.

The Debugger prints the current date and time and a status report:

MPX-32 DEBUG date:time
(Status Report)

The DEBUG prompt for an immediate command (.) is then displayed:

.command

Enter a DEBUG command.

The Debugger can also be activated by coding a service call, M.DEBUG, in the task itself. The call causes DEBUG to be loaded into the address space of the task. Control is passed to DEBUG and DEBUG displays a message at the terminal as described above.

To access the Debugger for a task that has been activated via a TSM RUN command, depress the Break key. TSM responds:

*** BREAK *** ON taskname AT location
CONTINUE, ABORT, OR DEBUG? D

If you enter D (for DEBUG), the Debugger is loaded into the address space of the task, control is passed to DEBUG, and DEBUG displays a message at the terminal as described above.

To access the Debugger as part of a batch job, create a job file using the EDITOR, punch cards, or other media. Use $DEBUG loadmod instead of $EXECUTE. The job file can be submitted:

from the OPCOM console:

" < Attention > "

??BATCH { F,jobfile / D,devmnc }

from the OPCOM program:

```
TSM >OPCOM

??BATCH    {F,jobfile}
           {D,devmnc}
```

from the EDITOR:

```
TSM > EDITOR
      .
      .
      .
EDT >BATCH[jobfile]
```

If the job file is the current EDITOR work file, issue just the BATCH command. See Section 4.6 for considerations when running the Debugger in batch.

4.5 Commands

The following rules apply to DEBUG commands, whether they are input from the terminal or a job file (file code #IN) or from a file specified in a FILE command (file code #03).

Each command is contained entirely in a single 80-byte record. If #IN is assigned to a device or file with a smaller or larger record size, input commands are respectively blank-filled or truncated on the right hand end. There are no provisions for the continuation of commands or for compound commands.

The command verb starts in the first position of the line or record. All commands have a command verb except for the default form of the Expression command, in which a blank in the first column indicates the absence of a command verb. If a command can be abbreviated, the acceptable abbreviation is indicated in the syntax statement by underlining.

The command verb is followed by a terminator (any non-alpha character) and the argument list, if any. Multiple arguments are separated by commas (,). Extra blanks in the command line are ignored unless they are used inside a C or G character string.

Any of the command arguments in Table 4-3 can be specified by the user in the form of an expression, as long as the expression is valid and conforms to any further restrictions mentioned in the description of the command being used:

Arg	Command
reg	CR
value	CM, CR
addr	BASE, CM, GO, LIST
low	DUMP, SNAP
high	DUMP, SNAP
screens	REVIEW
start	TRACE, TRACK
stop	TRACE, TRACK
trap	DELETE, GO, SET
expr	expression
cond	IF

Table 4-3 Valid Use of Expressions

Only diagnostic messages originating from the command logic are mentioned in the command descriptions. Diagnostics arising from expression errors are covered in Section 4.8. Note that any diagnostic message incurred by a command signals the termination, possibly premature, of the command.

The RESPONSE section of each command description outlines DEBUG's response to the command when it is executed (i.e., immediate). For possible responses to deferred commands see Section 4.3.5, Traps and Trap Lists.

Even if no mention is made of batch differences in a command description any reference to the terminal is understood in batch to refer to the lfc #IN for input and #OT for output. See Section 4.2.1 for a description of I/O assignments in batch.

Command	Function
ABSOLUTE	Evaluates all input expressions as absolute and displays all output addresses as absolute until a RELATIVE command is issued.
BASE	Creates, deletes, or modifies the definition of a user base.
BREAK	Transfers control to user task's break receiver.
CC	Displays or modifies condition codes in user task's PSD.
CLEAR	Clears symbolic bases, or deletes all traps set in user task.
CM	Changes contents of memory beginning at specified address to new 32-bit value(s).
CR	Changes contents of user register(s) beginning at specified register to new 32-bit value(s).
DELETE	Deletes specified traps.
DETACH	Detaches DEBUG from the user task. DEBUG transfers control to the task at specified address or last address executed in the task.
DUMP	Dumps a snapshot of specified area of task's memory, including task's PSD and general purpose registers to dynamically created SLO file (interactive) or existing SLO file (batch).
END	Terminates a trap list and returns control to the terminal.
EXIT	Terminates both DEBUG and user task and returns control to TSM.

44	expression	Specifies an expression. DEBUG evaluates it and outputs its value on screen (interactive) or SLO (batch). No command word is used.
45	FILE	Reads records from specified command file. Reverts to terminal at end of file.
46	FORMAT	Sets default format for untyped numeric constants in input expressions to decimal or hex.
47	GO	Resumes execution of user's task at specified address or last known user program counter value. Optionally sets a one-shot trap at specified address. The trap is deleted automatically by DEBUG after it occurs.
49	IF	Used for conditional execution of trap lists.
50	LIST	Lists on UT (interactive) or SLO (batch); trap(s) set in task, with trap list(s).
50	LOG	Copies the log file of screen I/O to a dynamically allocated SLO file. Interactive use only. In batch, entire session already recorded on SLO.
51	MSG	Specifies a comment to output on terminal (interactive) or SLO file (batch).
51	RELATIVE	Displays and interprets logical addresses relative to a base.
52	REVIEW	Displays log file screens one at a time. Interactive use only.
52	RUN	Restarts run (as opposed to single-step) operation for TRACE or TRACK.
53	SET	Sets word address of an instruction to be trapped. User then creates trap list.
54	SHOW	Lists addresses of all traps, bases, or option settings.

55	SNAP	Dumps a snapshot of specified area of task's memory, including task's PSD and general purpose registers to terminal screen (interactive) or existing SLO (batch).
55	STATUS	Displays a status report of user PSD and registers for each currently active task interrupt level.
56	STEP	Single-steps subsequent TRACK or TRACE.
56	TIME	Displays current date and time.
57	TRACE	Transfers control to user task and displays each instruction after it is executed.
60	TRACK	Transfers control to user's task and displays each branch instruction after it is executed.
61	WATCH	Like TRACE, but does not display instructions. Detects erroneous branches into areas such as the MPX-32 operating system.

4.5.1 The ABSOLUTE Command

The ABSOLUTE command is used to:

evaluate all input expressions as absolute until a RELATIVE command is executed

display all addresses as absolute hexadecimal logical addresses until a RELATIVE command is executed

See Section 4.3.2.

Syntax:

ABSOLUTE

Response:

The command is always valid.

No output.

DEBUG prompts for the next command.

4.5.2 The BASE Command

The BASE command is used to:

define a user base (add its name to the internal base definition table)

delete a user base name from the base table

redefine a user base (change the value specified in the base name's definition)

Up to 16 user bases are allowed. See Section 4.3.1.4.

Syntax:

BASE base[,addr]

where:

base — is a user base name. Must begin with the character $ and an alphabetic character. Can be up to eight alphanumeric characters maximum.

addr — supplies a logical address for a base. If no address is used, deletes specified base name. If "addr" is specified and "base" is already defined, "base" is redefined to represent "addr".

Response:

No output except diagnostics. Diagnostic messages inform the user if:

the user tries to define a new base and the base table is full (16 user bases)

"base" is not specified

"base" is a DEBUG base name

the user attempts to delete an undefined base

4.5.3 The BREAK Command

The BREAK command is used to transfer control from the Debugger to the user task's break receiver.

Syntax:

BREAK

Response:

The user break receiver gets control. DEBUG regains control upon the occurrance of the next break, trap, user abort, or break receiver exit.

No output except diagnostics. A diagnostic message informs the user if the user task has no break receiver.

The BREAK command is a trap list terminator.

4.5.4 The CC (Condition Code) Command

The CC command is used to display the four condition code bits in the DEBUG base $PSD (bits 0-31 of the user PSD) or to display the old condition code of $PSD and insert a new value.

Syntax:

CC [cc]

where:

cc is a string of four binary digits. If used, replaces the existing condition code in $PSD.

If no value is specified, DEBUG displays the present condition code.

Response:

A diagnostic message informs the user if the condition code is specified incorrectly.

DEBUG prompts for the next command.

4.5.5 The CLEAR Command

The CLEAR command is used to delete all user base definitions or delete all traps.

Syntax:

CLEAR { BASES / TRAPS }

where:

BASES indicates that all user base definitions are to be deleted.

TRAPS indicates that all traps are to be deleted.

Response:

A diagnostic message informs the user of any argument specification errors.

No output except for diagnostics; DEBUG prompts for the next command.

4.5.6 The CM (Change Memory) Command

The CM command is used to alter the contents of one or more consecutive words in the task's logical address space.

Syntax:

CM addr=value[,value],...

where:

addr specifies the address of the first or only word to be changed (bits 30 and 31 of addr are ignored and assumed to be 00).

value is the 32-bit value to be stored at the specified address. Successive values are stored in consecutive words beginning at "addr". Two consecutive commas with no intervening value can be used to skip the memory address corresponding to the missing value, leaving its contents unchanged.

Response:

Diagnostic messages inform the user if:

"addr" and "value" are not both present and valid

memory changes must be stopped because "addr" or an address derived from it (multiple values) violates a DEBUG address restriction, or because an error occurs in evaluating one of the "value" expressions.

Note that in the second case, the diagnostic message will make clear which memory words, if any, were successfully changed.

A SNAP is performed for the modified range and the new contents are displayed.

DEBUG prompts for the next command.

4.5.7 The CR (Change Register) Command

The CR command is used to alter the contents of one or more user registers.

Syntax:

CR Rn=value[,value],...

where:

Rn is a number in the range 0-7, specifying one of the user registers R0-R7.

value is the 32-bit value to be stored in the specified register. Succeeding value's, if any, are stored in consecutive user registers. Two consecutive commas with no intervening value can be used to skip the user register corresponding to the missing value, leaving its contents unchanged. If user register R7 has been altered or skipped and one or more unused value's remain, they are ignored.

Response:

A diagnostic message informs the user if: (1) a register specification is absent or not in the range 0-7; or (2) the first value is not specified.

DEBUG prompts for the next command.

4.5.8 The DELETE Command

The DELETE command is used to delete specified traps and restore replaced user instructions to their original locations.

Syntax:

DELETE trap

where:

trap is a trap address.

Response:

A diagnostic message informs the user if:

no "trap" is specified

"trap" is not an address at which a trap has been set by the SET command

Note that in the second case, the diagnostic message clarifies which address is not a trap address by displaying an exclamation point below the incorrect expression; any traps specified to the left of the incorrect expression will have been deleted.

As each trap is deleted, the user instruction replaced by the trap instruction is restored to its original location.

DEBUG prompts for the next command.

4.5.9 The DETACH Command

The DETACH command is used to detach DEBUG from the user task and transfer control to the task at the specified address or at $ (bits 13-31 of user PSD).

Syntax:

DETACH [addr]

where:

addr is the address within the user task to which control is transferred. If not specified, defaults to $.

Response:

All traps are deleted (there is no need to enter CLEAR TRAPS to restore user instructions replaced by trap instructions).

DEBUG files and memory are deallocated.

DEBUG transfers control to the specified address.

A diagnostic message informs the user if the specified address violates a DEBUG address restriction.

DETACH is a trap list terminator.

4.5.10 The DUMP Command

The DUMP command is used to output a range of memory on an SLO file.

ASCII format is used for the right hand side of the memory display.

On line: the dump is output to a dynamically created SLO file, using file code #02.

Batch: the dump is output to file code #0T.

Syntax:

DUMP [low[,high]]

where:

low and high — are expressions representing memory addresses. If "high" is not specified or is not greater than "low", only the single word at "low" is displayed.

If no addresses are specified the entire user task area is displayed. Bits 30 and 31 of the values of "low" and "high" are zeroed to produce word addresses.

Response:

The memory range between the addresses "low" and "high" is output. The user PSD and registers are also shown.

A diagnostic is displayed if any address in the range violates an address restriction.

4.5.11 The END Command

The END command is used to terminate a trap list. Using just a carriage return performs the same function.

Syntax:

END or <CR>

Response:

END is a trap list terminator.

4.5.12 The EXIT Command

The EXIT command is used to terminate debugging and return to the TSM> prompt. Both the user task and DEBUG exit.

Syntax:

EXIT

Response:

DEBUG calls the M.EXIT service after verifying that the user desires to exit the system (return to TSM) and determining whether a hard copy of the log file is desired.

EXIT is a trap list terminator.

In batch, if EXIT is used and/or end-of-file is encountered on #IN, DEBUG processing terminates.

4.5.13 Expression Command

The Expression command is used to display the value of an input expression.

Syntax:

$$\left\{\begin{matrix}\underline{A}\\ \underline{B}\\ \underline{N}\\ \underline{X}\end{matrix}\right\} \text{expr}$$

where:

A displays the low order 19 bits of "expr" as an address. The data are displayed as absolute or relative, based on the most recent ABSOLUTE or RELATIVE command.

B displays "expr" in binary

N displays "expr" as a signed decimal number

X displays "expr" in hexadecimal

expr is any expression

Response:

A diagnostic message informs the user of any error encountered in evaluating the expression and displays an exclamation point under the invalid term or operator.

After displaying the value of the expression, DEBUG prompts for the next command.

4.5.14 The FILE Command

The FILE command is used to input subsequent DEBUG commands from a command file instead of from the terminal.

Syntax:

FILE filename[,password]

where:

filename is the name of a command file on disc.

password is the password, if any, associated with the file.

Response:

A diagnostic message informs the user if:

"filename" is absent or invalid, or the file does not exist

the password is invalid

the user is not allowed access to the file, e.g., a password is associated with the file and has not been supplied

the command is read from a command file

If there are no errors, DEBUG assigns lfc #03 to the specified file and reads subsequent commands from #03 instead of #IN. When DEBUG reaches EOF on #03 or a break is recognized, command input reverts to #IN. The user name, if any, stored in T.USER in the task's TSA is used to access the command file.

Use of the FILE command terminates a trap list.

4.5.15 The FORMAT Command

The FORMAT command is used to set the default (assumed) format for untyped numeric constants in expressions to hexadecimal or decimal.

Syntax:

FORMAT $\left\{\begin{matrix} X \\ N \end{matrix}\right\}$

where:

X sets the input radix to hexadecimal, which is the original default when DEBUG is attached.

N sets the input radix to decimal.

Response:

A diagnostic message informs the user if the format specification is absent or invalid.

DEBUG prompts for the next command (no output).

4.5.16 The GO Command

The GO command is used to transfer control to the user task, optionally setting a one-shot trap.

Syntax:

GO [addr][,trap]

where:

addr — is the address within the user task to which DEBUG transfers control. If "addr" is not specified, the DEBUG base $ (bits 13-31 of the user PSD) is used.

trap — is the address within the user task at which DEBUG sets a one-shot trap. The trap is defined by the following trap list:

```
MSG**ONE-SHOT SET BY GO COMMAND**
DELETE trap
```

where "trap" is the address of the one-shot trap, displayed by DEBUG as a hexadecimal number. If a trap address is not specified by the user in the GO command, DEBUG does not set a trap before transferring control to the user task.

Response:

A diagnostic message informs the user if:

either the transfer address or trap address violate DEBUG address restrictions

a trap address is specified and a trap is already set there

no trap table space remains and a trap address is specified

the specified transfer address is two bytes greater than any trap address

"addr" is an odd number

"trap" is not on a word boundary

In any of these cases, the GO command is not executed.

If GO is successful, DEBUG transfers control to the user task at the specified address. If the last control transfer into DEBUG was caused by a trap and control is passed to the trap address for that trap, the user instruction replaced by the trap instruction is executed first. Control is then passed to the trap address plus one word unless the replaced user instruction is any instruction which terminates the TRACE, TRACK, or WATCH commands--such a replaced instruction may not be executed without first deleting the trap set on it.

Control remains with the user task until a trap, break, or user abort occurs, whereupon DEBUG regains control and reads the next immediate command from the terminal or a trap list as appropriate.

GO is a trap list terminator.

4.5.17 The IF Command

The IF command is used to make a trap list conditional, i.e., the trap list is executed only if specified conditions are met.

Syntax:

IF cond

where:

cond is any expression.

Response:

If the value of "cond" is 0, the trap list which follows the IF is not executed; otherwise it is executed. When "cond" is 0, the COUNT for the trap is incremented, the user instruction replaced by the trap instruction is executed, and control is passed back to the user task as in a "GO $" command.

When "cond" is nonzero, the occurrence of the trap is reported, the trap's COUNT is incremented, and DEBUG executes the remaining commands in the trap list.

A diagnostic message informs the user when IF is entered as an immediate command or when "cond" is absent or invalid.

The expression "cond" is evaluated. If the value is nonzero, the trap is reported and remaining commands in the trap list are executed. The relational operators listed in Section 4.3.1.6 produce a value of 1 if the relation is true, and a value of 0 if false.

The trap's COUNT is incremented whether the trap is reported or not.

A trap list may contain at most one immediate IF command. When IF is present, it must be the first command of the trap list.

If the value of "cond" is zero no trap is reported and the program continues executing as if the user issued a GO $ command.

4.5.18 The LIST Command

The LIST command is used to display the trap list for a specific trap.

Syntax:

LIST trap

where:

trap specifies a trap address.

Response:

A diagnostic message informs the user if "trap" is not a trap address.

4.5.19 The LOG Command

The LOG command is used to print the log file. For log file description, see Section 4.1.2.1.

Syntax:

LOG

Response:

All log file records which have not already been printed are copied to an SLO file. The SLO file is then closed and deallocated. All log file records thus copied are no longer accessible (their space is released). The LOG command is ignored in batch.

A diagnostic message informs the user if LOG is entered as a deferred command.

4.5.20 The MSG Command

The MSG command is used to display a message.

Syntax:

MSG message
*

where:

message is any character string.

Response:

The character string is displayed. MSG can be used, for example, on a command file or in a trap list.

4.5.21 The RELATIVE Command

The RELATIVE command is used to input and display adresses as they are relative to a base.

Syntax:

RELATIVE [base]

where:

base is a base name. If a base is not specified, the base specified in the last RELATIVE command is used. If no base has been specified in a RELATIVE command, the special base $DSS (DSECT start) is used.

Response:

Each address that is displayed is represented as a displacement from the nearest base which is not greater than the address.

Each relative input address is biased with the value of the the specified base.

A diagnostic informs the user if the specified base has not been defined.

4.5.22 The REVIEW Command

The REVIEW command is used to display the log file at the terminal.

Syntax:

REVIEW [screens]

where:

screens — is the number of screens from the current position in the log file for DEBUG to backspace before beginning the log file display. If the number of screens is not specified or is greater than the number of screens currently contained in the log file, the display begins at the first record in the log file.

Response:

DEBUG displays the log file one screen at a time.

When DEBUG reaches the end of the log file, the display is terminated and DEBUG prompts for the next command. None of the above terminal I/O is copied to the log file.

REVIEW is treated as a comment in batch.

A diagnostic message informs the user if:

REVIEW is entered as a deferred command

REVIEW is read from a command file

4.5.23 The RUN Command

The RUN command is used to trace or track until DEBUG reaches a full screen of output instead of single-stepping.

Syntax:

RUN

Response:

Until a STEP command is executed, the TRACE and TRACK commands run to a full screen of output before pausing.

4.5.24 The SET Command

The SET command is used to set a trap in the user task.

Syntax:

SET trap

where:

trap is the address at which DEBUG sets a trap.

Response:

The user instruction at the specified trap address is replaced by a trap instruction (SVC 1,X'66').

DEBUG stores subsequent commands in the trap list (i.e., subsequent commands are deferred) until a trap list terminator is read which corresponds to this SET. Trap list terminators and their interaction with the SET command are discussed in Section 4.3.5, Traps and Trap Lists.

A diagnostic message informs the user if:

The trap address is absent, is already a trap address, or violates an address restriction.

DEBUG's trap table is full and thus no more traps can be set until a trap is deleted.

"trap" is not on a word boundary.

4.5.25 The SHOW Command

The SHOW command is used to display current base definitions, trap addresses, or option settings.

Syntax:

SHOW [BASES | TRAPS | OPTIONS]

where:

If no argument is specified, all displays are produced.

BASES displays the current definitions of all special bases and user bases.

TRAPS displays all trap addresses.

OPTIONS displays the settings of the options controlled by the following commands:

ABSOLUTE/RELATIVE
RUN/STEP
FORMAT

Response:

A diagnostic message informs the user if any argument but BASES, TRAPS, or OPTIONS is used.

4.5.26 The SNAP Command

The SNAP command is used to output the contents of a range of logical addresses on the file or device assigned to #OT. The output format is side-by-side hexadecimal and ASCII.

Syntax:

[SNAP][low[,high]]

where:

If a range of addresses is not supplied, DEBUG snaps the logical address space of the task from $DSS to $DSE.

low specifies the first address to snap. If no address is specified, the snap begins at $DSS. Bits 30 and 31 are ignored and assumed to be 00.

high specifies the last address to snap. If not specified, only the single word at the low address is snapped. Bits 30 and 31 are ignored and assumed to be 00.

Response:

The specified memory contents are output to #OT.

4.5.27 The STATUS Command

The STATUS command is used to display a status report indicating the user PSD and user registers for each currently active task interrupt level.

Syntax:

STATUS

Response:

DEBUG displays a status report on the terminal.

4.5.28 The STEP Command

The STEP command is used before TRACE or TRACK to single-step through the execution of each instruction in the user task.

Syntax:

STEP

Response:

Until a RUN command is issued, all TRACE and TRACK commands will pause after each instruction display so the user can inspect each instruction and its results before the next instruction is executed.

STEP is ignored in batch.

4.5.29 The TIME Command

The TIME command is used to display the date and time of day.

Syntax:

TIME

Response:

DEBUG displays the calendar date as stored in the Communication Region (C.DATE) and the time of day as returned by the M.TDAY service.

4.5.30 The TRACE Command

The TRACE command is used to execute and display each user instruction and its results. To trace only branching instructions, use the TRACK command.

Syntax:

TRACE [start][,stop]

where:

start — is the address of the first user instruction to be executed. If starting address is not specified, the special base $ (bits 13-31 of the user PSD) is used.

stop — is the address of the last user instruction to be traced. If a stop address is not specified, the trace continues as described below.

Response:

DEBUG fetches user instructions beginning at the specified start address and executes them, displaying each instruction and some of its results and/or operands in a basic Assembler-like format.

Unless a RUN command is in effect, DEBUG pauses after each instruction is executed or simulated, if possible, and waits for a 1-character response from the user. To proceed to the next user instruction, enter only a carriage return. Any other response terminates TRACE. If a RUN command is in effect, TRACE does not pause after each instruction but proceeds immediately to the next instruction; thus the only opportunity to stop the display is at the end of each screen. Note that in batch, TRACE functions as if a RUN command were in effect.

This process continues until one of the following occurs:

An instruction has been fetched, executed, and displayed from the specified stop address. The user context indicates that the instruction has been executed, as shown in the status report announcing trace termination.

A user instruction is aborted, e.g., by a privilege violation or a map fault. TRACE executes most user instructions by transferring control to the user task for one instruction at a time. When these instructions execute, it is as if the user had entered "GO a,b" where "a" is the address of an instruction and "b" is the address of the next instruction (logically next, not necessarily "a"+1W). Any abort condition caused by such instructions is reported as it would be after a GO command and the trace is terminated. The user context is reported in a status report.

DEBUG fetches an instruction that breaks the trace (see Table 4-4). The instruction is displayed and TRACE is terminated. The user context still points to the untraceable instruction, as shown in the status report announcing trace termination.

The address of the next instruction to be fetched would violate an address restriction. No instruction is displayed, the trace is terminated, and the user context points to the bad address as shown in the status report announcing trace termination.

If the last control transfer to DEBUG is caused by a trap, and the starting address is $ (the user PSD), the user instruction replaced by the trap instruction at $ is traced as if it were at $, and the trace continued.

A diagnostic message informs the user if the starting address violates an address restriction or is two bytes greater than any trap address or "start" or "stop" is an odd address.

TRACE is a trap list terminator.

Table 4-4 Instructions that Break a Trace

AI
BEI
BRI
CD
CEMA
DAI
DI
EI
EWCS
HALT
JWCS
LEM
LPCM
LPSD
RDST
RI
RWCS
SEM
SCPU
TD
TMTR
TPR
TRP
UEI
WAIT
WWCS
All undefined opcodes

4.5.31 The TRACK Command

The TRACK command functions exactly like TRACE, except that it displays only branching instructions.

Syntax:

TRACK [start][,stop]

where:

start — is the address of the first user instruction to be executed. If the starting address is not specified, the special base $ (bits 13-31 of the user PSD) is used.

stop — is the address of the last user instruction to be executed. If a stop address is not specified, the track is continued as described for TRACE.

Response:

TRACK functions exactly like TRACE with the following exception:

Only instructions which can cause branching are displayed (BCT, TRSW, LPSD, etc.), and they are listed with an indicator, where appropriate, showing whether a conditional branch is being taken.

4.5.32 The WATCH Command

The WATCH command functions like TRACE, but does not display instructions. It is used to detect erroneous branches into areas such as Global Common or MPX-32.

Syntax:

WATCH [start] [,stop]

where:

start is the address of the first user instruction to be executed. If a starting address is not specified, the special base $ (bits 13-31 of the user PSD) is used.

stop is the address of the last user instruction to watch. If a stop address is not specified, the watch continues as described below.

Response:

DEBUG performs a TRACE but inhibits the usual instruction display. When, as often happens in a new program, an erroneous branch is taken, it is often into an area completely out of the program (e.g., a branch to location 0). Especially in the case of a privileged task, many instructions may precede the inevitable disaster. While the system crumbles, many of the most useful hints as to the cause (e.g., register contents) are destroyed. WATCH provides a convenient means of detecting such branches when they happen without all the terminal output caused by TRACE or TRACK.

4.6 Batch Considerations

This section parallels the other sections in this chapter, outlining the major differences which arise when DEBUG is attached to a batch task. The differences between interactive and batch functions for specific commands are discussed in the individual command descriptions in Section 4.5. File assignments are covered in Section 4.2.

Section	Batch Considerations
4.4	DEBUG may be attached at activation time to a batch task by using the Job Control command $DEBUG where $EXECUTE would normally be used.
4.1.2.1	Terminal I/O: Screen size is meaningless. There is no log file; input commands are read without prompts, and pseudo-prompts are added as for commands from a command file.
4.1.2.3	SLO files: The DUMP command does not produce a separate SLO file, but places its output on the SLO file allocated for the file code #OT; the LOG command is treated as a comment in batch since there is no log file.
4.1.4	DEBUG recognizes breaks during batch processing, but since there is no terminal, the only possible sources are the OPCOM BREAK command or an SVC 1,X'66' instruction coded by the user as part of his task (i.e., not planted by the SET command). Since in batch, the command stream is not interactive, but must be composed before execution, the user abort and break interception provided by DEBUG must be used with care.

4.7 Listings and Reports

See individual command descriptions.

4.8 Errors

See individual command descriptions and Appendix C.

4.9 Examples

Not supplied.

5. THE TEXT EDITOR (EDIT)

The MPX-32 Text Editor (EDIT) provides a comprehensive set of commands for building and editing text files, merging files or parts of files into one file space, copying existing text from one location to another, and in general for performing editing functions familiar to users of interactive systems.

EDIT is typically used to create source files and to build job control files and general text files. A job file built in the Editor can be copied directly into the batchstream using the Editor BATCH command.

5.1 General Description

Edit structure is based on the concept of a work file upon which the user operates with editing commands. Source text may be transferred between permanent disc files and the work file. Access to source text is based on line numbers contained within text lines.

The Text Editor only recognizes file names with 1-8 characters. Valid characters for file names are A-Z, 0-9, dot (.) and underscore (_). Although other characters will generally be accepted, their use is not recommended. Filenames should not begin with a dot or string of digits followed by a dot. Filenames must contain at least one alphabetic character.

5.2 Files and File Assignments

Not applicable.

5.3 Options

The TSM command OPTION LOWER can be used to accommodate a terminal that is entering upper and lower case text. See Section 5.4.6, Entering Lower Case Text. There are no other options that affect the use of EDIT.

5.4 Using the Editor

5.4.1 Addressing Techniques

There are several ways of specifying what lines to work on with an EDIT command, ranging from supplying a specific line number to specifying a group of lines and ranges of lines. The following sections describe various addressing techniques.

5.4.1.1 Special Characters

Several characters are available to use in place of a line number (these apply to workfile line numbers only):

F or FIRST	the first line in the file
L or LAST	the last line in the file
E or END	the last line in the file (L) plus the current increment set for adding lines at the end of the file (DELTA, normally 1.)
A or ALL	all lines in the file
C or CURRENT	the last line currently displayed on the terminal
N or NEXT	the line following the current line

When referring to external files, specific line numbers should be stated to avoid erroneous results, i.e., 200/300 not 200/E.

Because these characters and words are EDIT keywords, they cannot be used as file names.

5.4.1.2 Line and Range Addressing

To access a specific line, type the line number. To access a contiguous set of lines (range), type the first line number, a forward slash, and the last line number in the range, i.e.,

lineno

or

lineno/lineno

The characters "F", "L", or "E" can be used in place of a specific line number. The range implied by ALL is F/L. Usage of "C" and "N" as part of a range is not permitted and will cause incorrect results. These special characters are intended to be used only to display a single line.

When only part of a range is specified with an EDIT command, the rest of the range is implied. If you use a beginning line number followed only by a forward slash:

lineno/

the last line in the range defaults to the end of the last range specified in the previous command. Likewise, if you use just a forward slash and the last line number for a range specification:

/lineno

the first line number defaults to the first line number from the last range specified with the previous command. See also Section 5.4.1.5.

5.4.1.3 Groups

A group is any combination of line numbers and ranges, where each specification is separated from the next by a comma, e.g.,

lineno/lineno, lineno, lineno/lineno

There can be up to 24 specifications in a group. The above illustration shows three.

5.4.1.4 Content Identifiers

To limit access within a group to lines containing a specific string, type the string enclosed in backslashes, i.e.,

string

Only the lines containing the specified string are then accessed. A content identifier used with a group applies to the entire group, e.g.,

1/60, TRAP ,75, 209/L

specifies that all lines within the group that contain the content identifier TRAP are to be accessed. This applies to lines 1-60, line 75, and line 209 through the last line of the file.

5.4.1.5 Defaults

When no lines are specified with an EDIT command, the lines specified with the previous EDIT command are used by default, with the following exceptions:

- o If a group was specified previously, only the last line or range in the group (the specification to the right of the last comma) is taken for the current command.
- o Content identifiers do not carry over. If you have selected lines within a range in the previous command by using a content identifier, you will access the entire last range in the group without regard to the content identifier when you specify nothing.

5.4.1.6 Special Command Defaults

Several EDIT commands have special defaults:

- o Initially the default for COLLECT and INSERT is the line following the last line of the file (E). After that point, COLLECT and INSERT default to the line following the last line collected or inserted as the point to begin collecting or inserting lines. (When LIST follows one of these commands and no group is supplied, it defaults to the lines that have just been collected, or inserted.)

- o COPY and MOVE also have special defaults. If nothing is specified, they default to the first through last lines specified or implied in the last range of the previous command as source text and the line following the last line in the file (E) as the point where source should be copied or moved.

LIST, CHANGE, APPEND, REPLACE, PREFACE, and DELETE initially default to the first line of the file. After that, defaults are geared to the specification for the previous command as described previously in Section 5.4.1.5.

5.4.1.7 Description in Syntax Sections

The syntax description for each command in Section 5.6 uses the term 'group' as a means of flagging any of the addressing techniques described in Sections 5.4.1.1 through 5.4.1.4. Defaults are reiterated with each command description.

In general the various arguments associated with any command may be entered in any order.

5.4.2 Lines and Line Numbers

Line numbers take the form of decimal numbers in the range zero (0) to 9999.999, with at most three digits after the decimal point. When creating new text, it is usually convenient to use only integer line numbers, reserving fractional line numbers for subsequent insertions.

The user work file is limited to 10,000,000 lines. This is the actual limit if the user sequentially numbers lines 0.000, 0.001,...,9999.999, inclusive. Otherwise, the work file is limited by the highest line number.

5.4.2.1 Line Numbers Generated by the Editor

When the Editor generates line numbers, e.g., with the COLLECT, MOVE, COPY, or INSERT command, it does so according to what the user specifies for a beginning line number (base) and optional increment. The following rules apply:

- The least significant decimal position used in the base (or increment) applies to line numbers that are generated by the Editor. A specification in tenths implies lines 1 to .9 or some subset thereof. A specification in hundreths implies lines .01 to .99 or some subset thereof. A specification in thousandths implies lines .001 to .999 or some subset thereof.

- The Editor stops the current operation if it encounters an existing line number. As long as no existing lines are encountered, the Editor rolls over line numbers until it reaches an existing line number.

- An increment is an absolute number to add to the previous line number to obtain the next line number. It is not automatically relative to the base, i.e., if you specify a base (or beginning) line number in tenths, and want to increment in tenths, the increment must reflect the base decimal position. For example, if you specify line 2.2 as the starting line and specify .2 as an increment, the Editor generates lines 2.2, 2.4, 2.6, and 2.8. If instead, you specify 2 as an increment, the next line number generated after 2.2 will be 4.2. The default increment used in generating line numbers is 1, .1, .01, or .001 depending on the least significant position of the specified base number.

- DELTA increments apply to lines following the last line of the workfile.

- a special DELTA increment (1/10 of DELTA value) is used by the Editor to generate line numbers for COLLECT and INSERT commands when the line number supplied (or implied) by the user already exists.

5.4.2.2 Line Numbers at the Beginning and End of the Workfile

Unless otherwise specified, when you begin building a workfile, the Editor defaults to line 1 for beginning of file. Lines 0 - 0.999 are, however, valid line numbers and can be used subsequently to insert lines before line 1.

The special character 'E' provides access to the end of the workfile, as it specifies the last line in the file plus an increment. The line number for E depends on the setting of DELTA. DELTA is normally an increment of 1 over the last line in the file. It can be overridden to a different significant digit and increment (e.g., .02) by using the SET DELTA command.

5.4.2.3 Physical Position of Line Numbers

The Editor displays line numbers at the beginning of the line, but it does not store, save, or output them there. The line numbers are physically output in columns 73-80 of each line of text, so that when another program reads the file, it can ignore the line numbers.

For example, when any processor is run, it requires that a directive, a label, or some other significant command 'verb' begin in position 1 of a line. (In some cases, a symbol such as a slash precedes the verb; in this description and throughout the documentation, the symbol is considered an integral part of the verb.)

Because the Editor moves the line number to the end of each line, the verb falls in the correct position (column 1) for processing in the batch environment.

5.4.2.4 Text Output Without Line Numbers

The workfile can be listed or output without line numbers if desired for readability or required for subsequent processing by a user-developed task. No SYSTEMS processors require removing line numbers.

Physically, using UNN as an option on a command replaces the Editor line number in columns 73-80 with blanks.

5.4.3 Accessing Files Created Outside the Editor

The USE command is normally used before performing any other editing functions on these files. If a disc file being copied into the workfile via USE has records longer than 80 characters, the records are truncated to 80 characters. In addition, editing requires valid line numbers in positions 73-80 on files coming in from disc. If the line numbers do not exist, data in positions 73-80 are replaced by Editor line numbers. The line numbers generated for the records reflect the current value of DELTA, i.e., if DELTA is set at the default, numbers are provided starting at 1.0 and incrementing by 1.

If an unnumbered file is accessed that has not previously been edited, and it has more than 9999 records, records 10,000 and subsequent are ignored. DELTA can be reset so that up to 10,000,000 records are copied into the workfile. See the SET DELTA command.

To get a file from media other than disc into the Editor, use another utility, e.g., MEDIA, to copy it to disc prior to issuing the Editor USE command. (See Volume 2, Chapter 8.)

5.4.4 Accessing Password-Protected Files

Any file that is defined as RO (Read Only) protected can be read without supplying the password. Commands that read files include LIST, PRINT, PUNCH, USE, COPY, and BATCH. Read only files cannot, however, be written to without supplying a valid password. Commands that 'write' files include SAVE, SCRATCH, and STORE. A file that is PO (Password Only) protected cannot be read or written without supplying a valid password.

The Editor honors these rules by prompting for a password in any of the cases where it is required:

ENTER PASSWORD password

If the user does not type a valid password and the command (e.g., LIST, PRINT, etc.) involves a read, the command terminates. If the command (e.g., SAVE) involves a write, the prompt is returned until the user enters the valid password or types just a CR , in which case, the command terminates.

Password protection can be achieved via the Editor when performing a SAVE or STORE command and specifying the RO or PO option. See SAVE and STORE command descriptions. If the RO or PO option is not used, the password is retained when Editor STORE or SAVE commands are used.

5.4.5 Accessing System Files

A system file (one without a username) can be edited by any user who knows the password associated with the file, if any. Any user can create a system file by using the SYS keyword on a STORE or SAVE command. Password protection is described in the previous section.

5.4.6 Entering and Editing Upper/Lower Case Text

The Editor normally translates lower case input to upper case. TSM provides a command that inhibits the translation:

```
TSM > OPTION LOWER
```

This allows the user to enter and edit lower case characters.

When entering upper/lower case text, depress the shift key for upper case characters. Commands and keywords as well as text can be entered in any combination of upper and lower case; however, the names of files must be all upper case.

Files that are built with upper/lower case text must be edited with the same character conventions in effect. For example, a content identifier in all upper case will not match text with the same characters if any of the text characters are lower case.

5.4.7 Using the Break Key

The Editor responds to the Break key, which provides a convenient means of terminating a long display once you have the information you need, or stopping a global change or large set of deletions in process. Response to the break is not guaranteed in a specific timeframe. When the break interrupt is received, the command in process is terminated at the earliest safe termination point.

5.5 Accessing EDIT

To access the Editor use:

TSM > EDIT

The Editor prompts for a work file code:

ENTER WORK FILE CODE OR CR TO TERMINATE: filecode

The file code is a two-character prefix for the work file. Each character must be printable. If the code you supply has been used previously in accessing EDIT, the existing workfile with that prefix is retrieved with a message indicating whether it was cleared, saved, or changed (without a save) at the end of the last EDIT session. The EDT prompt is then displayed.

If the code you supply has not been used before, EDIT assumes that you want to create a new work file. It creates the file for you.

To exit the Editor, type a carriage return.

You can bypass the prompt for a work file code by entering the code when you access the Editor. A blank or comma is the legal delimiter, i.e.,:

TSM > EDIT filecode

Note: The special characters, the commands, and the keywords within commands cannot be used as a file name. Also, numeric names are not permitted.

5.6 EDIT Commands

EDIT commands are summarized in the following chart and described individually in sections which follow. EDIT commands must either be abbreviated to three characters or completely spelled out.

To delete one or more character(s) on a command line or on a line of text being edited, use a Backspace or a CTRL H key sequence. Type the right character(s).

To delete an entire command or text line, use the RUB or DELETE key, then re-enter the line.

The Break key can be used to interrupt operations in process.

Command		Function
APPEND	5-13	Appends text to end of a line or group of lines.
BATCH	5-14	Copies work file or specified file into batchstream.
CHANGE	5-15	Replaces a character string with a different one.
CLEAR	5-17	Clears work file.
COLLECT	5-18	Accumulates lines of text.
COMMAND	5-21	Displays the last four commands the user performed.
COPY	5-22	Copies existing text to work file.
DEBUG	5-26	Causes the user to exit the Editor and enter the Task Debugger.
DELETE	5-26	Deletes lines.
EXIT	5-27	Terminates current session in the Editor.
INSERT	5-28	Inserts lines of text.
LIST	5-30	Lists text on terminal screen.
MODIFY	5-32	Changes an existing line by allowing the user to space past good characters and replace bad characters.
MOVE	5-33	Moves lines of text within the workfile.
NUMBER	5-35	Renumbers lines in work file.
PREFACE	5-37	Inserts characters at beginning of line.
PRINT	5-39	Copies work file or specified permanent file to SLO file.

PUNCH	5-40	Copies work file or specified permanent file to SBO file.
REPLACE	5-41	Deletes lines and enters new ones.
RUN	5-14	Copies work file or specified file into batch stream. (Same as the BATCH command, see BATCH).
SAVE	5-42	Saves work file compressed on permanent file.
SCRATCH	5-44	Deletes a permanent file.
SET	5-45	Sets delta, tabs, verification, or show files status length.
SHOW	5-50	Shows current line increment, files, or tab settings.
STORE	5-52	Stores work file uncompressed on permanent disc file.
USE	5-54	Copies another permanent file into a cleared work file.
VERIFY	5-56	Checks validity of the current work file.
WORKFILE	5-57	Accesses a different work file.

5.6.1 APPEND Command

The APPEND command is used to append characters at the end of an existing line. More than one line can be accessed with the command. Additions are then made line by line.

Syntax:

APPEND [group]

where:

group is a line number, a range of lines, or a group of ranges and/or lines on which to append text. Lines to be modified can be further identified by content. (See Section 5.4.1.)

If not specified, the last range specified in the previous command is accessed by default.

Response:

The first line in the specified group is displayed, with the cursor positioned at the end of the line. Type the additional character(s) followed by a carriage return. The new characters are added to the line.

If a group has been specified, the next line in the group is displayed for modification, and so on, continuing to the last line in the group.

To terminate the APPEND command, enter a carriage return instead of appending a character to a line. To bypass modifying a line and go to the next specified line in a group, type a blank followed by a carriage return. The line remains unchanged, but the APPEND command is not terminated. For tabs, use the Editor tab character instead of CTRLI. (See the SET TABS command.)

Example:

```
EDT > APPEND 35
 35.   DEBUG COMMON    DBC(441)   \NO. WORDS FOR COMMON.
EDT >   <CR>
 35.   DEBUG COMMON    DBC(441)        NO. WORDS FOR COMMON.
EDT >
```

Errors:

If the characters you attempt to append to a line cause it to exceed 72 characters, the additional characters are not displayed and the following message appears:

TOO LONG, GO = Y

Type Y (Yes) or enter a carriage return to keep the truncated, modified line. Type N (No) to leave the line as it was before you attempted to append characters.

5.6.2 The BATCH or RUN Command

The BATCH command (the RUN command is equivalent) is used to copy a job file into the batchstream. The file can be the current workfile or another job file in the user's directory. In either case, the file must contain job control statements for a complete job. (See Batch Processing, Volume 1, Chapter 6.)

An alternative to this command is the OPCOM BATCH command as described in Volume 1, Chapter 4.

Syntax:

BATCH[jobfile][UNN]

where:

jobfile is any one-to-eight character name of a file in the user's directory. If no file is specified, defaults to the current workfile. (To copy another workfile in your directory into the batchstream, use the WORKFILE command, then BATCH.)

UNN changes the line numbers in physical line positions 73-80 to blanks. This is not required for batch processing with any of the SYSTEMS utilities. (See Section 5.4.2.3.)

Response:

The file is entered in the batchstream and the EDT > prompt is returned. (The OPCOM LIST command can be used to check on the status of the job.)

Errors:

If a job file specified for BATCH is PO (Password Only) protected, the password is required to read it. The prompt:

ENTER PASSWORD

is displayed. Enter the valid password followed by a carriage return. If the password is not valid, the BATCH command terminates.

Errors encountered during batch processing are shown on the listing produced for the job. (See BATCH PROCESSING, Volume 1, Chapter 6.)

5.6.3 The CHANGE Command

The CHANGE command is used to replace an existing string with another string. Existing characters following the replacement string are adjusted left or right to compensate for replacing a string of one length with a longer or shorter string. (It is not uncommon to exceed the 72-character line limit with this command. See Errors.)

Text in tabbed lines will be shifted left or right to adjust for a replacement string just like any other line. To compensate for the shift, you can type extra blanks in the existing string (if it is shorter than the replacement) or type extra blanks in the replacement string (if it is shorter than the existing string). This will maintain the original alignment of tabbed values.

Syntax:

CHANGE [group] \string\\[newstring]\ [NOLIST]

where:

group — is a line number, a range of lines, or a group of ranges and/or lines to be modified. (See Section 5.4.1.) A content identifier cannot be used with the CHANGE command.

If a group is not specified, the last range specified in the previous command is used by default.

string — identifies the existing string to replace. It must be enclosed in backslashes.

newstring — is the string to replace the existing string. It must also be enclosed in backslashes. To delete the existing string, replace it with nothing by using two consecutive backslashes.

NOLIST — inhibits the automatic display of lines as they are changed. Not recommended on large groups.

Response:

As each line is changed, the resulting line is displayed. Changed lines scroll up on the screen, and no intervention except the Break key is possible until a full screen of changes is displayed. At that point, the Editor pauses, waiting for a signal from the terminal. Use a carriage return to continue the specified changes. Or, use any other character to terminate the command. (A No response to an error prompt also terminates the command. See Errors.)

Example:

```
EDT> CHA\EXPR.\\EXPRESSION\ALL
14.      *   OUTPUTS: R6=0 GOOD EXPRESSION
15.      *   R7=VALUE OF EXPRESSION
674.     *   FOUND END OF EXPRESSION
EDT>
```

Comment:

In this example, the Editor searches the workfile for every occurrence of the term "EXPR." and changes each one to "EXPRESSION".

Errors:

If a change in a line results in a line longer than 72 characters, the modified line is displayed with the characters beyond 72 truncated. The following message appears:

```
TOO LONG, GO = Y
```

Type Y (Yes) or enter a carriage return to keep the truncated, modified line. Type N (No) to leave the line as it was before you attempted to make the change.

Typing N terminates the CHANGE command.

5.6.4 The CLEAR Command

The CLEAR command is used to clear the current contents of the workfile. It is used in conjunction with the USE command to access a different disc file or before building text into a new file interactively via a COLLECT command.

To preserve the contents of the workfile before clearing it (e.g., to retain editing changes made since you accessed the Editor), use the SAVE or STORE commands before using CLEAR. (See the SAVE and STORE command descriptions.)

CLEAR is also an option on the USE command. If you have not specifically cleared the workfile already with the CLEAR command or the CLEAR option, USE prompts CLEAR=N, to discourage clearing the workfile, knowing that it must be cleared before you can use another file and assuming that you want to take action to SAVE or STORE the current workfile if you have not done so already.

Note that a file does not have to be read into the workfile with a CLEAR/USE sequence in order to be listed, copied, or printed.

Syntax:

CLEAR

Response:

The workfile is cleared and its status is updated as reflected in the SHOW command.

Example:

1 EDT>CLEAR

2 EDT>USE XXX CLEAR

Comment:

1 The current contents of the workfile are cleared and the workfile is now 'empty'.

2 The current contents of the workfile are replaced by the contents of file XXX.

Errors:

To be supplied.

5.6.5 The COLLECT Command

The COLLECT command is used to enter new lines of text into the workfile.

Syntax:

COLLECT [lineno [/lineno]][BY increment]

where:

lineno — is the line number in the workfile at which collection is to begin. The least significant digit in the specified line number determines the line numbers the Editor generates for the collected text. (See Section 5.4.2.1.) If the line number you specify currently exists, the Editor adds 1/10 of DELTA, and provides that line number for you.

If a beginning line number is not specified, EDIT defaults to the line following the last line entered with the previous COLLECT or INSERT command or to E, if neither command has been issued. The line number which prompts for collected text reflects the least significant digit of line numbers used in the previous collection or insertion. (It does not reflect the increment used in the previous command, but defaults back to 1., .1, .01, or .001 as applicable, unless overridden with BY.)

Note that 0., .0, .00, or .000 up to 0.999 are acceptable line numbers. This is useful when inserting lines at the beginning of an existing workfile.

To collect lines at the end of the workfile, use 'E' to signify the last line in the file plus 1, e.g., COLLECT E. When E is used, line numbers for collected text will reflect the least significant position and increment of the DELTA value for line numbers. See the SET DELTA command.

/lineno — is the line number in the workfile at which collection is to end. This line number must not currently exist. If not specified, collection ends when the Editor reaches a line number that does already exist.

Line numbers used to prompt for collected text reflect the decimal position of the line number specified for beginning or ending the collection, whichever is least significant.

BY increment — is an absolute number to add to each line number to generate the next line number. The increment value can be any number starting at .001 up to 9999.999.

Example: If 0. is specified as a beginning line for collection and .05 is specified with BY, line numbers generated for collection are .00, .05, .10, .15, .20, etc.

The default increment is DELTA or 1/10 of DELTA depending on the least significant digit used in the line number specification(s).

For further description of increments, see Section 5.4.2.

Response:

The beginning line number is displayed as a prompt. Enter the text of the line followed by a carriage return. From this point on, line numbers are generated automatically. The next sequential line number is displayed. Continue entering lines. A blank followed by a carriage return creates a blank line.

To terminate the command, use a carriage return after the line number is displayed, or depress the Break key. The EDT> prompt is returned.

The command terminates automatically if the next line number in a collection sequence already exists or if you have specified a line at which to end collection and have entered that line.

Examples:

```
EDT>COL 22.01
22.01     text
22.02     text
22.03     text
22.04     <CR>
EDT>22/23
22.       text
22.01     new text
22.02     new text
22.03     new text
23.       text
EDT>
```

1

```
EDT>LIST L
374.      text
EDT>COL E
375.      text
376.      text
377.      <CR>
EDT>
```

Comment:

1. If you were to answer the EDT> with another COL command at this time, the Editor would default to the last command and start its collect with line number 22.04. Therefore, it is necessary to set the Editor up with a new command sequence to start the next collect. In this example we chose the E command.

Errors:

If more lines are to be collected than there are available lines, EDIT collects as many lines as it can. It then displays the line number where collection stopped with the message:

lineno NOT PROCESSED.

Use COLLECT again with a beginning line number specification and/or TO increment that generates enough new lines to accommodate the text that still must be collected. Start collecting again. (See Section 5.4.2.1 and the lineno descriptions under Syntax.)

The NUMBER command can be used if you have gotten to a .nnn position with no line numbers left for insertion. See the NUMBER command.

If you type more than 72 characters in a line, the additional characters are not displayed and the following message appears:

TOO LONG, GO = Y

Type Y (Yes) or enter a carriage return to keep the truncated line as displayed on the screen. Type N (No) to re-enter the line.

5.6.6 The COMMAND Command

The COMMAND command is used to display the user's last four Edit commands as they were seen by the Editor.

Syntax:

COMMAND

Response:

The last four commands entered into the Editor are displayed. This could be useful in determining a problem with a file. Possibly what you typed in as an instruction wasn't what you really wanted done.

Example:

```
EDT> COM
F,L
DEL 3/7
APP 9
COM
EDT>
```

If less than four commands were issued to the Editor, only those issued are displayed.

5.6.7 The COPY Command

The COPY command is used to copy existing lines of text to the work file, beginning at the specified line number. The lines to copy can be on the work file already or they can come from a file saved or stored previously.

To select specific lines from a file, the file must be saved or stored with line numbers. If the whole file is to be copied, a lack of line numbers makes no difference.

The lines from which text are copied are not deleted. To delete lines in one part of a workfile and copy them to a different part, use the MOVE command.

Files that have not been built in the Editor or accessed and saved previously via Editor USE and SAVE or STORE commands cannot be copied directly into a workfile.

To do so, save the current contents of the workfile, then access the desired file with the USE command. USE attaches line numbers in a form acceptable to the EDT. (See Section 5.4.2.3.)

Save or store the file back on disc, then copy the original file back into the workfile. The file can now be copied into the workfile.

Syntax:

COPY [group] [FROM][filename] [TO lineno] [BY increment] [LIST]

where:

group — is a line number, a range of lines, or a group of ranges and/or lines to copy into the workfile from another file or from within the workfile itself. Lines to be copied can be further identified by content. (See Section 5.4.1.) If copying from a file other than the current workfile, the name of the file must be supplied with FROM.

If nothing is specified for group, all lines in the file are copied.

FROM filename — specifies the name of a file from which lines are to be copied. This file must belong to the same user who issues the command.

If a file name is not specified, the current workfile is the default.

TO lineno — is the line number in the workfile at which copying is to begin. The least significant digit in the specified line number determines the line numbers the Editor generates for the text being copied. (See Section 5.4.2.1.) The line number you specify must not currently exist (must not have been used before or must have been previously deleted). Copying stops if the Editor generates a line number that already exists.

If a beginning line number is not specified, EDIT defaults to E, the line following the last line in the workfile. If collecting at end of file, line numbers for copied text will reflect the least significant position and increment of the DELTA value for line numbers. See the SET DELTA command.

Note that 0., .0, .00, or .000 up to 0.999 are acceptable line numbers. This is useful when copying lines to the beginning of an existing workfile.

BY increment — is an absolute number to add to each line number to generate the next line number. The increment value can be any number starting at .001.

Example: If 0. is specified as a beginning line for copying and .05 is specified with BY, line numbers generated for copied text are .00, .05, .10, .15, .20, etc.

The default increment is 1., .1, .01, or .001, depending on the least significant digit used in the line number specification.

For further description of increments, see Section 5.4.2.1.

LIST — specifies that lines be displayed as they are copied. It is helpful if you have to break out of a copy operation.

Response:

Lines are copied into the workfile as specified.

If lines are coming from a file other than the workfile, the name of the file and the message *FILE* are displayed.

Example:

1.
```
EDT > COP 93/95 TO 90.1
EDT > 90/96
90.     text
90.1    copied text
90.2    copied text
90.3    copied text
91.     text
92.     text
93.     text (same as 90.1)
94.     text (same as 90.2)
95.     text (same as 90.3)
96.     text
```

2.
```
EDT>COP 1/5 FROM BAT TO 1.01
EDT>1/2
1.      text
1.01    BAT text
1.02    BAT text
1.03    BAT text
1.04    BAT text
1.05    BAT text
2.      text
EDT>
```

3.
```
EDT> COP TO 30.1 5,12/14,20
EDT> 30/31
30.     text
30.1    Copied text (same as line 5)
30.2    Copied text (same as line 12)
30.3    Copied text (same as line 13)
30.4    Copied text (same as line 14)
30.5    Copied text (same as line 20)
31.     text
EDT>
```

Comments:

1. In this example text was copied from one position in the workfile to another position, therefore only the line numbers to be copied needed to be specified.

2. In this example, text from an existing file was copied to the workfile so both the file name (BAT) and the line numbers to be copied had to be specified.

3. In this example, text was copied from several positions in the workfile to one position. To do this, the 'TO' parameter must be specified before the group parameters. If you attempt this in the standard manner (i.e., format shown under the Syntax), lines 5 and 12 through 14 would be copied to the end of the file and only line 20 would be copied to 30.1 (see Defaults, Section 5.4.1.5).

Errors:

If more lines are to be copied than there are available lines, EDIT copies as many lines as it can. It then displays the line number on the file being accessed where copying stopped with the message:

lineno NOT PROCESSED.

Use COPY again with a TO specification that has a beginning line number (and/or TO increment) that generates enough new lines to accommodate the text that still must be copied. (See Section 5.4.2.1 and the TO and BY descriptions under Syntax.) Start copying again at the line number displayed above.

If a file is specified for COPY and it is PO (Password Only) protected, the password is required to read it. The prompt:

ENTER PASSWORD

is displayed. Enter the valid password followed by a carriage return. If the password is not valid, the COPY command terminates.

5.6.8 The DEBUG Command

The DEBUG command is used to exit from the Editor and generate memory dumps. This command was introduced for internal use only to aid in development and debugging of the Editor.

As a result, the keyword DEBUG is reserved and cannot be used as a file name.

Syntax:

DEBUG

5.6.9 The DELETE Command

The DELETE command is used to delete lines of text from the work file.

Syntax:

DELETE [group] [LIST]

where:

group — is a line number, a range of lines, or a group of ranges and/or lines to be deleted. Lines to be deleted can be further identified by content. (See Section 5.4.1.)

If not used, the last range specified in the previous command is deleted by default.

LIST — specifies that lines be displayed as they are deleted.

Response:

The specified lines are deleted from the workfile. The Break key can be used to stop deletions. (See Section 5.4.7.)

Example:

```
EDT>DEL 15/20
EDT>13/22
13.     text
14.     text
21.     text
22.     text
EDT>
```

Errors:

If an invalid group is specified, the Editor responds:

```
VOID RANGE - LINE RANGE NOT SPECIFIED OR ILLEGAL
EDT>
```

5.6.10 The EXIT Command

The EXIT command terminates the current EDIT session and returns you to the TSM> prompt. The current workfile is maintained as is for future editing.

Syntax:

EXIT or X

Example:

```
EDT > EXIT
TSM >
```

5.6.11 The INSERT Command

The INSERT command is used to add one or more new lines of text to a file. It is essentially the same as COLLECT except that INSERT assumes the addition one line at a time and COLLECT assumes the addition of more than one line.

Syntax:

INSERT[group] [BY increment]

where:

group — is a line number, a range of lines, or a group of ranges and/or lines you want to insert.

The least significant digit in the line number or range, specified in the group, determines the line numbers the Editor generates for the inserted text. (See Sections 5.4.1 and 5.4.2.) If the line number you specify currently exists, EDIT adds 1/10 DELTA to the existing line and uses it as the line number.

If no line numbers are specified, EDIT normally defaults to the line following the last line entered with the previous COLLECT or INSERT command. The line number which prompts for inserted text reflects the least significant digit of line numbers used in the previous collection or insertion. (It does not reflect the increment used in the previous command, but defaults back to 1., .1, .01, or .001 as applicable.)

Note that 0., .0, .00, or .000 up to 0.999 are acceptable line numbers. This is useful when inserting one or more Lines at the beginning of an existing workfile.

To insert a line at the end of the workfile, use 'E' to signify the last line in the file plus 1, i.e., INSERT E. When E is used, the line number for inserted text will reflect the least significant position and increment of the DELTA value for line numbers. See the SET DELTA command. This is the default if no previous INSERT or COLLECT command has been used and a group is not specified.

BY increment — is an absolute number to add to each line number to generate the next line number. The increment value can be any number, starting with .001.

Example: If 0./0.99 is specified as a range of lines to insert and .05 is specified with BY, line numbers generated for insertion are .00, .05, .10, .15, .20, etc.

The default increment is 1., .1, .01, or .001, depending on the least significant digit used in the range specification(s).

For further description of increments, see Section 5.4.2.1.

Response:

The beginning line number is displayed as a prompt. Enter the text of the line followed by a carriage return. From this point on, line numbers are generated automatically. The next sequential line number is displayed. Continue entering lines. A blank followed by a carriage return creates a blank line.

To terminate the command, use a carriage return after the line number is displayed or depress the Break key. The EDT> prompt is returned.

The command terminates automatically if the next line number in an insertion sequence already exists or when it reaches the line number you have specified for ending insertions and have entered that line.

Example:

```
EDT>INS 14.01
14.01   text
EDT>13/15
13.     text
14.     text
14.01   text
15.     text
EDT>
```

Errors:

If more lines are to be inserted than there are available lines, EDIT takes as many lines as it can. It then displays the line number where insertion stopped with the message:

lineno NOT PROCESSED.

Use INSERT again with a BY specification that has a beginning line number (and/or BY increment) that generates enough new lines to accommodate the text that still must be inserted. (See Section 5.4.2.1 and The group description under Syntax.) Start inserting again before the line number displayed above.

The NUMBER command can be used if you have gotten to a .nnn position with no line numbers left for insertion. See the NUMBER command.

If you type more than 72 characters in a line, the additional characters are not displayed and the following message appears:

TOO LONG, GO = Y

Type Y (Yes) or enter a carriage return to keep the truncated line as displayed on the screen. Type N (No) to re-enter the line.

5.6.12 The LIST Command

The LIST command is used to display lines from the current workfile or from any other file in the user's directory. The LIST command is implied when no command is supplied in response to the EDT> prompt.

Syntax:

[LIST] [group] [[FROM] filename] [UNN]

where:

group — is a line number, a range of lines, or a group of ranges and/or lines to list. Lines can be further identified by content. (See Section 5.4.1.)

If not specified, LIST normally defaults to the last range specified or implied in the previous command. Exception: Initially, LIST or a carriage return to the EDT > prompt lists the first line of the file. To list the entire file, use LIST ALL (or just ALL).

FROM filename — specifies the one-to-eight character name of a file in the user's directory. Using the keyword FROM is optional.

If FROM filename is not specified, listing defaults to the current workfile.

UNN — lists lines without displaying line numbers.

Response:

The specified or default group is scrolled onto the screen. If the screen is full, EDIT pauses. Enter a carriage return to continue the listing or any other character to terminate the listing. To interrupt the listing mid-screen, use the Break key.

If a file is specified for LIST and it is PO (Password Only) protected, the password is required to read it. The prompt:

ENTER PASSWORD

is displayed. Enter the valid password followed by a carriage return. If the password is not valid, the LIST command terminates.

Example:

```
EDT > COL
12.   text
13.   text
14.   text
15. <CR>
EDT > LIST
12.   text
13.   text
14.   text
EDT > LIST ALL
1.    text
2.    text
3.    text
.
.
.
12.   text
13.   text
14.   text
EDT >
```

Errors:

To be supplied.

5.6.13 The MODIFY Command

The MODIFY command is used to change an existing line by spacing past good characters and replacing bad characters. An up arrow (^) can be used to replace a character with a blank. MODIFY is used one line at a time. It cannot be applied to all lines in the file.

Replacement strings must be equal to or less than the number of characters being changed. If not less than or equal to the original string, or for global modifications, use the CHANGE command.

Syntax:

MODIFY [group]

where:

group — is a line number, a range of lines, or a group of ranges and/or lines to be modified. Lines to be modified can be further identified by content. (See Section 5.4.1.4.)

If not used, the last range specified in the previous command is accessed by default.

Response:

The first line in the specified group is displayed. The Editor reissues the line number as a prompt. Use the space bar to keep characters, type in new characters where needed, or use the uparrow key to replace characters with a blank. Using a carriage return before the end of the line keeps remaining characters 'as is'.

If a group has been specified, the next line in the group is displayed for modification, and so on, continuing to the last line in the group.

For tabs, use a CTRLI key sequence rather than the Editor tab character.

Example:

```
EDT>MOD 1/3
1.  THERE WILL BE A SHORTTMEETING ON TUESDAY.
1.                  ^
1.  THERE WILL BE A SHORT MEETING ON TUESDAY.
2.  PLEASE PLAN OT ATTEND.
2.                 TO
2.  PLEASE PLAN TO ATTEND.
3.  ATTENDANCE WILL NOT BE NOTED.
3.   <CR>
EDT>
```

1. Space past good characters, enter an up arrow to replace T with a space.
2. Space past good characters, enter correction.
3. Immediate carriage return leaves line 'as is' and returns the EDT> prompt.

5.6.14 The MOVE Command

The MOVE command is used to move one or more existing lines of text from one part of the work file to another. Each line is deleted from its original position after it has been moved successfully to the new position.

Syntax:

MOVE [group] [TO lineno] [BY increment] [LIST]

where:

group — is a line number, a range of lines, or a group of ranges and/or lines to move. Lines to move can be further identified by content. (See Section 5.4.1.)

If not specified, EDIT defaults to the last line or range in the group specified (or implied) in the previous command.

TO lineno — is the line number given to the first line of text being moved. The least significant digit in the specified line number determines the line numbers the Editor generates for the rest of the text. (See Section 5.4.2.1.) The line number you specify must not currently exist (must not have been used before or must have been previously deleted).

If a beginning line number is not specified, EDIT defaults to E, the line following the last line on the workfile. Line numbers for text that is moved will reflect the least significant position and increment of the DELTA value for line numbers. See the SET DELTA command.

Note that 0., .0, .00, or .000 up to 0.999 are acceptable line numbers. This is useful when moving lines to the beginning of a workfile.

BY increment — is an absolute number to add to each line number to generate the next line number. The increment value can be any number, starting at .001.

Example: If 0. is specified as a beginning line for moved text and .05 is specified with BY, line numbers generated for the text are .00, .05, .10, .15, .20, etc.

The default increment is 1., .1, .01, or .001, depending on the least significant digit used in the line number specification(s).

For further description of increments, see Section 5.4.2.1.

Response:

The Editor simultaneously moves text to its new position and deletes the text from its old position.

Example:

1.
```
EDT > MOV 60/62 TO 58.1
EDT > 58/63
58.     text
58.1    moved text (formerly line 60)
58.2    moved text (formerly line 61)
58.3    moved text (formerly line 62)
59.     text
63.     text
EDT >
```

2.
```
EDT> MOV TO 53.1 46,48/50,52
EDT> 45/54
45.     text
47.     text
51.     text
53.     text
53.1    moved text (formerly line 46)
53.2    moved text (formerly line 48)
53.3    moved text (formerly line 49)
53.4    moved text (formerly line 50)
53.5    moved text (formerly line 52)
54      text
```

Comment:

1. Lines 60/62 are deleted from their current position and moved to the specified position within the workfile.

2. In this example, text was moved from several positions in the workfile to one position. To do this, the 'TO' parameter must be specified before the group parameters. If you attempt this in the standard manner (i.e., format shown under the Syntax), lines 46 and 48 through 50 would be moved to the end of the file and only line 52 would be moved to 53.1 (see Defaults, Section 5.4.1.5).

Errors:

If more lines are to be moved than there are available lines, EDIT moves as many lines as it can. It then displays the original line number where copying stopped with the message:

lineno NOT PROCESSED.

Use MOVE again with a TO specification that has a beginning line number (and/or TO increment) that generates enough new lines to accommodate the text that still must be moved. (See Section 5.4.2.1 and the TO and BY descriptions under Syntax.) Start moving again at the line number displayed above.

5.6.15 The NUMBER Command

The NUMBER command is used to renumber all lines in the workfile, using the specified decimal position and increment. The default number for the first line in the file is 1; the default increment is also 1 (i.e., lines are numbered 1, 2, 3, etc.).

If the line number and increment do not provide sufficient line numbers for the entire workfile, a diagnostic is displayed; the workfile is unchanged.

Syntax:

NUMBER [lineno][BY increment]

where:

lineno — is the line number for the first line in the workfile. If a beginning line number is not specified, EDIT defaults to line 1.

Line numbers for the file will reflect the increment of the DELTA value for line numbers unless overridden with BY. See the SET DELTA command, and BY, next.

BY increment — is an absolute number to add to each line number to generate the next line number. The increment value can be any number, starting at .001.

Example: If 1 is specified as a beginning line number and .05 is specified with BY, line numbers generated are 1.00, 1.05, 1.10, 1.15, 1.20, etc.

For NUMBER the default increment is not dependent on the least significant digit used in the line number specification. It is dependent only upon DELTA. (See the SET DELTA command.)

Response:

A diagnostic is displayed if the line number and increment do not provide sufficient line numbers.

Example:

```
EDT>LIS 3/4
3.    text
3.01  text
3.02  text
3.03  text
4.    text
EDT > NUM
EDT > LIS 3/L
3.    text
4.    old line 3.01 text
5.    old line 3.02 text
6.    old line 3.03 text
7.    old line 4
 <break>
EDT>
```

5.6.16 The PREFACE Command

The PREFACE command is used to insert one or more characters at the beginning of an existing line. Additions are made character-by-character, resulting in a right shift in the existing line.

More than one line can be accessed with the command. Additions are then made line by line.

Syntax:

PREFACE [group]

where:

group is a line number, a range of lines, or a group of ranges and/or lines to be modified. Lines to be modified can be further identified by content. (See Section 5.4.1.)

If not used, the last range specified in the previous command is accessed by default.

Response:

The first line in the specified group is displayed. The Editor reissues the line number as a prompt. Type the new string, followed by a carriage return. Existing characters in the line are right shifted with the new string at the beginning of the line. The rest of the line remains unchanged. If a group has been specified, the next line in the group is displayed for modification, and so on, continuing to the last line in the group.

To terminate the PREFACE command, enter a carriage return in lieu of a new string.

For tabs, use a CTRLI key sequence rather than the Editor tab character.

Example:

```
EDT> PRE 1,15,20

 1.    ON MODIFYING A LINE
 1.    UP <CR>
 1.    UPON MODIFYING A LINE
15.    MODIFIED LINE SHIFTS
15.    A <b> <CR>
15.    A MODIFIED LINE SHIFTS
20.    TO BYPASS MODIFICATION
20.      <CR>
VOID RANGE
EDT>
```

Comments:

1. Note that b indicates typing blank spaces.

2. A carriage return terminates the PRE command and the line remains 'as is'.

Errors:

If you type more than 72 characters as a new string, the additional characters are not displayed and the following message appears:

```
TOO LONG, GO = Y
```

Type Y (Yes) or enter a carriage return to keep the truncated, modified line. Type N (No) to leave the line as it was before the replacement was attempted.

5.6.17 The PRINT Command

The PRINT command is used to print the current workfile or another file in the user's directory on the device assigned for system SLO files (usually a printer).

Syntax:

PRINT [filename] [UNN]

where:

filename — is a one- to eight-character name of a file in the user's directory. If no file is specified, defaults to the current workfile. (Note that other workfiles in your directory can be output by accessing them with the WORKFILE command.)

UNN — outputs the file without line numbers.

Response:

To be supplied.

Example:

To be supplied.

Errors:

If a file is specified for PRINT and it is PO (Password Only) protected, the password is required to read it. The prompt:

ENTER PASSWORD

is displayed. Enter the valid password followed by a carriage return. If the password is not valid, the PRINT command terminates.

5.6.18 The PUNCH Command

The PUNCH command is used to output the current workfile or another file in the user's directory on the device assigned for system SBO files (usually a card punch).

Syntax:

PUNCH [filename] [UNN]

where:

filename is a one- to eight-character name of a file in the user's directory. If no file is specified, defaults to the current workfile. (Note that other workfiles in your directory can be output by accessing them with the WORKFILE command.)

UNN outputs the file without line numbers. See Section 5.4.2.4.

Response:

To be supplied.

Example:

To be supplied.

Errors:

If a file is specified for PUNCH and it is PO (Password Only) protected, the password is required to read it. The prompt:

ENTER PASSWORD

is displayed. Enter the valid password followed by a carriage return. If the password is not valid, the PUNCH command terminates.

5.6.19 The REPLACE Command

The REPLACE command is used to replace existing lines in the workfile with different lines of text. Replacements are made line by line.

Syntax:

REPLACE [group]

where:

group is a line number, a range of lines, or a group of ranges and/or lines to be replaced. Lines to be replaced can be further identified by content. (See Section 5.4.1.)

If not specified, the last range specified in the previous command is accessed by default.

Response:

The first line in the specified group is displayed. The Editor reissues the line number as a prompt. Type the replacement line, followed by a carriage return. If a group has been specified, the next line in the group is displayed for replacement, and so on, continuing to the last line in the group.

To replace an existing line with all blanks, type a blank space followed by a carriage return in lieu of a replacement line. The REPLACE command will not terminate.

To terminate the REPLACE command, enter a carriage return in lieu of a replacement line or use the Break key.

Example:

```
EDT> REP 24
24. The replace commands are used to REPLACE
24. The REPLACE command is used to replace
EDT>
```

Errors:

If you type more than 72 characters as a replacement string, the additional characters are not displayed and the following message appears:

TOO LONG, GO = Y

Type Y (Yes) or enter a carriage return to keep the truncated, modified line. Type N (No) to leave the line as it was before the replacement was attempted.

5.6.20 The SAVE Command

The SAVE command is used to put a copy of the current workfile on disc as a permanent file in the user's directory. With SAVE, the text is compressed, i.e., consecutive blanks are replaced with a special string indicating the number of blanks compressed. The workfile remains in tact. The STORE command is identical to SAVE except that the file is not compressed and with STORE, you can specify unnumbered.

If you attempt to save onto a file name that already exists in your directory, you must explicitly scratch (delete) the existing contents of the file. This requirement curbs inadvertant writes over existing files.

The SAVE command is order dependent, therefore, the syntax must be entered in the order shown below:

Syntax:

SAVE [filename] [PO / RO] [SYS] [SCRATCH]

where:

filename is the one- to eight-character name of the file to create. Filename is optional.

If a name is not specified, the workfile is copied to the file name used in the most recent SAVE, STORE, or USE command, unless it was a system file. If the last file used with one of the above commands was a system file, the workfile is copied to a file of the same name in the user's directory unless you use SYS. See below.

PO creates the file with password only access restrictions.

RO creates the file with read only access restrictions.

Both the PO and RO options permit creating a restricted file, changing a nonrestricted file to a restricted file, or changing the password of a restricted file.

If a file has restricted access and the PO or RO option is specified, the message "ENTER PASSWORD:" is displayed. The valid password must be given before the file can be scratched. Upon entering the valid password, the message "ENTER NEW PASSWORD:" is displayed. Upon entering the same password or a new password, the file will be SAVED.

SYS creates the file named above as a system file.

SCRATCH explicitly clears the contents of the file named or implied with the SAVE command before the workfile is copied to that file space. If not specified, you are prompted for the scratch function. (See Response.)

Response:

The current contents of the workfile are copied into the file space specified with SAVE. If SCRATCH is not specified and a file of the same name already exists in the user's directory, the following message is displayed:

filename, SCRATCH = N

Type N (No) or enter a carriage return to terminate the save operation. Type Y (Yes) to scratch the existing contents of the file and store the workfile in their place.

If a file is password protected, the following prompt is displayed:

ENTER PASSWORD:

The valid password for the file must be entered. If what you type is not the valid password, the prompt is repeated. See Errors.

When the save is complete, the following message is displayed:

filename nnnnnn size type username speed device restriction
ii*WRKFL nnnnnn size type username speed device
ii*WRKFL SAVED xx lines

where filename is the name of the file saved, ii is the workfile code, nnnnnn is the starting address, size is the file size, type is the file type, username is the user name associated with the file, speed designates FAST or SLOW access, device is disc address, restrictions note any restrictions placed on access to the file, and xx lines is the number of lines of text in the workfile.

Example:

```
EDT > SAVE SSS
SSS          334400      4ED          MEYERS S  0800  RO
MM*WRKFL     289296      80FE         MEYERS S  0800
MM*WRKFL SAVED           120 LINES
EDT >
```

Errors:

If a file is password protected, you must supply the valid password to update the file. If you do not, the Editor keeps prompting for a valid password. Enter a carriage return to get out of the loop. The SAVE command will terminate and the workfile will not be saved.

5.6.21 The SCRATCH Command

The SCRATCH command is used to delete a user file or a system file from the directory. Both the filename and its contents are removed from the disc and disc directory. Space is thus freed for new files. If the file is password protected, the password must be supplied.

Syntax:

SCRATCH filename [SYS]

where:

filename is the one- to eight-character name of the file to be scratched.

SYS designates the name file is a system file.

Response:

If a file is password protected, the following prompt is displayed:

ENTER PASSWORD:

The valid password for the file must be entered. If what you type is not the valid password, the prompt is repeated. See Errors.

If a system file is found with the file name specified and SYS was not specified, the following message is displayed:

SYSTEM FILE FOUND BY THAT NAME. CORRECT FILE (Y OR N)?

A Y (Yes) response will scratch the file. N (No) response will not scratch the file and terminate the scratch operation.

Note: If there is a user file and a system file with the same file name and SYS is not specified, the user file is the file which will be scratched.

Example:

```
EDT> SCR AAA
EDT> SCR BAR SYS
EDT>
```

User file AAA is scratched and system file BAR is scratched.

Errors:

If a file is password protected, you must supply the valid password to scratch (delete) it. If you do not, the Editor keeps prompting for a valid password. Enter a carriage return to get out of the loop. The SCRATCH command will terminate, and the file will not be scratched.

5.6.22 The SET Commands

There are four SET commands in the Editor: SET DELTA, SET TABS, SET VERIFICATION and SET SHOW FILES. They are described separately in this section.

SET DELTA

The SET DELTA command is used to specify a default increment and significant digit other than 1.0 for line numbers which follow the last line of the workfile. The reset DELTA value remains in effect only for the current editing session. When you exit the Editor, DELTA is returned to 1.0.

Syntax:

SET DEL [increment]

where:

increment — is an absolute number to add to each line number to generate the next line number. The increment value can be any number, starting at .001 (up to 9999.999). Zero is an invalid increment.

Example: If .05 is specified as DELTA and line 15 is the last line on the workfile, lines generated at the end of file (when E is specified or implied) will be 15.05, 15.10, 15.15, 15.20, etc.

The default DELTA increment is 1., i.e., in the example above, the next line numbers generated would be 16, 17, 18 and so on. If an increment is not specified, the default value of DELTA is reset.

Response:

DELTA is modified. After the current session is complete, it will be reset to 1.

Example:

To be supplied.

Errors:

If 0 is entered as the increment value, the following message is displayed:

```
ERROR-->0.
EDT>
```

SET TABS

The SET TABS command is used to modify tab positions. In MPX-32, the M.KEY file can contain tab settings for each logon ownername as described in the KEY utility (Section 7). TSM defaults to these tabs if they exist. Or, if no tabs are set in M.KEY, TSM sets system tabs when you access the Editor.

Default tabs set by the Editor are in columns 10, 20, 36, 41, 46, 51, 61, and 72, which correspond to label, op code, and other fields defined on SYSTEMS Assembler coding sheets.

The most recent tabs set with SET TABS will remain in effect as long as you remain on the system (until you exit TSM), i.e., they will remain in effect as you access various processors, move from one system environment to another, etc. When you exit from TSM, the SET TABS are not saved. When you log on again, either the M.KEY tabs (or system tabs if none exist in M.KEY) are set.

When typing text with tabs, the tab character produced by a Control I (<CTRLI>) key sequence is interpreted by the TSM device handlers and replaced by the appropriate number of blanks. The cursor is adjusted by echoing the spaces to the terminal. This allows you to see tabbed spacing on the screen as you are entering the text.

The tab character used to define tab positions in the SET TABS command is either a backslash (\) or the character you define when you enter tab settings. The most recent tab character defined in SET TABS is the one used when entering a tabbed record (unless you want to use <CTRLI> . Like tab settings, the tab character remains in effect until you exit TSM. When you log on again, the tab character is a backslash.

The tab character is removed from the text when it is interpreted, so that no special treatement of tab characters is required from subsequent processors such as the Assembler.

Syntax:

SET TABS

Response:

The Editor displays the current tab settings. Assuming tabs have not been reset previously during the current session, numbers 1-9 indicate column groupings (10's, 20's, etc.) with the default tab character (\) for columns 10, 20, and 70, enclosed in blanks. Beginning in column 29, the most significant digit of the column number is also enclosed in blanks, e.g., b3b indicates the beginning of columns 30-39, with blanks falling in columns 29 and 31 as shown in the example.

The Editor prompts for new tab positions by locating the cursor in column 2. Type blanks for non-tab positions and any character desired in tab positions (a maximum of 8 tab positions can be set at one time). The character(s) you supply will override the backslash. If you use different characters for tab specifications, the last one typed is taken as the tab character. Any tab that is not specified explicitly is not set.

Example:

```
EDT> SET TABS

TABS   =12345678    2345678    2345678 3 2345 78 4 2345 78 5 234 ...
.SET   =bbbbbbbbbAbbbbbbBbbbbCbbbbbbbbbbbCbbbbbbBbbbbbbbbbbbC ...

TABS   =123456789C123456C8921C34567893123C567894C234567895123C ...
```

If a user has changed the tabs and wants to return to the Editor default settings, rather than perform a SET TABS command and reenter the settings, he can perform one of the following commands:

SET { OLD | OLDTAB | OLDTABS }

Response:

The tab settings are returned to the Editor default tab positions and the tab delimiter is returned to the default delimiter, i.e., backslash (\).

If the user wishes to change the tab delimiter but keep tab settings in their current positions, he can perform the following command:

```
SET TC
```

The Editor responds:

```
CHAR=
```

Enter the character you want to be the tab delimiter. To return to the default tab delimiter, perform the following command:

```
SET OTC
```

Errors:

If when using tabs, you exceed 72 characters in a line, the tabbed line is displayed with extra characters truncated followed by the message:

```
TOO LONG,  GO = Y
```

Type Y (Yes) or enter a carriage return to keep the truncated line as displayed. Type N (No) to re-enter the line.

If more than 8 tab positions are entered, only the first 8 are recognized and set.

SET VERIFICATION

The SET VERIFICATION command is used to set or inhibit the automatic verification of a file before a Number, Save or Store command is performed (see VERIFY command).

Syntax:

SET { VFA }
    { VFN }

where:

VFA — enables the automatic verification of a file before any Number, Save or Store command is performed and remains in effect until a SET VFN is performed or the Editor is exited.

VFN — inhibits the automatic verification of a file before any Number, Save or Store command is performed. Default.

Response:

When a SET VFA command is used, the following message is displayed upon issuing a Number, Save or Store command:

VERIFYING BEFORE NUM/SAV/STO. PLEASE WAIT.

There is no response to the SET VFN command other than the Number, Save or Store operation being performed.

Errors:

The following error messages can be generated by the validation (VFA) routine:

LINE COUNT ERROR - INTEGRITY UNCERTAIN

POINTER INVALID - INTEGRITY UNCERTAIN

HEADER SEQ INVALID - INTEGRITY UNCERTAIN

SEQ ERR - BAD SECTOR LINKAGE - INTEGRITY UNCERTAIN

INVALID SECTOR NUMBER - INTEGRITY UNCERTAIN

If any of these messages is displayed, the following message is also displayed:

CONTINUING MAY RESULT IN LOST DATA

SET SHOW FILES

The SET SHOW FILES command is used to obtain a shortened version of file status output from a Show or Show Files command.

Syntax:

```
SET {SFS }
    {SFN}
```

where:

SFS — causes a shortened version of file status output to be displayed when a Show or Show Files command is performed and remains in effect until a SET SFN is performed or the Editor is exited.

SFN — causes the normal version of file status output to be displayed when a Show or Show Files command is performed. Default.

Response:

The file status displayed in response to a SET SFS command is as follows:

```
EDT> SET SFS
EDT> SHO FILES
UTILPC      303828          20ED
STATUS      345328           8ED
SH*WRKFL    295552          80FE
 <break>

EDT > SHO
JM*WRKFL    298380          80FE
JM*WRKFL SAVED              5 LINES
```

The file status displayed in response to a SET SFN command is the same as the SHOW command SHOW FILES example shown in this chapter.

Errors:

None

5.6.23 The SHOW Command

The SHOW command is used to display the status of one particular file, current increment setting of DELTA, current tab settings, or names of current files in the user's directory. For further description of DELTA and tabs, see the SET command.

If just SHOW is used, the Editor displays the name of any file accessed (SAVED, STORED, or USED) in the current session plus the status of the current workfile.

Syntax:

SHOW [filename | DELTA | FILES | TABS]

where:

filename is a 1-8 character file name whose status will be displayed.

DELTA is an option which displays DELTA.

FILES is an option which displays names of permanent disc files belonging to the user.

TABS is an option which displays the current tab settings.

Response:

For SHOW, the following message is displayed:

```
ii*WRKFL nnnnnn size FE username speed device
ii*WRKFL status xx lines
```

where ii is the workfile code, nnnnnn is the starting address, size is the file size, FE designates a workfile, username is the username associated with the file, speed designates FAST or SLOW access, device is the disc address, the status message for the workfile is one of the following: CHANGED, CLEAR, SAVED, and xx lines is the number of lines of text in the workfile. CHANGED means the file has been edited but the edited version has not been saved. CLEAR means the file has been cleared and another disc file has not been copied into it SAVED means that a SAVE or STORE command has been issued with no intervening editing.

If a default file name exists for SAVE and STORE, its name is also displayed.

For SHOW FILES, the Editor lists each file in the user's directory. For each file the following is displayed:

> filename, starting disc address (beginning block number), number of blocks, type, as defined via the FILEMGR utility (files created in the Editor are automatically type ED or EE), user name associated with the file, speed of the file, device address, and restrictions, if any

Display of file names can be terminated with the Break key. Or, you can wait until a full screen is displayed. EDIT will pause. Enter a carriage return to continue the listing or any other key to terminate the command.

For SHOW TABS, positions 1-72 of the line are displayed with the current tab character displayed in positions corresponding to current tab stops.

For DELTA, the DELTA increment value used to generate lined numbers at the end of file is displayed.

Example:

```
      EDT > SHOW FILES
    UI*WRKFL     219775   118FE    MEYERS   F   0800
    MM$UI.5      277536    59ED    MEYERS   F   0800   RO
    MM$UI.7      275471    59EE    MEYERS   F   0800
1   <break>
      EDT> SHOW
    MM*WRKFL     254720    80FE    MEYERS   F   0800
2   MM*WRKFL CHANGED 3 LINES
      EDT > SHO DELTA
3     1.  (DELTA)
      EDT>
```

Comments:

1 The Break key is used to terminate the listing of files in the user's directory mid-screen. FE designates a work file, ED designates a SAVED file, and EE designates a STORED file.

2 No files have been saved, stored, or used in the current section. Only the status of the workfile is displayed.

3 The increment that will be used when adding text at the end of the workfile in 1, the default value of DELTA.

5.6.24 The STORE Command

The STORE command is used to put a copy of the current workfile on disc as a permanent file in the user's directory. The text is output as is and the workfile remains intact. The SAVE command is identical to STORE except that the saved file is compressed and cannot be stored unnumbered. (See the SAVE command.)

If you attempt to store onto a file name that already exists in your directory, you must explicitly scratch (delete) the existing contents of the file. This requirement curbs inadvertent writes over existing files.

The STORE command is order dependent, therefore the syntax must be entered in the order shown below:

Syntax:

STORE [filename] [RO/PO] [SYS] [UNN] [SCRATCH]

where:

filename is the one- to eight-character name of the file to create. Filename is optional.

If a name is not specified, the workfile is copied to the file name used in the most recent SAVE, STORE, or USE command, unless that file was a system file. If the last file used with one of the above commands was a system file, the workfile is copied to a file of the same name in the user's directory.

PO creates the file with password only access restrictions.

RO creates the file with read only access restrictions.

Both PO and RO options permit creating a restricted file, changing a nonrestricted file to a restricted file, or changing the password of a restricted file.

If a file has restricted access and the PO or RO option is specified, the message "ENTER PASSWORD:" is displayed. The valid password must be given before the file can be scratched. Upon entering the valid password, the message "ENTER NEW PASSWORD:" is displayed. Upon entering the same password or a new password, the file will be STORED.

SYS creates the file named above as a system file.

UNN The file can optionally be stored without line numbers. See Section 5.4.2.4.

SCRATCH explicitly clears the contents of the file named or implied with the STORE command before the workfile is copied to that file space.

If not specified, you are prompted for the scratch function. (See Response.)

Response:

The current contents of the workfile are copied into the file space specified with STORE. If SCRATCH is not specified and a file of the same name already exists in the applicable directory, the following message is displayed:

```
filename, SCRATCH = N
```

Type N (No) or enter a carriage return to terminate the command. Type Y (Yes) to scratch the existing contents of the file and store the workfile in their place.

If a file is password protected, the following prompt is displayed:

```
ENTER PASSWORD:
```

The valid password for the file must be entered. If what you type is not the valid password, the prompt is repeated. See Errors.

When the store is complete, the following message is displayed:

```
filename   nnnnnn size type username speed device restrictions
ii*WRKFL nnnnnn size type username speed device
ii*WRKFL SAVED xx lines
```

where filename is the name of the file stored, ii is the workfile code, nnnnnn is the starting address of the file, size is the size of the file, type is the file type, username is the user name associated with the file, speed designates FAST or SLOW access, device is the disc address, restrictions note any restrictions placed on access to the file, and xx lines is the number of lines of text in the workfile.

Example:

```
EDT> STO RRR
RRR          333124     4EE  HALE  F  0800
BB*WRKFL  326568     80FE  HALE  F  0800
BB*WRKFL SAVED      120 LINES
EDT>
```

Errors:

If a file is password protected, you must supply the valid password to store on it. If you do not, the Editor keeps prompting for a valid password. Enter a carriage return to get out of the loop and terminate the command.

5.6.25 The USE Command

The USE command is used to copy a permanent disc file into the current workfile.

The file can be one created or edited previously using the Editor, or the user can access a file that has not been edited previously in the interactive environment. (See Section 5.4.3.) If you attempt to use a file that was not edited previously and has more than 9999 physical records, DELTA should be reset to an increment less than 1.0 or the subsequent records (10,000 and up) will not be brought into the work file. (See the SET DELTA command.)

Syntax:

USE filename [CLEAR] [SCRATCH] [SYS]

where:

filename — is the one- to eight-character name of a permanent file. The records on the file must be blocked. Since the Editor's default assignment for incoming files is a temporary system file which is always blocked, this presents no problem to the user.

CLEAR — is an option which clears the current workfile before copying the contents of the specified file into the workfile. (It performs an identical function to the CLEAR command or a positive response to the CLEAR prompt described below.)

SCRATCH — is the equivalent of CLEAR.

SYS — indicates the file name specified is a system file. If there is a user file and a system file with the same filename and SYS is not specified, the user file is the one brought into the workfile. If SYS is not specified and a system file is found with the file name specified, the following message is displayed:

SYSTEM FILE FOUND BY THAT NAME. CORRECT FILE (Y OR N)?

For further discussion of system files, see Section 5.4.5.

Response:

The workfile is cleared and the contents of the specified disc file are brought in. If CLEAR has not been specified, the Editor prompts:

CLEAR = N

Type N (No) or enter a carriage return to terminate the command or Y (Yes) to continue. The prompt defaults to No to protect the user from inadvertently clearing out edited text that he may want to save. If there is any question of whether or not to clear the workfile, terminate the command. The SHOW command can be used to check the current status of the workfile (CLEAR, SAVED, or CHANGED). CHANGED indicates that edits have been made since the workfile was last saved.

A new or different workfile can be accessed. (See the WORKFILE command.)

If the specified disc file is PO (Password Only) protected, its password is required to access it.

Example:

```
EDT > USE RRR
EDT > USE XXX SYS CLE
EDT >
```

User file RRR is brought into the workfile then system file XXX is brought into the workfile.

If a file specified for USE is PO (Password Only) protected, the password is required to read it. The prompt:

ENTER PASSWORD

is displayed. Enter the valid password followed by a carriage return. If the password is not valid, the USE command terminates.

If the specified file is not a normal source/text file, that is, it does not have a file code type of EE, ED, or C0, the following message is displayed:

FILE TYPE NOT ED, EE, OR C0, PROCESS IT (Y OR N)?

If the user knows the file is indeed source or text, he may try to use it.

5.6.26 The VERIFY Command

The VERIFY command checks the validity of the current workfile.

Syntax:

VERIFY

Response:

The following error messages can be generated by the validation routine:

LINE COUNT ERROR - INTEGRITY UNCERTAIN

POINTER INVALID - INTEGRITY UNCERTAIN

HEADER SEQ INVALID - INTEGRITY UNCERTAIN

SEQ ERR - BAD SECTOR LINKAGE - INTEGRITY UNCERTAIN

INVALID SECTOR NUMBER - INTEGRITY UNCERTAIN

If any of these messages is displayed, the following message is also displayed:

CONTINUING MAY RESULT IN LOST DATA

A short verify is performed before each command and can generate two additional error messages:

NEXT FREE SECTOR IN HEADER AND FREEPAGE DO NOT MATCH

FREEPAGE IS IN HEADER AS AN ACTIVE SECTOR

5.6.27 The WORKFILE Command

The WORKFILE command is used to access a different workfile than the one currently in use. The current workfile is stored back on disc as is (the message describing its status is also retained). The new workfile is created or retrieved. Workfiles are named:

ii*WRKFL

where ii is the workfile code you specify after you activate the Editor or via a WORKFILE command. The file code must be two printable upper case characters uniquely identifying each workfile in the user's directory. The first character cannot be a number, however the second character can, i.e., A9*WRKFL is acceptable.

Syntax:

WORKFILE filecode

where:

filecode specifies the work file code identifying the workfile to be used.

Response:

When you use the WORKFILE command, you can enter the code for the new workfile. If you do not enter the file code, the Editor prompts for it:

ENTER WORK FILE CODE OR CR TO TERMINATE

Either enter a valid two-character workfile code or terminate the command.

If the filecode entered identifies an existing workfile for your user name, EDIT stores the current workfile back on disc and retrieves the specified workfile. The message indicating the current state of the workfile is displayed (SAVED, CLEARED, or CHANGED).

If the code entered does not identify an existing workfile, EDIT assumes that you want to create a new one.

To terminate the WORKFILE command, use a carriage return in response to the prompt for channel and subaddress. The Editor will keep the current workfile.

Example:

To be supplied.

5.7 EDIT Errors

See individual command descriptions. Possible error messages are as follows:

EDITOR FOUND UNPRINTABLE CHARS???
ILLEGAL USE OF AN EDIT RESERVED KEY WORD
ILLEGAL PARAMETER
CANNOT SCRATCH YOUR CURRENT WORKFILE
??? NOTING IN WORKFILE TO BE SAVED!
DISC FILE SPACE UNAVAILABLE
INCORRECT POINTERS IN WORKING FILE
COMMAND IGNORED - JOB QUEUE FULL
NOT A VALID SOURCE FILE
LAST LINE OVERFLOW TRY SMALLER DELTA
FILE IS IN USE BY ANOTHER
ZERO NUMBER DETECTED ON BY COMMAND
INVALID FILENAME OR WORKFILE NAME CANNOT BE SPECIFIED HERE
READ ERROR
FILE IS TOO BIG FOR EDITOR TO HANDLE
WRITE ERROR
BAD COMPRESSED RECORD DETECTED
UNABLE TO ALLOCATE FILE
SEQUENCE ERROR
NO CURRENT DEFAULT FILENAME - PLEASE SPECIFY A FILENAME
NO DISC SPACE AVAILABLE FOR WORKFILE
SYSTEM FILE FOUND BY THAT NAME. CORRECT FILE (Y,N)?
BAD LINE IN WORKFILE - AM RECOUNTING
COULD NOT SCRATCH FILE SPECIFIED
UNEXPLAINED EOM/EOF. DATA LOST. PROCESS AS EOF.
FILE TYPE NOT ED, EE, OR C0. PROCESS IT (Y,N)?
AM EXPANDING WORKFILE. PLEASE WAIT.
NEXT FREE SECTOR IN HEADER AND FREEPAGE DO NOT MATCH
FREEPAGE IS IN HEADER AS AN ACTIVE SECTOR
DETECTED EOM ON WRITE - LOGIC ERROR
(In the event this occurs, generate a SPR detailing events leading up to the generation of this error.)
CREATION FAILED - REASON 7 means directory is full.

6. THE FILE MANAGER (FILEMGR)

The MPX-32 File Manager (FILEMGR) is used to create or delete permanent disc file space, create or delete special Global partitions and/or a DATAPOOL partition (one that can be dynamically allocated in memory when required by tasks), or to provide system and user permanent file backup.

Temporary versus permanent files, types (system and user), allocation, and other basic concepts and supporting constructs for disc files are described in Volume 1, Chapter 7, with the File System Executive (FISE).

Note that files built in the MPX-32 Text Editor are created and expanded without running the FILEMGR utility. Any permanent files can also be deleted (scratched) via the Text Editor.

Note also that files cannot be copied from one user directory to another with the FILEMGR. The MEDIA utility is used for that purpose (see Volume 2, Chapter 10).

6.1 General Description

This section describes the System Master Directory (SMD), compares system files to user files, and describes how the FILEMGR saves and restores files.

6.1.1 The System Master Directory (SMD)

The SMD is located on disc and contains entries for all permanent files and Global Common/DATAPOOL partitions located on all discs configured in a user's system. Each entry in the SMD shows:

file name/partition name

user name, if any

beginning address for the file (block number)

password, if any

number of 192-word blocks in the file or protection granules in the partition

other information used by the system when the file or partition is accessed, e.g., access speed (FAST/SLOW), disc type, channel, subaddress, etc.

Temporary files can be created and used by the system and by user tasks. They are not logged in the SMD although they are tracked by the system. The temporary files are allocated in free space on a disc only for the duration of a task. The FILEMGR does not attempt to allocate permanent files in space concurrently being used by temporary files.

Temporary files cannot be created, deleted, saved, or restored via the FILEMGR. System services are available that enable a task to create temporary files, or to turn a temporary file into a permanent file.

6.1.2 System Files versus User Files

The FILEMGR allows the user to create, delete, save, or restore either system files or files in a specific user directory.

System files can be accessed for reads by any user on the MPX-32 system and are normally load modules, (including SYSTEMS processors, user tasks, etc.) operating system files and library files, or files that the user needs to share among several user names. They have no associated user name.

User files are private and cannot be shared. They are associated with a particular user, whose name is matched against the M.KEY file at logon or after logon. For further description of M.KEY, see the KEY utility, Volume 2, Chapter 7.

User files can be read or written only by the user who logs on or who declares himself/herself as the user via a USERNAME statement (batch, TSM, or FILEMGR). A task can use the M.USER system service to access a particular set of user files or before creating a permanent file in a specific user directory.

When using the FILEMGR, user files are denoted specifically by a U as part of a directive, e.g., SAVEU, RESTOREU, CREATEU, etc. If U is specified, the FILEMGR searches first in the directory of the user last specified in a USERNAME statement, i.e.,

$USERNAME name	(batch)
TSM USERNAME name	(interactive)
USERNAME name	(FILEMGR directive)

If creating a file space with CREATEU, the search for the name stops in the user directory and a new file is created if the name does not already exist.

If saving files, when a file cannot be found in the user's directory, the FILEMGR searches the system directory for the name.

System files (no username) are specifically denoted by a FILEMGR or other USERNAME directive with no user name supplied. The system directory is also assumed for the OPCOM console unless overridden by a USERNAME statement. System files are assumed by default when a command verb does not contain a U.

The directory entry of a file is maintained when it is restored, i.e., the user name used to save the file is the user's directory to which the file is restored.

Global Common/DATAPOOL partitions defined either at SYSGEN or via the FILEMGR belong to the system file catagory.

6.1.3 The Save/Restore Process

When files are saved, the FILEMGR builds a directory containing directory entries for all files saved in a group, where a group is one or more files specified with one SAVE or SAVEU directive. The FILEMGR uses the SMD and copies files in the group to an output medium preceded by the directory.

When files are restored, a directory entry is created for each file to be restored. The FILEMGR locates the file on the input medium and reads it to temporary space on disc. It matches the name against the SMD, and deletes the existing file, if any, that matches the name. The FILEMGR then creates a new permanent file in the SMD for the file that is being restored. The existing user name, if any, is retained from the saved version.

Reading first to temporary space ensures that an I/O error in the restoration process does not result in the loss of existing disc files. The user can opt to bypass this function as described in Section 6.3.

6.2 Files and File Assignments

Default assignments for the FILEMGR cover all needs for running the utility except for saves and restores.

The IN/OUT File or Device - Normally magnetic tape is used to save files for subsequent restoration to disc. Characteristics of disc-to-tape transfer are described in Section 6.4.5.

Job File - Contains Job Control Commands, including ASSIGN's, OPTION's, etc., and FILEMGR directives for the job. For sample job files used in the FILEMGR, see Section 6.9.

The alternative methods for reading the file to SYC (interactive and batch) are described in the File Assignment Table, Section 6.2.1, and Activation, Section 6.5.

6.2.1 File Assignments Chart

Table 6-1, columns 1-3, describes input files used by the File Manager, their associated file codes, if any, and the default assignments, if any. Columns 4-6 relate the File Manager input files to previous use of other processors as applicable. Where it is feasible to override a default assignment or supply more files than the defaults accommodate, columns 3-6 describe options. Output files are also included.

Table 6-1

File Manager File Assignments

Input/Output Description	FILEMGR Logical File Code	Default and Optional Assignments for Cataloging	How Built (Previous Processor Assignment)	How Specified for FILEMGR	Comment
Job File	SYC	Default: SYC=SYC	Work file built using the EDITOR. Permanent file built using EDITOR or MEDIA. Cards or other device media, e.g., magnetic tape, where the job file was copied from cards or a file via MEDIA. Supply the device mnemonic. Interactively. See Accessing the File Manager.	EDI>BATCH EDI>BATCH jobfile or ??BATCH {D,devmnc / F,jobfile}	For further description, see Accessing the File Manager. For further description of device mnemonics, see Appendix A.
Listing/Logs and Audit Trail	SLO	Default: SLO=SLO,6000 Option: Change number of printlines anticipated on SLO.	By FILEMGR. Outputs an audit trail of files saved (or restored).	To change the number of printlines to use on SLO, use ASSIGN2, e.g., ASSIGN2 SLO=SLO,600	
Input (for RESTORE's)	IN	No default IN= {devmnc / filename}	Previous run of FILEMGR. This is the file or device defined as lfc OUT when the files were saved.	ASSIGN3 for device assignments, e.g., ASSIGN3 IN=MT0100, or ASSIGN1 for file assignment	
Output (for SAVE's)	OUT	No default OUT={devmnc / filename}	By FILEMGR, as files are saved.	ASSIGN3 for device assignments, e.g., ASSIGN3 OUT=MT0100,1 or ASSIGN1 for file assignment	

6.3 Options

FILEMGR options can be specified with the $OPTION batch statement or the OPTION command in TSM. Options apply to SAVE, RESTORE, and SAVELOG directives as follows.

Option 1 - The tape assigned to lfc IN is pre-MPX-32. All files restored are assumed to have eight-word, RTM-formatted SMD entries. Eight-word MPX-formatted entries are written to the SMD. The primary change in the SMD entry is in the structure of the space definition. Also, the second halfword of Word 6 is now used. (See Volume 1, Chapter 7.)

Option 2 - Normally, when restoring files, the media copies (lfc IN) are written first to temporary file space on disc as described in Section 6.1.3. This option causes the FILEMGR to delete the existing disc file specified for the restore before copying the saved file back onto disc from lfc IN. For further description, see section 6.1.3.

Option 3 - The user can specify that a file not be saved when it is created. This option overrides that specification and allows NOSAVE files to be saved.

Option 4 - The user can override the username specified when the file was saved. This allows all files on the same tape to be restored to the current username in effect in File Manager. If a single or limited number of file names rather than all the files under a given user name are to be restored, it is sometimes necessary to use the "RESTOREU" to locate the file, rather than "RESTORE".

Option 5 - The user can override the username specified when the file was saved. This allows all files on the save tape to be restored as system files.

Option 6 - The user can override the default to any moving head disc during a restore to be any disc (moving head or fixed head).

6.4 Using the File Manager

6.4.1 Computing the Size of a File

When you use the CREATE or CREATEU directive to establish a file space, you define file size in blocks. A block consists of 192 words (768 bytes). On unblocked files, records are stored one per block, i.e., a 200 block file, unblocked, will contain 200 records.

The maximum record size for a blocked file is 254 bytes. A guide for approximating space required for a blocked file is:

records between 4 and 254 bytes long (1 and 63 words) are packed together up to a block boundary

records cannot span block boundaries

a new file always begins on a block boundary

two header and two trailer bytes are inserted on each packed record automatically for identification and tracking

Thus, in computing the space needed for blocked files, you allow for the extra four bytes in each packed record and for the number of records that can be packed into a block.

Fixed length records under 254 bytes long take up blocks on the file as follows:

$$\frac{768 \text{ (Bytes per Block)}}{\text{Record Length (bytes)} + 4} = \text{Number of Records per Block}$$

$$\frac{\text{Number of Records}}{\text{Number of Records per Block}} = \text{Number of Blocks}$$

For example, if each record is 80 bytes long, each block will hold 9 records (768/84). To hold 2000 records, the file must be 223 blocks long (2000/9, rounded).

The number of blocks required to accommodate variable-length records can be estimated by getting an average byte/record value and using that value as the record length in the setup shown above. For example, if you anticipate having approximately 150 variable-length records in a file, none exceed 254 bytes, and the average is about 50 bytes long:

$$\frac{768}{50 + 4} = 14 \qquad \frac{150}{14} = 11 \text{ blocks (rounded)}$$

Output to all disc files is assumed to be in blocked form unless specified otherwise when a file is assigned or allocated.

6.4.2 Using Wild Card Characters in File Names

When saving or restoring files, a question mark can be used in place of a character in a file name to match any character that falls in its position.

From one to eight question marks can be used.

The total number of characters specified in a filename is the upper limit of characters allowed in matching file names.

For example, using just five question marks as a 'file name' saves all files with five or fewer than five characters in the file names.

Using PJ?????? as a file name saves all files beginning with PJ.

6.4.3 Password-Protected Files

Files can be password-protected via the FILEMGR CREATE or CREATEU directive. Any file defined as Read Only (RO) protected can be read without supplying the password; the password is required to write to it.

Any file defined as Password Only (PO) requires that the password be supplied to access it for any operation.

6.4.4 Special Characters in File Names

If any file specified with SAVE or RESTORE has any of the following characters in the name, the name must be enclosed in single quotes:

```
:
;
(
)
/
```

For example:

```
SAVEU FILE='EM:02'
```

6.4.5 Notes on File-to-Tape Transfers

All SAVE's in a given session apply to the magnetic tape or set of tapes assigned to lfc OUT prior to the EXECUTE FILEMGR directive. (A set of tapes is implied by indicating multivolume on the device mnemonic, e.g., MT0100,SAVE,1, where ,1 implies Volume 1 of n physical tapes.)

When the FILEMGR is ready to execute, it issues a MOUNT message on the OPCOM console prompting the operator to mount an appropriate tape. When the tape is mounted, the operator responds on the OPCOM console and the FILEMGR proceeds.

SAVE's and RESTORE's must be coordinated by the user.

> Any SAVE or SAVEU directive, including SAVE FILE=filename, or SAVE FILE=prototype, prototype ,...n, puts out a group of one or more files on tape with one EOF mark at the end of the group.
>
> If a RESTORE directive in a subsequent session selects files or a group of files from a tape that contains several groups, the FILEMGR must know where that group is located physically on the tape.
>
> The FILEMGR assumes a sequential restoration in the order that files were saved. If files are restored outside the order in which they were saved, the user must use special FILEMGR directives.

Figure 6-1 illustrates the physical result of multiple SAVE/RESTORE operations.

The left side of the figure illustrates SAVE's used to output disc files to a magnetic tape. The right side illustrates how RESTORE's could be used to retrieve the files back to disc.

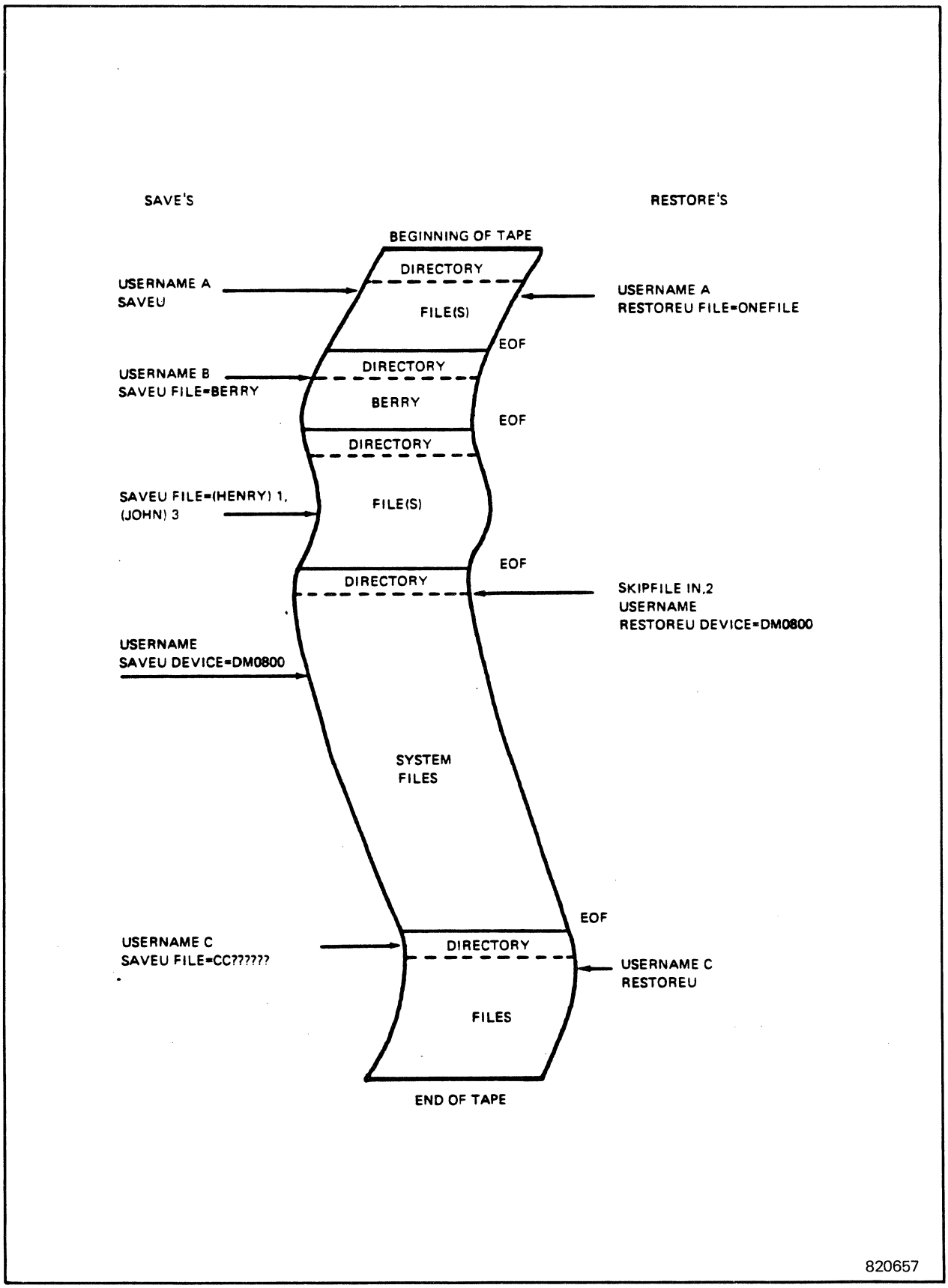


Figure 6-1. File - to - Tape Transfers

One tape contains five groups of files, each with a separate set of directory entries. All files are saved for User A. One file is saved for User B, followed by files from two other users. USERNAME and blank specifies system files. All system files on the moving head disc configured on channel 8, subaddress 00 are saved, then all User C files that begin with CC are saved.

When restoring files from the tape, one file is selected from User A. The FILEMGR goes past the EOF marking the end of User A's files. To restore the system files back to DM0800, the SKIPFILE directive tells the FILEMGR to move past two EOF's to the beginning of the group of system files.

Restoring User C's files requires no special directive, because the FILEMGR is already positioned at the beginning of that group.

The REWIND directive is used if you do not restore files in the same order they were saved, for example, if you were moving back to restore another User A file after User C restoration. Do not use REWIND in the middle of a multivolume restoration.

6.4.6 Device Specifications

Device mnemonics used with the FILEMGR must include the device code and channel. A subaddress is optional. The default subaddress is 00. For further description, see Appendix A.

6.5 Accessing the File Manager

To access the File Manager as part of a batch job, create a job file using the EDITOR, punch cards, or other media as described in Table 6-1. The job file can be read to SYC and the job activated in several ways:

from the OPCOM console:

"<Attention>"

??BATCH { F,jobfile / D,devmnc }

from the OPCOM program:

TSM>OPCOM

??BATCH { F,jobfile / D,devmnc }

from the EDITOR:

```
TSM>EDIT
      .
      .
      .
EDT>BATCH  jobfile
```

If the jobfile is the current EDITOR work file, issue just the BATCH command.

To activate the FILEMGR and run on-line, use the TSM ASSIGN and OPTION commands for FILEMGR assignments and options equivalent to those preceding the EXECUTE FILEMGR command on a jobfile, then proceed to issue FILEMGR directives.

```
TSM>FILEMGR
FIL>CREATEU MYFILE,DC,25,,FAST
FIL>DELETEU GEO.1
FIL>
```

6.6 FILEMGR Directives

File Manager directives are summarized below and described in detail on pages which follow. FILEMGR directives cannot be abbreviated. A directive must begin in column 1 of a record or command line. Commas are legal separators. Blanks are ignored.

Directive		Function
CREATE or CREATEU	6-12	Creates a permanent system file or user file on specified disc.
CREATEM		Defines a dynamic area of memory with a Global Common variable name (GLOBAL 00-99) or the name DATAPOOL. This area can be accessed by M.SHARE (first task activated) and M.INCL (any task thereafter). Can also define a memory partition in the user's extended address space.
DELETE or DELETEU	6-17	Deletes specified system or user file from SMD and deallocates disc space.
DELETEW	6-17	Deletes specified system or user file from SMD and deallocates disc space.

EXIT	6-19	In interactive mode, exits the File Manager and returns to TSM > . In batch, designates the end of File Manager directives in a job stream.
EXPAND or EXPANDU	6-19	Expands or contracts space of a permanent system or user file to n 192-word blocks.
LOG or LOGU	6-20	Lists all permanent files or user files defined in the SMD.
LOGC	"	Indicates collision mapping.
LOGS	"	Lists all system and user files or a subset of files defined in SMD.
MEMO	6-21	Creates a tape from which the dynamic task activation can be performed on a memory-only MPX-32 system.
PAGE	6-22	Puts page eject and header on audit trail.
RESTORE DEVICE or RESTOREU DEVICE	6-22	Restores all permanent system or user files to specified disc from device assigned to lfc IN. (This was the device assigned to lfc OUT when the SAVE command was used.) If a file being restored does not already exist in the SMD, it is added; else it is replaced.
RESTORE FILE or RESTOREU FILE	6-22	Restores specified permanent system or user file(s) to disc from the device assigned to lfc IN (usually magnetic tape). (This was the device assigned to lfc OUT when the SAVE command was used.) If the file(s) being restored already exist in the SMD, existing contents are replaced by the IN contents. If a file does not as yet exist, it is added.
REWIND	6-25	Positions input file or device (lfc IN) or output file or device (lfc OUT) at beginning.
SAVE DEVICE or SAVEU DEVICE	6-26	Saves all permanent system or user files from specified disc (except those created by SYSGEN) on device assigned to lfc OUT.

SAVE FILE or SAVEU FILE	6-26	Saves specified system or user file(s) on device assigned to lfc OUT.
SAVELOG	6-30	Lists files in directory on lfc IN; beginning at current location.
SDT	6-30	Specifies key load modules.
SKIPFILE	6-32	Advances past specified number of EOF's on the file or device assigned to lfc IN or lfc OUT.
USERNAME	6-32	Associates new username with FILEMGR operations.

6.6.1 The CREATE and CREATEU Directives

The CREATE directive is used to allocate file space for a system file. The CREATEU directive is used to allocate file space for a user file. For CREATEU, the file will be created in the directory of the user whose name was most recently supplied in a USERNAME statement (batch, TSM, or FILEMGR). If no username is associated for FILEMGR operations (e.g., if creating a file from the OPCOM console) with no intervening USERNAME directive or if USERNAME b has been specified a system file is created with CREATEU just as if a CREATE directive had been issued.

Syntax:

CREATE [U] filename , devmnc , blocks , [type], [FAST / SLOW], [NZRO], [NSAV]
[, {PO / RO} , password]

where:

filename	specifies the name of the file, eight characters maximum. File names must not contain blanks. Any printable ASCII characters are acceptable. Required.
devmnc	allocates the file on a specific disc by supplying the appropriate disc device code, channel, and subaddress as described in Appendix A. To let the FILEMGR allocate the file on any disc, use just DC as the device mnemonic.
blocks	the size of the file. Specifies the number of 192-word blocks required for it. See Computing the Size of a File, Section 6.4.1.
type	optionally specifies a two-character code to display or print with the file name (e.g., files created in the EDITOR carry the code ED). The code is entered here in hexadecimal, but is output in ASCII. Default is 00.

FAST	the file's directory entry must be retrieved from the SMD in one disc access.
SLOW	the file's directory entry can be retrieved from the SMD in one or more accesses. SLOW is the default.
NZRO	optionally suppresses initializing the disc file space to 0's. Default: the file space is initialized to all 0's.
NSAV	optionally suppresses output of this file when all files on the disc it is contained on are saved via a SAVE DEVICE directive. Default: the file is saved.
RO	the file space is read-only protected. A password must be supplied to write to it.
PO	the file space is read-write protected. A password must be supplied to access it. Default: the file space is not protected.
password	if RO or PO, specifies an eight-character maximum password.

6.6.2 The CREATEM Directive

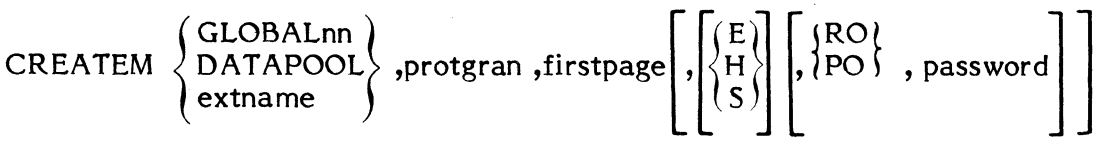
The CREATEM directive is used to define a Global Common partition, a DATAPOOL partition, or a partition in the user's extended address space (above the 128KW logical address space mapped for each task). Memory partitions defined via the FILEMGR are allocated dynamically when required by a task, i.e., they do not remain allocated in physical memory regardless of use as do SYSGEN-defined Global Common partitions or a SYSGEN-defined DATAPOOL partition. To use a memory partition defined via the FILEMGR, tasks must use M.SHARE or M.INCL system services.

A partition defined via the FILEMGR is 8KW minimum on a 32/7x and 2KW on a CONCEPT/32, whereas a SYSGEN-defined partition is structured in protection granule increments (512 words per protection granule).

A particular Global Common name, e.g., GLOBAL02, can be created only once. If created via SYSGEN, GLOBAL02 cannot be created again with the FILEMGR.

DATAPOOL is a good candidate for dynamic allocation and deallocation. MPX-32 has the ability to multicopy DATAPOOL map block(s) into more than one logical address space. If created in the FILEMGR, there can be more than one physical copy of DATAPOOL in memory at a time, depending on the association of tasks that access it simultaneously. Physical space is not taken up permanently (as it is with a SYSGEN-created DATAPOOL partition), thus it is reasonable to have multiple DATAPOOL partitions. Each task structures and shares a given DATAPOOL partition via a DATAPOOL dictionary. All tasks which access the same 'DATAPOOL' do so by specifying the same dictionary during cataloging and by using M.SHARE or M.INCL.

Syntax:



where:

GLOBALnn — creates a Global common partition (00-99) which can be located physically in any class of memory (E, H, or S) and is mapped into the address space of each task that accesses it via an M.INCL system service. One task must use M.SHARE to define the partition as sharable.

DATAPOOL — creates a DATAPOOL partition whose structure is defined via one or more DATAPOOL dictionaries. Like Global Common, the DATAPOOL area can be located physically in any class of memory (E, H, or S) and mapped into the logical address space of each task that accesses it via the M.INCL system service. One task must use M.SHARE to define the partition as sharable. (The same dictionary must be used in cataloging each task associated with M.SHARE or M.INCL.)

extname — is any one-to-eight character name to use for a memory partition in a task's extended address space. This partition may be mapped into memory above the first 128KW logical address space available to a task. Since the partition will be in extended memory, certain restrictions will apply. Refer to the MPX-32 Reference Manual Volume 1, Section 2.9.4.4 for programming restrictions that affect the use of extended address space.

Partitions in a task's extended address space can be located in any class of physical memory (E, H, or S).

The name used for a partition that is allocated in extended address space must not be GLOBALnn or DATAPOOL.

protgran — specifies the number of 512-word protection granules to include in the partition. (Sixteen protection granules equal one map block on a 32/7x and four protection granules equal one map block on a CONCEPT/32). Unused physical protection granules within the last 8KW map block on a 32/7x or 2KW on a CONCEPT/32 allocated to the partition will be write-protected from all sharing tasks. However, only one dynamic partition may be defined in any one map block.

firstpage — specifies the starting protection granule in the nonextended logical address space (pages 0-255) or in the extended address space (pages 256-479 on a 32/7x and 256-1019 on a CONCEPT/32) where the partition is to be mapped. Protection granules in the first several map blocks should not be specified, as they are used for the MPX-32 operating system.

Protection granules for GLOBAL and DATAPOOL partitions are normally allocated from top down in a task's logical address space, or below any SYSGEN-created common partitions. In a 32/7X, the last map block of extended address space is reserved for MPX-32 use. In a CONCEPT/32, the last two map blocks of extended address space are reserved for MPX-32 use. See Volume 1 for further description of the various structures of logical address spaces available to the user.

E — allocate the partition physically in Class E memory (first 128KW). If Class E not available, wait.

H	high speed. Allocate the partition in Class H memory. If H or E not available, wait.
S	slow. Allocate the partition physically in any class memory (E, S, or H). If no memory available, wait. Default: S.
RO	the partition is read-only protected. Tasks are not allowed to write to it without supplying a password.
PO	the partition is read and write protected. A password must be supplied to access it.
password	supplies the password, eight-characters maximum.

6.6.3 The DELETE and DELETEU Directives

The DELETE directive is used to scratch a system file from disc and free the disc space. The directive also removes the directory entry for the file from the SMD.

The DELETEU directive is used to delete a file from the directory of the user whose name was supplied in the most recent USERNAME statement (batch, TSM, or FILEMGR).

Syntax:

DELETE [U] filename [,password]

where:

filename	is the name of the file to delete.
password	is the password associated with the file, if any.

6.6.4 The DELETEW Directive

The DELETEW directive enables the user to scratch more than one file per directive from the disc, free disc space, and remove the directory entry for each file from the SMD. Up to 20 file prototypes can be specified per directive, continued on several lines or cards with each line or card containing a comma as the last non-blank character.

This directive can be used to delete system files and user files. However, user files with the user name SYSTEM must be deleted with the DELETE U directive because the DELETEW directive interprets the word SYSTEM as meaning a system file.

There are no defaults with the DELETEW directive. For each file set to be deleted, the word SYSTEM or a user name and the respective file name must be specified in the directive (see examples).

Syntax:

DELETEW [FILE=] prototype [,prototype]...

where prototype identifies a file set as follows:

(username[,key])['']filename['] [;password]

where:

username	can be changed for each file set in a directive. The user name and optional key must be enclosed in parentheses. The pseudo user name SYSTEM is used to specifically indicate system files and must be enclosed in parentheses.
filename	name of file to delete. A question mark (?) can be used in place of any character(s) of the file name to match any character in that position.

If a filename contains any of the following characters, the file name must be enclosed in single quotes:

: ; ( ) or /

;password — supplies the password required to access a file if it is either PO (Password Only) or RO (Read Only) protected.

Examples:

Example 1 - This example demonstrates how to delete all non-password protected system files that begin with GS. and WD/.

```
TSM>A4 SLO UT
TSM>FILEMGR
FIL>DELETEW FILE=(SYSTEM)GS.?????,(SYSTEM)'WD/?????'
FIL>X
TSM>
```

Example 2 - This example demonstrates how to delete all files belonging to user name FADEN with user key F and password SECRET.

```
TSM>A4 SLO UT
TSM>FILEMGR
FIL>DELETEW (FADEN,F)????????;SECRET
FIL>X
TSM>
```

Example 3 - This example demonstrates how to delete the file RISK belonging to user name HOLLY and all non-password protected system files that begin with GC..

```
$JOB
$EXECUTE FILEMGR
DELETEW FILE=(HOLLY)RISK,(SYSTEM)GC.?????
EXIT
$EOJ
```

6.6.5 The EXIT Directive

The EXIT directive is used to exit the File Manager and return to the TSM prompt when running in interactive mode.

When running in batch mode, the EXIT directive signifies the end of File Manager directives in a job stream.

Syntax:

EXIT

6.6.6 The EXPAND and EXPANDU Directives

The EXPAND directive is used to increase or decrease the size of an existing system file. The FILEMGR allocates a new file space on the disc that already contains the file, copies the existing contents of the file to the new space, deallocates the original file space, and updates the SMD. If the file space is increased in size, additional space is zeroed. If decreased in size, any contents that exceed the new space are truncated.

The EXPANDU directive expands a file from a specific user directory. The user name is the name supplied in the most recent USERNAME statement (batch, TSM, or FILEMGR).

Syntax:

EXPAND[U] filename, blocks[,password]

where:

filename	is the name of the file to expand or contract.
blocks	the size of the file. Specifies the number of 192-word blocks to allocate for the file. (For further description of file size, see Section 6.4.)
password	is the password associated with the file, if any.

6.6.7 The LOG, LOGU, LOGC, and LOGS Directives

The LOG and LOGU directives are used to obtain a list of all permanent files defined in the System Master Directory (SMD) or all user files, respectively. The data output includes the file name, the device on which the file resides, FAST/SLOW file access, file size, and starting address of the file.

Syntax:

LOG [U] [C][[FILE=]prototype][,prototype]...

where:

	If no parameters are specified, the resulting list contains data on all permanent files defined in the SMD.
U	indicates active user files of the username currently associated with the File Manager.
C	indicates collision mapping by printing a 'C' in the rightmost position on the audit trail.
FILE=	limits the list to a specific file or set of files. Up to 20 file prototypes can be specified with a single LOG directive. Note: prototypes can be continued on several lines (or cards). Each line or card must have a comma as the last non-blank character.
prototype	identifies a subset of files are described in the RESTORE and RESTOREU directives syntax.

The LOGS directive lists all system and user files or a subset of files that currently exist in the System Master Directory (SMD). For each file, LOGS indicates the device and sector number. It flags overlapping files. It can be used before and/or after other FILEMGR directives.

Syntax:

LOGS

6.6.8 The MEMO Directive

The MEMO Directive is used to create a tape or floppy disc from which dynamic task activation can be performed on a memory-only MPX-32 system.

During dynamic task activation, the system allocator reads the preamble (first 192-word block) of the first load module on the input medium. If the load module name in the preamble is the same as the load module name specified in this directive, the remainder of the load module is loaded. If the names do not match, the allocator advances down the input medium to the preamble of the next load module. This process continues until either the appropriate load module name is found or an end-of-file (written by the File Manager) is encountered. In either case, the input medium will be rewound and deallocated when task loading has completed.

Syntax:

MEMO loadmod[,loadmod],...

where:

loadmod	is the load module name of the task to be activated. A maximum of 20 load module names can be specified per tape or floppy disc.

6.6.9 The PAGE Directive

The PAGE directive is used to force a page eject and output a header on the audit trail. (A header is output automatically for the first page of the audit trail.)

Syntax:

PAGE

6.6.10 The RESTORE and RESTOREU Directives

The RESTORE directive copies system files saved via the SAVE directive back to disc. A tape or other media assigned to lfc OUT with SAVE is assigned to lfc IN to restore the files. For further description, see Section 6.4 The RESTORE directive can be used to restore:

all system and user files on IN to any available moving head disc

all system and user files on IN to a specific disc or type of disc

an arbitrary list of system (and user) files from IN to a specific disc or type of disc

The RESTOREU directive restores files from IN belonging to a particular user back onto disc (under the same user name). The user name used to retrieve the files is the name supplied in the most recent job control USERNAME statement (batch, TSM, or FILEMGR). With RESTOREU, you can restore:

all files belonging to a particular user or all system files (no user name associated) from IN to any available moving head disc

all files belonging to a particular user or all system files (no user name associated) from IN to a specific disc or type of disc

an arbitrary list of user and/or system files from IN to a specific disc or type of disc

With a list, files from other user names can also be restored.

Syntax:

RESTORE[U][[DEVICE=devmnc],][[FILE=prototype][,prototype] ,...]

where:

If nothing is specified, files are restored to the disc from which they were saved. If that disc is not available, files are restored to any available disc as if the DC device mnemonic had been used.

DEVICE — used to restore the files on lfc IN to a specific disc or type of disc.

devmnc — the device, channel, and subaddress of the disc(s) as described in Appendix A.

Files can be restored to any available disc by using the device mnemonic DC.

FILE= — limits the restoration to a specific file or set of files. Up to 20 file prototypes can be specified with a single RESTORE directive. Note: prototypes can be continued on several lines (or cards). Each line or card must have a comma as the last non-blank character.

prototype — identifies a file or file set as follows:

[(username)] ['] filename ['] [;password]

where:

username — can be changed for each file or file set in a list. The username must be enclosed in parentheses. Default: if RESTORE, no user name; if RESTOREU, will save specified file or files from directory of user last specified in TSM, batch, or FILEMGR USERNAME statement.

filename — name of file to restore. A question mark (?) can be used in place of any character(s) of the file name to match any character in that position. See Section 6.4.

' — if a file name contains any of the following characters, the file name must be enclosed in single quotes:

: ; ( ) or /

;password — if the file is either PO (Password Only) or RO (Read Only) protected, supplies the password required to access it.

Examples:

```
RESTORE
```

Comment:

Restores all files from lfc IN back to disc. If a file on IN does not currently exist, it is created and added to the SMD. Files are restored to the disc from which they were saved.

If a file already exists, it is replaced by the version on the device assigned to lfc IN.

```
USERNAME MOORE
RESTOREU FILE = DEBUG.?
USERNAME JOHNSON
RESTOREU
```

Comment:

Restores all files belonging to Moore named DEBUG.n to disc, where n can be any character or no character.

Restores all files on IN belonging to Johnson.

```
RESTORE DM0100
```

Comment:

Restores all system and user files on IN to disc DM01. If some of the files exist on other discs, they are deleted from those discs after they are safely restored on DM01.

6.6.11 The REWIND Directive

The REWIND directive is used to rewind a magnetic tape. It is used primarily when restoring files from a tape in an order different than the order they were saved as described in Section 6.4.

The FILEMGR does not rewind a tape automatically at the end of either a set of SAVE's or RESTORE's. If the tape has not been rewound offline, it can be rewound via this directive.

Syntax:

REWIND { IN / OUT }

where:

IN	specifies the device assigned to lfc IN.
OUT	specifies the device assigned to lfc OUT.

6.6.12 The SAVE and SAVEU Directives

The SAVE directive is used to back up permanent disc files on the medium associated with lfc OUT. Normally files are saved on, and restored from, magnetic tape.

The SAVE directive can be used to back up:

all files on all discs configured in the system

all files on a particular device

an arbitrary list of system and user files, where no user name supplied defaults to a system file

Or with SAVEU, you can back up:

all files for a particular user (all discs or a specific disc)

all system files (all discs or a specific disc)

an arbitrary list of user and system files, where no username supplied defaults to the most recent USERNAME statement (batch, TSM, or FILEMGR)

The SAVEU directive saves files taken from the directory of the user whose name was supplied in the most recent USERNAME statement (batch, TSM, or FILEMGR). To save just system files, use USERNAME without any name supplied.

As files are saved, the FILEMGR builds a directory containing essentially the same information contained on the SMD. It outputs this directory at the beginning of each group of files saved on the medium assigned to lfc OUT. An error message and a zero-filled block on OUT indicating end-of-file (EOF) is produced if a save directive is specified and no files are saved, i.e., SAVE DEVICE=DM0801.

An 'audit trail' of all files saved in a particular FILEMGR session is provided automatically on SLO. This list should be kept and used in restoring the files, because the sequence in which files are saved is important when they are restored. For further description, see Sections 6.4 and 6.7.

To increase or decrease the number of blocks in the saved version of a file, its new size (number of blocks) can be specified when it is saved. The file space will be recreated when the file is restored and will be the size you specify in the save. The ability to modify the size of a file is available only when saving a specific file.

Syntax:

SAVE[U] [{DEVICE=devmnc / FILE=prototype ,prototype ,...}]

where:

if no parameters are supplied, all system and user files (SAVE) or all user files (SAVEU) without password protection are saved on lfc OUT.

DEVICE= limits the saved files to a particular disc.

devmnc — the device code, channel, and subaddress of the disc as described in Appendix A.

FILE= — limits the SAVE to a specific file or set of files. Up to 20 file prototypes can be specified. Note: prototypes can be continued on several lines (or cards). Each line or card must have a comma (,) as the last non-blank character.

prototype — identifies the file or file set as follows:

[(username)] ['] filename ['] [;password][/SIZ:blocks]

where:

username — can be changed for each prototype.

The username must be enclosed in parentheses.

Default: SAVE, no username, i.e., will look for system files.

Default: SAVEU, the username supplied in the most recent USERNAME statement (batch, TSM, or FILEMGR). If cannot match the file in the user directory, will search for a system file.

Or the pseudo username SYSTEM can be used specifically to indicate system file(s).

filename — name of file to save. A question mark (?) can be used in place of any character(s) of the file name to match any character in that position. See Section 6.4.

' — if a file name contains any of the following characters, the file name must be enclosed in single quotes:

: ; ( ) or /

;password — if the file is password protected, supplies the password required to read it.

/SIZ:blocks — specifies the desired size of the file in number of 192-word blocks. If existing file is larger, truncates the saved version; if existing file is smaller, add blocks on the saved version and initializes them to 0's.

Default: the size of the existing file is the size of the saved version.

For further description of file size, see Section 6.4.

Examples:

```
SAVE
```

Comment:

Saves all files (system and user) on all discs configured in the system.

```
SAVE DEVICE=DM0100
```

Comment:

Saves all system and user files from the moving head disc on channel 01, subaddress 00.

```
USERNAME B
SAVEU
```

Comment:

Saves all files belonging to User B on all discs configured in the system.

```
USERNAME
SAVEU
```

Comment:

Saves all system files from all discs configured in the system.

```
USERNAME BROWN
SAVEU FILE=PJ$?????,(MOORE)DEBUG.?,EXPR
```

Comment:

Saves all the files in Brown's directory that start with PJ$, plus the file named EXPR. Also saves all files named DEBUG. from the directory of a user named MOORE. All DEBUG. files are saved, regardless of the character following the dot. If a DEBUG. file in Moore's directory contains an eighth character, the file will not be saved. (See Section 6.4.2.)

```
USERNAME MILLER
SAVEU FILE=?????,'FTN:SRC'
```

Comment:

Saves all files from the Miller directory that have five or less characters in the file name. Also saves the Miller file named FTN:SRC, which is enclosed in single quotes because of the embedded colon in the file name.

```
USERNAME SMITH
SAVEU
USERNAME MOORE
SAVEU
```

Comment:

Saves all files in Smith's directory, then saves all files in Moore's directory.

6.6.13 The SAVELOG Directive

The SAVELOG directive can be used when restoring files from magnetic tape or before restoring files, simply to check the contents of a tape. It displays (or lists) the files grouped in the current directory on the tape assigned to lfc IN. After listing those files, it returns the tape to the beginning of the current directory.

This directive is useful during restoration because it allows you to match restore directives against the actual saved files on a tape, or it can be used simply to check the contents of a tape, e.g., to ensure that the right tape is mounted for the files you want to restore.

If a tape contains several directories, the SKIPFILE directive can be used to get to and list the next directory, e.g., if a tape had three directories,

```
SAVELOG
SKIPFILE IN
SAVELOG
SKIPFILE IN
SAVELOG
REWIND IN
```

would output all directory entries to SLO. Or if SAVELOG is inserted between RESTORE directives, each directory list would precede the RESTORE operations shown on SLO.

Syntax:

```
SAVELOG
```

Example:

To be supplied

6.6.14 The SDT Directive

The SDT directive is used to specify key load modules. It also checks to ensure that each file specified is a valid load module.

Syntax:

```
SDT sysfile,BOOTxx,[loadmod],...
```

where:

sysfile	specify the load module that contains the resident operating system. You will have run SYSGEN to create a tailored version of the operating system and will specify the name of the output file generated with SYSGEN as the first load module. (Supply the file name used with the SYSGEN SYSTEM directive.)

BOOTxx — specify the SYSTEMS-supplied load module, BOOT7X or BOOT27, which loads the resident system onto disc. BOOTxx also provides restart logic on disc for reloading from disc. It is required for all configurations of the SDT.

loadmod — specify key load modules required to get a system that the user can communicate with.

For MPX-32, absolute minimum load modules are: FILEMGR, OPCOM, J.INIT, J.SOUT, and J.TSM. Additional load modules should be included with the key modules or the SDT to maximize convenience for the user: J.JOBC, J.SSIN, JSSIN2, J.TINIT. These are followed by an EOF written by the FILEMGR after the last load module specified with SDT.

For memory-only MPX-32 systems, the SDT is built as follows:

SDT sysfile,BOOTMEMO[,OPCMM] [,loadmod],...

where:

sysfile — specifies the name of the load module that contains the resident operating system created by running SYSGEN. This is the file named used wit the SYSGEN SYSTEM directive.

BOOTMEMO — specifies the SYSTEMS-supplied load module 'BOOTMEMO' which performs the necessary system initialization and creation of the task SYSBUILD which in turn performs the loading and activation of user tasks supplied on the SDT.

OPCMM — specifies the memory-only operator communications task. This module is created by assembling the source module SJ.OPCOM into the object module OJ.OPCOM under the user name MEMONLY, with the memory-only conditional assembly flag (C.MEMO) set with the MPX-32 macro library. The resulting object module is then cataloged as the load module OPCMM so as not to erase the MPX-32 operator communications task (OPCOM). This module is optional, but if desired, it must be the first task name on the SDT after BOOTMEMO.

loadmod — specifies the load module names of user tasks to be loaded from the SDT and activated after system initialization. Up to 18 tasks (17 if OPCMM is specified) can be included in the SDT directive for immediate loading and activation.

Note: The SDT cannot be multi-volume. The boot loader is not capable of processing the header record.

6.6.15 The SKIPFILE Directive

The SKIPFILE directive is used to advance past one or more end-of-files (EOF's) on the file or device assigned to lfc IN or lfc OUT.

Syntax:

SKIPFILE $\left\{\begin{matrix}\text{IN}\\\text{OUT}\end{matrix}\right\}$ [,n]

where:

n	number of EOF's to skip. Default: one.

Example:

For sample use, see Figure 6-1 and accompanying text and the SAVELOG description.

6.6.16 The USERNAME Directive

The USERNAME directive is used to establish a directory to associate with subsequent FILEMGR directives.

Initially, if running from a terminal via TSM, the user name is defaulted to the owner name established at logon. If running from the OPCOM console, no user name is established (implying the user name 'SYSTEM'). The initial user name can be changed by supplying a different username in any USERNAME statement (batch, TSM, or FILEMGR).

The USERNAME directive without any name supplied associates system files with subsequent FILEMGR directives.

Syntax:

USERNAME [username] [,key]

where:

username	is the name of a valid user on the M.KEY file. Default if no name supplied is no user name, i.e., system files.
key	specifies a valid key if required to use this user name.

6.7 Examples

Example 1 - This example saves all user files beginning with XY onto a tape.

```
TSM >ASSIGN3 OUT=MT
TSM >ASSIGN4 SLO=UT
TSM >FILEMGR
FIL> SAVEU FILE=XY??????
FIL> EXIT
TSM>
```

Example 2 - This example creates a fast file named TEST of 100 sectors.

```
TSM> ASSIGN4 SLO=UT
TSM> FILEMGR
FIL> CREATEU TEST,DM,100,,FAST
FIL> EXIT
TSM>
```

Example 3 - This example restores the tape files in Example 1 back to disc.

```
EDT> COL
$JOB
$ASSIGN3 IN=MT
$EXECUTE FILEMGR
RESTOREU FILE=XY??????
EXIT
$EOJ
$$
EDT >RUN
```

6.8 Errors

See Appendix C.

6.9 Listings

The FILEMGR automatically outputs an audit trail of all directives issued, resulting operations, and errors on SLO. In addition, if you use a FILEMGR LOG, LOGU, and/or SAVELOG directive, those results are output to SLO.

On SAVE's it is a good idea to retain the audit trail listing so that you have the right order of saved files on the tape to use in restoration.

7. M.KEY EDITOR (KEY)

KEY is a utility used to build an M.KEY file for the MPX-32 system. The M.KEY file specifies valid owner names/usernames* on the system and optionally sets, for each owner name/username:

a key to restrict access to the owner name during logon and to restrict access to the user name when accessing files

OPCOM indicators restricting the owner's use of OPCOM commands

an indicator that prevents the owner from cataloging 'privileged' tasks (tasks that use privileged system services or privileged variations of unprivileged system services)

an indicator that prevents the owner from activating tasks cataloged as privileged

default tab settings

default alphanumeric project names/numbers for accouting purposes

After KEY has been run, only those owners/users established in the M.KEY file are allowed to log on to the system and access files. The owner name CONSOLE cannot be restricted in any way. CONSOLE is also the one exception to an owner name being identical to a user name. There is no user name associated with the CONSOLE owner.

7.1 General Description

KEY processes an input file containing ownernames/usernames, keys, and other information described previously and outputs the data to the M.KEY file referenced by the system at logon.

*Owner names are associated with tasks in the System Dispatch Queue and, except for jobs, cannot be changed from logon through logoff. Usernames are associated with file names in the System Master Directory (SMD) and can be changed via USERNAME commands/directives in Job Control, TSM, and various utilities (as documented).

7.2 Files and File Assignments

The primary file required to use KEY is an input file containing M.KEY information for each owner.

The structure of the input file is described in Section 7.4

The input file is prepared using the EDITOR STORE UNNUMBERED command. It is assigned to the logical file code INP, as in:

```
ASSIGN1 INP=filename
```

A system file named M.KEY is also required. It can be password-protected. This is the output file used by the KEY utility and accessed by the system. The file code used by KEY for this file is OUT, as in:

```
ASSIGN1 OUT=M.KEY,pass,U
```

Output to M.KEY must be unblocked.

M.KEY file space can, for example, be created and password-protected via the FILEMGR.

7.3 Using KEY

This section describes how to build the input file for KEY. For ease of use, the input file should be built as a system file, as in:

```
EDT>COLLECT
.
.
.
EDT>STORE INFILE UNN SYS
```

7.3.1 Input Record Syntax

ownername,key,opcommands,opcommands,tabs,tabs,projno

where:

ownername — one-to-eight character name used to log on the MPX-32 system. Also provides the user name for file access.

key — one-to-eight character key to associate with the owner name/user name.

opcommands — hex characters representing the bit pattern of OPCOM commands and privileges available to this owner name. Two words are used, and are separated by a comma or blanks. The bit settings in a word can optionally be left unchanged by entering a zero for the word; however, if you set one bit in a word, all bit settings in that word must be specified.

If a bit is set, a function or privilege is denied to tasks with this ownername.

The module number of each command verb corresponds to its bit position. For example, the command verb "MODIFY" (module OC18), is represented by bit 18. Module numbers are indicated below.

With the exception of EXIT (see Volume 1, Section 4.1.4), if a command verb bit is set, the command is not available to an owner name and generates an "INVALID COMMAND VERB" message.

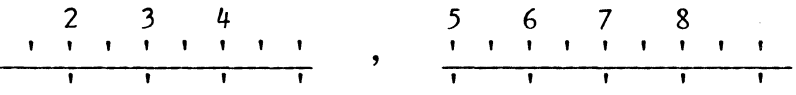
Bit	Command
00	EXIT
01	ABORT
02	ACTIVATE
03	BATCH
04	BREAK
05	CONNECT
06	CONTINUE
07	DEPRINT
08	DEPUNCH
09	DISABLE
10	DISCONNECT
11	DUMP
12	ENABLE
13	ENTER
14	HOLD
15	KILL
16	LIST
17	MODE
18	MODIFY
19	OFFLINE
20	ONLINE
21	PURGEAC
22	REDIRECT
23	REMOVE
24	REPRINT
25	REPUNCH
26	REQUEST
27	DELETETIMER
28	SAVEAC
29	SEARCH
30	SEND
31	SETTIMER
32	SNAP
33	START
34	STATUS
35	SYSASSIGN
36	TIME
37	URGENT
38	RESUME
39	ESTABLISH

Bit	Privilege
40	Disables access to tasks with a different owner name.
41	Disables activation of privileged tasks other than OPCOM (Note: With this bit set File Manager cannot be activated).
42	Disables cataloging of load modules as privileged. To use the privileged option will be considered illegal and the cataloger will abort the job.
43	Disables use of the TSM RESTART command.
44-63	Reserved.

tabs — hex characters representing the tab positions for this owner name. Eight tabs can be set. Each word defines four tab positions. The words are separated by a comma or blank. The first byte of 00 indicates end of tabs. Tabs must be entered in ascending order and not exceed the width of a terminal line.

projno — one-to-eight character alphanumeric project name/number to associate with the owner name.

Tab Stop Setting (Hex)



7.3.2 Sample Input File

1. GIPSON,DG

   HAWK,RH,0,0,151A2640
2. BEVIER,JB,0,0,0A142430,36400000,ALLFILES
   FADEN,GF,0,0,1A152640
   MYERG

3. MEYERS,M,40002000,10000000
   PARIS,C
   HALE,C
   MOORE

Notes:

1. Owners/users with only name and key and no other specifications have access to all OPCOM commands, have no default tabs (until they enter the Editor), and no privilege restrictions.

2. This owner has access to all OPCOM commands, default tab settings for tabs in positions 10, 20, 36, 48, 54, and 64, and a default project name established as ALLFILES.

3. This owner cannot use the following OPCOM commands: ABORT, MODIFY, and SYSASSIGN.

7.4 Accessing the M.KEY Editor

To access KEY, assign appropriate files to IN and OUT as described in Section 7.2. Then activate KEY. To do so as part of a batch job, create a job file using the the EDITOR, punch cards, or other media. The job file can be read to SYC and the job activated in several ways:

from the OPCOM console:

" <Attention> "

??BATCH { F,jobfile / D,devmnc }

from the OPCOM program:

TSM > OPCOM

??BATCH { F,jobfile / D,devmnc }

from the EDITOR:

TSM > EDIT
.
.
.
EDT> BATCH [jobfile]

If the jobfile is the current EDITOR work file, issue just the BATCH command.

To activate KEY and run on line, use the TSM ASSIGN command for KEY assignments, then activate KEY.

TSM > KEY
TSM >

7.5 Example

How to build the input file is described in Section 7.3. A sample input file (INFILE) is described in Section 7.3.2.

```
TSM > ASSIGN1 INP=INFILE
TSM > ASSIGN1 OUT=M.KEY,KEYPASS,UNBLOCKED
TSM > KEY
TSM >
```

8. THE SUBROUTINE LIBRARY EDITOR (LIBED)

The System Subroutine Library (LIBED) is a set of assembled object modules available from SYSTEMS. It is called MPXLIB and contains FORTRAN math subroutines and I/O formatting routines. The corresponding directory for MPXLIB is named MPXDIR. The library object modules can be accessed as externals by tasks written in various languages, including Assembly language. The Subroutine Library Editor (LIBED) is used to add object modules to the library.

The user can also use LIBED to create his own library of object modules and a directory. He can then access modules within the library as externals. LIBED is also used to delete existing modules from a library.

References to subroutines (object modules) in the library (MPXLIB and/or user) are resolved when a task that accesses them is cataloged. (See the Cataloger, Volume 2, Chapter 2.)

8.1 General Description

With LIBED, the object modules to include on a library or to replace on an existing library are assigned to the logical file code Library General Object (LGO), where the default assignment is to SGO. This allows the user to assemble or compile one or more object modules with output to SGO and use SGO as input to LIBED in the same job, if desired.

LIBED has three directives: LOG is used to specify if a list of all module names and external definitions is desired, DELETE is used to specify modules to be deleted, and EXIT is used to terminate directive input and cause previous directives to be processed.

Existing modules are replaced with new modules automatically whenever the replacement module exists on the file assigned to LGO.

Use of the $OPTION statement determines whether a library is being created or not, as described in Section 8.3. If creating a library, LIBED does not process directives.

The subroutine library directory is used by the Cataloger to locate modules on a given library.

8.1.1 LIBED Directives Summary

Directive	Function
DELETE	Deletes a module from the library.
EXIT	Terminates directive input.
LOG	Provides a log of all modules and their external definition.

8.2 Files and File Assignments

The following logical file codes are used for LIBED assignments. Note that a file or device assigned to LIB or DIR must be assigned as unblocked. All other file or device assignments are blocked (the MPX-32 default).

8.2.1 The Object Module File (LGO)

LIBED takes input from the file or device assigned to LGO. This file contains assembled or compiled object modules. The default assignment is LGO = SGO.

8.2.2 The Directive File (CTL)

The LOG or DELETE directives are supplied on the file or device assigned to CTL. The default assignment is CTL = SYC.

8.2.3 The Subroutine Library File (LIB)

Object modules are output to the file or device assigned to LIB. If object modules are being deleted, the file or device assigned to LIB is both the input and output source. Object modules are automatically replaced if they exist on the file or device assigned to LGO as well as on the file or device assigned to LIB.

8.2.4 The Directory (DIR)

The file or device assigned to DIR is used for the library directory. The directory is created or maintained by LIBED. The user must be certain that the proper directory is paired with the corresponding library when making assignments for LIBED or the Cataloger.

8.2.5 Listed Output (LLO)

The file or device assigned to LLO is used for LIBED listings. (See Section 8.7.) The default assignment is LLO = SLO,1000.

Table 8 - 1
LIBED File Assignments

Input/Output Description	Logical File Code	Assignments for LIBED	Previous Processor Assignment	How Specified for LIBED	Comment
Object Module Input File	LGO	Default: ASSIGN2 LGO = SGO Options: ASSIGNn LGO = {filename / devmnc}	The Assembler defaults output to an SGO file. It is used automatically if assembling code in the same job, or output from the Assembler can be directed to a device or permanent file and accessed by assignment to LGO.		
Directives	CTL	Default: CTL = SYC	Work file built using EDIT. Permanent file built using EDIT or MEDIA. Cards. Other device medium e.g., magnetic tape, where jobfile was copied from cards or a file via MEDIA. Interactively. See "Accessing LIBED."	EDT>BATCH EDT>BATCH jobfile ??BATCH {D,devmnc / F,jobfile} Same route shown for cards.	For further description see "Accessing LIBED"
Subroutine Library	LIB	Default: LIB=MPXLIB Options: ASSIGNn LIB = {MPXLIB / userlib},U	If disc file is used for the library it must be pre-established via the FILEMGR utility. If the size of a file is increased, it must also be handled via the FILEMGR.		The Subroutine Library and directory created via LIBED are used when a task is cataloged that accesses object modules on the library. Both files must be unblocked.

Table 8 - 1 (Cont'd)
LIBED File Assignments

Input/Output Description	Logical File Code	Assignments for LIBED	Previous Processor Assignment	How Specified for LIBED	Comment
Library Directory	DIR	DIR=MPXDIR,,U Options: ASSIGNn DIR = {MPXDIR / userdir},,U	A directory is produced for each subroutine library. The directory (e.g., MPXDIR) already exists or is to be created via this run of LIBED.		See above.
Listed Output	LLO	ASSIGN2 LLO = SLO,1000 Options: ASSIGNn LLO = {filename / devmnc}	N/A		

8.3 Options

If no options are specified, LIBED processes directives from the file or device assigned to CTL. If an object module on the file or device assigned to LGO has the same name as an existing module on the library (LIB assignment), it is replaced. If not, it is added to the library. LIBED updates the directory.

LIBED options are specified through the job control statement, $OPTION or the TSM OPTION command.

OPTION 1 or $OPTION 1 is used to create a new library and directory from the object modules in the file assigned to LGO. Processing under this option ignores the current contents of the library and directory (if any) and ignores the file assigned through the CTL file code. When this option is specified, a LOG of all module names and external definitions is produced automatically.

OPTION 2 or $OPTION 2 is used to make only a statistics run. It produces an analysis of allocated and remaining disc space in the library and directory assigned to file codes LIB and DIR.

8.4 Using LIBED

Subroutine libraries are a convenient mechanism for accessing code (object modules) used by different tasks. Up to seven libraries can be accessed when a task is cataloged.

To be used most efficiently in a subroutine library, an object module should be given a specific, unique name when it is assembled or compiled (PROG = uniquename), so that is can be accessed with a Cataloger PROGRAM directive. Also, cataloger INCLUDE and EXCLUDE directives access object modules by the name supplied in Assembler DEF statements. These names should be unique to avoid problems.

Note that the name 'subroutine' library is really a misnomer, as both main segments (programs) and subroutines (definitions) can be included and edited on the library, i.e., they are all discrete 'object modules'.

8.5 Accessing LIBED

To access the Subroutine Library Editor as part of a batch job, create a job file using the EDITOR, punch cards, or other media as described in Table 8-1. The job file can be read to SYC and the job activated in several ways:

from the OPCOM console:

" <Attention> "

??BATCH { F,jobfile / D,devmnc }

from the OPCOM program:

TSM > OPCOM

??BATCH { F,jobfile / D,devmnc }

from the EDITOR:

```
TSM > EDIT
        .
        .
        .
EDT > BATCH [jobfile]
```

If the jobfile is the current EDITOR work file, issue just the BATCH command.

To activate the Subroutine Library Editor and run online, use the TSM ASSIGN and OPTION commands to make the Subroutine Library Editor assignments and chose options equivalent to those preceding the EXECUTE LIBED command on a jobfile, then proceed to issue Subroutine Library Editor directives, if applicable. (SELECT and OBJECT statements are not available when running the Subroutine Library Editor online.)

```
TSM> LIBED
LIB>
```

8.6 Subroutine Library Editor Directives

All directives are read from the device assigned through logical file code CTL. All directives begin in column 1 and fields are terminated by blanks.

8.6.1 DELETE Directive

The DELETE directive causes the specified program (object module) to be deleted from the library assigned to lfc LIB, and its external definitions to be removed from the directory assigned to lfc DIR.

Syntax:

```
DELETE name comments
```

where:

name — is the one- to eight-character (ASCII) program name to be deleted. This entry must be left-justified with no leading blanks.

comments — any desired comments.

8.6.2 EXIT Directive

The EXIT directive terminates directive input and causes the previous directives to be processed.

Syntax:

```
EXIT
```

8.6.3 LOG Directive

The LOG directive causes a log of all module names and their external definitions to be output to the device or file assigned to lfc LLO.

Syntax:

```
LOG
```

8.7 Listings

The Subroutine Library Editor generates the following listed output:

- o A directive list
- o An optional log of module names and external definitions
- o An optional analysis of available library and directory disc space
- o A list of modules specified for deletion that were not located in the specified library

8.8 Errors

Not supplied.

8.9 Examples

Example 1

This example creates a subroutine library on the file USERLIB and directory on the file USERDIR. The library contains all modules following the $OBJECT Job Control statement. A log of the module names and external definitions are produced on the LLO device. The files USERLIB and USERDIR must be created via the File Manager.

```
$JOB CREATE MEYERS
$NOTE CREATE NEW USER LIBRARY
$OBJECT
(User Object Modules)
$ASSIGN1 LIB=USERLIB,,U
$ASSIGN1 DIR=USERDIR,,U
$OPTION 1
$EXECUTE LIBED
$EOJ
$$
```

Example 2

This example produces a log of all library module names and external definitions, on the SLO device. In addition, statistics on available space in the library and directory are printed.

```
$JOB LOG MEYERS
$ASSIGN1 LIB=USERLIB,,U
$ASSIGN1 DIR=USERDIR,,U
$EXECUTE LIBED
LOG
$EOJ
$$
```

Example 3

This example updates the standard MPX-32 System Subroutine Library and directory MPXLIB and MPXDIR. All modules following the $OBJECT statement either replace an old module of the same name or are added as a new module. The module name UMOD1 is deleted. A log of all module names and external definitions and an analysis of remaining disc space in MPXLIB and MPXDIR is produced.

```
$JOB UPDATE MEYERS
$OBJECT
(User Object Modules)
$EXECUTE LIBED
LOG
DELETE UMOD1
$EOJ
$$
```

Example 4

In this example, the binary output from a compilation is taken directly from the SGO file and used as input to the Subroutine Library Editor. Modules replace modules of the same names; modules that do not match existing names are inserted.

```
$JOB UPDATE MEYERS
$NOTE UPDATE OWNER LIB FROM COMPILATION
$OPTION 5
$EXECUTE FORTRAN
(Source Program)
$A1 LIB=ULIB,,U
$A1 DIR=UDIR,,U
$EXECUTE LIBED
$EOJ
$$
```

Example 5

This example produces a log (using Control C as the EOF character) and returns the user to the TSM prompt automatically when execution is complete.

```
TSM>ASSIGN1 LIB=USERLIB,,U
TSM>ASSIGN1 DIR=USERDIR,,U
    (If these ASSIGN1 statements are not specified, the resulting log details
    contents of the System Subroutine Library to the terminal.)
TSM>ASSIGN4 LLO=UT
TSM>EXECUTE LIBED
LIB > LOG
LIB><Control C>
TSM>
```

9. THE MACRO LIBRARY EDITOR (MACLIBR)

The Macro Library Editor (MACLIBR) is used to create and maintain system or user macro libraries.

MACLIBR allows the user to:

- o delete or replace macros by name
- o insert macros
- o build a macro library from scratch

9.1 General Description

Macros are sequences of Assembly language instructions that are given unique names (e.g., M.ALOC). When a macro name is used in source code, the macro is retrieved from the macro library by the Assembler and expanded into the associated instructions. In a macro, the user can also define up to 256 variable parameters to pass to the macro. In a task that uses the macro, the task supplies appropriate parameters and instructions are expanded as defined in the macro. For further description, see the Macro Assembler Reference Manual, particularly Section 6.

The system macro library for MPX-32 is named M.MPXMAC. Supplied with the operating system, M.MPXMAC contains all macros required to expand system services (Volume 1, Chapters 5, 7, and 8) into Assembler level code with service calls (SVC's). The system macro library for RTM is also provided with MPX-32, and is called M.MACLIB. It can be used with RTM-based source code that uses Call Monitor (CALM) services.

Macros begin with a DEFM statement, which can be proceeded by a header record further describing the macro. The user can list the DEFM statements for a library via the /MACLIST directive. The first line of each macro (header or DEFM) is shown on the normal MACLIBR audit trail.

Within the macro, the user can optionally define parameters to pass to the macro (dummy symbols preceded by percent signs). Other dummy symbols are labels used for conditional processing. The /MACLIST directive allows the user to output all dummy symbols used in a macro.

For further description of macros, see the Macro Assembler Reference Manual, publication number 323-001220.

9.1.1 MACLIBR Command Summary

Directive	Function
/APPEND	Adds macro(s) at end of a library file.
/CREATE	Generates a macro library.
/DELETE	Deletes a macro from a library.
/DISPLAY	Lists a macro.
/END	Defines end of INSERT, REPLACE, APPEND, and/or CREATE sequence. After APPEND or CREATE, has same effect as EXIT.
/EXIT	Last update directive. Performs update and returns control to calling task.
/INSERT	Inserts macro(s) ahead of specified macro.
/LOG	Lists names and numbers of all macros after all updates complete.
/REPLACE	Replaces an existing macro with a new one of the same name.

9.2 Files and File Assignments

Input and output files for MACLIBR are described in this section and the accompanying files assignments chart, Table 9-1.

9.2.1 Macro Library (MAC)

A macro library can reside on either a permanent disc file or a magnetic tape file. If the macro library file is a disc file, the File Manager utility (see Chapter 6) must be used to create the macro library file prior to generating the macro library.

The file assigned to MAC must be unblocked.

Both the Macro Assembler and the Macro Library Editor default the assignment of the macro library to the M.MPXMAC permanent file. During assembly, the macro library is searched each time a macro that is not coded in-line is encountered.

The user has the capability to create multiple macro libraries. When generating a macro library other than M.MPXMAC, the user must reassign the file code, MAC, to the desired macro library file via Job Control or TSM $ASSIGN1 or $ASSIGN3 statements. (Similarly, the Macro Assembler can access a macro library other than M.MPXMAC by reassignment of the file code, MAC.)

9.2.2 Macro Input File (SI)

Macro source is supplied on the file or device assigned to SI. Macros must not be compressed format. Each macro may have a maximum of 256 parameters. The maximum number of macros is 65,535.

9.2.3 Directives (DIR)

MACLIBR directives are supplied on the file or device assigned to lfc DIR. The default assignment is to an SYC file. See Table 9-1.

9.2.4 Audit Trail (LO)

The file or device for an audit trail is assigned to lfc LO. The default assignment is to SLO. For further description of the audit trail, see Section 9.7.

9.2.5 Scratch File

A scratch file the same size as the library file assigned to MAC is allocated dynamically by MACLIBR. The scratch file is used to build and edit macros from the source into an existing or new library.

Table 9 - 1
MACLIBR Files and File Assignments

Input/Output Description	Logical File Code	Assignments for MACLIBR	Previous Processor Assignment	How Specified for MACLIBR	Comment
Source Code	SI	Default: ASSIGN4 SI=DIR Options: ASSIGNn SI = { filename / devmnc } ,,U	Source files can be created via EDIT, MEDIA, etc., as described in alternative routes to SYC with the DIR file code.		By default, MACLIBR equates the source code and directives files so that all come from SYC. Note: if you use a separate file for the SI assignment, specify UNBLOCKED (,,U).
Directives	DIR	Default: DIR=SYC	Work file built using EDIT. Permanent file built using EDIT or MEDIA. Cards. Other device medium e.g., magnetic tape, where jobfile was copied from cards or a file via MEDIA. Interactively. See "Accessing MACLIBR"	EDT>BATCH EDT>BATCH jobfile ??BATCH { D,devmnc / F,jobfile } (or) Same route shown for cards.	For further description see "Accessing MACLIBR."

Table 9 - 1
MACLIBR Files and File Assignments

Input/Output Description	Logical File Code	Assignments for MACLIBR	Previous Processor Assignment	How Specified for MACLIBR	Comment
Macro Library	MAC	Default: ASSIGN1 MAC = M.MPXMAC,,U Options: ASSIGN1 MAC = {M.MACLIB / userlib} ,,U	If an existing library file is not being used, the file for the library must be pre-established via the FILMGR utility.		The macro library file must be unblocked. M.MPXMAC is used for MPX-32 compatible macros (SVC service calls) and MACLIB provides RTM-compatible CALM service calls.
Listed Output	LO	Default: ASSIGN2 LO = SLO,2000 Options: ASSIGNn LO = {filename / devmnc}			

9.3 Options

Applicable MACLIBR options are specified through the Job Control or TSM OPTION command as follows:

$OPTION 7The file assigned to lfc DIR is unblocked.

$OPTION 8The input file assigned to lfc SI is unblocked.

If either option is specified, it must also be specified in the ASSIGN statement.

9.4 Using the Macro Library Editor

MACLIBR processes files sequentially, i.e., the macro specified with any directive must be located further 'down' on the library file than the macro specified with the previous directive. For example, you cannot add a macro in the middle of a library then replace a macro at the beginning. Care is required in preparing the directives file and the file assigned to SI so that both follow the sequence of the library being updated.

The user is cautioned against allowing duplicate names for macros. If two macros have the same name and you want, for example, to delete one of them, there is no guarantee that the right one will be deleted. MACLIBR makes no checks for duplicate names. Listed output from the previous edit can be used to make a name check.

9.5 Accessing the Macro Library Editor

To access MACLIBR as part of a batch job, create a job file using the EDITOR, punch cards, or other media as described in Table 9-1. The job file can be read to SYC and the job activated in several ways:

from the OPCOM console:

" <Attention> "

??BATCH { F, jobfile / D, devmnc }

from the OPCOM program:

TSM > OPCOM

??BATCH { F, jobfile / D, devmnc }

from the EDITOR:

TSM > EDIT
.
.
.

EDT > BATCH [jobfile]

If the jobfile is the current EDITOR work file, issue just the BATCH command.

To activate the Cataloger and run online, use the TSM ASSIGN commands to make Cataloger assignments equivalent to those preceding the EXECUTE MACLIBR command on a jobfile, then proceed to issue Cataloger directives. (SELECT and OBJECT statements are not available when running MACLIBR online.)

```
TSM > MACLIBR
MAC> MACLIBR
MAC>
```

9.6 Macro Library Editor Directives

Macro Library Editor directives are described in detail on subsequent pages.

Most Macro Library Editor directives can either be abbreviated to the first four characters or completely spelled out. If a directive or parameter can be abbreviated, the acceptable abbreviation is indicated in syntax statements by underlining.

Both a comma and blanks between parameters are legal delimiters. Commas need be used only where shown.

Directives are processed serially until an /EXIT directive or an end-of-file is encountered. At least one blank column or space must separate the end of the directive verb and required parameter. A required parameter must not be placed beyond column 20 nor exceed 8 characters. Directives which reference a macro by name must be in the same order as they appear on the macro library file. The only exception is in the case of the /DISPLAY and /LOG directives, which may occur anywhere.

MACLIBR writes the updated macro library to a dynamically allocated scratch file. When the updating sequence is complete, the /EXIT directive will cause the scratch file to be copied to the file assigned to MAC. For /APPEND and /CREATE directives, an /END directive or an end-of-file will serve as an /EXIT directive and cause the scratch file to be copied to the file assigned to MAC. The /INSERT, /DELETE, and /REPLACE directives will cause the remainder of the macro library to be retained if the name specified cannot be found. All modifications to that point will be placed on the macro library file.

9.6.1 /APPEND Directive

The /APPEND directive is used to add macros to the end of the macro library. All macros from the current file position to the end of the macro library will remain the same. Macros are read from the SI file until an /END directive or an end-of-file is encountered. Because no further updating is possible, this will act as an EXIT directive to terminate MACLIBR.

Syntax:

/APPEND

9.6.2 /CREATE Directive

The /CREATE directive is used to generate a macro library. Macros are read from the file assigned to SI until an /END directive or an end-of-file is encountered. MACLIBR then terminates.

Syntax:

/CREATE

9.6.3 /DELETE Directive

The /DELETE directive is used to delete the named macro from the macro library. All macros from the current file position to the name macro remain the same. The next directive is processed after the macro has been found and deleted.

Syntax:

/DELETE macro

where:

macro is the 1- to 8-character ASCII name of the macro to be deleted.

9.6.4 /DISPLAY Directive

The /DISPLAY directive is used to list the statements of the named macro or of all macros if a name is not specified. This directive may be placed anywhere. After the macro has been displayed or found to be nonexistent, the macro library is repositioned to where it was before the display. The next directive is then processed.

Syntax:

/DISPLAY [macro]

where:

macro is the 1- to 8-character (ASCII) name of the macro to be displayed.

9.6.5 /END Directive

The /END directive is used to define the end of an /INSERT, /REPLACE, /APPEND, or /CREATE sequence. After an /INSERT or /REPLACE sequence, the next directive is processed. For /APPEND or /CREATE, processing is the same as for an /EXIT directive.

Syntax:

/END

9.6.6 /EXIT Directive

The /EXIT directive defines the last directive. If any updates have been performed, the scratch file is copied to the file assigned to MAC. If no updates have been performed, MACLIBR terminates. If a /LOG END directive has been included, the updated library is logged.

Syntax:

/EXIT

9.6.7 /INSERT Directive

The /INSERT directive is used to insert macro(s) ahead of the macro specified by the directive. All macros from the current file position to the specified macro remain the same. Macros are read from the file assigned to SI until the /END directive or an end-of-file is encountered. The next directive is then processed.

Syntax:

/INSERT macro

where:

macro is the 1- to 8-character (ASCII) name of the macro before which the new macro will be inserted.

9.6.8 /LOG Directive

The /LOG directive is used to output the name and number of all macros to the file or device assigned to LO. This directive may be placed anywhere in a Macro Library Editor directive stream. If the END option is specified, the logging operation is performed after all updates are complete. If the END option is not specified, the macro library is logged exclusive of any updates. When the logging operation is complete the macro library is repositioned to the point prior to logging. The next directive is then processed.

Syntax:

/LOG [END]

where:

END The log is output after all updates are complete. If not used, the log does not reflect updates.

9.6.9 /MACLIST Directive

The /MACLIST directive is used to entirely or partially suppress the listing of each source macro. This directive does not affect listed output of macros that have already been formatted via /DISPLAY or /LOG. When dummy symbols are output, their corresponding hexadecimal assignments are included.

Syntax:

/MACLIST [option]

where:

option is one of the following parameters:

ON	=	Complete listing
OFF	=	Suppress listing
ID	=	List each macro DEFM statement
BODY	=	List each macro and exclude output of dummy symbols
SYMS	=	List each macro DEFM statement and include output of dummy symbols

If no option is specified, MACLIBR defaults to ON and provides a complete listing.

9.6.10 /REPLACE Directive

The /REPLACE directive is used to replace the named macro with a new macro from the file assigned to SI. All macros from the current file position to that of the specified macro remain the same. Macros are read from the file assigned to SI until an END directive or and end-of-file is encountered. The next directive is then processed.

Syntax:

/REPLACE macro

where:

macro is the 1- to 8-character (ASCII) name of the macro to be replaced.

9.7 Listings

The Macro Library Editor outputs an audit trail that includes directives, a list of all macros, each macro, and a series of MACLIBR operation counters as follows:

CODE	DESCRIPTION
BR	Number of 192-word blocks read from the file assigned to MAC
BW	Number of 192-word blocks written to the scratch file
MD	Number of macros deleted
MR	Number of macros replaced
MI	Number of macros inserted and appended
BU	Number of 192-word blocks used on file assigned to MAC, after updating
NM	Macro number of next macro

The MACLIBR counter values appear at the end of the audit trail.

9.8 Errors

Unless the message, MAC UPDATE COMPLETE, has been printed, the following error diagnostic messages will cause any editing operation to be inhibited.

ARGUMENT 'N1' MATCHES ARGUMENT 'N2':

This is a warning that macro parameters in the N1 and N2 positions of the parameter list are equal.

CURRENT MAC POSITION:

This message is issued, followed by the current position of the Macro Library file, when a /LOG directive is encountered.

DIRECTIVE FILE READ ERROR:

Error condition detected while reading the directive file.

DUMMY PARAMETERS OVERFLOW:

A macro has exceeded the maximum of 256 parameters.

DYNAMIC ALLOCATION OF UTI SCRATCH FILE FAILED:

A scratch file the same size as the MAC file could not be allocated. This error is most likely the result of insufficient disc space.

EOF/EOM ON DIRECTIVE FILE:

An end-of-file on the directive file was encountered before normal termination by an /EXIT or /END directive.

ILLEGAL DIRECTIVE:

The directive is not a legal directive.

MAC FILE SIZE INCREASE REQUIRED:

The updated macro library is larger than the Macro Library file.

NAME NOT FOUND:

A macro specified on a /REPLACE, /INSERT, /DELETE, or /DISPLAY directive was not found on the library file assigned to MAC. The macro may not exist or the file may be already positioned beyond that macro. /INSERT, /REPLACE, and /DELETE directives must occur in the sequence in which the named macros are found on the macro library file.

REPOSITIONED TO:

On the completion of a /DISPLAY or /LOG directive, this message is printed preceding the present position on the Macro Library file.

9.9 Examples

Example 1- This sequence generates a new macro library with directive and source input from the card reader.

```
$JOB CREATE  MEYERS
$ASSIGN3 SI=CR          Assign Source File to Card Reader
$ASSIGN4 DIR=SI         Assign Directive File to Card Reader
$EXECUTE MACLIBR
$EOJ
$$
```

Cards

```
/CREATE
(Macro Source)
/END
```

Example 2- This sequence logs all macros by number and name.

```
$JOB LOG OWNER
$EXECUTE MACLIBR
/LOG
/EXIT
$EOJ
```

Example 3- This sequence displays the macro named M.EQUS.

```
$JOB DISPLAY OWNER
$EXECUTE MACLIBR
/DIS  M.EQUS
/EXIT
$EOJ
```

Example 4-This sequence appends the macro named M.TEST.

```
$JOB APPEND OWNER
$EXECUTE MACLIBR
/MAC  BODY              List with No Dummy Symbol Output
/APPEND
(M.TEST Source)
/END
$EOJ
```

Example 5- This sequence updates the macro library using /REPLACE, /DELETE and /INSERT directories.

```
$JOB UPDATE OWNER
$EXECUTE MACLIBR
/LOG                        List Current Macros
/LOG END                    List Updated Macro Library
/REP  M.EQUS                Replace M.EQUS
(Replacement Macro Source)
/END
/DELETE  M.EXIT             Delete M.EXIT
/INS M.FADD                 Insert macro before M.FADD
(Macro Source to be Inserted)
/END
/EXIT
$EOJ                        Copy from Scratch to MAC
```

10. MEDIA CONVERSION UTILITY (MEDIA)

The Media Conversion Utility (MEDIA) enables a user to manipulate data and/or files. With MEDIA, the user can copy one file to another file, copy a file from one type of medium to another (e.g., copy a magnetic tape file to a disc file), or dump (really a special type of copy) a file from one type of medium to another (e.g., from magnetic tape to a line printer).

MEDIA can also be used to manipulate data. For example, the user can rearrange data from one group of columns on an input deck to another group of columns on an output deck. Another feature allows the user to convert data from one type of code to another, e.g., from EBCDIC to ASCII.

10.1 MEDIA Directives Summary

A summary of MEDIA directives follows. For further details, see Section 10.6.

Directive		Function
BACKFILE	10-6	Positions file backward n files.
BACKREC	10-6	Positions file back n records.
BUFFER	10-7	Names buffer (B03-B09) and specifies size or resets buffer's current read address to start address.
CONVERT	10-7	Converts contents of buffer from ASCII to BCD, BCD to ASCII, ASCII to EBCDIC, EBCDIC to ASCII, or 026 to 029.
COPY	10-8	Copies input records from file or device to output file or device.
DUMP	10-8	Copies a file by converting to ASCII coded hex and outputting to the line printer or SLO.
END	10-9	At end of statements, transfers control to specified previous statement.
EXIT	10-9	Leaves MEDIA.
GOTO	10-10	Conditional transfer to another directive based on counter value, error, or EOF. Or, transfer can be unconditional.
INCR	10-11	Adds specified value to counter (K1-K20).
MESSAGE	10-11	Sends message to OPCOM console.
MOVE	10-12	Moves bytes in one buffer to another buffer.

10.2 Files and File Assignments

All device and file assignments are made to logical file codes via job control $ASSIGNn statement(s) which precede the $EXECUTE MEDIA statement or by equivalent TSM ASSIGN statements. Table 10-1 describes MEDIA file assignments. Up to 64 static assignments may be specified for MEDIA.

Table 10-1.
MEDIA File Assignments

Input/Output Description	Logical File Code	Assignments for MEDIA	Previous Processor Assignment	How Specified for MEDIA	Comment
Directive File	*IN	Default: ASSIGN2 *IN = SYC	Work file built using EDIT. Permanent file built using EDIT or MEDIA. Cards. Other device medium e.g., magnetic tape, where jobfile was copied from cards or a file via MEDIA. Interactively. See "Accessing MEDIA".	EDT > BATCH EDT > BATCH jobfile or ??BATCH {D,devmnc / F,jobfile} Same route shown for cards.	For further description see "Accessing MEDIA".
Input File(s)	User-defined	No default. User ASSIGN's file(s) or device(s) to user-specified lfc(s), e.g., ASSIGN3 INP = CR0001.	Input files can be cards, permanent files, built via EDIT or MEDIA, or other device media such as magnetic tape.		Input files are manipulated by referring to the logical file code (lfc) specified in an ASSIGN statement. MEDIA recognizes a hexadecimal 'OF' record on card reader or card reader/punch devices only. Whether a file or tape is blocked or unblocked should be specified with ASSIGN.
Listed Output File	*OT	Default: ASSIGN2 *OT = SLO,500	N/A		
Output File(s)	User-defined.	No default. User ASSIGN's file(s) or device(s) to user-specified lfc('s), e.g., ASSIGN1 OUT = MYFILE.	Disc files must be created (via the FILEMGR or equivalent utility) before they can be assigned for MEDIA.		Output files are manipulated by referring to the logical file code (lfc) specified in an ASSIGN statement. See notes on 'OF' and blocking for input files.

10.3 Options

The $OPTION statement is not applicable to MEDIA.

10.4 Using MEDIA

Directive and processing errors result in diagnostic messages, but an abort (MD01/MD02) is generated at the end of the run instead of a normal termination so that conditional batch processing directives may be used.

If an I/O error occurs, the status for the device is printed.

If a loop was being executed where record/file information is accumulated, this information will be printed even if an error (and now an abort) occurs.

10.4.1 Labels

Any MEDIA directive can be preceded by a label (up to 8 digits long) that will allow the user to branch to it via a GOTO directive. If used, the label must be numeric and it must start in column 1. If a label is not used, the directive can start in column 1. The label number need have no relationship to the physical sequence of directives on the control file, i.e., it is a label and not an absolute sequence number. If a label is used, it must be terminated with a comma.

10.5 Accessing MEDIA

To access MEDIA as part of a batch job, create a job file using the EDITOR, punch cards, or other media as described in Table 10-1. The job file can be read to SYC and the job activated in several ways:

from the OPCOM console:

" <Attention> "

??BATCH { F,jobfile / D,devmnc }

from the OPCOM program:

TSM > OPCOM

??BATCH { F,jobfile / D,devmnc }

from the EDITOR:

```
TSM>EDIT
   .
   .
   .
EDT>BATCH [jobfile]
```

If the jobfile is the current EDITOR work file, issue just the BATCH command.

To activate MEDIA and run online, use the TSM ASSIGN commands to make MEDIA assignments equivalent to those preceding the EXECUTE MEDIA command on a jobfile, then proceed to access MEDIA and enter MEDIA directives. (SELECT and OBJECT statements are not available when running MEDIA online.) The logical file *OT (MEDIA diagnostic output) can be redirected to the terminal for online use.

```
TSM>MEDIA
MED>
```

10.6 MEDIA Directives

10.6.1 General

MEDIA directives establish the logical flow of the MEDIA utility. Each directive has a label field, a directive field, and a parameter field. If used the label field must start in column one and be terminated with a comma. The directive field may start in any column, including column one. Fields and parameters within a field are separated by commas. Blank columns are ignored. All numeric parameters are specified in decimal and are limited to seven digits.

MEDIA contains two types of predefined areas which the user may reference within the MEDIA directives. They are predefined buffer areas and counters.

Two predefined buffers exist within MEDIA. These buffers are referenced by the names B01 and B02, and each buffer is defined as 2048 words in length. The user also has the option of defining additional buffer areas (B03-B09) by using the BUFFER statement (up to 3K bytes total for B03-B09). Buffer names other than B01-B09 are illegal.

Twenty predefined counter cells exist within MEDIA. These counters are referenced by the names K1 through K20. Counters may be used as program flags, record counters, file counters, etc., and may contain any positive decimal value within the range of 0 to 99,999,999.

A maximum of 256 directives may be specified by the user for MEDIA.

10.6.2 BACKFILE Directive

The BACKFILE directive causes the specified file to be positioned backwards by the number of files specified in the count field.

Syntax:

BACKFILE, lfc, count

where:

lfc	is the 1- to 3-character logical file code.
count	is the 1- to 8-digit field specifying the number of files to be skipped over.

10.6.3 BACKREC Directive

The BACKREC directive causes the specified file to be positioned backwards by the number of records specified in the count field.

Syntax:

BACKREC, lfc, count

where:

lfc	is the 1- to 3-character logical file code.
count	is the 1- to 8-digit field specifying the number of records to be skipped over.

10.6.4 BUFFER Directive

The BUFFER directive defines the specified buffer according to the specified size or resets the current buffer read address to the buffer starting address. The space allocated to the specified buffer is allocated from a 3000-byte buffer pool, which is the maximum allowable additional buffer.

Syntax:

BUFFER, buffer, $\left\{\begin{matrix}\text{nbytes}\\ \text{R}\end{matrix}\right\}$

where:

buffer	is the 3-character name of the buffer, B03-B09. Buffer names must be defined via the BUFFER directive before being used.
R	specifies reset buffer pointer.
nbytes	is the size of the buffer, in decimal bytes.

10.6.5 CONVERT Directive

The CONVERT directive results in the conversion of the specified buffer to the specified code.

Syntax:

CONVERT, buffer, code [,nbytes]

where:

buffer	is the 3-character name of the buffer, B01-B09. Buffer names must be defined via the BUFFER directive before being used.
code	is the 4-character code specifying the type of conversion:

2629	026 to 029
ASBC	ASCII to BCD
BCAS	BCD to ASCII
ASEB	ASCII to EBCDIC
EBAS	EBCDIC to ASCII

nbytes	the number of bytes to be converted. If this parameter is absent, the count is obtained from the total number of bytes read and/or moved into the buffer.

10.6.6 COPY Directive

The COPY directive causes the specified input file to be copied, record by record, to the specified output file until an EOF is encountered on either file. The end-of-file is not copied to the output file.

Syntax:

COPY, lfc1, lfc2

where:

lfc1	is the 1- to 3-character lfc identifying the input file to be copied.
lfc2	is the 1- to 3-character lfc identifying the file to which the copy is made.

10.6.7 DUMP Directive

The DUMP directive causes the file specified by file code 1 to be input record by record, converted to ASCII-coded hexdecimal with side-by-side ASCII translation, and output to the file specified by file code 2. The dump is terminated when an end-of-file is encountered on either file or when the optionally specified number of records have been dumped.

Syntax:

DUMP, lfc1, lfc2 [,recordcount]

where:

lfc1	is the 1- to 3-character identifier of the file to be dumped.
lfc2	is the 1- to 3-character identifier of either the line printer or SLO. Normally, *OT is the lfc assignment which is referenced.
recordcount	is the number of records in the file to be dumped. If not specified, the dump terminates at EOF on lfc 1.

10.6.8 END Directive

The END directive indicates the end of the MEDIA directives and control is transferred to the specified directive. If no directive number is specified, control will go to the first MEDIA directive.

Syntax:

END [,label]

where:

label	is a numeric label associated with a directive. Transfers control to that directive.

10.6.9 EXIT Directive

The EXIT directive indicates that a normal program exit is to be taken.

Syntax:

EXIT

10.6.10 GOTO Directive

The GOTO directive transfers control to the specified directive unconditionally when no optional arguments are specified. Conditional control transfer occurs when the specified counter is equal to the specified value, whenever an input/output error occurs on the specified file, or whenever an end-of-file is encountered on the specified file. If none of the conditional specifications are satisfied, control proceeds to the next MEDIA directive. When the ERR parameter is specified, the GOTO directive must directly follow an input/output directive such as READ or WRITE.

Syntax:

```
GOTO, label, [counter, value]
             [EOF, lfc    ]
             [ERR, lfc    ]
```

where:

label	is the numeric label, if any, associated with a directive. Transfers control to that directive.
counter	is a user-controlled indicator of the state of the program. Valid counter names are K01-K20. (See the INCR and SETC directives.)
value	is a countervalue that specifies when control is to be transferred.
EOF	specifies that conditional transfer should occur at end of file.
lfc	is the 1- to 3-character logical file code indicating which file to apply EOF or ERR conditions to.
ERR	specifies that conditional transfer should occur upon an I/O error.

10.6.11 INCR Directive

The INCR directive causes the specified value to be added to the specified counter.

Syntax:

INCR, counter, value

where:

counter	is the 3-character name of the counter: K01 through K20.
value	is the 1- to 8-digit decimal number to be used as an increment.

10.6.12 MESSAGE Directive

The MESSAGE directive results in the output of the specified alphanumeric string to the OPCOM console. A maximum of 256 bytes of message information can be stored.

Syntax:

MESSAGE, 'message'

where:

'message'	is a 3- to 72-character alphanumeric message to be displayed on the OPCOM console; the single quotes are required.

10.6.13 MOVE Directive

The MOVE directive causes the specified number of bytes to be written into the specified position of buffer 2 from the specified position of buffer 1 (zero origin). The starting byte position for buffer 1 and buffer 2 may be specified as an absolute byte number or as a counter.

Syntax:

MOVE, nbytes, buffer1, {counter / startbyte} , buffer2 [,counter / ,startbyte]

where:

nbytes	is the number of bytes to be moved.
buffer1	is the buffer 3-character name of the buffer containing the data to be moved, B01-B09. Buffer names must be defined via the BUFFER directive before being used.
counter	is the name of a counter indicating the starting byte position. Valid counter names are K1-K20. If neither counter or byte number is specified, the current read address for buffer name 2 is used.
startbyte	is an absolute indicator of the starting byte position.
buffer2	is the 3-character name of the buffer to which data are to be moved, B03-B09. Buffer names must be defined via the BUFFER directive before being used. If neither counter nor starting byte number is specified, the current read address for buffername2 is used.

10.6.14 OPTION Directive

The OPTION directive allows the specification of nonstandard options for the specified file. Options which are assumed by default are defined according to Table 10-2.

Syntax:

OPTION, lfc [,BLOCKED] [,B2OF / ,B2ON] [,B8OF / ,B8ON] [,E / ,O] [,H / ,L]

where:

lfc	is the 1- to 3-character logical file code specified by the user in a job control or TSM ASSIGN statement.
BLOCKED	is the blocked option. This option is permitted, but does not affect processing. The ASSIGNs are the only way to control blocked/unblocked.
B2OF B2ON	indicates the bit 2 option is either on or off (refer to Table 10-2).
B8OF B8ON	indicates the bit 8 option is either on or off (refer to Table 10-2).
E O	indicates the even/odd parity option (refer to Table 10-2).
H L	indicates the density option (refer to Table 10-2).

Table 10-2. Media Option Definitions

DEVICE	BLOCKED OPTION	BIT 2 OPTION	BIT 8 OPTION	PARITY	DENSITY
CARD READER READER/PUNCH	N/A	*B2OF-Automatic Mode Select B2ON-Interpret Bit 8	N/A B8OF-Forced ASCII B8ON-Forced Binary	N/A	N/A
PAPER TAPE READER	N/A	*B2OF-Read Formatted Skipping Leader B2ON-Read Unformatted	N/A B8OF-Do Not Skip Leader B8ON-Skip Leader	N/A	N/A
PAPER TAPE PUNCH	N/A	*B2OF-Punch in Formatted Mode B2ON-Punch Unformatted	N/A N/A	N/A	N/A
LINE PRINTER, TELETYPE	N/A	*B2OF-First Character is Carriage Control B2ON-No Carriage Control	N/A	N/A	N/A
9-TRACK MAG TAPE	Blocked I/O	N/A	N/A	N/A	N/A
7-TRACK MAG TAPE	Blocked I/O	*B2OF-Read/Write Packed (Binary) Mode B2ON-Interpret Bit 8	N/A B8OF-Inter-change (BCD B8ON-Packed (Binary)	E (Even Parity) O(Odd Parity)	H(800 bpi L(556 bpi
MOVING-HEAD, FIXED-HEAD DISC	Blocked I/O	N/A	N/A	N/A	N/A

*Standard Options
N/A - Not Applicable

10.6.15 READ Directive

The READ directive causes one record to be read from the specified file into buffer B01 or into the optionally specified buffer, starting at the current buffer address. The current buffer address is advanced after read and is reset only by a write from the specified buffer or by a buffer reset via the BUFFER directive.

Syntax:

READ, lfc ,[buffer][,count]

where:

lfc	is the 1- to 3-character logical file code.
buffer	is the 3-character name of the buffer, B01-B09. Buffer names must be defined via the BUFFER directive before being used.
count	is the number of bytes (B), halfwords (H), or words (W) to be read (e.g., B22, H10, H2048, W192).

10.6.16 REWIND Directive

The REWIND directive causes the specified file to be rewound. If the file is being sent to the line printer, a top-of-form is performed.

Syntax:

REWIND, lfc

where:

lfc	is the 1- to 3-character logical file code.

10.6.17 SETC Directive

The SETC directive sets the specified counter to the value specified in the value field.

Syntax:

SETC, counter, value

where:

counter	is the 3-character name of the counter: K1 through 20.
value	up to 8 decimal digits specifying the value to which counter is to be set.

10.6.18 SKIPFILE Directive

The SKIPFILE directive causes the specified file to be positioned forward by the number of files specified in the count field.

Syntax:

SKIPFILE, lfc, count

where:

lfc	is the 1- to 3-character logical file code.
count	is the decimal number of files to be skipped.

10.6.19 SKIPREC Directive

The SKIPREC directive causes the specified file to be positioned forward by the number of records specified in the count field.

Syntax:

SKIPREC, lfc, count

where:

lfc	is the 1- to 3-character logical file code.
count	is the decimal number of records to be skipped.

10.6.20 VERIFY Directive

The VERIFY directive causes the file specified by lfc 1 to be compared, record by record, with the file specified by lfc 2. The verification is terminated when an end-of-file is encountered on either file. Records which do not compare result in output indicating the record numbers which do not compare.

Syntax:

VERIFY, lfc1, lfc2

where:

lfc1	is the 1- to 3-character logical file codes identifying the file to compare to lfc 2.
lfc2	If records of unequal length are to be verified, the file specified by lfc 1 must contain the shorter record size.

10.6.21 WEOF Directive

The WEOF directive causes an end-of-file to be written on the specified file.

Syntax:

WEOF, lfc

where:

lfc	is the 1- to 3-character logical file code.

10.6.22 WRITE Directive

The WRITE directive causes one record to be written from buffer B01 or from the optionally specified buffer to the specified file. The WRITE statement resets the current buffer address and byte count for the output buffer.

Syntax:

WRITE, lfc ,[buffer [,count]]

where:

lfc	is the 1- to 3-character logical filecode.
buffer	is the 3-character name of the buffer: B01 through B09. If buffer name is not supplied, B01 is used by default. Buffer names must be defined via the BUFFER directive before being used.
count	is the number of bytes (B), halfwords (H), or words (W) to be written (e.g., B22, H10, H2048, W192). If count is not specified, the total number of bytes, halfwords, or words read and/or moved into the buffer will be used as the output count.

10.7 Listings

Not supplied; they are dependent on the assignments and processing that were requested.

10.8 Errors

During the compilation or execution of any MEDIA conversion program, all detected errors are flagged with two digit codes. A complete list of the diagnostic codes follows:

Code	Definition
01	Control specification invalid
02	File code unassigned
03	Illegal conversion code specified
04	Illegal count specification
05	No available FCB, excessive file assignments
06	No available blocking buffers, excessive system file assignments
07	Illegal buffer name
08	Buffer already defined
09	Insufficient buffer space available
10	Undefined buffer
11	Illegal device assignment
12	Illegal counter name specified
13	Insufficient message storage space available
14	Illegal byte number or number of bytes specifications
15	Illegal optional parameter
16	Missing parameter
17	Incorrect message format
18	Illegal decimal or hexadecimal character
20	No end statement
21	Excessive number of control statements specified
22	Fatal control statement error(s)
23	Undefined statement number encountered
24	Execution of END statement attempted

Code	Definition
25	Length of READ/MOVE exceeds buffer size
26	WRITE statement which is not preceded by READ statement must specify count
27	End-of-medium encountered in input file
28	End-of-medium encountered on output file
29	Convert statement specified zero byte count
30	Duplicate statement number

10.9 Examples

Example 1 is used to copy tape "MAST" to the output tape "COPY", and write an end-of-file to "COPY"; both tapes are rewound and then verified.

```
$JOB EXAMPLE1 MEYERS
$ASSIGN3 IN=MT,MAST,1,U
$ASSIGN3 OT=MT,COPY,1,U
$EXECUTE MEDIA
COPY,IN,OT
WEOF,OT
REWIND,IN
REWIND,OT
VERIFY,IN,OT
EXIT
END
$EOJ
$$
```

Example 2 is used to read a card deck punched in 026 code, convert the data to 029 code, and then punch the data in 029 code.

```
$JOB EXAMPLE2 MEYERS
$ASSIGN3 01=CR
$ASSIGN3 02=CP
$EXECUTE MEDIA
MESSAGE,'PLACE 026 DECK IN CARD READER'
5,READ,01
GOTO,6,EOF,01
CONVERT, B01,2629
WRITE,02
GOTO,5
6,EXIT
END
$EOJ
$$
```

---

026 Card Deck

EOF (Card with holes 2,3,4, and 5 punched in Column 1.)

Example 3 is used to dump the first 50 records of the second file on tape "T132" to an SLO file. A maximum of 5000 lines will be output.

```
$JOB EXAMPLE3 MEYERS
$ASSIGN3 AB=MT, T132
$ASSIGN2 DP=SLO,5000
$EXECUTE MEDIA
REWIND, AB
SKIPFILE, AB, 1
DUMP, AB, DP, 50
EXIT
END
$EOJ
$$
```

Example 4 is used to directly output the first 40 columns of a maximum of 100 cards in the batchstream to the line printer.

```
$JOB EXAMPLE4 MEYERS
$ASSIGN2 IN=SYC
$ASSIGN3 OT=LP
$EXECUTE MEDIA
OPTION, OT,,B2ON
SETC, K1, 0
3,READ, IN,,B40
GOTO,5,EOF,IN
WRITE,OT
INCR,K1,1
GOTO,5,K1,100
GOTO,3
5,EXIT
END
CARD DECK
$EOJ
$$
```

Example 5 is used to read two source program card decks, one through the card reader and one through the card reader/punch. Columns 20-40 of Deck 1 will be moved to columns 10-30 of the output image. Columns 65-80 of Deck 2 will be moved to columns 31-46 of the output image. The output image will be written to a tape in 120-byte records.

```
$JOB EXAMPLE5 MEYERS
$ASSIGN3 IN1=CR
$ASSIGN3 IN2=CD
$ASSIGN3 OT=MT,SAVE
$EXECUTE MEDIA
BUFFER,B04,120
4,READ,IN1,B01,B40
GOTO,5,EOF,IN1
READ, IN2, B02,B80
MOVE, 21,B01, 19, B04, 9
MOVE, 16, B02, 64, B04
WRITE, OT, B04, B120
BUFFER, B01, R
BUFFER, B02, R
GOTO, 4
5, WEOF, OT
REWIND, OT
EXIT
END
$EOJ
$$
```

Deck 1 in Card Reader with EOF Card

Deck 2 in Card Reader/Punch

Example 6 shows online usage (underlined text is input).

```
TSM > A1 IN=FILE1 OUT=FILE2
TSM > A4 *OT=UT
TSM > MEDIA
MED > COPY,IN,OUT
COPY,IN,OUT
MED > EXIT
EXIT
MED > END or  <CTRLC>
END

MEDIA COMPILATION COMPLETE: EXECUTION STARTED EXECUTION
COMPLETE 0001 FILE COPIED
TSM >
```

11. SOURCE UPDATE UTILITY (UPDATE)

The UPDATE utility is used to add, replace, or delete lines of source code within a particular file. It can also be used to maintain sets of source files by adding or deleting complete files.

UPDATE was designed to use in building and editing tapes containing multiple source files for software libraries. It can be used to edit or build any set of source files onto a single tape or disc file. Files can be positioned to end of file marks (EOF) by UPDATE directives such as /BKSP or they can be positioned symbolically by referring to a header record that identifies a particular file (library format). Library format is a simple structure where source code is preceded by a header record and terminated with a single EOF record. Any group of source files that has been built in 'library' format can be positioned symbolically.

No matter how a file is built (e.g., via MEDIA) if it is structured in library format, it can be positioned symbolically with UPDATE. (UPDATE also provides the ability to insert header records during processing.)

11.1 General Description

Updating is a two-pass process. In the first pass, UPDATE reads control directives and dating statements that may be interspersed from the SYC file. All of the directives detected within this control stream are scanned for errors. The control stream (with error diagnostics, if any) is copied to a work file (normally an SLO file) for the actual update processing.

If directive errors have occurred, UPDATE exits after it encounters an /EXIT directive in the first pass. The user receives a listing of the control stream plus error diagnostics. When an /EXIT directive is encountered and directive errors have not been detected, the updating sequence is entered. Updating continues until all of the stored directives have been sequentially processed. At this point, UPDATE exits.

11.1.1 UPDATE Directives

A summary of UPDATE directives follows. For further details, see Section 11.6.

Directive		Function
/ADD	11-12	Adds source lines which follow (up to next directive) after the specified line of source.
/AS1	11-13	Reassigns input or output file codes to another permanent disc file.
/AS3	11-13	Reassigns input or output files to another configured peripheral device.
/BKSP	11-14	Backspaces n files on current input medium.
/BLK	11-14	Causes sequence field (bytes 73 through 80) of each record to be filled with blanks.

11.2 Files and File Assignments

All files that do not have cataloged assignments can be assigned via the job control $ASSIGNn statement. Cataloged file assignments may be overridden by the $ASSIGNn statement. Table 11-1 provides details of UPDATE file usage.

Note that upon entry, UPDATE marks all unassigned file codes as unavailable to the user.

Input files can be presented in either a compressed or standard source format. The output file can be produced in either format at the user's option. Also, an optional listing can be printed as the output file is generated.

The output file (SO) and primary input file (SI1) are assumed blocked unless otherwise specified via the $OPTION job control statement.

Table 11-1
UPDATE File Assignments

Input/Output Description	Logical File Code	Assignments for Update	Previous Processor Assignment	How Specified for Update	Comment
Directives and Corrections	SYC	Default:	Work file built using EDITOR.	EDT BATCH	
		SYC = SYC	Permanent file built using EDIT or MEDIA	EDT BATCH jobfile or	For further description see "Accessing Update"
			Cards.	?? BATCH {D,devmnc / F,jobfile}	
			Other device medium e.g., magnetic tape, where jobfile was copied from cards or a file via MEDIA.	Same route shown for cards.	
			Interactively. See "Accessing Update"		
Primary Input File	SI1	No default. $ASSIGNn SI1 = {filename / devmnc}	N/A		A hexadecimal 'OF' record is taken as an EOF on any input source. Source can be compressed or noncompressed. (See Section 11.4.1 and Option 1.) Input is assumed blocked (MPX-32 and UPDATE default) unless UNBLOCKED is specified on assignment. If unblocked, $OPTION 8 must also be set.
Second and Third Input Files	SI2 and SI3	No defaults. ASSIGNn SI2 = {filename / devmnc} SI3 = {filename / devmnc}	N/A	With /SELECT /SELECT {SI2 / SI3}	See comment on blocking directives, as in:above. Note that additional input files can be assigned dynamically via /AS4 and /AS3 UPDATE directives.

Table 11-1
UPDATE File Assignments (Continued)

Input/Output Description	Logical File Code	Assignments for Update	Previous Processor Assignment	How Specified for Update	Comment
Output File for Updated Source	SO	No default ASSIGNn SO = {filename / devmnc}	N/A	Via an ASSIGN statement	If a disc file, must be created via the FILEMGR or other means before running UPDATE. UPDATE can generate compressed output. (See OPTION 1). Other options can be used if merging files. An output file can be produced in library format (see Section 11.4.2.) See comment on blocking for SI1. Option 9 is not recommended. Note: additional output files can be assigned dynamically via /AS1 and /AS3 UPDATE directives.
Work File for Intermediate Storage, and Directive and Error Listings.	UTY	Default: ASSIGN2 UTY = SLO,1000	N/A	Default output is to 1000 record SLO file.	
History, History Summary, and Output Image Listing for Update Process	LO	Default: ASSIGN2 LO= SLO,2000 Options: ASSIGNn LO= {filename / devmnc}	By UPDATE. Image outlines structure of updated output file.	Default output is to 2000 record SLO file, which is output to the first device available for auto selection. Output can be redirected to a file or device via ASSIGN1 or ASSIGN3 LO = statement.	When LO is assigned to SLO and UPDATE reaches EOM, it attempts to allocate 2000 additional lines. If unsuccessful or if an SLO file is not assigned, listed output terminates.

11.3 Options

The set or reset state of an option is declared by the following methods:

- o Job control $OPTION statement
- o UPDATE directives such as /LIST and /NOLIST
- o UPDATE utility defaults

Once declared, the state of an option remains in effect until it is changed by a directive.

Job control options are controlled via the $OPTION statement (batch) or by a TSM OPTION command. The $OPTION statement is specified in the following general format, where n represents the numeric value of the option to be set:

$OPTION n n n

The user is allowed to specify multiple options for each UPDATE execution, as follows:

$OPTION 1 - Creates the output file (SO) in compressed source format.

$OPTION 2 - Creates the output file (SO) in library format.

$OPTION 3 - Prints the control stream (i.e., statements input from the SYC file). The control stream will always be printed when directive errors have been detected.

$OPTION 4 - Inhibits writing end-of-file marks detected on the input file to the output file. The option could be used to strip off multiple files that are to be assembled or compiled. This option cannot be used in conjunction with $OPTION 2.

$OPTION 8 - Indicates that the primary input file (SI1) is not blocked. If SI1 file is unblocked, the user must specify $OPTION 8 and U (unblocked) on the Assignment statement for SI1.

$OPTION 9 - Indicates that the output file (SO) is not blocked. If SO file is unblocked, the user must specify $OPTION 9 and U (unblocked) on the Assignment statement for SO. If this option is not present, the file is assumed blocked.

11.4 Using UPDATE

UPDATE processes files sequentially, i.e., the line number specified with any directive must be equal to or greater than the line specified with the previous directive. For example, you cannot add lines after line 100 then go back and delete line 52. (The equal to specification applies only in the case where you delete a line; delete can be followed by an ADD specifying the same line.)

Care is required in preparing a directives file so that it follows the sequence of the file being updated.

11.4.1 Compressed Source Formatting

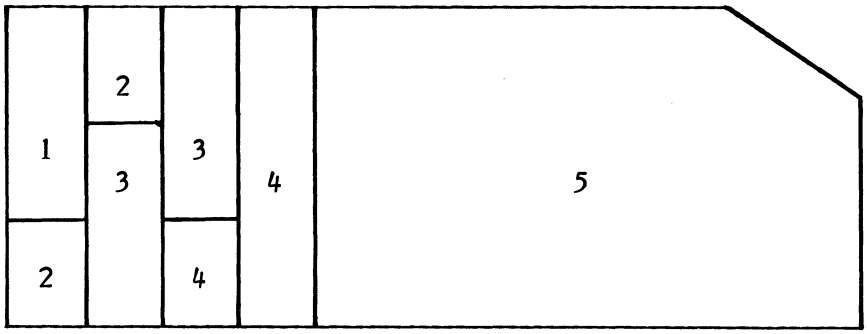
UPDATE is capable of accepting compressed source input. The output file can be produced in compressed or standard source format.

The source compression ratio will vary depending on the amount of comments contained in the source program. The user should expect a compression of between 4:1 and 12:1 for the average range of source decks. All compressed records output by UPDATE will be 120 bytes long. The maximum input record size is 192 bytes. Compressed source can be output to or processed from disc, magnetic tape, paper tape, or cards (see Section 11.2).

11.4.2 Library Mode of Operation

The purpose of the library mode of operation is to create a standard format to be used for building library files. This mode is specified by a job control $OPTION 2 statement.

The library mode of operation allows input files to be presented in any format, but ensures that the output file is created in library format (see Figures 11-2, 11-3, and 11-4). Incorrectly formatted header records will cause the job to be aborted. If continuous end-of-file marks are encountered on the input file, one end-of-file mark will be written on the output file. Upon completion of the library update, a unique end-of-library file record will be written on the output file, and UPDATE will exit.



(1) 8-bit type code

X'BF' and X'9F' designate compressed source image

X'9F' indicates last record of a compressed source module

(2) 8-bit count of bytes remaining in record

(3) 16-bit checksum of record

(4) 16-bit sequence number

(5) Contiguous bytes of data

820659

Figure 11-1 Compressed Record Card Format

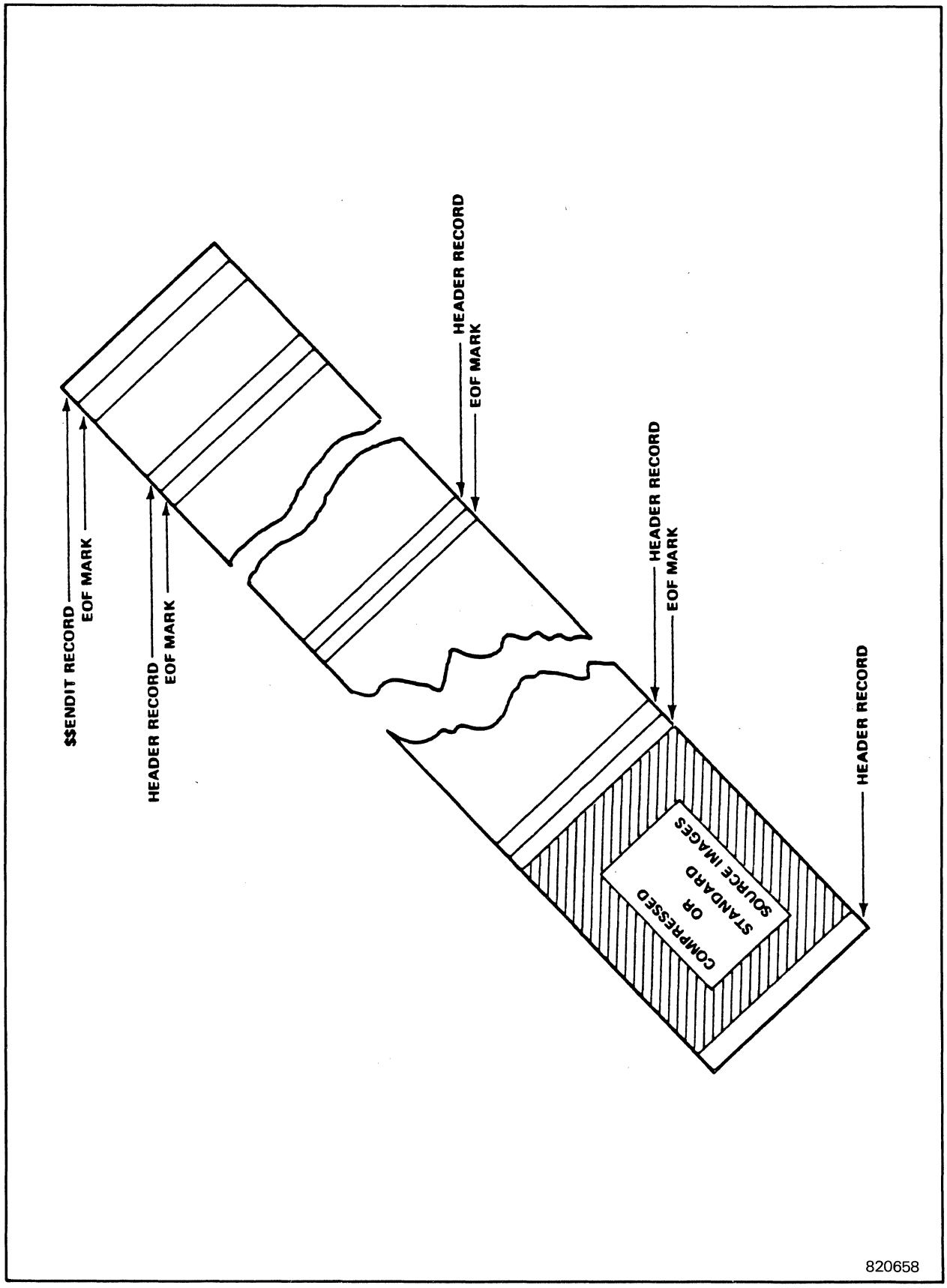


Figure 11-2. Library Format (Magnetic Tape)

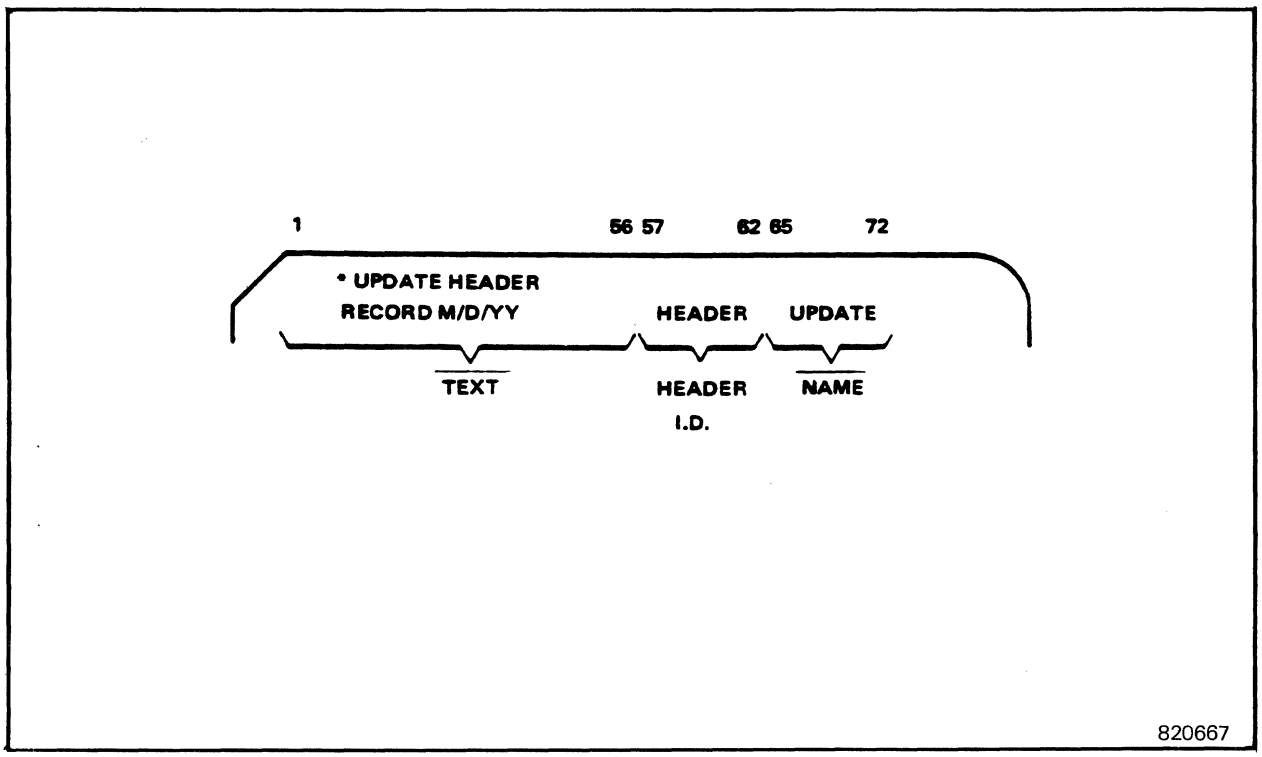


Figure 11-3. Header Record Format

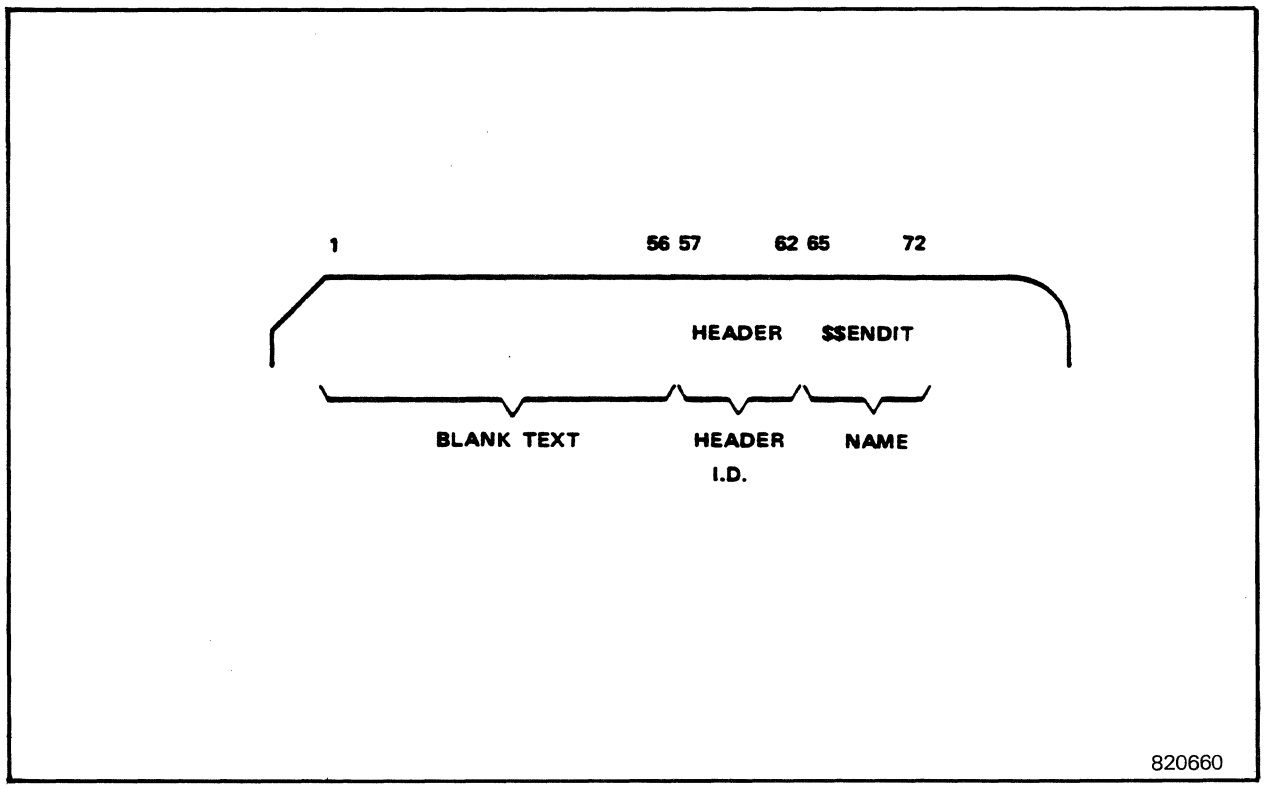


Figure 11-4. Library End-Of-Tape Format

11.5 Accessing UPDATE

To access the UPDATE utility as part of batch job, create a job file using the EDITOR, punch cards, or other media as described in Table 11-1. The job file can be read to SYC and the job activated in several ways:

from the OPCOM console:

" < Attention > "

??BATCH { F,jobfile / D,devmnc }

from the OPCOM program:

TSM > OPCOM

??BATCH { F,jobfile / D,devmnc }

from the EDITOR:

```
TSM > EDIT
        .
        .
        .
EDT>BATCH[jobfile]
```

If the jobfile is the current EDITOR work file, issue just the BATCH command.

To activate UPDATE and run online, use the TSM ASSIGN commands to make assignments equivalent to those preceding the EXECUTE UPDATE command on a jobfile, then proceed to issue UPDATE directives. (SELECT and OBJECT statements are not available when running UPDATE online.)

```
TSM > ASSIGNn...
        .
        .
        .
[TSM > OPTION n...]
TSM > UPDATE
UPD > /directive
UPD > etc.
```

11.6 UPDATE Directives

11.6.1 General

Control directives for UPDATE have the following general forms:

```
/Command Specification
```

A slash (/) in column one identifies the record as a control directive. Command is the symbolic identification of the control directive. Specification is a variable field which contains information that further describes the operation to be performed.

The command field may be abbreviated to three characters and must be terminated by a blank. The specification fields are terminated by commas and/or blanks.

Special characters should not be placed in column one of the command file. When placed in column one, they can be interpreted incorrectly, resulting in an UPDATE abort. For example, the "$" in column one is interpreted by MPX-32 as a JCL command, which will result in an UPDATE abort exit.

11.6.2 /ADD Directive

The /ADD directive indicates that the supplied statements are to be added immediately following the specified source line. All statements following the /ADD directive and up to the next control directive are inserted in the output file.

Syntax:

```
/ADD start
```

where:

start identifies the source line after which the additions are to be inserted. A specification of zero (i.e., /ADD 0) will result in additions being made to the beginning of an existing file.

11.6.3 /AS1 Directive (Reassign Lfc to Disc File)

The /AS1 directive reassigns input or output file codes to another permanent disc file. The mode (blocked or unblocked) is not changed.

If the /AS1 directive is printed with the updates and the directive contains more than 19 characters, only the first 19 characters are printed on the output.

Note: This directive reassigns only previously assigned file codes and will not assign unassigned file codes.

See also the /USR and /SELECT directives.

Syntax:

```
/AS1 lfc,filename[,password]
```

where:

lfc is the logical file code.

filename is the 1- to 8-character name of a disc file.

password is the 1- to 8-character password associated with the file.

11.6.4 /AS3 Directive (Reassign Lfc to Device)

The /AS3 directive reassigns input or output files to another configured peripheral device. The mode (blocked or unblocked) is not changed.

Note: This directive reassigns only previously assigned files and will not assign unassigned files.

See also the /SELECT directive.

Syntax:

```
/AS3 lfc,devmnc
```

where:

lfc is the logical file code.

devmnc is one of the following device mnemonic of a configured peripheral device: CD, CP, CR, M7, M9, or MT.

11.6.5 /BKSP (Backspace) Directive

The /BKSP directive provides the capability for backspacing input files. The utility will backspace the number of specified end-of-files and position to the beginning of that file in order to be properly positioned in the file. Detection of beginning-of-medium will cause backspacing to be terminated and the next control directive will be executed.

Syntax:

/BKSP lfc [,n]

where:

lfc is the logical file code associated with the device or file to be backspaced.

n is an optional parameter that denotes the number of files to be backspaced. If not specified, n is assumed to be 1.

11.6.6 /BLK (Blank Sequence Field) Directive

The /BLK directive causes the sequence field (bytes 73 through 80) of each record to be filled with blanks. This directive is in effect until a No Blank Sequence Field (/NBL) directive is read, and takes precedence over Sequence (/SEQ) directives.

This directive may be placed anywhere in the control stream.

Syntax:

/BLK

Example:

```
/BLK
/COPY 1
/SKIP 1
/COPY 1
/BKSP SI1,1
/COPY 1
```

In this example, all three files would be copied without sequence numbers.

11.6.7 /COPY Directive

The /COPY directive gives the user the capability of copying complete input files. Files that have been created in library format can be copied to the end of a library by the directive: /COPY END.

Syntax:

/COPY header

where:

header is a 1- to 8-character delimiter. If header is an alphanumeric string, UPDATE will copy all files up to the header record defined by header. If header is a numeric string, UPDATE will copy the number of files represented by the string.

11.6.8 /DELETE Directive

The /DELETE directive indicates that the specified sequential source lines are to be omitted from the output file.

Syntax:

/DELETE start [,end]

where:

start identifies the first source line to be deleted.

end is an optional parameter that identifies the last source line to be deleted. If not specified, end will assume to have the same value as start.

11.6.9 /END Directive

The /END directive indicates that the end of an /ADD, /DELETE, or /REPLACE sequence for a file has occurred. The remaining source lines will be copied inclusive of the next EOF. At this point, UPDATE reads the next directive.

Syntax:

/END

11.6.10 /EXIT Directive

The /EXIT directive indicates that the end of the UPDATE control stream has been reached. If directive errors have been detected, UPDATE will exit. If directive errors have not been detected, the update sequence will be entered. When all control directives have been sequentially processed, UPDATE will check for the library option. If the library option is set, the utility will write a unique end-of-library file record on the file assigned to SO, rewind the file, and exit. This directive must be used to terminate an UPDATE control stream.

Syntax:

/EXIT

11.6.11 /INSERT Directive

The /INSERT directive causes one complete file to be copied from the current input device.

The user is allowed a maximum of 56 bytes of text for the new header. Text begins at the next character after the first delimiter.

The new header record will replace the old header record in a file that already has a header record.

If the /INSERT directive is printed with the updates and the directive contains more than 19 characters, only the first 19 characters are printed on the output.

Syntax:

/INSERT [name ,text]

where:

If nothing is specified, the old header will be used. If no header exists, the first line of code is treated as a header.

name,text are optional parameters used to create text a header record in the following format:

1-56	56-62	65-72
text	header identification	name

11.6.12 /LIST Directive

The /LIST directive controls the generation of listed output. This directive may be placed anywhere in the control stream. By default, no lists will be generated during the updating sequence.

When /LIST is specified without any optional parameters, an audit trail of the UPDATE operation will be generated. When the /LIST directive (without parameters) is followed by a /COPY directive, a top-of-form eject is performed before the first record of each file is listed.

More than one /LIST directive can be specified per control stream. However, the additional /LIST directives do not reset options. The /NOLIST directive must be used to reset options.

When the FRST parameter is specified without the UPDT parameter, the file number and the record count for each file is printed. Note that /LIS FRST is intended for use only when checking file sizes. Use of /ADD, /DEL, or /REP can cause erroneous results.

See also the /USR directive.

Syntax:

/LIST [FRST][,UPDT]

where:

FRST is an optional parameter that indicates that the first record of each file is to be listed. Header records will be the first record in files created in library format.

UPDT is an optional parameter that indicates that all control directives and records affected by /ADD, /DELETE, or /REPLACE directives are to be listed.

11.6.13 /MOUNT Directive

The /MOUNT directive allows the user to mount/dismount input tapes. When performing symbolic editing, the directive is permitted only after the /END directive.

The information on the directive (from column 1 to 70, exclusive of trailing blanks) is output to the console teletype, followed by the message

UPDATE - MOUNT INPUT VOL ON MT UNIT! R,A,H ?

The UPDATE utility is held until you enter a response.

Syntax:

/MOUNT lfc[,text]

where:

lfc is the logical file code to which a magnetic tape is assigned for input.

text user comments can be output to the OPCOM console up to, but not including, column 70.

11.6.14 /NBL (No Blank Sequence Field) Directive

The /NBL directive terminates the blanking of the sequence number field (i.e., terminates a /BLK directive). This directive may be placed anywhere in the control stream.

Syntax:

/NBL

Example:

```
/COPY 1
/SKIP 1
/NBL
```

11.6.15 /NOLIST Directive

The /NOLIST directive controls the generation of the control stream. Specified options FRST and UPDT have the same significance as for the /LIST directive. These options are reset for /NOLIST. If neither is specified, all listing output is terminated. When the FRST parameter is specified without the UPDT parameter, the file number and the record count for each file is printed.

See also the /USR directive.

Syntax:

/NOLIST [FRST][,UPDT]

where:

FRST is an optional parameter that inhibits the listing of the first record of each file.

UPDT is an optional parameter that inhibits the listing of all control directives and records affected by /ADD, /DELETE, or /REPLACE directives.

11.6.16 /NOSEQN Directive

The /NOSEQN directive will terminate the sequence of source statements. This directive may be placed anywhere in the control stream.

Syntax:

/NOSEQN

11.6.17 /REPLACE Directive

The /REPLACE directive indicates that the specified set of source lines from the input file are to be omitted from the output file and replaced with an alternate set. The alternate set may be smaller or larger than the original set of source lines. All statements following the /REPLACE directive and up to the next control directive are inserted on the output file.

Syntax:

/REPLACE start[,end]

where:

start identifies the first source line to be replaced.

end is an optional parameter that identifies the last source line to be replaced. If not specified, end will be assumed to have the same value as start.

11.6.18 /REWIND Directive

The /REWIND directive provides the capability for rewinding input or output files.

If the library option is set, the SO file must not be specified.

Syntax:

/REWIND lfc[,lfc],...

where:

lfc is the logical file code associated with the device to be rewound.

11.6.19 /SCAN Directive

The /SCAN directive sets the number of characters to scan on the remaining directives. This optional directive should be first to ensure unwanted data is not scanned.

Syntax:

/SCAN n

where:

n is the number of characters to be scanned; the value must be from 6 to 80.

11.6.20 /SELECT Directive

The /SELECT directive gives the user the capability of selecting alternate input devices. The utility always assumes that input will be from the file or device assigned to SI1. When an input device is selected, all subsequent control directives (with the exception of /BKSP, /MOUNT and /REW) will pertain to the selected device or file.

Syntax:

/SELECT lfc[,BLOCKED]

where:

lfc is the logical file code to which input is assigned.

BLOCKED is an optional parameter which though syntatically correct, should not be used in the MPX-32 environment. In MPX-32, the default is blocked and the Assignment statement should be used to specify an unblocked file.

11.6.21 /SEQUENCE Directive

The /SEQUENCE directive provides the capability to sequentially number source statements. This directive may be placed anywhere in the control stream. By default, statements will not be sequenced.

Syntax:

/SEQUENCE [id], [inc], [HOLD]

where:

id is an optional parameter of one to three alphanumeric characters which will be placed in columns 73-75 of all source output records. At least one character must be non-numeric. If omitted, id is assumed to be blanks.

inc is an optional parameter of one to four numeric characters which will be placed in columns 76-80 of all source output records. The value of inc will be used to increment the sequence number. If omitted, inc is assumed to be 1.

HOLD is an optional parameter that indicates that sequencing will be continued until another /SEQUENCE or a /NOSEQUENCE directive is encountered. If the HOLD option is specified, the beginning sequence number for all subsequent files will be reset to zero and the file will be sequenced as specified by the id and inc options. If the HOLD option is omitted, sequencing will be discontinued after the current file.

11.6.22 /SKIP Directive

The /SKIP directive provides the capability for skipping complete input files.

Syntax:

/SKIP header

where:

header is a 1- to 8-character delimiter. If header is alphanumeric, UPDATE will skip all input files up to the header record defined by header. If header is a numeric string, the utility will skip the number of files represented by the string.

11.6.23 /USR Directive

The /USR directive is used to access files belonging to another user.

Note that if the user name changes at any point in the UPDATE directive sequence, /USR must precede the first directive in the control stream that requires user name identification, i.e., the user name supplied via TSM or job control must be re-established to the default user name to be used by UPDATE for second pass processing.

When a user name is not re-established, UPDATE displays a reminder message:

> USERNAME WITH A KEY WAS CHANGED - THE USER MUST USE /USR TO RESET

Syntax:

> /USR [username[,key]]

where:

username — is the 1- to 8-character username (blank signifies no username).

key — is the 1- to 8-character user's key (if any).

11.6.24 /WEOF Directive

The /WEOF directive gives the user the capability of writing an EOF on the output (SO) file. This directive is not allowed when output is in library format.

Syntax:

> /WEOF

11.7 Listings

Not supplied; they are dependent on the assignments and processing that were requested.

11.8 Errors

The following error diagnostic messages will inhibit UPDATE from executing or cause an abort condition during execution.

INVALID OPTION SPECIFICATION

$OPTION 4 cannot be specified in conjunction with $OPTION 2.

lfc FILE IS NOT ASSIGNED

lfc is the logical file code of the implied input file. There is no file or device assigned to this file code.

INVALID UPDATE DIRECTIVE

The directive is not a valid update directive.

INVALID UPDATE DIRECTIVE SEQUENCE

An /END directive did not follow an /ADD, /REPLACE, or /DELETE sequence.

UPDATE SEQUENCE ERROR

The START or /END address on an /ADD, /REPLACE, or /DELETE statement is not sequential.

xxxxxxxx NOT FOUND

The program represented by xxxxxxxx cannot be found.

IMPROPER HEADER FORMAT

An attempt was made to create a library file that contained an improperly formatted HEADER RECORD.

IMAGE NOT COMPRESSED

Illegal mixture of standard and compressed source images.

CHECKSUM ERROR

A checksum error was detected in the record just read.

lfc FILE AT END-OF-MEDIUM

The file or device assigned to this file code is at EOM

lfc FILE, UNRECOVERABLE I/O ERROR

An I/O error has been encountered on the file or device assigned to this file code.

EOF DETECTED BEFORE SPECIFIED LINE NUMBER

An EOF was detected before the line number indicated in an /ADD, /REPLACE, or /DELETE directive could be found.

LIBRARY MODE, DIRECTIVE NOT ALLOWED

A /WEOF directive was detected in the control stream and the library mode is in affect.

lfc FILE ASSIGNED TO UTY FILE

The UTY file code cannot be assigned by the user.

lfc FILE, INVALID BLOCKING BUFFER CONTROL POINTER

An input operation for a blocked file assigned to this lfc was not successful. Either the file is unblocked or the file has been destroyed.

PERMANENT FILE NONEXISTENT

Indicates that a specified file name cannot be located in the SMD.

INVALID DEVICE TYPE

A device type is not supported by this utility.

FILE ALLOCATION DENIED

Allocation for file was denied.

DEVICE ALLOCATION DENIED

Allocation for device was denied.

UNABLE TO ASSIGN username

Indicates the specified user name cannot be assigned.

11.9 Examples

Example 1 is used to update the first file on the tape "INPUT" and place the updated file on the tape "OPUT".

```
$JOB EXAMPLE1 MEYERS
$ASSIGN3 SI1=MT,INPT SO=MT,OPUT
$EXECUTE UPDATE
/ADD 25
(statements to be added)
/DELETE 27, 28
/REPLACE 45,51
(replacement statements)
/END
/EXIT
$EOJ
$$
```

Example 2 is used to add and delete statements. /ADD 0 is used to make additions to the beginning of an existing file. An updated and compressed copy of PROG11 will be output on tape "OPUT", and a listing of all statements affected by the update will be updated. Note that INPT is in library format, which allows the user to skip to the header PROGII. The output file, OPUT, will not be in library format because OPTION 2 is not specified.

```
$JOB EXAMPLE2 MEYERS
$OPTION 1
$ASSIGN3 SI1=MT,INPT SO=MT,OPUT
$EXECUTE UPDATE
/LIST UPDT,FRST
/SKIP PROG11
/ADD 0
(statements to be added)
/DELETE 1,5
/ADD 5
(statements to be added)
/END
/EXIT
$EOJ
$$
```

Example 3 is used to copy all files up to and including the eleventh file onto the tape "OPUT" and to modify and insert a program from the SI2 file. The primary input files will be reselected, and the remaining SI files will be copied. A listing of the updates made to the inserted file will be output.

```
$JOB EXAMPLE3 MEYERS
$ASSIGN3 SI1=MT,INPT SI2=MT,TAPE SO=MT,OPUT
$EXECUTE UPDATE
/LIST UPDT
/COPY 11
/SELECT SI2
/REPLACE 24, 33
(replacement cards go here)
/END
/SELECT SI1
/COPY 51
/EXIT
$EOJ
$$
```

Example 4 is used to convert a punched card deck in compressed format to a standard format punched deck. The standard deck will be sequenced by 10, and the 3-character id of ABC will be placed in columns 73-75 of each card during the conversion.

```
$JOB EXAMPLE4 MEYERS
$ASSIGN2 SO=SBO,1000
$ASSIGN4 SI1=SYC
$EXECUTE UPDATE
/SEQN ABC, 10
/COPY 1
/EXIT
(compressed deck)
$EOJ
$$
```

Example 5 is used to position the output tape (that was created in library format) at the end of the last file. The program "NEW" will be sequenced by one and inserted (with a header record) on the output file. A new library End-of-Tape marker will be written on the output file.

```
$JOB EXAMPLE5 MEYERS
$OPTION 2
$ASSIGN3 SI2=MT,INPT,,U SO=MT,OPUT
$ASSIGN4 SI1=SO
$EXECUTE UPDATE
/SEQN
/SKIP END
/SELECT SI2
/INSERT NEW1, *NEW 1 REV C JULY 12, 1972
/EXIT
$EOJ
$$
```

Example 6 is used to list all the header records on the library tape "INPT". The file number and record count for each file are printed.

```
$JOB EXAMPLE6 MEYERS
$ASSIGN3 SI1=MT,INPT
$EXECUTE UPDATE
/LIST FRST
/COPY END
/EXIT
$EOJ
$$
```

Example 7 is used to copy PROG13, PROG16, and PROG17 from "INPT" to "OPUT", and to output listings of the three programs. The programs that are skipped will not be listed.

```
$JOB EXAMPLE7 MEYERS
$ASSIGN3 SI1=MT,INPT SO=MT,OPUT
$EXECUTE UPDATE
/LIST
/SKIP PROG13
/COPY 1
/SKIP PROG16
/COPY 2
/EXIT
$EOJ
$$
```

Example 8 is used to (1) copy the first, fifth, sixth, and twelfth files from "INPT"; (2) output compressed source to the disc file "PERMFILE"; (3) inhibit writing of EOFs detected on the input file to the output file; and (4) assemble all four files.

```
$JOB EXAMPLE8 MEYERS
$OPTION 1 4
$ASSIGN3 SI1=MT,INPT
$ASSIGN1 SO=PERMFILE
$EXECUTE UPDATE
/COPY 1
/SKIP 3
/COPY 2
/SKIP 5
/COPY 1
/WEOF
/EXIT
$EOJ
$JOB ASSEMBLE EXAMPLE8
$ASSIGN1 SI=PERMFILE
$EXECUTE ASSEMBLE
$EOJ
$$
```

Example 9 is used to insert the new revision level of PROG10 in the new master file and to list all the header records on the new master file.

```
$JOB EXAMPLE9 MEYERS
$OPTION 2
$ASSIGN3 SI1=MT,MAST SI2=MT,INSE SO=MT,NEW1
$EXECUTE UPDATE
/LIST FRST
/COPY PROG10
/SELECT SI2
/INSERT PROG10,*PROG10 REVB JULY 12, 1971
/SELECT SI1
/SKIP PROG11
/COPY END
/EXIT
$EOJ
$$
```

APPENDIX A
MPX-32 DEVICE ACCESS

Throughout the reference manual, the generic descriptor 'devmnc' is used to indicate that a device can be specified.

Under MPX-32, device addresses are specified using a combination of three levels of identification. They are device type, device channel/controller address, and device address/subaddress.

A device can be specified using the generic device type only, which will result in allocation of the first available device of the type requested.

A second method of device specification is achieved by using the generic device type and specifying the channel/controller address. This results in allocation of the first available device of the type requested on the specified channel or controller.

The third method of device selection requires specification of the device type, channel/controller, and device address/subaddress. This method allows specification of a particular device.

1. Special Device Specifications and Handling

1.1 Magnetic Tape

For magnetic tape, a reel identifier, multivolume number, and unblocking can be part of the device mnemonic.

Syntax:

lfc= device [,reel | ,reel,volume | ,reel,volume,U | ,reel,,U]

where:

device	is any one of the four levels of device specification described above.
reel	specifies a one- to four-character identifier for the reel. This parameter is required in batch. This parameter is not required in TSM and if not specified, the default is SCRA (Scratch).
volume	if multivolume tape, indicates volume number. Default: not multivolume (0).
U	the tape is optionally unblocked. Default: blocked.

Commas in this specification are significant. If an option is not specified, e.g., a reel identifier, but another option is specified, e.g., U, commas must be inserted for all non-specified options in between, e.g.,

```
MT1000,,,U
```

There must be no embedded blanks within the entire device mnemonic.

When the task is activated that has an assignment to tape, a MOUNT message indicates the name of the task and other information on the OPCOM console:

{TASK / jobno} {,taskname,taskno} MOUNT reel VOL volume ON devmnc DEV,R,A,H?

where:

jobno	if the task is part of a batch job, identifies the job by job number.
taskname	is the name of the task to which the tape is assigned.
taskno	is the number of the task.

reel — if the assignment is a multivolume tape, indicates the reel identifier specified in the assignment. This parameter is required in batch. This parameter is not required in TSM and if not specified, the default is SCRA.

volume — identifies the volume number to mount if multivolume tape.

devmnc — is the device mnemonic for the tape unit selected in response to the assignment. If a specific channel and subaddress are supplied in the assignment, the specific tape drive is selected and named in the message. Otherwise, a unit is selected by the system and its complete address is named in the message.

DEV,R,A,H — the device listed in the message can be allocated and the task resumed (R), a different device can be selected (DEV), the task can be aborted (A), or the task can be held with the specified device deallocated (H). If an 'R' response is given and a high speed XIO tape drive is being used, its density can be changed when the software select feature is enabled on the tape unit front panel. If specified, it will override any specification made at assignment. Values are:

N or 800	indicates 800 bpi nonreturn to zero inverted (NRZI)
P or 1600	indicates 1600 bpi phase encoded (PE)
G or 6250	indicates 6250 bpi group coded recording (GCR) Default.

Example usage: RN, R1600, etc.

Note: Do not insert blanks or commas.

Response:

To indicate the drive specified in the MOUNT message is ready and proceed with the task, mount the tape on the drive and type R (Resume), optionally followed by a density specification if the drive is a high speed XIO tape unit. To abort the task, type A (Abort). To hold the task and deallocate the specified device, type H (Hold). The task can then be resumed by the OPCOM CONTINUE command, at which time a tape drive will be selected by the system and the MOUNT message redisplayed.

To select a tape drive other than the drive specified in the message, enter the mnemonic of the drive you want to use. Any of the three levels of device identification can be used. The MOUNT message is reissued. Mount the tape and type R if satisfactory, or if not satisfactory, abort, override, or hold as just described.

1.2 Temporary Disc File Size

For a temporary disc file, size must be specified and unblocking is optional.

Syntax:

lfc = device,size [,U]

where:

size	specifies the number of 192-word blocks required.
U	the file is optionally unblocked. Default: blocked.

Examples of the three methods of device specification follow:

Type 1 - Generic Device Class

$ASSIGN3 DEV=M9,,1

In this example, the device assigned to logical file code (lfc) "DEV" will be any 9-track tape unit on any channel. The multivolume reel number is 1. The reel identifier is SCRA.

Type 2 - Generic Device Class and Channel/Controller

$ASSIGN3 DEV=M910,MORK,,U

In this example, the device assigned to logical file code (lfc) "DEV" will be the first available 9-track tape unit on channel 10. The specification is invalid if a 9-track tape unit does not exist on the channel. The reel identifier is supplied. This is not a multivolume tape. It is, however, unblocked.

Type 3 - Specific Device Request

$ASSIGN3 DEV=M91001

In this example, the device assigned to logical file code (lfc) "DEV" will be the 9-track tape unit 01 on channel 10. The specification is invalid if unit 01 on channel 10 is not a 9-track tape. The tape reel identifier is SCRA; the tape is blocked and is not multivolume.

2. GPMC Devices

GPMC/GPDC device specifications are in keeping with the general structure just described. For instance, the terminal at subaddress 04 on GPMC 01 whose channel address is 20 would be identified as follows:

```
$ASSIGN3 DEV=TY2004
```

3. NULL Device

A special device type "NU" is available for NULL device specifications. Files accessed using this device type generate an end-of-file (EOF) upon attempt to read and normal completion upon attempt to write.

4. OPCOM Console

Logical file codes are assigned to the OPCOM console by using the device type "CT".

5. Special System Files

There are four special mnemonics provided for access to special system files: SLO, SBO, SGO and SYC. These are assigned via the $ASSIGN2 statement, as is:

```
$ASSIGN2 OUT=SLO,printlines
```

For non-batch tasks, SLO and SBO files are allocated dynamically by the system and used to disc buffer output to a device selected automatically. For batch tasks, use of SLO and SBO files is identical, except that automatic selection of a device can be overridden by assigning a specific file or device.

SGO and SYC assignments are used for batch processing. See Section 7.6.

Dev Type Code	Device	Device Description
00	CT	Operator Console (Not Assignable)
01	DC	Any Disc Unit
02	DM	Any Moving Head Disc
03	DF	Any Fixed Head Disc
04	MT	Any Magnetic Tape Unit
05	M9	Any 9-Track Magnetic Tape Unit
06	M7	Any 7-Track Magnetic Tape Unit
07	CD	Any Card Reader-Punch
08	CR	Any Card Reader
09	CP	Any Card Punch
0A	LP	Any Line Printer
0B	PT	Any Paper Tape Reader-Punch
0C	TY	Any Teletypewriter (Other than Console)
0D	CT	Operator Console (Assignable)
0E	FL	Floppy Disc
0F	NU	Null Device
10	CA	Communications Adapter (Binary Synchronous/Asynchronous)
11	U0	Available for user-defined applications
12	U1	Available for user-defined applications
13	U2	Available for user-defined applications
14	U3	Available for user-defined applications
15	U4	Available for user-defined applications
16	U5	Available for user-defined applications
17	U6	Available for user-defined applications
18	U7	Available for user-defined applications
19	U8	Available for user-defined applications
1A	U9	Available for user-defined applications
1B	LF	Line Printer/Floppy Controller (used only with SYSGEN)

Table A-1: Device Type Codes

6. Samples

A description of device selection possibilities would be constructed as follows:

DISC

DC	Any Disc
DM	Any Moving Head Disc
DM08	Any Moving Head Disc on Channel 08
DM0801	Moving Head Disc 01 on Channel 08
DF	Any Fixed Head Disc
DF04	Any Fixed Head Disc on Channel 04
DF0401	Fixed Head Disc 01 on Channel 04

TAPE

MT	Any Magnetic Tape
M9	Any 9-track Magnetic Tape
M910	Any 9-track Magnetic Tape on Channel 10
M91002	9-track Magnetic Tape 02 on Channel 10
M7	Any 7-track Magnetic Tape
M712	Any 7-track Magnetic Tape on Channel 12
M71201	7-track Magnetic Tape 01 on Channel 12

CARD EQUIPMENT

CD	Any Card Reader-Punch
CR	Any CR
CR78	Any CR on Channel 78
CR7800	CR on Channel 78 Subaddress 00
CP	Any CP
CP7C	Any CP on Channel 7C
CP7C00	CP on Channel 7C Subaddress 00

LINE PRINTER

LP	Any LP
LP7A	Any LP on Channel 7A
LP7A00	LP on Channel 7A Subaddress 00

APPENDIX B
SYSTEM SERVICES CROSS REFERENCE CHARTS

USER LEVEL SYSTEM SERVICES - MACRO NAME

	MACRO	DESCRIPTOR	SVC 1,XX	MODULE, E.P.	REF MANUAL SECTION
	M.ACTV	ACTIVATE TASK	52	H.MONS,15	8.2.1
	M.ADRS	MEMORY ADDRESS INQUIRY	44	H.MONS,3	8.3.1
*	M.ALOC	ALLOCATE FILE OR PERIPHERAL DEVICE	40	H.MONS,21	7.8.1
	M.ANYW	WAIT FOR ANY MSG, END ACTION, OR BRK	7C	H.MONS,37	8.2.2
	M.ASYNCH	SET ASYNCHRONOUS TASK INTERRUPT	1C	H.MONS,68	8.2.3
	M.BACK	BACKSPACE RECORD	35	H.IOCS,9	7.8.2
		OR			
		FILE	36	H.IOCS,19	
	M.BORT	ABORT SPEC. TASK	56	H.MONS,19	8.2.4
		OR			
		SELF	57	H.MONS,20	
		OR			
		WITH EXT. MESSAGE	62	H.MONS,28	
	M.BRK	BREAK/TASK INTERRUPT LINK	6E	H.MONS,46	8.2.5
	M.BRKXIT	EXIT FROM TASK INTERRUPT LEVEL	70	H.MONS,48	8.2.6
**	M.CDJS	SUBMIT JOB FROM DISC FILE	61	H.MONS,27	8.2.7
	M.CLSE	CLOSE FILE	39	H.IOCS,23	7.8.3
	M.CONADB	CONVERT ASCII DECIMAL TO BINARY	28	H.TSM,7	5.6.3.1
	M.CONAHB	CONVERT ASCII HEX TO BINARY	29	H.TSM,8	5.6.3.2
	M.CONBAD	CONVERT BINARY TO ASCII DECIMAL	2A	H.TSM,9	5.6.3.3

*Reduced functionality under Memory-Only MPX-32
**Not supported under Memory-Only MPX-32

USER LEVEL SYSTEM SERVICES - MACRO NAME

(CONTINUED)

	MACRO	DESCRIPTOR	SVC 1,XX	MODULE, E.P.	REF MANUAL SECTION
	M.CONBAH	CONVERT BINARY TO ASCII HEX	2B	H.TSM,10	5.6.3.4
	M.CONN	CONNECT TASK TO INTERRUPT	4B	H.MONS,10	8.2.8
**	M.CREATE	CREATE PERM FILE	75	H.FISE,12	7.8.4
	M.CWAT	SYSTEM CONSOLE WAIT	3D	H.IOCS,26	7.8.5
*	M.DALC	DEALLOCATE FILE OR DEVICE	41	H.MONS,22	7.8.6
	M.DATE	DATE AND TIME INQUIRY	15	H.MONS,70	8.2.9
**	M.DEBUG	LOAD AND EXECUTE INTERACTIVE DEBUGGER	63	H.MONS,29	8.2.10
**	M.DELETE	DELETE PERM FILE	77	H.FISE,14	7.8.7
	M.DELTSK	DELETE TASK	5A	H.MONS,31	8.2.11
	M.DEVID	GET DEVICE MNEMONIC OR TYPE CODE	14	H.MONS,71	8.2.12
	M.DISCON	DISCONNECT TASK FROM INTERRUPT	5D	H.MONS,38	8.2.13
	M.DLTT	DELETE TIMER ENTRY	47	H.MONS,6	8.2.14
	M.DSMI	DISABLE MESSAGE TASK INTERRUPT	2E	H.MONS,57	8.2.15
	M.DSUB	DISABLE USER BREAK INTERRUPT	12	H.MONS,73	8.2.16
**	M.DUMP	MEMORY DUMP REQUEST	4F	H.MONS,12	8.3.2
	M.EAWAIT	END ACTION WAIT	1D	H.EXEC,40	8.2.17
	M.ENMI	ENABLE MESSAGE TASK INTERRUPT	2F	H.MONS,58	8.2.18

*Reduced functionality under Memory-Only MPX-32
**Not supported under Memory-Only MPX-32

USER LEVEL SYSTEM SERVICES - MACRO NAME

(CONTINUED)

	MACRO	DESCRIPTOR	SVC 1,XX	MODULE, E.P.	REF MANUAL SECTION
	M.ENUB	ENABLE USER BREAK INTERRUPT	13	H.MONS,72	8.2.19
	M.EXCL	FREE SHARED MEMORY	79	H.ALOC,14	8.3.3
	M.EXIT	TERMINATE TASK EXECUTION	55	H.MONS,18	8.2.20
**	M.FADD	PERMANENT FILE ADDRESS INQUIRY	43	H.MONS,2	7.8.8
	M.FD	FREE DYNAMIC EXTENDED INDEX DATA SPACE	6A	H.ALOC,9	8.3.5
	M.FE	FREE DYNAMIC TASK EXECUTION SPACE	68	H.ALOC,11	8.3.6
	M.FILE	OPEN FILE	30	H.IOCS,1	7.8.9
**	M.FSLR	RELEASE SYNCHRONIZATION FILE LOCK	24	H.FISE,25	7.8.10
**	M.FSLS	SET SYNCHRONIZATION FILE LOCK	23	H.FISE,24	7.8.11
	M.FWRD	ADVANCE RECORD	33	H.IOCS,7	7.8.12
		OR			
		FILE	34	H.IOCS,8	
	M.FXLR	RELEASE EXCLUSIVE FILE LOCK	22	H.FISE,23	7.8.13
**	M.FXLS	SET EXCLUSIVE FILE LOCK	21	H.FISE,22	7.8.14
	M.GADRL	GET ADDRESS LIMITS	65	H.MONS,41	8.3.7
	M.GD	GET DYNAMIC EXTENDED INDEXED DATA SPACE	69	H.ALOC,8	8.3.8
	M.GE	GET DYNAMIC TASK EXECUTION SPACE	67	H.ALOC,10	8.3.9

*Reduced functionality under Memory-Only MPX-32
**Not supported under Memory-Only MPX-32

USER LEVEL SYSTEM SERVICES - MACRO NAME

(CONTINUED)

	MACRO	DESCRIPTOR	SVC 1,XX	MODULE, E.P.	REF MANUAL SECTION
	M.GMSGP	GET MSG PARAMETERS	7A	H.MONS,35	8.2.21
	M.GRUNP	GET RUN PARAMETERS	7B	H.MONS,36	8.2.22
	M.HOLD	PROGRAM HOLD REQUEST	58	H.MONS,25	8.2.23
	M.ID	GET TASK NUMBER	64	H.MONS,32	8.2.24
*	M.INCL	GET SHARED MEMORY	72	H.ALOC,13	8.3.10
	M.INT	ACTIVATE TASK INTERRUPT	6F	H.MONS,47	8.2.25
**	M.LOG	PERMANENT FILE LOG	73	H.MONS,33	7.8.15
	M.MYID	GET TASK NUMBER	64	H.MONS,32	8.2.26
**	M.OLAY	LOAD OVERLAY	50	H.MONS,13	8.2.27
		OR			
		LOAD AND EXECUTE OVERLAY	51	H.MONS,14	
**	M.PDEV	PHYSICAL DEVICE INQUIRY	42	H.MONS,1	7.8.16
**	M.PERM	CHANGE TEMP FILE TO PERMANENT	76	H.FISE,13	7.8.17
	M.PGOW	TASK OPTION WORD INQUIRY	4C	H.MONS,24	8.2.28
	M.PRIL	CHANGE PRIORITY LEVEL	4A	H.MONS,9	8.2.29
	M.PTSK	PARAMETER TASK ACTIVATION	5F	H.MONS,40	8.2.30
	M.RCVR	RECEIVE MESSAGE LINK ADDRESS	6B	H.MONS,43	8.2.31
	M.READ	READ RECORD	31	H.IOCS,3	7.8.18
	M.RELP	RELEASE DUAL PORTED DISC	27	H.IOCS,27	7.8.19

*Reduced functionality under Memory-Only MPX-32
**Not supported under Memory-Only MPX-32

USER LEVEL SYSTEM SERVICES - MACRO NAME

(CONTINUED)

	MACRO	DESCRIPTOR	SVC 1,XX	MODULE, E.P.	REF MANUAL SECTION
	M.RESP	RESERVE DUAL PORTED DISC	26	H.IOCS,24	7.8.20
	M.RRES	RELEASE CHANNEL	3B	H.IOCS,13	7.8.21
	M.RSML	RESOURCEMARK LOCK	19	H.MONS,62	7.8.22
	M.RSMU	RESOURCEMARK UNLOCK	1A	H.MONS,63	7.8.23
	M.RSRV	RESERVE CHANNEL	3A	H.IOCS,12	7.8.24
	M.RWND	REWIND FILE	37	H.IOCS,2	7.8.25
	M.SETS	SET USER STATUS WORD	48	H.MONS,7	8.2.32
	M.SETT	CREATE TIMER ENTRY	45	H.MONS,4	8.2.33
*	M.SHARE	SHARE MEMORY WITH ANOTHER TASK	71	H.ALOC,12	8.3.11
	M.SMSGR	SEND MESSAGE TO SPECIFIED TASK	6C	H.MONS,44	8.2.34
	M.SMULK	UNLOCK AND DEQUEUE SHARED MEMORY	1F	H.ALOC,19	8.3.12
	M.SRUNR	SEND RUN REQUEST	6D	H.MONS,45	8.2.35
	M.SUAR	SET USER ABORT RECEIVER ADDRESS	60	H.MONS,26	8.2.36
	M.SUME	RESUME TASK EXECUTION	53	H.MONS,16	8.2.37
	M.SUSP	SUSPEND TASK EXECUTION	54	H.MONS,17	8.2.38
	M.SYNCH	SET SYNCHRONOUS TASK INTERRUPT	1B	H.MONS,67	8.2.39
	M.TBRKON	TRAP ONLINE USER'S TASK	5C	H.TSM,6	5.6.2

*Reduced functionality under Memory-Only MPX-32
**Not supported under Memory-Only MPX-32

USER LEVEL SYSTEM SERVICES - MACRO NAME

(CONTINUED)

	MACRO	DESCRIPTOR	SVC 1,XX	MODULE, E.P.	REF MANUAL SECTION
	M.TDAY	TIME-OF-DAY INQUIRY	4E	H.MONS,11	8.2.40
**	M.TSCAN	SCAN TERMINAL INPUT BUFFER	5B	H.TSM,2	5.6.1
	M.TSTE	ARITHMETIC EXCEPTION INQUIRY	4D	H.MONS,23	8.2.41
	M.TSTS	TEST USER STATUS WORD	49	H.MONS,8	8.2.42
	M.TSTT	TEST TIMER ENTRY	46	H.MONS,5	8.2.43
	M.TURNON	ACTIVATE PROGRAM AT GIVEN TIME OF DAY	1E	H.MONS,66	8.2.44
	M.TYPE	CONSOLE TYPE	3F	H.IOCS,14	7.8.26
	M.UPSP	UPSPACE	10	H.IOCS,20	7.8.27
	M.USER	USERNAME SPECIFICATION	74	H.MONS,34	7.8.28
	M.WAIT	WAIT I/O	3C	H.IOCS,25	7.8.29
	M.WEOF	WRITE EOF	38	H.IOCS,5	7.8.30
	M.WRIT	WRITE RECORD	32	H.IOCS,4	7.8.31
	M.XBRKR	EXIT FROM TASK INTERRUPT LEVEL	70	H.MONS,48	8.2.45
	M.XIEA	NO-WAIT I/O END ACTION RETURN	2C	H.IOCS,34	7.8.32
	M.XMEA	EXIT FROM MESSAGE END ACTION ROUTINE	7E	H.MONS,50	8.2.46
	M.XMSGR	EXIT FROM MESSAGE RECEIVER	5E	H.MONS,39	8.2.47
	M.XREA	EXIT RUN REQUEST END ACTION ROUTINE	7F	H.MONS,51	8.2.48

**Not supported under Memory-Only MPX-32

USER LEVEL SYSTEM SERVICES - MACRO NAME

(CONTINUED)

	MACRO	DESCRIPTOR	SVC 1,XX	MODULE, E.P.	REF MANUAL SECTION
	M.XRUNR	EXIT RUN RECEIVER	7D	H.MONS,49	8.2.49
	M.XTIME	TASK CPU EXECUTION TIME	2D	H.MONS,65	8.2.50
	N/A	ERASE OR PUNCH TRAILER	3E	H.IOCS,21	7.8.33
**	N/A	DEBUG LINK SERVICE	66	H.MONS,42	8.2.51
	N/A	EXECUTE CHANNEL PROGRAM	25	H.IOCS,10	7.8.34
	N/A	RELEASE FHD PORT	27	H.IOCS,27	7.8.35
	N/A	RESERVE FHD PORT	26	H.IOCS,24	7.8.36
	N/A	SET TABS IN UDT	59	H.TSM,5	N/A

**Not supported under Memory-Only MPX-32

USER LEVEL SYSTEM SERVICES - ALPHABETIC

		SVC 1,XX	MODULE	E.P.	REF MANUAL SECTION
ABORT SELF	M.BORT	57	H.MONS	20	8.2.4
ABORT SPECIFIED TASK	M.BORT	56	H.MONS	19	8.2.4
ABORT WITH EXTENDED MESSAGE	M.BORT	62	H.MONS	28	8.2.4
ACTIVATE PROGRAM AT GIVEN TIME OF DAY	M.TURNON	1E	H.MONS	66	8.2.44
ACTIVATE TASK INTERRUPT	M.INT	6F	H.MONS	47	8.2.25
ACTIVATE TASK	M.ACTV	52	H.MONS	15	8.2.1
ADVANCE FILE OR RECORD	M.FWRD	34 33	H.IOCS H.IOCS	8 7	7.8.12 7.8.12
ALLOCATE FILE OR PERIPHERAL DEVICE	M.ALOC	40	H.MONS	21	7.8.1
ARITHMETIC EXCEPTION INQUIRY	M.TSTE	4D	H.MONS	23	8.2.41
BACKSPACE FILE OR RECORD	M.BACK	36 35	H.IOCS H.IOCS	19 9	7.8.2 7.8.2
BREAK/TASK INTERRUPT LINK	M.BRK	6E	H.MONS	46	8.2.5
CHANGE PRIORITY LEVEL (PRIV)	M.PRIL	4A	H.MONS	9	8.2.29
CHANGE TEMP FILE TO PERMANENT	M.PERM	76	H.FISE	13	7.8.17
CLOSE FILE	M.CLSE	39	H.IOCS	23	7.8.3
CONNECT TASK TO INTERRUPT	M.CONN	4B	H.MONS	10	8.2.8
CONSOLE TYPE	M.TYPE	3F	H.IOCS	14	7.8.26
CONVERT ASCII DECIMAL TO BINARY	M.CONADB	28	H.TSM	7	5.6.3.1

UNPRIVILEGED USER LEVEL SYSTEM SERVICES - ALPHABETIC

(CONTINUED)

		SVC 1,XX	MODULE	E.P.	REF MANUAL SECTION
CONVERT ASCII HEX TO BINARY	M.CONAHB	29	H.TSM	8	5.6.3.2
CONVERT BINARY TO ASCII DECIMAL	M.CONBAD	2A	H.TSM	9	5.6.3.3
CONVERT BINARY TO ASCII HEX	M.CONBAH	2B	H.TSM	10	5.6.3.4
CREATE PERM FILE	M.CREATE	75	H.FISE	12	7.8.4
CREATE TIMER ENTRY	M.SETT	45	H.MONS	4	8.2.33
*DATE AND TIME INQUIRY	M.DATE	15	H.MONS	70	8.2.9
DEALLOCATE FILE OR DEVICE	M.DALC	41	H.MONS	22	7.8.6
DEBUG LINK SERVICE	N/A	66	H.MONS	42	8.2.51
DELETE PERM FILE	M.DELETE	77	H.FISE	14	7.8.7
DELETE TASK	M.DELTSK	5A	H.MONS	31	8.2.11
DELETE TIMER ENTRY	M.DLTT	47	H.MONS	6	8.2.14
DISABLE MESSAGE TASK INTERRUPT	M.DSMI	2E	H.MONS	57	8.2.15
*DISABLE USER BREAK INTERRUPT	M.DSUB	12	H.MONS	73	8.2.16
DISCONNECT TASK FROM INTERRUPT	M.DISCON	5D	H.MONS	38	8.2.13
ENABLE MESSAGE TASK INTERRUPT	M.ENMI	2F	H.MONS	58	8.2.18
*ENABLE USER BREAK INTERRUPT	M.ENUB	13	H.MONS	13	8.2.19
END ACTION WAIT	M.EAWAIT	1D	H.EXEC	40	8.2.17
ERASE OR PUNCH TRAILER	N/A	3E	H.IOCS	21	7.8.33

*NEW

UNPRIVILEGED USER LEVEL SYSTEM SERVICES - ALPHABETIC

(CONTINUED)

		SVC 1,XX	MODULE	E.P.	REF MANUAL SECTION
EXECUTE CHANNEL PROGRAM	N/A	25	H.IOCS	10	7.8.34
EXIT FROM MSG RCVR	M.XMSGR	5E	H.MONS	39	8.2.47
EXIT FROM MSG END ACTION ROUTINE	M.XMEA	7E	H.MONS	50	8.2.46
EXIT FROM TASK INTERRUPT LEVEL	M.BRKXIT M.XBRKR	70	H.MONS	48	8.2.6 8.2.45
EXIT RUN RCVR	M.XRUNR	7D	H.MONS	49	8.2.49
EXIT FROM RUN REQUEST END ACTION ROUTINE	M.XREA	7F	H.MONS	51	8.2.48
FREE DYNAMIC EXT INDEX DATA SPACE	M.FD	6A	H.ALOC	9	8.3.5
FREE DYNAMIC TASK EXECUTION SPACE	M.FE	68	H.ALOC	11	8.3.6
FREE SHARED MEMORY	M.EXCL	79	H.ALOC	14	8.3.3
GET ADDRESS LIMITS	M.GADRL	65	H.MONS	41	8.3.7
*GET DEVICE MNEMONIC OR TYPE CODE	M.DEVID	14	H.MONS	71	8.2.12
GET DYNAMIC EXTENDED INDEXED DATA SPACE	M.GD	69	H.ALOC	8	8.3.8
GET DYNAMIC TASK EXECUTION SPACE	M.GE	67	H.ALOC	10	8.3.9
GET MSG PARAMETERS	M.GMSGP	7A	H.MONS	35	8.2.21
GET RUN PARAMETERS	M.GRUNP	7B	H.MONS	36	8.2.22
GET SHARED MEMORY	M.INCL	72	H.ALOC	13	8.3.10
GET TASK NUMBER	M.ID M.MYID	64	H.MONS	32	8.2.24 8.2.26

*NEW

UNPRIVILEGED USER LEVEL SYSTEM SERVICES - ALPHABETIC

(CONTINUED)

		SVC 1,XX	MODULE	E.P.	REF MANUAL SECTION
LOAD AND EXECUTE INTERACTIVE DEBUGGER	M.DEBUG	63	H.MONS	29	8.2.10
LOAD OVERLAY AND	M.OLAY	5O	H.MONS	13	8.2.27
LOAD AND EXECUTE OVERLAY		51	H.MONS	14	8.2.27
MEMORY ADDRESS INQUIRY	M.ADRS	44	H.MONS	3	8.3.1
MEMORY DUMP REQUEST	M.DUMP	4F	H.MONS	12	8.3.2
NO-WAIT I/O END ACTION RETURN	M.XIEA	2C	H.IOCS	34	7.8.32
OPEN FILE	M.FILE	30	H.IOCS	1	7.8.9
PARAMETER TASK ACTIVATION (PRIV)	M.PTSK	5F	H.MONS	40	8.2.30
PERMANENT FILE ADDRESS INQUIRY	M.FADD	43	H.MONS	2	7.8.8
PERMANENT FILE LOG	M.LOG	73	H.MONS	33	7.8.15
PHYSICAL DEVICE INQUIRY	M.PDEV	42	H.MONS	1	7.8.16
PROGRAM HOLD REQUEST	M.HOLD	58	H.MONS	25	8.2.23
READ RECORD	M.READ	31	H.IOCS	3	7.8.18
RECEIVE MESSAGE LINK ADDRESS	M.RCVR	6B	H.MONS	43	8.2.31
RELEASE CHANNEL (PRIV)	M.RRES	3B	H.IOCS	13	7.8.21
RELEASE DUAL PORTED DISC	M.RELP	27	H.IOCS	27	7.8.19
RELEASE EXCLUSIVE FILE LOCK	M.FXLR	22	H.FISE	23	7.8.13
RELEASE FHD PORT (PRIV)	N/A	27	H.IOCS	27	7.8.35

*NEW

UNPRIVILEGED USER LEVEL SYSTEM SERVICES - ALPHABETIC

(CONTINUED)

		SVC 1,XX	MODULE	E.P.	REF MANUAL SECTION
RELEASE SYNCHRONIZATION FILE LOCK	M.FSLR	24	H.FISE	25	7.8.10
RESERVE CHANNEL (PRIV)	M.RSRV	3A	H.IOCS	12	7.8.24
*RESERVE DUAL PORTED DISC	M.RESP	26	H.IOCS	24	7.8.20
RESERVE FHD PORT (PRIV)	N/A	26	H.IOCS	24	7.8.36
RESOURCEMARK LOCK	M.RSML	19	H.MONS	62	7.8.22
RESOURCEMARK UNLOCK	M.RSMU	1A	H.MONS	63	7.8.23
RESUME TASK EXECUTION	M.SUME	53	H.MONS	16	8.2.37
REWIND FILE	M.RWND	37	H.IOCS	2	7.8.25
SCAN TERMINAL INPUT BUFFER	M.TSCAN	5B	H.TSM	2	5.6.1
SEND MESSAGE TO SPECIFIED TASK	M.SMSGR	6C	H.MONS	44	8.2.34
SEND RUN REQUEST	M.SRUNR	6D	H.MONS	45	8.2.35
SET ASYNCHRONOUS TASK INTERRUPT	M.ASYNCH	1C	H.MONS	68	8.2.3
SET EXCLUSIVE FILE LOCK	M.FXLS	21	H.FISE	22	7.8.14
SET SYNCHRONIZATION FILE LOCK	M.FSLS	23	H.FISE	24	7.8.11
SET SYNCHRONOUS TASK INTERRUPT	M.SYNCH	1B	H.MONS	67	8.2.39
SET TABS IN UDT	N/A	59	H.TSM	5	N/A
SET USER STATUS WORD	M.SETS	48	H.MONS	7	8.2.32
SET USER ABORT RECEIVER ADDRESS	M.SUAR	60	H.MONS	26	8.2.36

*NEW

UNPRIVILEGED USER LEVEL SYSTEM SERVICES - ALPHABETIC

(CONTINUED)

		SVC 1,XX	MODULE	E.P.	REF MANUAL SECTION
SHARE MEMORY WITH ANOTHER TASK	M.SHARE	71	H.ALOC	12	8.3.11
SUBMIT JOB FROM DISC FILE	M.CDJS	61	H.MONS	27	8.2.7
SUSPEND TASK EXECUTION	M.SUSP	54	H.MONS	17	8.2.38
SYSTEM CONSOLE WAIT	M.CWAT	3D	H.IOCS	26	7.8.5
TASK CPU EXECUTION TIME	M.XTIME	2D	H.MONS	65	8.2.50
TASK OPTION WORD INQUIRY	M.PGOW	4C	H.MONS	24	8.2.28
TERMINATE TASK EXECUTION	M.EXIT	55	H.MONS	18	8.2.20
TEST TIMER ENTRY	M.TSTT	46	H.MONS	5	8.2.43
TEST USER STATUS WORD	M.TSTS	49	H.MONS	8	8.2.42
TIME-OF-DAY INQUIRY	M.TDAY	4E	H.MONS	11	8.2.40
TRAP ONLINE USER'S TASK	M.TBRKON	5C	H.TSM	6	5.6.2
UNLOCK AND DEQUEUE SHARED MEMORY	M.SMULK	1F	H.ALOC	19	8.3.12
*UPSPACE	M.UPSP	10	H.IOCS	20	7.8.27
USERNAME SPECIFICATION	M.USER	74	H.MONS	34	7.8.28
WAIT I/O	M.WAIT	3C	H.IOCS	25	7.8.29
WAIT FOR ANY MSG, END ACTION, OR BRK	M.ANYW	7C	H.MONS	37	8.2.2
WRITE EOF	M.WEOF	38	H.IOCS	5	7.8.30
WRITE RECORD	M.WRIT	32	H.IOCS	4	7.8.31

*NEW

USER LEVEL SYSTEM SERVICES - SVC ORDER

SVC 1	DESCRIPTION	MODULE,EP	MACRO	REF MANUAL SECTION
00-0F	RESERVED			
10	UPSPACE	H.IOCS,20	M.UPSP	7.8.27
11	RESERVED			
12	DISABLE USER BREAK INTERRUPT	H.MONS,73	M.DSUB	8.2.16
13	ENABLE USER BREAK INTERRUPT	H.MONS,72	M.ENUB	8.2.19
14	GET DEVICE MNEMONIC OR TYPE CODE	H.MONS,71	M.DEVID	8.2.12
15	DATE AND TIME INQUIRY	H.MONS,70	M.DATE	8.2.9
16	RESERVED			
17	ADI I/O	H.ADIO,8	N/A	N/A
18	ADI EAI	H.ADIO,9	N/A	N/A
19	RESOURCEMARK LOCK	H.MONS,62	M.RSML	7.8.22
1A	RESOURCEMARK UNLOCK	H.MONS,63	M.RSMU	7.8.23
1B	SET SYNCHRONOUS TASK INTERRUPT	H.MONS,67	M.SYNCH	8.2.39
1C	SET ASYNCHRONOUS TASK INTERRUPT	H.MONS,68	M.ASYNCH	8.2.3
1D	END ACTION WAIT	H.EXEC,40	M.EAWAIT	8.2.17
1E	ACTIVATE PROGRAM AT GIVEN TIME OF DAY	H.MONS,66	M.TURNON	8.2.44
1F	UNLOCK AND DEQUEUE SHARED MEMORY	H.ALOC,19	M.SMULK	8.3.12
20	RESERVED			
21	SET EXCLUSIVE FILE LOCK	H.FISE,22	M.FXLS	7.8.14

USER LEVEL SYSTEM SERVICES - SVC ORDER (CONTINUED)

SVC 1	DESCRIPTION	MODULE,EP	MACRO	REF MANUAL SECTION
22	RELEASE EXCLUSIVE FILE LOCK	H.FISE,23	M.FXLR	7.8.13
23	SET SYNCHRONIZATION FILE LOCK	H.FISE,24	M.FSLS	7.8.11
24	RELEASE SYNCHRONIZATION FILE LOCK	H.FISE,25	M.FSLR	7.8.10
25	EXECUTE CHANNEL PROGRAM	H.IOCS,10	N/A	7.8.34
26	RESERVE FHD PORT RESERVE DUAL PORTED DISC	H.IOCS,24 H.IOCS,24	N/A M.RESP	7.8.36 7.8.20
27	RELEASE FHD PORT RELEASE DUAL PORTED DISC	H.IOCS,27 H.IOCS,27	N/A M.RELP	7.8.35 7.8.19
28	CONVERT ASCII DECIMAL TO BINARY	H.TSM,7	M.CONADB	5.6.3.1
29	CONVERT ASCII HEX TO BINARY	H.TSM,8	M.CONAHB	5.6.3.2
2A	CONVERT BINARY TO ASCII DECIMAL	H.TSM,9	M.CONBAD	5.6.3.3
2B	CONVERT BINARY TO ASCII HEX	H.TSM,10	M.CONBAH	5.6.3.4
2C	NO-WAIT I/O END ACTION RETURN	H.IOCS,34	M.XIEA	7.8.32
2D	TASK CPU EXECUTION TIME	H.MONS,65	M.XTIME	8.2.50
2E	DISABLE MESSAGE TASK INTERRUPT	H.MONS,57	M.DSMI	8.2.15

USER LEVEL SYSTEM SERVICES - SVC ORDER

(CONTINUED)

SVC 1	DESCRIPTION	MODULE,EP	MACRO	REF MANUAL SECTION
2F	ENABLE MESSAGE TASK INTERRUPT	H.MONS,58	M.ENMI	8.2.18
30	OPEN FILE	H.IOCS,1	M.FILE	7.8.9
31	READ RECORD	H.IOCS,3	M.READ	7.8.18
32	WRITE RECORD	H.IOCS,4	M.WRIT	7.8.31
33	ADVANCE RECORD	H.IOCS,7	M.FWRD	7.8.12
34	ADVANCE FILE	H.IOCS,8	M.FWRD	7.8.12
35	BACKSPACE RECORD	H.IOCS,9	M.BACK	7.8.2
36	BACKSPACE FILE	H.IOCS,19	M.BACK	7.8.2
37	REWIND FILE	H.IOCS,2	M.RWND	7.8.25
38	WRITE EOF	H.IOCS,5	M.WEOF	7.8.30
39	CLOSE FILE	H.IOCS,23	M.CLSE	7.8.3
3A	RESERVE CHANNEL (PRIV)	H.IOCS,12	M.RSRV	7.8.24
3B	RELEASE CHANNEL (PRIV)	H.IOCS,13	M.RRES	7.8.21
3C	WAIT I/O	H.IOCS,25	M.WAIT	7.8.29
3D	SYSTEM CONSOLE WAIT	H.IOCS,26	M.CWAT	7.8.5
3E	ERASE OR PUNCH TRAILER	H.IOCS,21	N/A	7.8.33
3F	CONSOLE TYPE	H.IOCS,14	M.TYPE	7.8.26
40	ALLOCATE FILE OR PERIPHERAL DEVICE	H.MONS,21	M.ALOC	7.8.1
41	DEALLOCATE FILE OR DEVICE	H.MONS,22	M.DALC	7.8.6
42	PHYSICAL DEVICE INQUIRY	H.MONS,1	M.PDEV	7.8.16

USER LEVEL SYSTEM SERVICES - SVC ORDER

(CONTINUED)

SVC 1	DESCRIPTION	MODULE,EP	MACRO	REF MANUAL SECTION
43	PERMANENT FILE ADDRESS INQUIRY	H.MONS,2	M.FADD	7.8.8
44	MEMORY ADDRESS INQUIRY	H.MONS,3	M.ADRS	8.3.1
45	CREATE TIMER ENTRY	H.MONS,4	M.SETT	8.2.33
46	TEST TIMER ENTRY	H.MONS,5	M.TSTT	8.2.43
47	DELETE TIMER ENTRY	H.MONS,6	M.DLTT	8.2.14
48	SET USER STATUS WORD	H.MONS,7	M.SETS	8.2.32
49	TEST USER STATUS WORD	H.MONS,8	M.TSTS	8.2.42
4A	CHANGE PRIORITY LEVEL (PRIV)	H.MONS,9	M.PRIL	8.2.29
4B	CONNECT TASK TO INTERRUPT	H.MONS,10	M.CONN	8.2.8
4C	TASK OPTION WORD INQUIRY	H.MONS,24	M.PGOW	8.2.28
4D	ARITHMETIC EXCEPTION INQUIRY	H.MONS,23	M.TSTE	8.2.41
4E	TIME-OF-DAY INQUIRY	H.MONS,11	M.TDAY	8.2.40
4F	MEMORY DUMP REQUEST	H.MONS,12	M.DUMP	8.3.2
50	LOAD OVERLAY	H.MONS,13	M.OLAY	8.2.27
51	LOAD AND EXECUTE OVERLAY	H.MONS,14	M.OLAY	8.2.27
52	ACTIVATE TASK	H.MONS,15	M.ACTV	8.2.1
53	RESUME TASK EXECUTION	H.MONS,16	M.SUME	8.2.37

USER LEVEL SYSTEM SERVICES - SVC ORDER

(CONTINUED)

SVC 1	DESCRIPTION	MODULE,EP	MACRO	REF MANUAL SECTION
54	SUSPEND TASK EXECUTION	H.MONS,17	M.SUSP	8.2.38
55	TERMINATE TASK EXECUTION	H.MONS,18	M.EXIT	8.2.20
56	ABORT SPECIFIED TASK	H.MONS,19	M.BORT	8.2.4
57	ABORT SELF	H.MONS,20	M.BORT	8.2.4
58	PROGRAM HOLD REQUEST	H.MONS,25	M.HOLD	8.2.23
59	SET TABS IN UDT	H.TSM,5	N/A	N/A
5A	DELETE TASK	H.MONS,31	M.DELTSK	8.2.11
5B	SCAN TERMINAL INPUT BUFFER	H.TSM,2	M.TSCAN	5.6.1
5C	TRAP ONLINE USER'S TASK	H.TSM,6	M.TBRKON	5.6.2
5D	DISCONNECT TASK FROM INTERRUPT	H.MONS,38	M.DISCON	8.2.13
5E	EXIT FROM MESSAGE RECEIVER	H.MONS,39	M.XMSGR	8.2.47
5F	PARAMETER TASK ACTIVATION (PRIV)	H.MONS,40	M.PTSK	8.2.30
60	SET USER ABORT RECEIVER ADDRESS	H.MONS,26	M.SUAR	8.2.36
61	SUBMIT JOB FROM DISC FILE	H.MONS,27	M.CDJS	8.2.7
62	ABORT WITH EXTENDED MESSAGE	H.MONS,28	M.BORT	8.2.4

USER LEVEL SYSTEM SERVICES - SVC ORDER

(CONTINUED)

SVC 1	DESCRIPTION	MODULE,EP	MACRO	REF MANUAL SECTION
63	LOAD AND EXECUTE INTERACTIVE DEBUGGER	H.MONS,29	M.DEBUG	8.2.10
64	GET TASK NUMBER	H.MONS,32	M.ID M.YID	8.2.24 8.2.26
65	GET ADDRESS LIMITS	H.MONS,41	M.GADRL	8.3.7
66	DEBUG LINK SERVICE	H.MONS,42	N/A	8.2.51
67	GET DYNAMIC TASK EXECUTION SPACE	H.ALOC,10	M.GE	8.3.9
68	FREE DYNAMIC TASK EXECUTION SPACE	H.ALOC,11	M.FE	8.3.6
69	GET DYNAMIC EXTENDED INDEXED DATA SPACE	H.ALOC,8	M.GD	8.3.8
6A	FREE DYNAMIC EXTENDED INDEX DATA SPACE	H.ALOC,9	M.FD	8.3.5
6B	RECEIVE MESSAGE LINK ADDRESS	H.MONS,43	M.RCVR	8.2.31
6C	SEND MESSAGE TO SPECIFIED TASK	H.MONS,44	M.SMSGR	8.2.34
6D	SEND RUN REQUEST	H.MONS,45	M.SRUNR	8.2.35
6E	BREAK/TASK INTERRUPT LINK	H.MONS,46	M.BRK	8.2.5
6F	ACTIVATE TASK INTERRUPT	H.MONS,47	M.INT	8.2.25
70	EXIT FROM TASK INTERRUPT LEVEL	H.MONS,48	M.BRKXIT M.XBRKR	8.2.6 8.2.45
71	SHARE MEMORY WITH ANOTHER TASK	H.ALOC,12	M.SHARE	8.3.11
72	GET SHARED MEMORY	H.ALOC,13	M.INCL	8.3.10
73	PERMANENT FILE LOG	H.MONS,33	M.LOG	7.8.15

USER LEVEL SYSTEM SERVICES - SVC ORDER

(CONTINUED)

SVC 1	DESCRIPTION	MODULE,EP	MACRO	REF MANUAL SECTION
74	USERNAME SPECIFICATION	H.MONS,34	M.USER	7.8.28
75	CREATE PERM FILE	H.FISE,12	M.CREATE	7.8.4
76	CHANGE TEMP FILE TO PERMANENT	H.FISE,13	M.PERM	7.8.17
77	DELETE PERM FILE	H.FISE,14	M.DELETE	7.8.7
78	RESERVED FOR FUTURE USE			
79	FREE SHARED MEMORY	H.ALOC,14	M.EXCL	8.3.3
7A	GET MSG PARAMETERS	H.MONS,35	M.GMSGP	8.2.21
7B	GET RUN PARAMETERS	H.MONS,36	M.GRUNP	8.2.22
7C	WAIT FOR ANY MSG, END ACTION, OR BRK	H.MONS,37	M.ANYW	8.2.2
7D	EXIT RUN RECEIVER	H.MONS,49	M.XRUNR	8.2.49
7E	EXIT FROM MESSAGE END ACTION ROUTINE	H.MONS,50	M.XMEA	8.2.46
7F	EXIT FROM RUN REQUEST END ACTION ROUTINE	H.MONS,51	M.XREA	8.2.28

APPENDIX C

MPX-32 ABORT AND CRASH CODES

Address Specification Trap Handler (H.IP0C)

CODE	DESCRIPTION
AD01	Address specification occurred within the operating system.
AD02	Address specification occurred within the current task.
AD03	Trap occurred while no tasks were in active state.
AD04	Trap occurred within another interrupt trap routine.

CODE	DESCRIPTION
AL01-AL06	Reserved
AL07	The combined number of file assignments for a task exceeds number specified. The cataloged assignments are combined with those defined by $ASSIGN statements. See cataloger FILES directive and recatalog if needed.
AL08	An assigned permanent file is nonexistent.
AL09	An assigned device is not configured in the system. An assigned device is off-line.
AL10	Reserved
AL11	Reserved
AL12	Unable to load program because of I/O error or addressing inconsistencies in load module preamble.
AL13	An unrecoverable I/O error has occurred during the read of the task preamble into the TSA.
AL14	Reserved
AL15	An assigned device type is not configured in the system.
AL16	A resident request has been issued for a task requiring an SLO, SBO, SGO or SYC file. Resident tasks cannot use system files.
AL17	Reserved
AL18	Reserved
AL19	A file code to file code assignment (ASSIGN4) has been made to an undefined file code. A file code must be defined before a second file code can be equated by an ASSIGN4.
AL20	User attempted deallocation of TSA.
AL21	Destroyed task MIDL was detected while attempting to allocate dynamic execution space.
AL22	A software checksum error has occurred during task loading.
AL23	An invalid user name is cataloged with the task. The user name is either not contained in the user name file M.KEY or a correct user key is not present. Also: task has attempted to deallocate TSA.

AL24	Access to an assigned permanent file is by password only, and a valid password was not included on the cataloged assignment or Job Control statement assignment.
AL25	Undefined Resource Requirement Summary (RRS) type (internal format of an assignment statement is wrong).
AL26	The task has requested more blocking buffers than were specified during catalog. See Cataloger BUFFER directive and recatalog if needed.
AL27	There are no free entries in shared memory table for GLOBAL, DATAPOOL, CSECT, or other shared areas.
AL28	Task is attempting to share an undefined GLOBAL or DATAPOOL memory partition.
AL29	Task is attempting to exclude undefined memory partition.
AL30	The requested device is already assigned to the requesting task via another file code. Use ASSIGN4 or deallocate before reallocating.
AL31	Logical file code has already been allocated by caller (e.g., a card reader may already be assigned to lfc IN and a magnetic tape cannot be assigned to the same file code). Use ASSIGN4 or deallocate before reallocating.
AL32	Dynamic common block may not be assigned via ASSIGN1 directive.
AL33	Shared memory definition conflicts with caller's address space.
AL34	Shared memory partition not defined in SMD.
AL35	Attempt to share an SMD entry that is not a memory partition.
AL36	Invalid password specified for shared memory partition.
AL37	Attempt to exclude undefined shared memory partition.
AL38	Attempt to activate a privileged task by unauthorized owner.
AL39	Shared memory entry not found.
AL40	Partition definition not found on SMD.
AL41	SMD definition not a dynamic definition.
AL42	Invalid password for this partition.
AL43	Task has attempted to allocate an unshared resource that was not available during task activation in a memory-only environment.

AL44	Unable to resume 'SYSBUILD' task during initial task activation in a memory-only environment.
AL45	Unable to deallocate input device after dynamic task activation in a memory-only environment.
AL46	Task has attempted to share memory via a dynamic memory partition in a memory-only environment.
AL47	Dynamic memory partitions cannot be greater than 1 megabyte.
AL48	The user has attempted to exclude a shared partition whose associated map blocks are not designated as being shared in the task's TSA.
AL49	The task's DSECT space requirements overlap the task's TSA space requirements.
AL50	The task's DSECT space requirements overlap the task's CSECT space requirements, or if no CSECT, load module is too large to fit in user's address space.
AL51-AL54	Reserved
AL55	The sum of the CSECT, DSECT, and the operating system sizes is greater than the total amount of memory configured.
AL56	Unrecoverable I/O error to the SMD.
AL57	File Lock Table (FLT) is full.

CODE	DESCRIPTION
AS01	Physical end-of-file encountered on write to the General Object (GO) file.
AS02	Physical end-of-file encountered on write to the Binary Output (BO) file.
AS03	Physical end-of-file encountered on write to the Listed Output (LO) file.
AS04	Physical end-of-file encountered on write to the scratch (UT1) file (i.e., $ASSIGN3 UT1 = DC, ????).
AS05	Physical end-of-file encountered on write to the cross-reference (UT2) file (i.e., $ASSIGN3 UT2 = DC,????).
AS06	There does not exist a prime number of three-word entries in the allocated core for the symbol table.
AS07	Unrecoverable I/O error on the Binary Output (BO) file.
AS08	Unrecoverable I/O error on the General Object (GO) file.
AS09	Unrecoverable I/O error on the Listed Output (LO) file.
AS10	Unrecoverable I/O error on the Source Input (SI) file.
AS11	Unrecoverable I/O error on the intermediate compressed source (UT1) file.
AS12	Physical end-of-file encountered on write to the Compressed Source Output (CS) file.
AS13	Checksum error on compressed source input either during pass 1 while reading compressed source from the Source Input (SI) file or during pass 2 while reading the intermediate scratch compressed source (UT1) file.
AS14	The file the Assembler is using as the macro library was not successfully created by the macro library generator. The file is invalid.
AS15	Unrecoverable I/O error on the Macro Library (MAC) file.
AS16	Unrecoverable I/O error on the cross-reference (UT2) file.
AS17	Unrecoverable I/O error on the Compressed Source Output (CS) file.

AS18	Invalid blocking buffer control pointer encountered on the Binary Output (BO) file.
AS19	Invalid blocking buffer control pointer encountered on the General Object (GO) file.
AS20	Invalid blocking buffer control pointer encountered on the Listed Output (LO) file.
AS21	Invalid blocking buffer control pointer encountered on the Source Input (SI) file.
AS22	Invalid blocking buffer control pointer encountered on the Scratch Compressed Source (UT1) file.
AS23	Invalid blocking buffer control pointer encountered on the Compressed Source Output (CS) file.
AS24	Invalid blocking buffer control pointer encountered on the cross-reference (UT2) file.
AS25	The macro library (MAC) file is unblocked.
AS26	End of file on MA2 file.
AS27	Unrecoverable I/O error on MA2 file.
AS28	Invalid blocking buffer control pointer on MA2 file.
AS29	MAC assigned to illegal device.
AS30	MA2 assigned to illegal device.
AS31	Potential abort conditions have been detected during program assembly. An abort status flag within the job's TSA has been set during assembler termination processing. Conditional job control directives may be used to test status prior to job continuation.
AS32	Unrecoverable I/O error on the prefix (MPXPRE) file.
AS33	Invalid blocking buffer control pointer encountered on the prefix (MPXPRE) file.

Auto-Start Trap Processor (H.IPAS)

CODE	DESCRIPTION
AU01	Trap occurred on auto-start.
AU02	Trap occurred in another interrupt trap routine.
AU03	Reserved
AU04	Reserved
AU05	User was unmapped when trap occurred.

Debugger

CODE	DESCRIPTION
DB01	End of medium error on lfc #OT in batch mode.
DB02	Fatal I/O error on lfc contained in the abort message.

Call Monitor Interrupt Processor (H.IP27 and H.IP0A)

CODE	DESCRIPTION
CM01	Physical end-of-file encountered on write to the compressed source output (CS) file. Call monitor interrupt processor cannot locate the CALM instruction.
CM02	Expected CALM instruction does not have CALM (X'30') opcode.
CMO3	Invalid CALM number.
CM04	CALM number too low (out of bounds).
CM05	CALM number too big (out of bounds).

CODE	DESCRIPTION
CT01	Physical end-of-file encountered on subroutine library. The lfc of the library in question is displayed. This results from the library being updated by another user while it is allocated by the Cataloger.
CT02	Load module file specified with CATALOG cannot be allocated.
CT03	Unrecoverable I/O error encountered on the DATAPOOL dictionary file assigned to DPD.
CT04	Listed output space is deleted and additional SLO space cannot be allocated.
CT05	Unrecoverable I/O error on file or device assigned to SBM for SYMTAB output.
CT06	An error occurred during the cataloging process and the reason is described in the SLO.

CODE	DESCRIPTION
DP01	Unrecoverable I/O error while reading or writing the DATAPOOL dictionary.
DP02	Dictionary file code (DPD) unassigned.
DP03	Unrecoverable error on error audit trail (ER) file.
DP04	Unrecoverable error on audit trail (LO) file.
DP05	Unable to allocate additional SLO space for audit trail after initial file is filled.
DP06	Unable to allocate additional SLO space for error audit trail after initial file is filled.
DP07	Invalid directive.
DP08	The name 'DATAPOOL' is not defined as a core partition.
DP09	Dictionary overflow.
DP10	Unable to reassign the DPD file.
DP11	End-of-tape or illegal end-of-file encountered on IN.
DP12	Physical end-of-media encountered on OT.
DP13	Unrecoverable error on IN.
DP14	Unrecoverable error on OT.
DP15	File code OT unassigned and the SAVE function requested.
DP16	File code IN unassigned and the REMAP function requested.
DP17	Sequence error on dictionary entry record (accessed through file code IN).
DP18	Checksum error on dictionary entry record (accessed through file code OT).
DP19	Invalid specification on REMAP directive.
DP20	Invalid specification on DPD directive.
DP21	Unrecoverable error on directive input (SYC) file.
DP22	Dictionary size is less than the required minimum (five records).
DP99	A non-fatal error occurred.

Error Condition Codes for DPEDIT

CODE	DESCRIPTION
EC11	Attempt to delete a symbol not found in the dictionary.
EC12	Attempt to delete a symbol that is used as a base for another variable.
EC13	A change is requested for a symbol used as a base that may result in a change in the relative address.
EC14	The calculated relative address does not fall on the specified boundary (precision).
EC15	The referenced base symbol is not in the datapool dictionary.
EC16	Attempt to add a symbol that is already defined in the dictionary.
EC17	The calculated relative address is not within the range of the datapool core partition.
EC18	The datapool variable does not reside in the dictionary at the location computed by the hash coding scheme.
EC19	Invalid specification on directive.
EC20	Log function deleted, not enough memory to sort data.
EC21	Log function deleted, the scratch sort file is not enough to contain the necessary data.
EC22	Log function deleted, unrecoverable I/O error on the scratch sort file.
EC23	Attempted to change a symbol not found in the dictionary.
EC24	Computed relative address does not agree with actual.
EC25	Entries are multiply defined.
ERnn	Error encountered in processing data card fields. The column number in which the error was detected is specified by "nn".

CODE	DESCRIPTION
ED01	User terminal I/O hardware error.
ED02	Internal line linkage invalid.
ED03	Reserved (RTM Only).
ED04	Internal logic error.

CODE	DESCRIPTION
FM01	Invalid command verb on directive.
FM02	Required argument(s) absent from a directive.
FM03	Create requested for an existing file.
FM04	Device specified is invalid for this command.
FM05	Decimal number specification contains non-decimal digits.
FM06	Hexadecimal number specification contains non-hexadecimal characters.
FM07	Specified file is password protected and correct password not specified.
FM08	Attempt to expand a core partition.
FM09	Cannot create or expand file due to unavailability of disc space for allocation.
FM10	Attempt to create a fast file which mapped into an existing fast file in the SMD.
FM11	DELETE, SAVE, or EXPAND requested for a disc file which does not exist.
FM12	Insufficient file assignment table space for I/O to the SMD.
FM13	Unrecoverable I/O error to the SMD.
FM14	Unrecoverable I/O error to the SYC file.
FM15	Unrecoverable I/O error to the SLO file.
FM16	Unrecoverable I/O error to the IN file.
FM17	Unrecoverable I/O error to the OUT file.
FM18	Invalid argument.
FM19	Cannot allocate required file.
FM20	Unrecoverable I/O error on SAVE/RESTORE file.
FM21	Unexpected EOF on IN file.
FM24	File specified for RESTORE not found.
FM25	Too many prototypes specified.

FM26	EOF expected on IN file not found.
FM41	End of medium on lfc SLO.
FM42	Invalid username or key.
FM99	Directive errors have been detected during execution of the File Manager. An abort status flag within the job's TSA has been set during File Manager termination processing. Conditional job control directives may be used to test status prior to job continuation.

File System

CODE	DESCRIPTION
FS01	Unrecoverable I/O error to the System Master Directory (SMD).
FS02	Unrecoverable I/O error to a disc allocation map.
FS03	Attempt to add a new file, but the System Master Directory (SMD) is full.
FS04	A disc allocation map checksum error was detected.
FS05	Attempt to allocate disc space that is already allocated.
FS06	Attempt to deallocate disc space that is not allocated.
FS07	Reserved.
FS08	Unrecoverable I/O error occurred while zeroing a file during creation.

Fortran

CODE	DESCRIPTION
FT01	Fortran scratch file *U1 must be expanded (i.e., $ASSIGN3 *U1=DC,????).
FT02	Fortran scratch file *U2 must be expanded (i.e., $ASSIGN3 *U2=DC,????).
FT03	Binary output (BO) file must be expanded if a disc file (i.e., $ASSIGN2 BO=SLO,???? or a direct assignment to the card punch - $ASSIGN3 BO=CP)..
FT04	The compiled program caused the SGO file to overflow. The size of the SGO file can be increased via SYSGEN or may be assigned to tape via Operator Communications.
FT05	End of medium on nonspooled SLO file. File must be expanded if a disc file or a direct assignment made to the line printer - $ASSIGN3 LO=LP.
FT06	Potential abort conditions have been detected during program compilation. An abort status flag within the job's TSA has been set during compiler termination processing. Conditional job control directives may be used to test status prior to job continuation.

CODE	DESCRIPTION
HT01	Trap occurred in user's map area.
HT02	Trap occurred in another interrupt trap routine.
HT03	Trap occurred while no other tasks were in the active state.
HT04	Reserved.
HT05	User was unmapped when trap occurred.

CODE	DESCRIPTION
IO01	An I/O operation has been attempted for which the FCB is not properly linked to a File Assignment Table (FAT) entry. Since this linkage is established by IOCS when the file is opened, either the user task has not properly opened the file or the FCB has been inadvertently destroyed subsequent to the time the open file operation was performed.
IO02	An I/O operation has been attempted on an unopened file. This abort code will normally be issued when a user has opened a file, subsequently closed the file, then attempted an I/O operation on the file.
IO03	An unprivileged task is attempting to read data into an area of core which is not allocated for its use. This type of abort is usually caused by an invalid TCW in the task's FCB.
IO04	The control specifications in the FCB specify random access. However, the random access address contained in the FCB does not fall within the limits of the file.
IO06	Invalid blocking buffer control cells have been encountered during a read operation performed on a blocked file. This type error is normally caused in one of the three ways: (1) The user's blocking buffer has been inadvertently destroyed. (2) The file being read is not a properly blocked file. (3) A data transfer error has occurred on input of data from the file.
IO07	The task has attempted to perform an operation which is not valid for the device to which the user's file is assigned (e.g., a read operation specified for a file assigned to the line printer).
IO08	Reserved
IO09	The task has attempted to perform a rewind operation on the system SYC file.
IO10	The task has attempted a write End-of-File operation on a file which has been opened in the Read-Only access mode.
IO11	The task has attempted a write End-of-File operation on the system SYC file.

IO12	The task has attempted an erase or punch trailer operation on a file which has been opened in the Read-Only access mode.
IO13	The task has requested an illegal operation to be performed on a system file (backspace file, upspace, erase or punch trailer, eject, advance record, advance file, or backspace record).
IO14	A task running in the unprivileged mode has attempted to reserve an I/O channel.
IO15	A task has requested a type operation and the Type Control Parameter Block (TCPB) specified indicates that an operation associated with that TCPB is already in progress.
IO17	The task has attempted an open operation on a file, and no File Pointer Table (FPT) entry exists with a matching file code. This type of abort is most often caused by an improper or missing file assignment directive at catalog or linking load time. This type abort may also occur if the logical file code portion of the task's FCB has been inadvertently destroyed.
IO18	Reserved
IO21	IOCS has encountered an unrecoverable I/O error in attempting to process an I/O request on behalf of a task.
IO22	An illegal IOCS entry point has been entered by a task.
IO24	A task has specified an illegal address or transfer count in the FCB TCW. This type of error is usually the result of trying to output to a halfword device from a data area which is not on a halfword boundary. This error may also occur if the task attempts to transfer other than an even multiple of halfwords to or from a halfword device.
IO25	The task has requested a data transfer operation (read or write) with a Transfer Control Word (TCW) which specifies a quantity of zero.
IO26	Illegal sequence of operations while in read mode on either a system file or a blocked file.
IO27	Illegal sequence of operations while in write mode on either a system file or a blocked file.
IO28	Attempt to advance a record while in the write mode on a blocked file.
IO29	Attempt to advance a file while in the write mode on a blocked file.
IO30	Illegal or unexpectecd volume number encountered on magnetic tape.

IO32	Calling task has attempted to perform a second read on a $ statement through the SYC file.
IO33	An Invalid Device Address has been specified in the Task's Input/Output Control Header (IOCH).
IO34	An unprivileged task has requested the link service.
IO35	An unprivileged task has specified an IOCB list greater than 30 IOCB's in length.
IO36	A SYSGEN error has occurred, and the handler HAT address is not in the Controller Definition Table (CDT).
IO37	Job sequence number not found in the job table for task attempting to open SYC or SGO file.
IO38	The task has requested a write operation to be performed on a file which has been opened in the read-only access mode. Permanent files to which a task has read but not write access are opened read-only even though read-write is specified when the file is opened.
IO39	Blocked file indicated in FCB (or implied via assignment to a system file) but no blocking buffers available.
IO40	User TCW is in error due to one or more of the following conditions: 1. Unable to construct a valid TCW because the transfer count is too large. 2. Transfer count not an even multiple of transfer type. 3. Data address not bounded for transfer type (types = W, HW, B).
IO43	Input/Output Control List (IOCL) or data address not in contiguous 'E' memory (ASYNC,BSYNC).
IO46	Dynamic storage space for IOCDs within IOQ exhausted.
IO47	Class 'E' device TCW is not in class 'E' memory. This type of error indicates a map failure.
IO48	Reserved
IO49	Device access failure on OPEN.
IO53	The user has attempted to write to SYC file in Batch mode.
IO54	An attempt has been made to use the same logical file code in two or more File Control Blocks.

CODE	DESCRIPTION
JC01*	Unrecoverable read error from job's SYC file.
JC02*	Unrecoverable write error on SLO file.
JC03*	Unrecoverable write error on job's SGO file.
JC04*	Unable to build FAT/FPT for SLO or for SBO link file which indicates a program error.
JC05*	Unable to allocate disc space for SLO file.
JC06*	An entry is not available in the System Output Directory (M.SOD) for the definition of the job's SLO or SBO link file.
JC07	This job's Job Table entry has been destroyed which indicates a program error.
JC08*	Unable to allocate the job's SYC file.
JC09	Unrecoverable I/O error to SMD returned on call to File System Executive (H.FISE,10).
JC10*	Unrecoverable I/O error to disc allocation map returned by File System Executive (H.FISE,3 or H.FISE,4).
JC11*	Unable to allocate job's SGO file.

* Whenever a Job Control task aborts with one of these codes, the associated job is deleted.

CODE	DESCRIPTION
LD01	Load code section error.
LD02	Code section checksum error.
LD03	Bias code error.
LD04	Code matrix checksum error.
LD05	Load data section error.
LD06	Data section checksum error.
LD07	Bias data error.
LD08	Data matrix checksum error.

CODE	DESCRIPTION
LE01	Directive error. The last directive printed on the directive list is in error.
LE02	Object record sequence error. The name of the module with the error will be the last line printed on the log.
LE03	Object record checksum error. The name of the module with the error will be printed as the last line of output on the log.
LE04	Object module format error. The module whose name is the module following that one has an invalid record code in record 1.
LE05	Incomplete object module. An object module, whose name is the last printed line of the log, has no terminal record (Hex DF).
LE06	Unrecoverable error on the SYC file.
LE07	Unrecoverable error on the LGO file.
LE08	Unrecoverable error on the LLO file.
LE09	Unrecoverable error on the LIB file.
LE10	Unrecoverable error on the DIR file.
LE11	Unrecoverable error on either the dynamically assigned temporary library file or temporary directory file.
LE12	Allocation denial on request for temporary disc space used to perform update function (i.e., disc space unavailable).
LE13	Delete table overflow. A combined maximum of 255 modules may be deleted/replaced/added in any one LIBED run.
LE14	End-of-medium encountered on temporary library. This error indicates the need to expand the subroutine library assigned to the LIB file code. The FILEMGR may be used to perform this function prior to another LIBED run.
LE15	End-of-medium encountered on temporary directory. This error indicates the need to expand the subroutine library directory assigned to the DIR file code. The FILEMGR may be used to perform this function prior to another LIBED run.
LE16	End-of-medium encountered on LIB file. The assigned file must be reallocated.
LE17	End-of-medium encountered on DIR file. The assigned file must be reallocated.

LE18	End-of-medium encountered on either LIB or DIR file. This error may occur on either a log, statistics, or update run and indicates that a previous CREATE run terminated prior to completion with an uncorrectable I/O error or a LE16/LE17 error.

MEDIA

CODE	DESCRIPTION
MD01	Potential abort conditions have been detected during media conversion operation. An abort status flag within the job's TSA has been set during compilation or execution processing. Conditional job control directives may be used to test status prior to job continuation. See output on logical file Code *OT for details about the abort condition.
MD02	At EOF on a SLO file.

MACLIBR

CODE	DESCRIPTION
ME99	Potential abort conditions have been detected during library editing operation. An abort status flag within the job's TSA has been set during editing processing. Conditional job control directives may be used to test status prior to job continuation.

Memory Parity Trap (H.IP02)

CODE	DESCRIPTION
MP01	Memory error occurred in a task's logical address space.
MP02	Memory error occurred in another interrupt trap routine (nested traps, context lost).
MP03	Memory error occurred while no tasks were in the active state.
MP04	Memory error occurred in a map block reserved for the O/S.
MP05	Error occurred while current task was in the unmapped mode.

CODE	DESCRIPTION
MS01	Permanent file address inquiry service found a number of allocation units in the Unit Definition Table that do not correspond to any known disc.
MS02	Invalid function code specified for request to create a timer entry. Valid codes are ACP (1), RSP or RST (2), STB (3), RSB (4), and RQI (5).
MS03	A privileged task bit Set/Reset address is outside of the operating system or a static memory partition, or an unprivileged task bit Set/Reset address is outside of a static memory partition.
MS04	Task has attempted to create a timer entry to request an interrupt with a priority level outside the range of X'12' to X'7F', inclusive, or the requesting task is unprivileged.
MS05	Invalid function code has been specified for request to set user status word.
MS06	Unprivileged task has attempted to reset a task priority level or a privileged task has attempted to reset a task priority to a level outside the range of 1 to 64, inclusively.
MS07	Cannot load overlay segment due to software checksum or data error.
MS08	Overlay is not in the SMD.
MS09	Task has attempted to connect a task to an interrupt level not defined for indirectly connected tasks.
MS10	Overlay has an invalid preamble.
MS11	An unrecoverable I/O error has occurred during overlay loading.
MS12	Overlay is password protected.
MS16	Task has requested dynamic allocation with an invalid function code.
MS17	File name contains characters outside range of X'20' to X'5F', inclusively.
MS21	Multi-volume magnetic tape allocation request made to scratch (SCRA) tape.
MS22	Multi-volume magnetic tape allocation request made on shared tape drive.

MS23	Task has issued a MOUNT MESSAGE ONLY allocation request to a non-allocated drive or to a device which is not a magnetic tape.
MS24	Task has specified an illegal volume number (0 if tape is multivolume; non-zero if tape is single volume).
MS25	Operator has aborted task in response to mount message.
MS28	A permanent file log has been requested, but the address specified for storage of the directory entry is not contained within the calling task's logical address space.
MS29	Task has attempted to load the interactive Task Debugger overlay in a memory-only environment.
MS30	Task has attempted to obtain a permanent file log in a memory-only environment.
MS31	User attempted to go to an any wait state from an end action routine.
MS32	Invalid register set-up detected in M.ID.
MS89	An unprivileged task has attempted to reestablish an abort receiver (other than M.IOEX).
MS90	Task has made a run request end action routine exit while the run request interrupt was not active.
MS91	Task has attempted normal exit with a task interrupt still active.
MS92	Task has attempted to queue a message during its exit sequence.
MS93	An invalid Receiver Exit Block (RXB) address was encountered during message exit.
MS94	An invalid Receiver Exit Block (RXB) return buffer address was encountered during message exit.
MS95	Task has made a message exit while the message interrupt was not active.
MS96	An invalid Receiver Exit Block (RXB) address was encountered during run receiver exit.
MS97	An invalid Receiver Exit Block (RXB) return buffer address was encountered during run receiver exit.
MS98	Task has made a run receiver exit while the run receiver interrupt was not active.
MS99	Task has made a message end action routine exit while the message interrupt was not active.

CODE	DESCRIPTION
RS47	Invalid time interval request.
RS48	Invalid activation request.
RS49	Invalid run request.
RS53	Invalid task number.
RS60	Invalid address specified.
RS65	Invalid delete request.
RS66	Invalid abort request.
RS67	Invalid resource mark request.
RS68	Invalid disconnect request.
RS69	Skip file or record operation requested on non-existent FCB.
RS70	Allocation error (appears only if IOSTAT and $n parameters have been omitted).
RT01	Unformatted read I/O error.
RT02	Formatted read I/O error.
RT03	Unformatted write I/O error.
RT04	Formatted write I/O error.
RT05	Reference made to non-existent device type or address.
RT06	Unit out of 0-999 range.
RT07	No left parenthesis on format.
RT08	Transfer index out of range (option 7 or M:ERRFLG can be used to avoid an abort).
RT09	Format error.
RT10	The I/O transfer requirements for the data buffer are incompatible with the amount of available data.
RT11	Format parenthesis level in excess of two.
RT13	Argument list exceeds logical read record.

RT14	Incorrect descriptor in format.
RT15	Integer descriptor but non-integer argument (option 7 or M:ERRFLG can be used to avoid an abort).
RT16	Hexadecimal descriptor but non-hexadecimal argument (option 7 or M:ERRFLG can be used to avoid an abort).
RT17	D, E, F, G descriptor, not real or complex argument (option 7 or M:ERRFLG can be used to avoid an abort).
RT18	Logical descriptor but non-logical argument (option 7 or M:ERRFLG can be used to avoid an abort).
RT19	Attempt to read past EOF/EOM.
RT20	Attempt to write past EOF/EOM.
RT21	Attempt to read past EOF/EOM.
RT22	Attempt to write past EOF/EOM.
RT23	Attempt to backspace following EOF/EOM.
RT24	Rewind after EOF/EOM.
RT25	Formatted record read.
RT26	Unformatted record read.
RT27	Doubleword integer overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT28	Byte integer input with negative sign (option 7 or M:ERRFLG can be used to avoid an abort).
RT29	Byte integer overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT30	Halfword integer overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT31	Full word integer overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT32	Illegal character in D, E, F, G input (option 7 or M:ERRFLG can be used to avoid an abort).
RT33	Underflow in floating conversion (option 7 or M:ERRFLG can be used to avoid an abort).
RT34	Overflow in floating conversion (option 7 or M:ERRFLG can be used to avoid an abort).

RT35	Argument list overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT36	Argument list overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT40	Attempt to free busy IOCH/IOCB entry.
RT41	Attempt to link busy IOCH/IOCB entry.
RT42	IOCH/IOCB table overflow.
RT43	Wait I/O returned before I/O termination.
RT44	Status parameter not linked to ADI device prior to I/O request.
RT46	ADI table address not on halfword boundary.
RT50	Missing or omitted parameter.
RT51	Parameter out of range.
RT52	End of search list reached.
RT55	Error found in math library routine.
RT61	List-directed I/O (input) encountered, character string split between two records.
RT62	Internal file read/write past EOF/EOM with no END option specified.
RT63	Block number exceeds maximum block number in file.
RT64	Record overflow.
RT65	Record length exceeds maximum allowable.
RT66	Record length not specified for random access or specified for sequential file.
RT67	Implicit open not allowed for or random access I/O.
RT68	Reference to sequential operation on a file opened for direct access not allowed.
RT69	Error(s) encountered on open.
RT80	Subscript error (i.e., subscript not a decimal number, illegal punctuation, excessive subscripts, or subscript out of range).
RT81	NAMELIST identifier error (i.e., column 1 non-blank, ampersand character not present, name does not immediately follow ampersand character, or non-blank following name).

RT82	Symbolic name error (no equal sign after variable/array name).
RT83	Data item error (i.e., excessive values for symbol or expected to find symbol).
RT84	Illegal value (i.e., illegal punctuation, missing comma, zero Hollerith count, or illegal character in value).
RT85	Attempt to read past EOF/EOM.
RT86	Attempt to write past EOF/EOM.
RT87	Symbolic name not defined in NAMELIST statement.
RT88	Repeat count error.
RT89	Symbolic name exceeds eight characters.
RT90	Invalid read/write operation.
RT91	End-of-file status return pursuant to random access record.
RT92	Random access partition number out-of-range (i.e., partition number not between 1 and 95, inclusive).
RT93	Random access number out-of-range (i.e., record number not between 1 and 65,535, inclusive).
RT94	Random access transfer length (write/read) or record size definition (define) out-of-range (i.e., transfer record length not between 1 and 65,535 bytes, inclusive).
RT95	Invalid random access argument list length.
RT96	FCB table overflow (16 or more files for RTM; 31 or more files for MPX-32).
RT97	Diagnostic output message exceeds 100 lines. To allow more diagnostic messages, statically assign the DO file (i.e., $ASSIGN2 DO=SLO,500).
RT98	Denial return when attempting to allocate file for diagnostic output message.
RT99	Insufficient blocking buffer space (each unit assignment to a system file requires one blocking buffer unless one file is assigned to another, i.e., via $ASSIGN4).

System Binary Output

CODE	DESCRIPTION
SB01	An I/O error has been encountered on the device assigned as the system binary (punched) output device.
SB02	The system output program has encountered an unrecoverable I/O error in attempting to read a punched output file from disc.
SB03	Denial of file code to file code allocation for J.SOUT2 indicates loss of system integrity.
SB04	System binary output aborted by operator.
SB05	No timer entry for system binary output (system fault).
SB06	Five echo check errors detected while attempting to punch a single card.

System Check Trap Processor

CODE	DEFINITION
SC01	System check trap occurred at an address located within the operating system.
SC02	System check trap occurred within the current task's space.
SC03	System check trap occurred at a time when there were no tasks currently being executed (C.PRNO equals zero).
SC04	System check trap occurred within another trap (C.GINT does not equal 1).

CODE	DESCRIPTION
SG01	Invalid loader function code in binary object module from the System Resident Module (OBJ) file.
SG02	Invalid binary record read from System Resident Module (OBJ) file (byte 0 must be X'FF' or X'DF').
SG03	Sequence error in module being read from temporary file.
SG04	CHECKSUM error in module being read from temporary file.
SG05	Unable to find CDT and/or UDT for I/O module load.
SG06	Unable to obtain additional memory required for resident system image module loading.
SG07	Unable to obtain memory required for resident system image construction.
SG08	Non-relocatable byte string encountered in binary module being processed from temporary file.
SG09	Unable to allocate temporary file space.
SG10	Overrun of SYSGEN address space by system being generated. Probable erroneous size specification in PATCH or POOL directive.
SG11	Sequence error while reading object module from file assigned to 'OBJ'.
SG12	CHECKSUM error while reading object module from file assigned to 'OBJ'.
SG13	Unable to allocate disc space for SYMTAB file.
SG14	Unable to allocate disc space for SYSTEM IMAGE file.
SG15	Maximum number (240) of symbol table/patch file entries exceeded.
SG16	Missing SYSTEM or SYMTAB directive.
SG17	Invalid IPU interval timer priority. Must not be between X'78' and X'7F'.
SG18	Maximum size of 80K for target system has been exceeded.

SG19	Attempt to define interrupt vectoring routine as system reentrant. Only device handlers may be system reentrant.
SG20	Unable to find "link" device in UDT.
SG21	Insufficient room in memory pool for download file list.
SG23	Insufficient shared memory table entries specified with SHARE directive. Number of entries must be equal to or greater than the number of partitions specified with /PARTITION NAME directives.
SG24	Attempt to define partition starting mapblock number in operating system area.
SG25	Attempt to define partition starting mapblock number in non-configured physical memory.
SG26	Attempt to use a module incompatible with the target machine type. The offending module name is the last entry on the listing followed by three asterisks (***).
SG27	The device specified in either the SMD, SWP, SID, LOD or POD directives is not included in the configuration being built.
SG28	The null device specification which is required to be included in every configuration is missing.
SG99	Directive errors encountered.

CODE	DESCRIPTION
SM01	The System Input Directory (M.SID) which is created at SYSGEN, does not exist. The directory may be created with the File Manager, but the file must be zeroed during creation.
SM02	The Systems Output Directory (M.SOD) which is created at SYSGEN, does not exist. The directory may be created with the File Manager, but the file must be zeroed during creation.
SM03	Unable to build a FAT/FPT for a system output task which is attempting to allocate an SLO or SBO file. Indicates a program error.
SM04	The Job Table entry associated with a job for which end-of-job processing is being performed has been destroyed. Indicates a program error.
SM05	Entry linkage is not consistent on the System Output Directory (M.SOD). The contents of M.SOD have been destroyed or a program error exists.
SM06	Entry linkage has been destroyed on the System Input Directory (M.SID).
SM07	Entry linkage has been destroyed on the System Output Directory (M.SOD).
SM08	Unrecoverable I/O error on spooled link file.
SM09	Unrecoverable I/O error on System Input Directory (M.SID).
SM10	Unrecoverable I/O error on System Output Directory (M.SOD).
SM11	Unrecoverable I/O error to a disc allocation map returned on call to File System Executive (H.FISE,4).
SM12	Attempt to activate System Output task unsuccessful.
SM13	Unrecoverable I/O error to the SMD returned on call to File System Executive (H.FISE,1).
SM14	Attempt to access a system input or output file in a memory-only environment.

System Input Task (J.SSIN)

CODE	DESCRIPTION
SN01	Blocking buffer or FAT space is not available.
SN02	Unrecoverable I/O error from the disc file being used as the SYC file.
SN03	System Input Directory (M.SID) does not exist or an unrecoverable I/O error was encountered in attempting to access it.
SN04	Job Sequence Number has been duplicated. Indicates a program error.
SN05	Spooled Input Directory (M.SID) is full.
SN06	A permanent file specified on the OPCOM BATCH command does not exist.
SN07	Unrecoverable I/O error to the SMD returned on call to the File System Executive (H.FISE,1 or H.FISE,10).
SN08	Unrecoverable I/O error to the allocation map returned on call to the File System Executive (H.FISE.3 or H.FISE,4).

System Output Task (J.SOUT)

CODE	DESCRIPTION
ST01	Unrecoverable write error on destination device for SLO or SBO records.
ST02	Unable to perform file code to file code allocation for separator file code.
ST03	Unable to issue magnetic tape mount message via allocation service.

Whenever a System Output task aborts, the task may be restarted with the OPCOM/REPRINT or REPUNCH commands.

SVC Trap Processor (H.IP06)

CODE	DESCRIPTION
SV01	Abort of unprivileged task using M.CALL.
SV02	Invalid SVC number abort.
SV03	Abort of unprivileged task attempting use of a "privileged-only" service.
SV04	Invalid SVC type abort.
SV05	Abort of unprivileged task attempting M.RTRN.

Swap Scheduler Task (J.SWAPR)

CODE	DESCRIPTION
SW01	Unrecoverable I/O error.
SW02	Reserved
SW03	Reserved
SW04	No 'E' memory available for SWAPR's buffer file.
SW05	No FAT or FPT to allocate.
SW06	Task has requested inswap but was never outswapped.
SW07	EOM detected on swap file.

'SYSBUILD'

CODE	DESCRIPTION
SY01	Unable to allocate or open input device during initial task loading. (Memory only MPX-32)
SY02	Unable to activate task. (Memory only MPX-32)
SY03	Unable to deallocate or close input device after initial task loading.
SY04	IPL device is undefined.
SY05	File is too small for the tape contents.
SY06	Transfer count on read is zero.
SY07	Unable to create a permanent file.
SY08	Unable to allocate file.

CODE	DESCRIPTION
UD01	Potential abort conditions have been detected during update processing. An abort status flag within the job's TSA has been set during execution processing. Conditional job control directives may be used to test status prior to job continuation.
UD02	User requested abort from mount prompt.

Miscellaneous Abort Codes

CODE	DESCRIPTIONS
BT01	Block mode timeout trap.
EX01	An abort has occurred in the task exit sequence.
EX02	An abort has occurred during the task abort sequence and has been changed to a delete (kill) task request.
MC01	Machine check trap.
MF01	A map fault trap has occurred. This is the result of a bad memory reference outside of the user's addressable space.
MP01	Memory error occurred in a task's logical address space. This is an internal or CPU failure. Rerun task.
NM01	Indicates a CPU failure.
OC01	The operator has requested that the task be aborted.
PV01	Privilege violation trap.
TS01	User requested removal from a BREAK request.
TS02	User requested removal from a Wait State queue.
TS03	Task running from specified terminal was aborted when the terminal disconnected.
UI01	Undefined instruction trap.

When system crash occurs as a result of a trap handler entry, the CPU halts with the registers containing the following information:

R0=PSD Word 0 (when trap generated)
R1=PSD Word 1 (when trap generated)
R2=Real address of instruction causing trap
R3=Instruction causing trap
R4=CPU status word (from trap handler)
R5=Crash code:

MP01=X'4D503031'	(See H.IP02 Codes)
NM01=X'4E4D3031'	(Non-Present Memory - H.IP03)
UI01=X'55493031'	(Undefined Instruction - H.IP04)
PV01=X'50563031'	(Privilege Violation - H.IP05)
MC01=X'4D433031'	(Machine Check - H.IP07)
SC01=X'53433031'	(System Check - H.IP08)
MF01=X'4D463031'	(Map Fault - H.IP09)
CP01=X'42543031'	(Cache Parity - H.IP10) 32/87 only
BT01=X'42543031'	(Block Mode Timeout - H.IP0E)
HT01=X'48543031	(Privileged Halt Trap - H.IPHT) CONCEPT/32 only
SW01=X'53573031'	(See SWAPR codes)

R6=Real address of register save block
R7=C'TRAP'=X'54524150'

For further description, see Volume 1, Section 2.10.

APPENDIX D

NUMERICAL INFORMATION

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 756 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 380 086 994 171 142 578 125

APPENDIX E

POWERS OF INTEGERS

TABLE OF POWERS OF SIXTEEN

16^n	n	16^{-n}
1	0	0.10000 00000 00000 00000 × 10
16	1	0.62500 00000 00000 00000 × 10^{-1}
256	2	0.39062 50000 00000 00000 × 10^{-2}
4 096	3	0.24414 06250 00000 00000 × 10^{-3}
65 536	4	0.15258 78906 25000 00000 × 10^{-4}
1 048 576	5	0.95367 43164 06250 00000 × 10^{-6}
16 777 216	6	0.59604 64477 53906 25000 × 10^{-7}
268 435 456	7	0.37252 90298 46191 40625 × 10^{-8}
4 294 967 296	8	0.23283 06436 53869 62891 × 10^{-9}
68 719 476 736	9	0.14551 91522 83668 51807 × 10^{-10}
1 099 511 627 776	10	0.90949 47017 72928 23792 × 10^{-12}
17 592 186 044 416	11	0.56843 41886 08080 14870 × 10^{-13}
281 474 976 710 656	12	0.35527 13678 80050 09294 × 10^{-14}
4 503 599 627 370 496	13	0.22204 46049 25031 30808 × 10^{-15}
72 057 594 037 927 936	14	0.13877 78780 78144 56755 × 10^{-16}
1 152 921 504 606 846 976	15	0.86736 17379 88403 54721 × 10^{-18}

TABLE OF POWERS OF TEN

10^n	n	10^{-n}
1	0	1.0000 0000 0000 0000
A	1	0.1999 9999 9999 999A
64	2	0.28F5 C28F 5C28 F5C3 × 16^{-1}
3E8	3	0.4189 374B C6A7 EF9E × 16^{-2}
2710	4	0.68DB 8BAC 710C B296 × 16^{-3}
1 86A0	5	0.A7C5 AC47 1B47 8423 × 16^{-4}
F 4240	6	0.10C6 F7A0 B5ED 8D37 × 16^{-4}
98 9680	7	0.1AD7 F29A BCAF 4858 × 16^{-5}
5F5 E100	8	0.2AF3 1DC4 6118 73BF × 16^{-6}
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC × 16^{-7}
2 540B E400	10	0.6DF3 7F67 5EF6 EADF × 16^{-8}
17 4876 E800	11	0.AFEB FF0B CB24 AAFF × 16^{-9}
E8 D4A5 1000	12	0.1197 9981 2DEA 1119 × 16^{-9}
918 4E72 A000	13	0.1C25 C268 4976 81C2 × 16^{-10}
5AF3 107A 4000	14	0.2D09 370D 4257 3604 × 16^{-11}
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D × 16^{-12}
23 86F2 6FC1 0000	16	0.734A CA5F 6226 F0AE × 16^{-13}
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449 × 16^{-14}
DF0 B683 A764 0000	18	0.1272 5DD1 D243 ABA1 × 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35 × 16^{-15}

APPENDIX F

ASCII INTERCHANGE CODE SET

Row	Col	0	1	2	3	4	5	6	7
Bit Positions 4 5 6 7		0	0	0	0	0	0	0	0
		0	0	0	0	1	1	1	1
		0	0	1	1	0	0	1	1
		0	1	0	1	0	1	0	1
0000	0	NUL 12-0-9-8-1	DLE 12-11-9-8-1	SP No punch	0 0	@ 8-4	P 11-7	. 8-1	p 12-11-7
0001	1	SOH 12-9-1	DCI 11-9-1	! 12-8-7	1 1	A 12-1	Q 11-8	a 12-0-1	q 12-11-8
0010	2	STX 12-9-2	DC2 11-9-2	" 8-7	2 2	B 12-2	R 11-9	b 12-0-2	r 12-11-9
0011	3	ETX 12-9-3	DC3 11-9-3	# 8-3	3 3	C 12-3	S 0-2	c 12-0-3	s 11-0-2
0100	4	EOT 9-7	DC4 9-8-4	$ 11-8-3	4 4	D 12-4	T 0-3	d 12-0-4	t 11-0-3
0101	5	ENQ 0-9-8-5	NAK 9-8-5	% 0-8-4	5 5	E 12-5	U 0-4	e 12-0-5	u 11-0-4
0110	6	ACK 0-9-8-6	SYN 9-2	& 12	6 6	F 12-6	V 0-5	f 12-0-6	v 11-0-5
0111	7	BEL 0-9-8-7	ETB 0-9-6	' 8-5	7 7	G 12-7	W 0-6	g 12-0-7	w 11-0-6
1000	8	BS 11-9-6	CAN 11-9-8	(12-8-5	8 8	H 12-8	X 0-7	h 12-0-8	x 11-0-7
1001	9	HT 12-9-5	EM 11-9-8-1	) 11-8-5	9 9	I 12-9	Y 0-8	i 12-0-9	y 11-0-8
1010	A	LF 0-9-5	SUB 9-8-7	. 11-8-4	: 8-2	J 11-1	Z 0-9	j 12-11-1	z 11-0-9
1011	B	VT 12-9-8-3	ESC 0-9-7	+ 12-8-6	; 11-8-6	K 11-2	[12-8-2	k 12-11-2	{ 12-0
1100	C	FF 12-9-8-4	FS 11-9-8-4	, 0-8-3	< 12-8-4	L 11-3	\ 0-8-2	l 12-11-3	| 12-11
1101	D	CR 12-9-8-5	GS 11-9-8-5	- 11	= 8-6	M 11-4	] 11-8-2	m 12-11-4	} 11-0
1110	E	SO 12-9-8-6	RS 11-9-8-6	. 12-8-3	> 0-8-6	N 11-5	^ 11-8-7	n 12-11-5	~ 11-0-1
1111	F	SI 12-9-8-7	US 11-9-8-7	/ 0-1	? 0-8-7	O 11-6	– 0-8-5	o 12-11-6	DEL 12-9-7

Some positions in the ASCII code chart may have a different graphic representation on various devices as:

ASCII	IBM 029
!	¦
[	¢
]	!
^	>

Control Characters:

NUL	–	Null	DC3	–	Device Control 3
SOH	–	Start of Heading (CC)	DC4	–	Device Control 4 (stop)
STX	–	Start of Text (CC)	NAK	–	Negative Acknowledge (CC)
ETX	–	End of Text (CC)	SYN	–	Synchronous Idle (CC)
EOT	–	End of Transmission (CC)	ETB	–	End of Transmission Block (CC)
ENQ	–	Enquiry (CC)	CAN	–	Cancel
ACK	–	Acknowledge (CC)	EM	–	End of Medium
BEL	–	Bell (audible or attention signal)	SS	–	Start of Special Sequence
BS	–	Backspace (FE)	ESC	–	Escape
HT	–	Horizontal Tabulation (punch card skip) (FE)	FS	–	File Separator (IS)
LF	–	Line Feed (FE)	GS	–	Group Separator (IS)
VT	–	Vertical Tabulation (FE)	RS	–	Record Separator (IS)
FF	–	Form Feed (FE)	US	–	Unit Separator (IS)
CR	–	Carriage Return (FE)	DEL	–	Delete
SO	–	Shift Out	SP	–	Space (normally nonprinting)
SI	–	Shift In	(CC)	–	Communication Control
DLE	–	Data Link Escape (CC)	(FE)	–	Format Effector
DC1	–	Device Control 1	(IS)	–	Information Separator
DC2	–	Device Control 2			