60471200

CONTROL DATA

CYBER CROSS SYSTEM VERSION 1 BUILD UTILITIES REFERENCE MANUAL

CDC® OPERATING SYSTEMS:

NOS 2

NOS/BE 1

60471200

CONTROL DATA

CYBER CROSS SYSTEM VERSION 1 BUILD UTILITIES REFERENCE MANUAL

CDC® OPERATING SYSTEMS:

NOS 2

NOS/BE 1

REVISION RECORD

Revision	Description
A (04/76)	Manual released.
B (06/76)	Manual Update (ECO 06420).
C (11/77)	Corrections on pages 3-2 and 3-11.
D (08/79)	Manual revised to incorporate CYBER Cross NOS R6.
E (02/80)	Manual revised to change above references from R6 to R5.
F (10/80)	Manual revised to incorporate on-line console removal and all PSRs to level 528. Manual title changed from CYBER Cross System, Version 1 Link Editor and Library Maintenance Programs Reference Manual. This revision obsoletes all previous editions.
G (11/22/82)	Manual revised at PSR level 580 to reflect release of CYBER Cross System Version 1.2, which runs under NOS Version 2.1 Release 6.1. The description of the AUTOLINK and EXPAND utilities has been removed from this manual and placed in the Communications Control Program Internal Maintenance Specification.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. Box 3492
SUNNYVALE, CALIFORNIA 94088-3492

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Front Cover	-
Title Page	-
ii	G
iii/iv	G
v	G
vi	G
vii	G
viii	G
ix	G
1-1	G
1-2	G
2-1 thru 2-6	G
3-1 thru 3-11	G
4-1 thru 4-11	G
A-1	F
A-2	G
B-1 thru B-7	G
C-1 thru C-4	G
D-1 thru D-3	G
E-1	F
E-2	F
F-1 thru F-4	G
G-1	G
H-1 thru H-7	G
Index-1 thru -3	G
Comment Sheet/Mailer	G
Back Cover	-

PREFACE

This manual was formerly called the CYBER Cross Link Edit/Library Maintenance Reference Manual. Under its new form and title, this manual describes three CYBER Cross System build utilities that aid in generating load files for a CONTROL DATA® 255x Network Processor Unit (NPU). These load files contain the on-line system for a network processing unit. The load files are of two types:

A Communications Control Program (CCP) used as a part of a NOS Network

A Communications Control INTERCOM (CCI) used as a part of a NOS/BE Network

The utilities are run on a CDC® CYBER host: either the CYBER 180 Computer Systems; the CYBER 170 Computer Systems; the CYBER 70 Computer Systems; or the 6000 Computer Systems under either the NOS 2 operating system or the NOS/BE operating system. The reader is assumed to be familiar with the command structure of these operating systems.

The utilities described in this manual are:

The library maintenance utility (MPLIB) used with CCI, which allows the user to generate a new library of object code modules. While this utility could be used with CCP, the current installation procedures do not use it.

The link utility (MPLINK), which assigns space for modules and links modules together. This utility produces a memory image load module file that can later be converted to a load file. The utility is used with both CCP and CCI.

The edit utility (MPEDIT), which allows the user to initialize values in the memory image load module. The utility is used with both CCP and CCI.

The memory image load module file that is produced for an NPU by MPLINK and MPEDIT is composed of object code modules that were initially generated by one of three CYBER Cross programs:

The CYBER Cross PASCAL compiler

The CYBER Cross macro assembler

The CYBER Cross micro assembler

Therefore, to fully use the capabilities of the utilities, the reader of this manual should be familiar with:

The installation procedures using a NOS or NOS/BE operating system

The CYBER Cross version of the PASCAL compiler

The CYBER Cross macro assembler and micro assembler

The NOS 2 and NOS/BE manual abstracts are pocket-sized manuals containing brief descriptions of the contents and intended audience of all manuals for NOS 2 and NOS/BE and their product sets. The manual abstracts can help a particular user determine the manuals that are of greatest interest.

The Software Publications Release History can help a user determine which revision level of software documentation corresponds to the Programming Systems Report (PSR) level of installed site software.

Related material is contained in the publications listed below. These publications are listed alphabetically within groupings that indicate relative importance to readers of this manual. Applicable operating systems are also indicated.

The following manuals are of primary interest:

Publication	Publication Number	NOS 1	NOS 2	NOS/BE 1
CYBER Cross System Macro Assembler Reference Manual	96836500	X	X	X
CYBER Cross System Micro Assembler Reference Manual	96836400	X	X	X
Network Products NAM Version 1 Network Definition Language Reference Manual	60480000	X	X	
NOS Version 2 Installation Handbook	60459320		X	
NOS/BE Version 1 Installation Handbook	60494300			X

Publication	Publication Number	NOS 1	NOS 2	NOS/BE 1
NOS Version 2 Reference Set, Volume 3, System Commands	60459680		X	
NOS/BE Version 1 Reference Manual	60493800			X
NOS Version 2 System Maintenance Reference Manual	60459300		X	

The following manuals are of secondary interest:

Publication	Publication Number	NOS 1	NOS 2	NOS/BE 1
CCP Internal Maintenance Specification	60490029		X	X
NOS Version 2 Diagnostic Index	60459390		X	
NOS/BE Version 1 Diagnostic Index	60456490			X
NOS Version 2 Manual Abstracts	60485500		X	
NOS/BE Version 1 Manual Abstracts	84000470			X
Software Publications Release History	60481000	X	X	X

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

NOTATIONS

Throughout this manual, the following conventions are used to present statement formats, command formats, request formats, operator type-ins, and diagnostic messages:

UPPERCASE	Uppercase letters indicate words, acronyms, keywords, or mnemonics required by the network software as input to it, or produced as output.
lowercase	Lowercase letters identify variables for which values are supplied by a console operator, a programmer using batch input, or by the system itself.
...	Ellipses indicate omitted entities that repeat the form and function of the last entity given. In PASCAL statements, ellipses are indicated by...).
[ ]	Square brackets enclose entities that are optional. If omission of an entity causes the use of a default value, the default is noted.
{ }	Braces enclose entities from which one must be chosen.

All numbers are given in decimal notation unless otherwise specified. Hexadecimal numbers in text are indicated by a subscript 16; for instance, $C000_{16}$. Hexadecimal parameters in commands require the display code designator, the dollar sign ($); for example, $C000.

INTRODUCTION 1

This manual describes the utility programs used to build the memory image load module file and to maintain the object code library file for 255x Network Processor Unit (NPU) on-line programs. These utilities are used with the standard Communications Control Program (CCP) or Communications Control Intercom (CCI) installation procedures described in the NOS and NOS/BE Installation Handbooks. The utilities are called and executed automatically from those installation procedures. The utilities are:

A library maintenance utility (MPLIB) used only with CCI

A link utility (MPLINK) used with both CCP and CCI

An edit utility (MPEDIT) used with both CCP and CCI

All the utilities execute on a CYBER host computer operating under either NOS or NOS/BE.

Figure 1-1 shows the logical flow of the utilities in producing a downline load file for a CCP system.

LIBRARY MAINTENANCE UTILITY (MPLIB)

This utility uses object code either from a previous library or from one of the CYBER Cross compilers or assemblers to generate a new library file. The library consists of object code which includes a directory to all the modules on the libraries.

LINK UTILITY (MPLINK)

MPLINK assigns space and links together all the modules that are to be a part of the load file. Each module is assigned an execution space on a memory image load module file, which, after initial values are assigned, can be converted to the load file for the NPU.

EDIT UTILITY (MPEDIT)

After MPLINK generates a memory image load module file (ABSOLMP) and a symbol table file (SYMTAB), the edit utility initializes values in selected variables. The variables to be initialized and the initialization values are specified by an MPEDIT program that uses a PASCAL-like syntax.

INPUTS TO THE UTILITIES

There are two types of inputs to the utilities:

Directives files. These directives are arranged into a batch input file.

Other files. The most important of these files consist of object code modules that are built into the on-line CCP or CCI system.

GENERAL COMMAND FORMAT FOR THE UTILITIES

Directives are used as the command inputs to the utilities. The general form of any directive is a command identifier (keyword) followed by a set of parameters:

```
KEYWORD,param1,param2,...,paramn
```

Parameters are separated by commas. Parameters usually specify either values (such as the size of NPU memory) or files (such as an application program name).

GENERAL DATA FORMAT INPUT TO THE UTILITIES

The data inputs to the utilities are modules in object code format that are generated by the CYBER Cross PASCAL compiler, by the CYBER Cross macro assembler, or by the CYBER Cross micro assembler. When used, the object code modules can be:

Separate files newly produced from one of the assemblers or compilers

Separate files read from magnetic tape or mass storage

Selected records read from a program library

OUTPUTS FROM THE UTILITIES

When used with NOS installation techniques, the principal output from the link and edit utilities is an initialized memory image load module on mass storage in the host computer. This load module can then be converted by the Load File Generator (LFG) into a downline loaded and formatted file.

A similar use of the link and edit utilities operating under NOS/BE produces an initialized memory image load module on mass storage in the host computer. This load module can be converted into a downline loaded and formatted file.

Optional outputs from various utilities include load maps, listings of all modules, and diagnostic reports.

The principal output from the library maintenance utility is a new, indexed library of the selected on-line NPU modules in object code format. All program libraries are held on the CYBER host's mass storage. Object code modules selected from any library are used to build a new load file for the on-line CCI system. The library maintenance utility could be used to build a new CCP library.

Another output of the library maintenance program is a listing of the programs on the library, together with information about these programs.

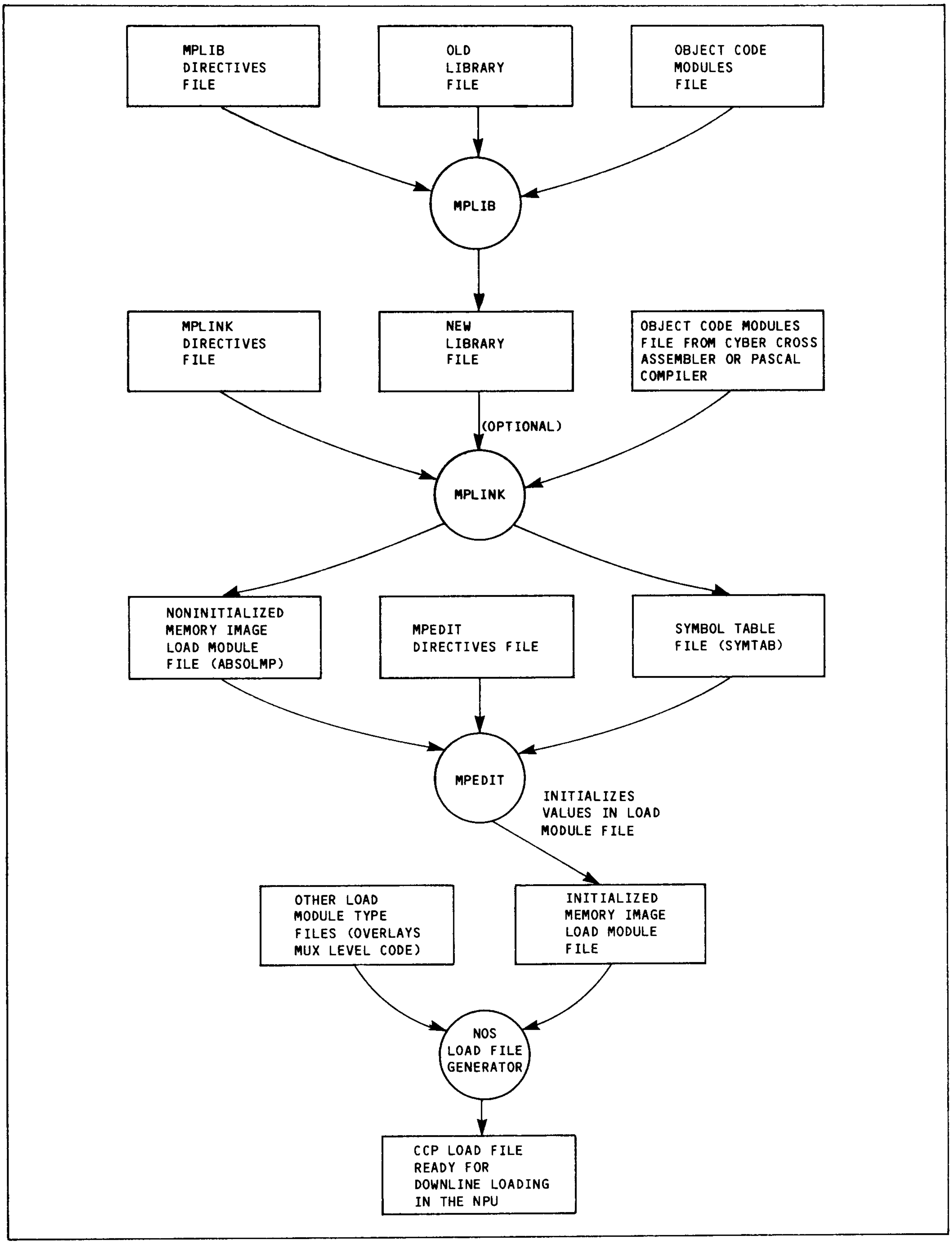

MPLIB DIRECTIVES FILE
OLD LIBRARY FILE
OBJECT CODE MODULES FILE
MPLIB
MPLINK DIRECTIVES FILE
NEW LIBRARY FILE
OBJECT CODE MODULES FILE FROM CYBER CROSS ASSEMBLER OR PASCAL COMPILER
(OPTIONAL)
MPLINK
NONINITIALIZED MEMORY IMAGE LOAD MODULE FILE (ABSOLMP)
MPEDIT DIRECTIVES FILE
SYMBOL TABLE FILE (SYMTAB)
MPEDIT
INITIALIZES VALUES IN LOAD MODULE FILE
OTHER LOAD MODULE TYPE FILES (OVERLAYS MUX LEVEL CODE)
INITIALIZED MEMORY IMAGE LOAD MODULE FILE
NOS LOAD FILE GENERATOR
CCP LOAD FILE READY FOR DOWNLINE LOADING IN THE NPU

Figure 1-1. Producing a CCP Downline Load File

LIBRARY MAINTENANCE UTILITY

INTRODUCTION

The library maintenance program (MPLIB) can be used with the Communications Control Program (CCP) or the Communications Control INTERCOM (CCI) but currently is used only with CCI. The utility uses a set of directives operating on a set of object code files to generate a new library. MPLIB uses two files:

An object code file. This file contains the object code for modules that are to be added to the new library or that are to replace existing object code modules of the same names which are already on the old library. This file is required. If there is no old library, the modules on the object code file create a library.

The old library file. Although this file is optional, it is usually present.

The user has the option of ordering listings of the new and/or the old libraries.

MPLIB INPUTS

The library maintenance utility requires two files:

A directives file

The object code file that contains the modules used to generate the library

If a library has already been built, MPLIB also requires the old library file.

Several calling parameters are associated with the library maintenance utility. These parameters are discussed in the Executing MPLIB subsection.

MPLIB DIRECTIVES FILE

This file contains the directives in the order of execution. The directives:

Cause modules to be selected for the library or deleted from an existing library

Allow the operator to order a listing of the new and/or the old library

Terminate the library maintenance operation

The default name of this file is INPUT.

MPLIB OBJECT CODE FILE

This file contains the object code of modules that:

Build a new library for the first time

Replace existing modules of the same name on the old library (the new library contains these modules)

The relocatable object code format of these modules is given in appendix F. Object programs that replace old programs of the same name are added to the new library at the same relative position as the replaced module. New object code programs that were not on the previous library are added to the end of the new library in the order in which they occur on the object file. MPLIB adds a directory record to this file.

The default name of this file is LGO.

MPLIB OLD LIBRARY FILE

This file was created by a previous run of MPLIB. Once a library file has been created, it cannot be modified by MPLIB. A library file is modified by creating a new file.

The format of a library file is shown in figure 2-1. The first record of the file consists of the file name and the library directory. Each program in the library has an entry in the directory. Following this is the object code of each program. Each program is contained in a single record; the records appear in the same sequence as their entries in the directory.

The default name of this file is OLDLIB.

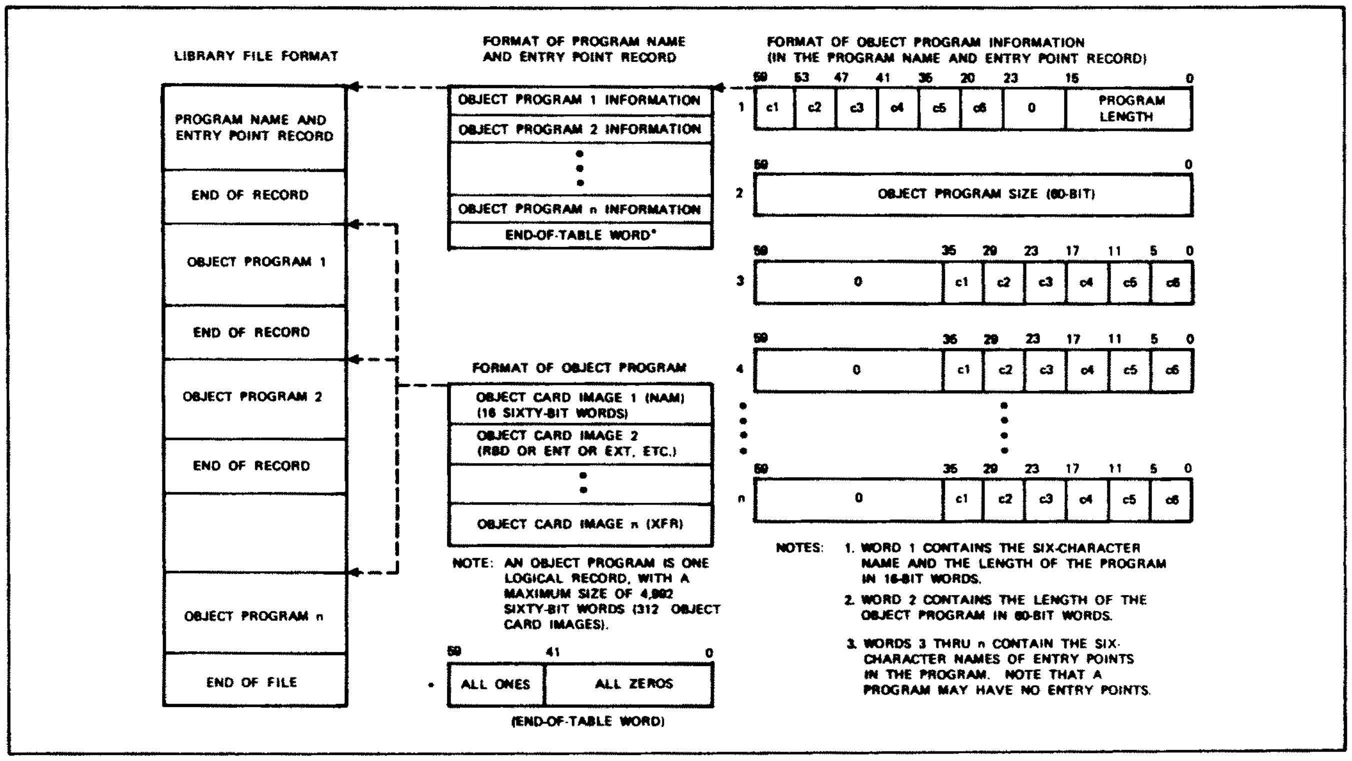
MPLIB OUTPUT FILES

MPLIB produces two output files: the new library file and an optional listing of the new and the old library file.

NEW LIBRARY FILE

The new library file has the same format as the old library file. However, it contains additional modules and substituted modules (as specified by the directives) and lacks modules that the directives have deleted.

The default name of the new library is NEWLIB.



LIBRARY FILE FORMAT
PROGRAM NAME AND ENTRY POINT RECORD
END OF RECORD
OBJECT PROGRAM 1
END OF RECORD
OBJECT PROGRAM 2
END OF RECORD
OBJECT PROGRAM n
END OF FILE
FORMAT OF PROGRAM NAME AND ENTRY POINT RECORD
OBJECT PROGRAM 1 INFORMATION
OBJECT PROGRAM 2 INFORMATION
OBJECT PROGRAM n INFORMATION
END-OF-TABLE WORD*
FORMAT OF OBJECT PROGRAM
OBJECT CARD IMAGE 1 (NAM) (16 SIXTY-BIT WORDS)
OBJECT CARD IMAGE 2 (RBD OR ENT OR EXT, ETC.)
OBJECT CARD IMAGE n (XFR)
NOTE: AN OBJECT PROGRAM IS ONE LOGICAL RECORD, WITH A MAXIMUM SIZE OF 4,992 SIXTY-BIT WORDS (312 OBJECT CARD IMAGES).
59 41 0
* ALL ONES ALL ZEROS
(END-OF-TABLE WORD)
FORMAT OF OBJECT PROGRAM INFORMATION (IN THE PROGRAM NAME AND ENTRY POINT RECORD)
59 53 47 41 35 29 23 15 0
1 c1 c2 c3 c4 c5 c6 0 PROGRAM LENGTH
59 0
2 OBJECT PROGRAM SIZE (60-BIT)
59 35 29 23 17 11 5 0
3 0 c1 c2 c3 c4 c5 c6
59 35 29 23 17 11 5 0
4 0 c1 c2 c3 c4 c5 c6
59 35 29 23 17 11 5 0
n 0 c1 c2 c3 c4 c5 c6
NOTES: 1. WORD 1 CONTAINS THE SIX-CHARACTER NAME AND THE LENGTH OF THE PROGRAM IN 16-BIT WORDS.
2. WORD 2 CONTAINS THE LENGTH OF THE OBJECT PROGRAM IN 60-BIT WORDS.
3. WORDS 3 THRU n CONTAIN THE SIX-CHARACTER NAMES OF ENTRY POINTS IN THE PROGRAM. NOTE THAT A PROGRAM MAY HAVE NO ENTRY POINTS.

Figure 2-1. Format of an MPLIB Library File

LIBRARY LISTINGS

The optional new and old library listings are requested by an MPEDIT directive. If the directive is used, the listing file is sent to the output file. The user can order printed copies of the file using the techniques described in the NOS/BE reference manual.

A library listing consists of program names, program lengths, and program entry points for each of the programs on the library. A sample listing is shown in figure 2-2.

If both the new and the old library listings are ordered, the old library is the first part of the output file and the new library is the second part.

EXECUTING MPLIB

The utility is executed by attaching the MPLIB permanent file and then using the name call statement (see the NOS/BE reference manual).

Five parameters, which specify files, are associated with the MPLIB name call statement. The format of the statement is:

```
    MPLIB[,lfn1,lfn2,lfn3,lfn4,lfn5.]
```

where:

lfn1 is the object code input file name (default name is LGO)

lfn2 is the input directives file name (default name is INPUT)

lfn3 is the output listing file name (default name is OUTPUT)

lfn4 is the old library file name (default name is OLDLIB)

lfn5 is the name of the new library file created by the MPLIB run (default name is NEWLIB)

The parameters are positional. Therefore, if a parameter is omitted, its delimiting commas must be retained. If a call uses all default parameters, all commas are omitted. Such a default call is terminated with a period to indicate the lack of delimiting commas.

Example 1:

```
    MPLIB.
```

This call uses an object code file, LGO (lfn1), which is operated on by the directives file, INPUT (lfn2), to produce a new library, NEWLIB (lfn5), from an old library, LGO (lfn4). It outputs any listings to the library file OUTPUT (lfn3).

Example 2:

```
    MPLIB,,,OLDOBJ,NEWOBJ.
```

This call uses default parameters for the object, directives, and listing files. However, it names the new and old libraries.

This call uses an object code file, LGO (lfn1), which is operated on by the directives file, INPUT (lfn2), to produce a new library, NEWOBJ (lfn5), from an old library, OLDOBJ (lfn4). It outputs any listings to the listing file, OUTPUT (lfn3).

```
                 MPLIB Directives

                 *ALL.                            COPY ALL OF LGO              SCFLIB
                 *LST,NEW.                                                     SCFLIB
                 *END.                                                         SCFLIB


           NEW LIBRARY

           PROGRAM    LENGTH    ENTRY POINTS

           ZEROX       0040        ZEROX      PBPSWI      0013      PBPSWI               BIV020
                                                                                          BIV040
           BEGINX      000B        BEGINX     PBPUTP      0017      PBPLTP               BIV050
                                                                                          BIC230
           PBINTR      0040        PBINTR     PBGETP      0011      PBGETP               BIC250
                                                                                          BIC400
           JUMPS       0010        JUMPS      PBSTPM      0013      PBSTPM               BIC410
                                                                                          BIC420
           ADDRES      001A        CIBADD     UNLOCK      000F      UNLOCK               BIC430
                                   CCPLEV                                                 BINTXT
                                   CCPCYC     QULOCK      0011      QULOCK               BIN005
                                   ADDRLC                                                 BIN110
                                   ADDRES     PBRTCD      00CB      PBRTCD               BIN140
                                   CCPVER                                                 BIN180
                                   ADDRSU     QENTRY      004B      QENTRY               BIN440
                                                                                          BISTID
           PBLN00      0029        PBLN00     QEXIT       001E      QEXIT                BISTBI
                                                                                          BISTRS
           PBLN01      001F        PBLN01     BUFMAI      0076      PBRELC               BISTRC
                                                                    PBREL1
           PBLN02      001F        PBLN02                           PBGET1   INSTA7 01A7 BIX100
                                                                    PBRELZ               BIX140
           PBLN03      001F        PBLN03                                                 BIC100
                                              LISTSR      0028      PBLSGE               BIC210
           PBLN06      002D        PBLN06                           PBLSPU               BIY350
                                                                                          BICLAS
           PBLN08      001F        PBLN08     PBSTOP      001C      PBSTOP               BIDCDN
                                                                                          BIOVER
           PBGETC      0008        PBGETC     PBSLJ       0007      PBSLJ                BIBUTH
                                                                                          BITERM
           PBPUTC      0007        PBPUTC     PTTPIN      002A      PTTPIN               BIC200
                                                                                          BIN230
           PBCALL      000B        PBCALL     PBSCLA      002F      PBSCLA               BIN250
                                                                                          BIT100
           PNSTOR      0022        PNSTOR     MODMST      00CE      MILT8                BIX130
                                                                    MILT9                BIX150
           PBFILE      002B        PBDF                             MILT0                BIX160
                                   PBBF                             MILT4                BIX170
                                                                    MILT1                BIX180
           PILMT       003C        PILMT                            MILT3                BIX190
                                                                    MILT6                BIX200
           PBAMAS      0012        PBAMAS                           MILTA                BIX210
                                                                                          BIX240
           PBLMAS      000D        PBLMAS     ASMUSE      000C                            BIX250
                                                                                          BIX300
           PBOMAS      00CE        PBOMAS     HSPRUI      00CF      HDCD9                BIX320
                                                                    HINSPT               BIX350
           PBSMAS      000C        PSBMAS                           HTERM2               BIX440
                                                                                          BIX450
           PBBEXI      000B        PBBEXI     HS2PRU      C14C      H2ISPT               BIERRO
                                                                                          BIT001
           PBAEXI      000E        PBAEXI     HSPRUT      027D      HXTPT                BID009
                                                                    HNTTPT
           PBSETP      0022        PBSETP                           H1ATPT   TPST78 01C1 B27TPT
                                                                    H2PTPT               B37TPT
           PBCLRP      000C        PBCLRP
                                              HSPMGS      000C      NAKHMS   BSCMSG 002D ACKOMS
```

Figure 2-2. Sample MPLIB Library Listing (Sheet 1 of 2)

```
PBCOMP     001F     PBCOMP                          ACKHMS              ACK1MS
                                                                        EOTBMS
PBRDPG     0010     PBRDPG     INPST7     021D      BISTAR              WACKBM
                                                    BIDISC              TTDBMS
                                                    BIENTR              DISCBM
                                                    BIB010              ENQBMS
                                                                        NAKBMS
PDEXIT     004C     PDEXIT     PNTMLD     0098      PNTMLD

PDSTTR     02B2     PDSTTR     PNCNTL     0056      PNCNTL

PDTERM     0011     PDTERM     PNLCR      0026      PNLCR

OLDIAG     048F     OLDIAG     PNSTAT     002F      PNSTAT

OLDMUX     0012     OLDMUX     PN1LNS     0077      PN1LNS

PTMSQU     003A     PTMSQU     PN2LNS     005C      PN2LNS

PTMSCA     007D     PTMSCA     PNLNST     00B7      PNLNST

PTDELM     004B     PTDELM     PN1TML     009F      PN1TML

PTTYMU     0007     PTTYMU     PNTMLS     009A      PNTMLS

PTTYTI     054F     PTTYTI     PNBRDC     0138      PNBRDC

PTTYTC     000E     PTTYTC     PN1BRD     00E7      PN1BRD

PNAWAI     001F     PNAWAI     PNOVLO     011B      PNOVLO

PNRTN      000A     PNRTN      PNOVLD     0089      PNOVLD

PNSMBA     0056     PNSMBA     PNOVLT     001A      PNOVLT

PNLNBA     004D     PNLNBA     PNFRCE     0027      PNFRCE

PNRVRS     0021     PNRVRS     PNPSTA     00A2      PNPSTA

PNT000     0037     PNT000     PNDSTA     0129      PNDSTA

PNQREL     0035     PNQREL     PNSMGE     0125      PNSMGE

PNGTCB     004C     PNGTCB     QDEBUG     0021      QDEBUG

PNTCBS     0066     PNTCBS     PBCLR      0028      PBCLR

PNDLTC     009F     PNDTLC     PBDISP     006B      PBDISP

PNDISC     0009     PNDISC     PBLOAD     004A      PBLOAD

PNSMTR     000A     PNSMTR     PBILL      0007      PBILL

PNSMWL     0171     PNSMWL     PBHALT     01C1      PBHALT

PNSMDI     00E4     PNSMDI     PBMON      0063      PBMON

PNCONF     0294     PNCONF     TOTIME     0011      TOTIME

PNLNCH     01AE     PNLNCH     TOSTAR     0008      TOSTAR

PNTMLC     02EC     PNTMLC     TOSTOP     0008      TOSTOP

PNDELE     00D9     PNDELE     PIDTBL     0081      PIDTBL

PNENAB     006F     PNENAB     MAIN$      0018      MAIN$

PNDISA     0105     PNDISA

PNLINE     00F5     PNLINE
```

Figure 2-2. Sample MPLIB Library Listing (Sheet 2 of 2)

MPLIB DIRECTIVES

There are six MPLIB directives. Four of them choose the programs that form the new library. One of them selects the optional listings. One of them terminates the directives list. Except for the terminating statement, the directives can occur in any order. Table 2-1 summarizes the MPLIB directives.

TABLE 2-1. SUMMARY OF MPLIB DIRECTIVES

Name	Function
*ALL	Adds all the programs on the object code file to the new library file.
*PUT	Adds selected progams to the new library file. Also replaces old library file programs with object code file programs of the same name.
*DEL	Suppresses copying of specified programs from the old library file to the new library file.
*SUP	Suppresses copying of specified programs from the object code file to the new library file. Used only with the *ALL directive.
*LST	Requests a listing of either the new library or the old library.
*END	Terminates the directives file and ends the library building operation.

All directives begin with an asterisk (*); any directive can be terminated with an optional period. The general form of an MPEDIT directive is:

```
*DIRECTIVENAME[,parameter.]
```

All directives begin in position 1.

If there are no directives in the directive file, the old library is copied to the new library without alteration.

*ALL - ADDS ALL THE OBJECT CODE FILE PROGRAMS TO THE NEW LIBRARY

The *ALL directive causes MPLIB to copy all the programs on the object code file to the new library file. It is used to create a new library for the first time. It can also be used for adding a set of new modules (such as a terminal interface program) to an existing library.

The format of the directive is:

```
*ALL[.]
```

*PUT - ADDS PROGRAMS TO THE LIBRARY FROM THE OBJECT CODE FILE

The directive is used to select programs for inclusion in the new library. It is also used to replace programs on the old library file with programs of the same names from the object code file. The utility can be used in the latter manner to update a program library.

The *PUT directive can add either a single program or a sequence of programs from the object code file to the new library file.

The single program format of the directive is:

```
*PUT,mod[.]
```

where mod is the program name. It starts with a letter. If the name exceeds six characters (letters or numbers), MPLIB discards the seventh and following characters. The names on the old library are no longer than six characters.

If a program is replaced with a new version of the program, the new program has the same place in the new library index and in the library file.

The series program format of the directive is:

```
*PUT,mod1-mod2[.]
```

where mod1 is the name of the first program in the object code sequence, and mod2 is the name of the last program in the sequence. Names are truncated to six characters as necessary. If the programmer wishes to replace a series of programs on the old program library, the names of all programs in the sequence must be identical to the new names. This form of the directive is particularly useful for adding new applications to a library, or replacing applications where the modular structure of the application has not changed.

A dollar sign ($) symbol can be substituted for the name of the first or last program on the object code file.

*DEL - DELETES PROGRAMS FROM THE LIBRARY

The *DEL directive suppresses copying the specified program or programs from the old library to the new library.

The single program deletion format is:

```
*DEL,mod[.]
```

where mod is the program name. If the name exceeds six characters, MPLIB discards the seventh and following characters.

The series program deletion format is:

```
*DEL,mod1-mod2[.]
```

where mod1 is the first of the programs to be deleted during copying, and mod2 is the last program to be deleted in the sequence. All programs on the library index between (and including) the named programs are deleted. (A listing of the old library can be produced by a *LST directive to find the order of programs on the library.)

A dollar sign ($) symbol can be substituted for the name of the first or last program on the library file.

*SUP - SUPPRESSES COPYING PROGRAMS FROM THE OBJECT CODE FILE TO THE LIBRARY

The *SUP directive is used only with the *ALL directive. It allows the user to suppress copying of the specified program or programs from the object code file to the new library.

The single program suppression format is:

```
*SUP,mod[.]
```

where mod is the program name. If the name exceeds six characters, MPLIB discards the seventh and following characters.

The series program suppression format is:

```
*SUP,mod1-mod2[.]
```

where mod1 is the first of the programs to be deleted during coping, and mod2 is the last program to be deleted in the sequence. All programs on the object code file between (and including) the named programs are deleted.

A dollar ($) symbol can be substituted for the name of the first or last program on the object code file.

*LST - LISTS A LIBRARY

The *LST directive is used to request either a listing of the new library or a listing of the old library.

The format for the old library listing is:

```
*LST,OLD[.]
```

The format for the new library listing is:

```
*LST,NEW[.]
```

If both listings are requested, the old library occurs first on the output file.

*END - ENDS THE LIBRARY BUILDING OPERATION

The *END directive terminates the directives file. The format of the directive is:

```
*END[.]
```

MPLIB ERROR MESSAGES

Library maintenance utility error messages are listed in table B-1 of appendix B. The significance of the message and the action that should be taken when the message appears are also given in that table.

LINK UTILITY

INTRODUCTION

The link utility (MPLINK) uses an input directives file and an input object code file to generate two outputs (see figure 1-1 in section 1):

A memory image load module file consisting of object code modules. The local file name (lfn) for this file is ABSOLMP. The file's modules are located in memory image order; that is, they have the absolute addresses they would have if they loaded into a Network Processing Unit (NPU). Initializable variables have the same values that they have on the object code input file. These variables are initialized later to selected values by the edit utility.

A symbol table (lfn = SYMTAB) consisting of the module names and entry points.

The edit utility uses both these files plus its own input directives to generate an initialized version of the memory image load module file. This load module can then be used to generate the downline load files used by the host to load a Communications Control Program (CCP) or a Communications Control Intercom (CCI) system into an NPU.

If standard CCP installation procedures are used, this program is generated by the build procedures from the release tapes; its use is generally invisible to the system installer (see the NOS and NOS/BE Installation Handbooks).

The MPLINK input directives file can be:

User-supplied

Generated by SCF procedures during a standard CCI installation

The link utility supplies special output listings including memory maps, symbol lists, input directives, and a hexadecimal listing of the memory image file.

NPU ADDRESSING

MPLINK assigns each module to an execution area in main memory, in extended memory, or in an overlay area of main memory. To uniquely address 128K (131072) words, an 18-bit address is provided (only 17 bits of the address are used). However, when paging registers are used, an 11-bit address will locate any word on a 2K (2048) word physical page. The remaining seven high-order bits are used to designate the logical page. Note that both CCP and CCI use an 8K (8196) word logical page.

The NPU has two addressing modes: paged and absolute. In either mode, the operating system calculates a 16-bit address for each memory reference.

In the absolute mode, the 16-bit value is the effective address in the range 0000 to $FFFF_{16}$.

In the page address mode, the page registers are used to achieve an effective 18-bit memory address in the range 0000 to $3FFFF_{16}$ (only the range 0000 to $1FFF_{16}$ is used).

PAGE ADDRESSING MODE

In page addressing mode, the NPU memory is subdivided into physical pages, each of which is 2K words long. The location of a word within a page requires an 11-bit address (range 000 to $7FF_{16}$) and is called the page displacement. Page displacement is the least significant bits of an NPU address.

Each page is assigned a unique identification (range 00 to $7F_{16}$) called the page number. The page number uses the most significant bits of an NPU address.

Page displacement taken together with page number gives an 18-bit addressing capability.

During page addressing mode, a page number is associated with one of the 32 hardware paging registers (range 00 to $1F_{16}$). This association requires five bits of the address and is handled by an MPLINK directive that associates the page numbers with the page registers. Page register selection (five bits) together with page displacement (11 bits) gives the normal 16 bits of memory address referencing.

There are two sets of 32-page registers. Either set (0 or 1) can be active at any one time. The set being used is selected by the executing program. Figure 3-1 correlates the 18-bit address to page register and page displacement.

MPLINK assumes that all memory address specifications are in the page addressing mode. Therefore, each addressing specification has four distinct components:

Page displacement

Page number

Page register

Page register set

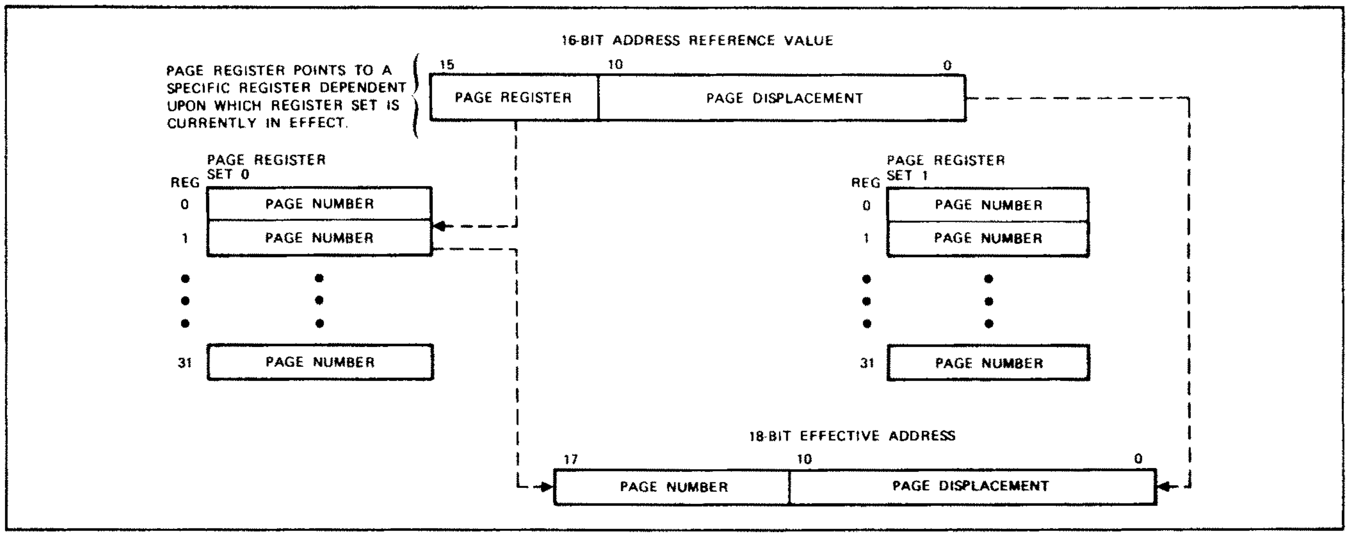


Figure 3-1. Page Register Selection

ABSOLUTE ADDRESSING MODE

The link and edit utilities do not support an absolute addressing mode directly. However, the default mode causes MPLINK to generate a program that effectively is an absolute addressing mode. In this case, address assignments are made entirely from page register set 0.

In default mode, the page register contents are the same as the page register numbers; that is, page register 0 has a zero value, page register 1 has a 1 value, etc. The resulting address resolution provides the same addresses that would be generated if absolute addressing mode was used.

SPECIFYING A MEMORY ADDRESS

The memory address is specified in three parts:

Number of Bits	Address Part
18	An effective address consisting of a page number plus page displacement
5	Page register
1	Page register set

The format of the address is:

effective address:page register:register set

Note that address parts are separated by colons. Each part is a numerical value in one of seven formats:

A decimal constant. This is preceded by a sign if necessary. Examples: 10, -734.

A positive hexadecimal constant. This is preceded by a dollar sign ($). Examples: $2000, $4FAC.

A linked module name.

An entry point name.

An overlay area name.

A local variable name.

An address function.

A previously linked module name with an explicit address assignment, a defined entry point name, a defined overlay area name, a defined local variable name, or an address function can be used as any part of the specified address. The effective address associated with any of these names represents the numerical address value.

Address Functions

MPLINK provides five functions that can be used with an operand or an address expression to generate a part of an address. The functions are requested by means of keywords. If this method is used, specification must have the following format:

/keyword(name)

where keyword is a reserved word used by the link utility for a specific operation, and name identifies a module, an entry point, or an overlay. Table 3-1 summarizes the allowable keywords.

TABLE 3-1. SUMMARY OF ADDRESS FUNCTION KEYWORDS

Keyword	Value Returned by the Function
PGDISP	11-bit page displacement
PGNUM	7-bit page number
PGREG	5-bit page register
PGSET	1-bit page register set
OVID	Last two characters of the overlay name in which the overlay module resides

For example:

/PGNUM(PNSMWL)

Returns the 7-bit number of the page used by the service module (PNSMWL).

Address functions can be used only if the module, entry point, or overlay has been explicitly named and the assignment of the address related to the name has preceded the reference.

Examples of a full address specification are:

$13B75:$A:1

The 18-bit effective address is composed of a page number = 27_{16}, and a page displacement of 375_{16}.

Page register 10 is to be used.

Page register set 1 is to be used.

MODA:/PGREG(MODB):1

The 18-bit effective address is taken from the starting execution address of MODA.

The page register where MODB is located is to be used.

Page register set 1 is to be used.

$1A45:/PGREG(MODA):/PGSET(MODB)

The 18-bit effective address is composed of a page number (3) and a page displacement of 045_{16}.

The page register where MODA is located is to be used.

The page register set where MODB is located is to be used.

Abbreviating Address Specification

It is not always necessary to specify the second (page register) and third (page register set) parts of the memory address. If only the first part of the address is specified (either as a constant or an address function), the page register is equal to the page number portion of the effective address (upper seven bits), and the page register set is assumed to be the same as in the previous memory address specification. For example:

$421F:8:0

$421F:8

$421F::0

$421F

All specify the same address: the page displacement is $21F_{16}$, the page number is 8, the page register is 8, and the page register set is 0.

Addresses are specified similarly if an address function is used. For example, MODA is equivalent to MODA:PGREG(MODA):PGSET(MODA).

ADDRESS ASSIGNMENT

MPLINK maintains an internal location counter for the four-component memory address. The location counter (which is used to assign space within the memory image file) is initially set to zero. The components are updated automatically as the memory image file is generated. Specifying a memory address within a link or overlay directive is the only method used by the link or edit utilities to explicitly assign an address.

As 16-bit words of a module's object code are moved into the memory image load module file (possibly with address resolution), the words are assigned consecutive memory addresses. If assigning the next address causes a memory page overflow condition, the internal location counter is adjusted to the first word (displacement = 0) of the next consecutive page. At the same time, the page register value is incremented by one.

Note that memory page overflow is not an error condition unless the resultant page register value is greater than 31 or the page number is greater than 127.

Unless MPLINK is given a specific load address for linking a module, the memory address assigned to a module is the next available memory address held in the internal location counter.

An area of memory that is designated as an overlay area can have several different module groupings for the area. Such a module grouping is called an overlay. On-line CCP or CCI execution of these different groupings occurs at different times.

Each overlay has a unique, two-letter identifier. If the user does not specify the identifier, MPLINK assigns the next alphabetic character in sequence (range AA through ZZ) when the next overlay is built. The binary equivalent of the identifier is returned to the user with each OVID keyword assignment.

The first overlay module of an overlay group is assigned the memory address at the start of the overlay area. Subsequent overlay modules of the same group are assigned space directly following the previously linked overlay module. MPLINK assigns overlay modules in this fashion until the next *L directive with an explicitly declared address occurs (the *L directive declares the overlay name). The user must explicitly declare all overlays using this directive.

MPLINK INPUT FILES

The user must supply MPLINK with two input files: one file contains directives, the other file contains object code modules. The user has the option of supplying a library file in addition to, or instead of, the object code module file. The MPLINK procedural flow is shown in figure 3-2. Examples of the use of MPLINK are given in appendix G.

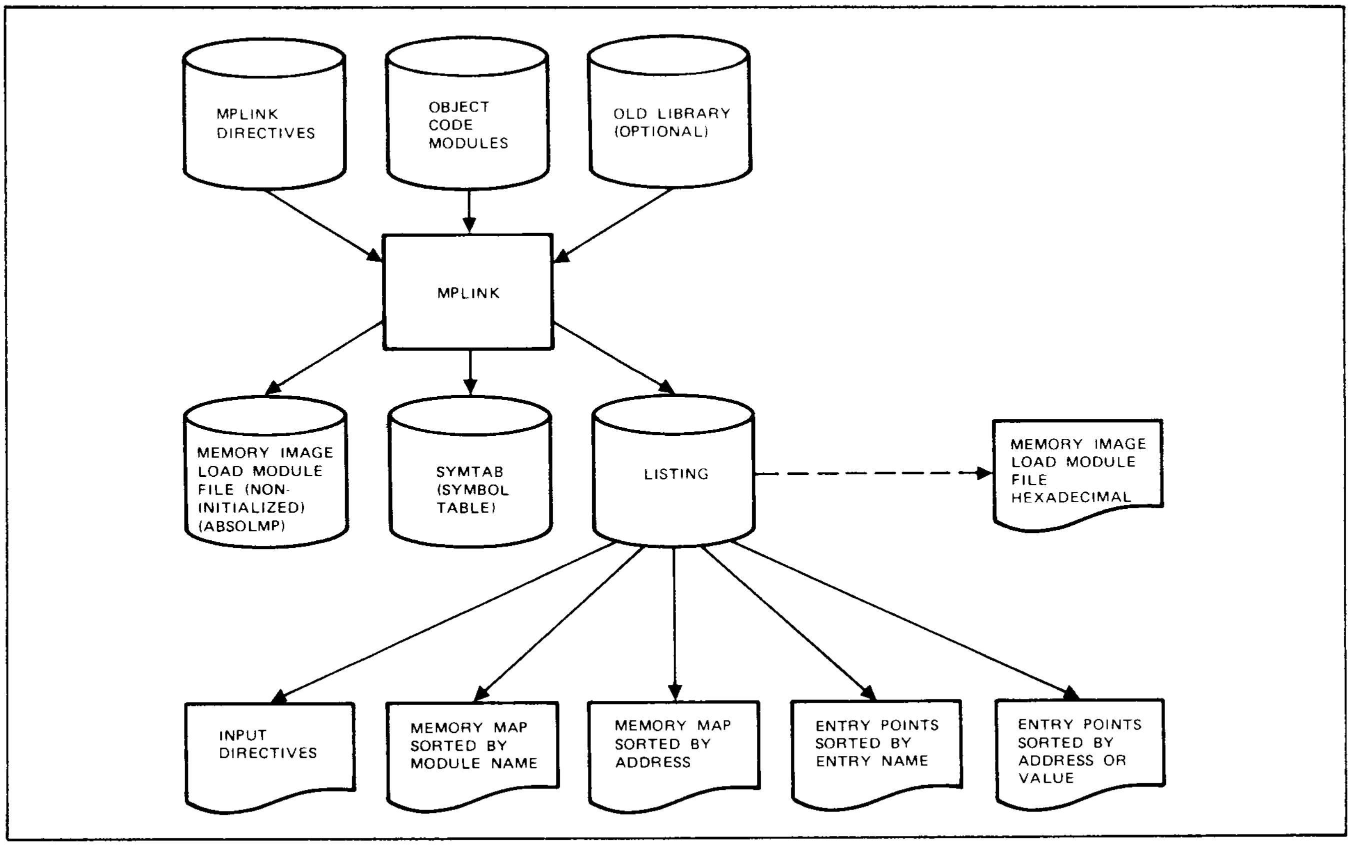

MPLINK DIRECTIVES
OBJECT CODE MODULES
OLD LIBRARY (OPTIONAL)
MPLINK
MEMORY IMAGE LOAD MODULE FILE (NON-INITIALIZED) (ABSOLMP)
SYMTAB (SYMBOL TABLE)
LISTING
MEMORY IMAGE LOAD MODULE FILE HEXADECIMAL
INPUT DIRECTIVES
MEMORY MAP SORTED BY MODULE NAME
MEMORY MAP SORTED BY ADDRESS
ENTRY POINTS SORTED BY ENTRY NAME
ENTRY POINTS SORTED BY ADDRESS OR VALUE

Figure 3-2. MPLINK Procedural Flow

MPLINK DIRECTIVES FILE

This file can be generated by one of the following:

CCP or CCI: The user can generate his own file of input directives using the MPLINK directives described later in this section.

CCI: The installer uses the SCF procedures with the standard installation processes.

In all cases, the directive file is the first input file presented to MPLINK.

MPLINK OBJECT CODE INPUT FILE

The required input file contains the object code modules to be included in the build.

These modules in this file were previously put in this form by a CYBER Cross assembler or compiler (macro assembler, micro assembler, or PASCAL compiler). See the appropriate CYBER Cross Language Reference Manual. The format of this relocatable object code is given in appendix F.

Optionally, in CCI, the input modules file can be a library file version of the modules in object code. This file was previously produced by MPLIB. The library file is always presented to MPLINK as the NEWLIB file. The library is used by MPLINK to resolve all unsatisfied external references.

Note that any object code file must be rewound prior to delivery to MPLINK; MPLINK does not rewind the files automatically.

MPLINK OUTPUT FILES

Three types of output files are produced by MPLINK:

Memory image load module file (ABSOLMP)

System table (SYSTAB)

Listings

MEMORY IMAGE LOAD MODULE FILE (ABSOLMP)

The file contains all the object code modules specified by the MPLINK directives. In this absolute memory image file:

All modules are assigned to a specific execution address.

All modules are assigned to a selected page register.

All relocatable addresses are converted to absolute addresses.

All external references are resolved.

All overlay modules are grouped in specified overlay areas.

SYMBOL TABLE FILE (SYMTAB)

The SYMTAB file contains the entire set of entry symbols and module names defined by MPLINK. Each entry has a value (either a memory address, a displacement, or a constant), a field start location, and a field length.

MPLINK LISTINGS

MPLINK automatically produces five listings; a sixth listing can be produced at the user's option. The listings are:

A copy of the input directives file.

A module memory map sorted by module name. A sample partial listing is shown in figure 3-3.

A module memory map sorted by module address.

An entry symbol list sorted by entry name. A sample partial listing is shown in figure 3-4.

An entry symbol list sorted by address.

A hexadecimal list of the memory image load file (optional). This is requested by the *DMP directive.

```
                              CYBER MINI CROSS SYSTEM - LINK EDITOR -

                           MODULE MEMORY MAP - SORTED BY MODULE NAME

   *MODULE*   *ADDRESS*      *MODULE*   *ADDRESS*      *MODULE*   *ADDRESS*      *MODULE*   *ADDRESS*

    AACAPL      493C          PBCOMP      53C1          PBPOPO      7A7D          PINIT       FBAF
    AAEAPL      48FC          PBCOPY      57EE          PBPROP      7A55          PINWIN      F724
    AAEBCD      48FC          PBDELE      653E          PBPSWI      53F1          PIPROT      F67F
    AASBAP      4A3C          PBDLTX      58CB          PBPUTP      540B          PISIZC      F6D5
    AASCST      48BC          PBDNAB      76D7          PBPUTY      5878          PITMRS      FE05
    AASTAP      49FC          PBFILE      5325          PBQBLK      4E4F          PIWLIN      FBF8
    ABAPLA      49BC          PBFMAD      7F84          PBQ1BL      4E78          PLCBIN    1605C:205C
    ACAPLA      4C3C          PBFMAH      6235          PBRDPG      53E0          PLIOST    162C0:22C0
    ADDRFS      0150          PBFRNC      7A94          PBRTCD      5466          PLIPML    1621A:221A
    AEAPLA      4B7C          PBGETP      5422          PBSCLA      7297          PLIPTC    16006:2006
    AFBCDA      4AFC          PBHALT      8096          PBSETP      5393          PLIP      162F2:22F2
    ASCF26    1539C:339C      PBHDRB      76FF          PBSLJ       4EDB          PLREAD    160DD:20DD
    ASCE29    1535C:335C      PBIIPO      7758          PBSMAS      5368          PLTKOP    16000:2000
    ASTDAS      4A7C          PBILL       808F          PBSTOP      5578          PMCDRV      6CDE
    ASYERR    11729:3729      PBINSE      657D          PBSTPM      5433          PMCOIN      6C59
    ASYLEM      47B1          PBINTP      76CB          PBSTRI      579F          PMMLFH      6470
    ASYMSG      47A8          PBINTR      0100          PBSWLE      76C0          PMTISE      7664
    ATAPLA      497C          PBIOPO      7956          PBTICK      681B          PMWOLP      72C6
    A128EB    153DC:33DC      PBLCPB      5732          PBTIMA      69A2          PNAWAI      2198
    A7TO6P      4CBC          PBLCBT      6A21          PBTIMO      6B05          PNBMPS      6C37
    BEGINX      F671          PBLLEN      6B8F          PBTMRS      6751          PNBRDC      38FE
    BUFMAI      54DA          PBLLRM      68D0          PBTOAD      640F          PNCECN      4448
    CLEANU    16221:2221      PBLMAS      5350          PBTOAH      6320          PNCEFI      42AE
    EBCA12    1551C:351C      PBLNKD      7C01          PBTODE      6956          PNCNTL      372E
    E26ASC    1549C:349C      PBLNKU      7BA5          PBTOQU      68B3          PNCONF    15902:3902
    E29ASC    1541C:341C      PBLN00      1F9B          PBTOSR      68DD          PNDELE      2F7D
    FCSRCB    1559C:859C      PBLN01      1FC4          PBTWLE      4E36          PNDEQU      61D4
    GLOBL$      0DAD          PBLN02      52C8          PBUPAB      76D4          PNDIRA      60AF
    HASPMS      85BE          PBLN03      52E7          PBUPDA      66F3          PNDIRD      6134
    HSPTCB    155BE:35BE      PBLN04      5699          PBXFER      5988          PNDISA      311B
    HSR4IP      85CA          PBLN06      4CFC          PB100M      652F          PNDISC      2458
    HSR4IT    14F6B:2F6B      PBLN07      56A5          PB16AD      69E9          PNDLTC      2346
    HSR4TP    150F4:30F4      PBLN08      5306          PB18AD      59BD          PNENAB      3077
    IC        16010:2010      PBLN09      56B1          PB18BI      5A2A          PNFRCE      3F9A
    ISPOLD      1F5B          PBLN10      568D          PB18C0      5AD8          PNGTCB      2294
    JUMPS       0140          PBLN11      56C9          PGDSTA      4EE2          PNLCR       376D
    LIPSMA      848C          PBLN12      56D5          PGHALT    17D47:3D47      PNLINE      32CA
    LISTSR      5550          PBLN13      56E1          PGIVTC    15E5C:3E5C      PNLLCN      265C
    MAIN$       F659          PNLN14      56ED          PGSWIT      5184          PNLLIN      7ADA
    MODMST      55B2          PNLN15      56F9          PIAPPS      FC77          PNLLLI      7E66
    PBADJU      5C7C          PBLOAD      8045          PIBUF1      FB22          PNLLLO      7ECB
    PBAEXI      5385          PBLOST      76F5          PIBUF2      FF98          PNLLRC      7DEE
    PBAMAS      1FED          PBMAX       577F          PIDTBL      868F          PNLLRE      7D39
    PBBEXI      5377          PBMEMB      5705          PIFR1       F99A          PNLLSN      7D58
    PBBFAV      5C05          PBMIN       575F          PIGETA      F8A4          PNLLST      3644
    PBCALL      1F8D          PBMLIA      5077          PIINIT      FC37          PNLLTC      7F0C
    PBCLKI      6514          PBMON       8085          PILCBS      F849          PNLNBA      4148
    PBCLRP      53B5          PBOMAS      535D          PILINI      FD63          PNLNCN      27E2
    PBCLR       801D          PBPAGE      592B          PILMT       F6E8          PNLNST      391E
    PBCOIN      6C8E          PBPIPO      774D          PIMLIA      FE97          PNOVLD      4000
```

Figure 3-3. Sample MPLINK (Partial) Memory Map Sorted by Module Name

```
                                CYBER MINI CROSS SYSTEM - LINK EDITOR -

                                 ENTRY SYMBOL LIST - SORTED BY ENTRY NAME

*ENTRY**R/A**ADDRESS/VALUE**BIT S/L*     *ENTRY**R/A**ADDRESS/VALUE**BIT S/L*     *ENTRY**R/A**ADDRESS/VALUE**BIT S/L*

AACAPL   R       493C                     AEAUT1   R       4648                    AISP4C   R       4512
AACDAD   A       0000                     AEAUT2   R       464E                    AISP4E   R       44FA
AACDPT   A       0000                     AEAUT3   R       4654                    ALARM1   A       0001
AACORR   R       48BC                     AEBLS    R       45B4                    ALARM2   A       0002
AAEAPL   R       48FC                     AECHR1   R       466D                    ALARM3   A       0003
AAEBCD   R       48FC                     AECIN1   R       46DA                    ALCAPL   R       4C7C
AANREA   A       0007                     AECIN2   R       46DC                    ALCORA   R       4ABC
AAOUTP   A       0003                     AECIN3   R       46E7                    ALEAPL   R       4BBC
AAREAD   A       0004                     AECIN4   R       46E9                    ALEBCD   R       483C
AASBAP   R       4A3C                     AECKMD   R       45AB                    ASASCI   A       0022
AASTAP   R       49FC                     AECODI   R       46C0                    ASAUTO   A       001B
ABAPLA   R       49BC                     AECOD2   R       46A0                    ASCE26   R    1539C:339C
ACAPL    R    116FD:36ED                  AECSE    R    116D7:36D7                 ASCE29   R    1535C:335C
ACARTO   A       0C78                     AECSLL   R       475F                    ASCINT   R       176C
ACAUTO   A       0001                     AECSL1   R       477A                    ASDISC   A       0003
ACCAPL   A       0004                     AECSL2   R       477C                    ASSLL    A       0004
ACCASE   R    11887:3687                  AEEIN1   R       46F5                    ASSOL    A       0001
ACCORR   A       0003                     AEEIN2   R       46F7                    ASXPT    A       0002
ACDELM   A       0002                     AEEIN3   R       4702                    ASXSOI   A       0005
ACEAPL   A       0002                     AEEIN4   R       4704                    ASYNCE   R       153A
ACEBCD   A       0001                     AEELL    R       4505                    AS2741   A       0020
ACELL    A       0000                     AEELT1   R       45CE                    ATAPLA   R       497C
ACEPL    A       0001                     AEELT3   R       45E0                    ATELL    A       0000        F:7
ACKMSG   R       858E                     AEEPL    R       45F1                    ATEPL    A       0000        7:7
ACKPTP   R       1068                     AEEPT1   R       45F8                    ATPDI2   R    1131B:331B
ACLIH2   A       0D00                     AEESL1   R       4780                    ATPDI5   R    1133F:333F
ACLIM1   A       0A00                     AEESL2   R       4782                    ATPD16   R    11363:3363
ACPBLI   A       0001                     AEINPT   R       4583                    ATPDIB   R    11387:3387
ACPROB   A       0002                     AEIN1    R       458C                    ATPPR1   R    112D3:32D3
ACPCLR   A       000C                     AEMBT    R       455F                    ATPPR4   R    112F2:32F2
ACPEVE   A       0000                     AESEDI   R       473C                    AUCAPL   R       4C3C
ACPICS   A       0C50                     AESLL    R       4570                    AUCORA   R       4A7C
ACPTOW   A       0060                     AES110   R       4658                    AUEAPL   R       4B7C
ACPLR1   R       1C0D                     AES150   R       4668                    AUEBCD   R       4AFC
ACPOBL   A       0058                     AES151   R       4662                    AVASCE   R    11729:3729
ACPOMA   A       006C                     AES300   R       4674                    AVASCP   R       1E5D
ACPONS   A       0048                     AES301   R       467A                    AVB7TO   R       1E74
ACPRMA   A       0010                     AEXBLS   R       4635                    AVCNTR   R       1E3D
ACRITT   A       000A                     AEXDLM   R       463A                    AVCORE   R    1173A:373A
ADDRES   R       0150                     AEXDTA   R       462A                    AVCORR   R       1E5E
ADDRLC   R       015F                     AEXPTO   R       462F                    AVCRLF   R       1E52
ADDRSU   R       0160                     AEXSOI   R       4604                    AVCRMS   R       1E50
ADEADT   A       0014                     AE4XDL   R       47A7                    AVEBCD   R       1E5F
ADST1    A       0001                     AE4XIN   R       4792                    AVEBCE   R    1174A:374A
ADST2    A       0002                     AIDLET   A       0003                    AVEOLS   R       1E29
AEABLS   R       474A                     AIDLE    A       0001                    AVEOLT   R       1E2D
AEAPL    R    11709:3709                  AINPLB   A       000E                    AVEOTM   R       47AB
AEASCI   R       46A7                     AINPSB   A       000D                    AVEOTP   R       1E58
AEATTN   R       4769                     AISPT1   R       44D5                    AVINTA   R       1E60
AEATT1   R       4775                     AISPT6   R       452A                    AVIS4C   R       1E54
```

LEGEND:

ENTRY	Name of the entry symbol. It is up to six letters and numbers long. If a local entry, the slash (/) is omitted.
R/A	Entry type; R=relocatable, A=absolute, L=local.
ADDRESS/VALUE	Address of the entry or its value. Addresses are a displacement from the first word of the file. If address is 64K (65,536) or less, true address appears. If address is above 64K, two addresses appear as shown: true:paged. For example, 10000:2000 has a true address of 10000_{16} and a page address of 2000_{16}.
BIT S/L	Field start position/field length. Both of these values are given in bits. For start position, bit 15 is the leftmost bit, bit 0 is the rightmost. Both start and length have a range 0-F_{16}.

Figure 3-4. Sample MPLINK Memory Map Sorted by Entry Name (Partial)

EXECUTING MPLINK

MPLINK is executed by attaching the MPLINK permanent file (local file name is MPLINK), and then executing the file name call statement MPLINK.

Four optional parameters are available with the MPLINK file name call statement:

```
MPLINK[D=infile,R=outfile,CSET=cset,X=1]
```

where:

D is the logical file that presents the input directives to MPLINK (default name is INPUT).

R is the logical file that receives the listings (default name is OUTPUT).

CSET is the host display code set to be used.

CSET=63 selects the CDC 63-character display code set (this is the default value).

CSET=64 selects the CDC 64-character display code set.

X=1 aborts the job if errors occur. If X=1 is omitted (default action) the job will not abort if errors occur.

Appendix G gives examples of executing MPLINK.

MPLINK DIRECTIVES

All MPLINK processing is controlled by the MPLINK directives entered in the input directives file. The general format of a directive is:

```
*dirname,
param1,...,parami[paramj...paramn] comment
```

where:

An asterisk (*) indicates the beginning of an MPLINK directive.

dirname is the name of the directive.

param1 is a parameter.

The first parameter is separated from the directive name by a comma. Additional parameters are separated from one another by commas. Some parameters are optional; optional parameters are enclosed in brackets [,param]. Parameter types are discussed below. Specific parameters are defined in the descriptions of individual directives. A period (.) terminates the command portion of the MPLINK directive. Comments start after the period and include all characters until the *, which starts the next directive.

MPLINK DIRECTIVE PARAMETERS

An MPLINK parameter is either:

A name of:

A module

An entry point in a module

A synonym equating an entry point to an external

The system being built

An overlay area

A local variable

An overlay identifier

A memory address

Names

A parameter name must begin with a letter. It can contain any number of following letters or numbers; however, MPLINK uses only the first six characters. Therefore, all unique names must differ in their first six characters. Identical names lead to an MPLINK error.

For the purpose of parameter names, the dollar sign ($) is considered to be a number.

Overlay Identifiers

An overlay identifier always consists of two letters. If an identifier is not assigned in a directive statement, MPLINK generates its own overlay identifiers.

Memory Addresses

The allowable forms of memory addressing were discussed earlier in this section.

SUMMARY OF MPLINK DIRECTIVES

Table 3-2 is an alphabetical summary of the MPLINK directives. Detailed descriptions of the MPLINK directives follow.

*L - Specifies Modules to be Linked

This directive links modules. The standard form of the link directive is:

```
*L,mod,addr.
```

TABLE 3-2. SUMMARY OF MPLINK DIRECTIVES

Name	Function
*CB	Defines the upper boundary for linking programs.
*COM	Defines the blank common area used by macroassembler modules.
*COR	Defines the size of the combined main and extended NPU memories.
*DAT	Defines a common data area for PASCAL global variables.
*DMP	Produces the hexadecimal listing of the memory image load file.
*DSTK	Defines a stack area to be used for PASCAL reentrant/recursive procedures.
*DVAR	Defines a dynamic variable area for use with PASCAL variables.
*END	Last statement of the input directives file; ends the file.
*ENT	Associates a memory address with an entry point name.
*L	Links one or more modules, or links all the unlinked modules on a library file.
*LIB	Defines the library file used to resolve unsatisfied externals.
*LL	Specifies a lower limit memory address; modules cannot be located below this address.
*OVLY	Identifies and establishes the limits of an overlay area.
*RL	Links reverse-loaded modules with the module ending address specified in the directive.
*SYN	Equates an arbitrary name with an entry point name or a defined module name.
*SYSID	Specifies the name for the build.
*UL	Specifies an upper limit memory address; modules cannot be located above this address.
*VE	Assigns a variable expression value to a local variable.

This form causes MPLINK to locate object code module mod at starting address addr. As it is located, other linking operations also occur: addresses are made absolute and externals are resolved.

There are five alternate ways of writing a link directive:

*L,mod.

Links the object code module (mod) starting at the word following the last word of the module most recently linked by MPLINK. Note that if trailing parameters are omitted, their delimiting commas can also be omitted.

*L,,addr.

Links all object code modules in the object code input file except those which are expressly linked by other *L directives. The modules are linked in the order in which they occur on the input object code file. The first module encountered starts linking at the specified address (addr). Note that the delimiting commas for the omitted mod parameter must be retained.

*L,mod1-mod2,addr.

Links all the modules starting with mod1 extending through mod2 on the object code input file. The first module (mod1) is located at the specified address. The other modules are located in order following that module. If either mod1 or mod2 cannot be located, an error occurs.

*L,mod1-mod2.

Same as the previous form, except mod1 is located at the address following the last word of the module previously linked.

*L.

Links all object code modules in the input object code file except those which are expressly linked by other linking (*L,parameters, or *RL) directives. The modules are linked in the order in which they occur on the input object code file. The first module encountered starts at the word following the last word of the module most recently linked.

*RL - Specifies Modules to be Reverse Linked

The reverse linking directive locates a module so that the last word of the module is placed in the specified address. There are two alternate forms for the directive:

*RL,mod,addr.

Links the module so that the last word is placed in addr.

*RL,mod1-mod2,addr.

Links a series of modules on the object code input file, starting with mod1 and extending through mod2. The last word of mod2 is located at addr. The module ahead of mod2 is linked next, with its last word immediately preceding the first word of mod2. Other modules are linked similarly until all modules in the sequence (including mod1) are linked. If either mod1 or mod2 cannot be found on the object code input file, an error occurs.

*CB - Defines Linking Boundary

This boundary directive prohibits linking programs above the specified address. The format of the directive is:

```
*CB,addr.
```

More than one *CB directive can be used in the directives file. If a second (or subsequent) *CB,addr is used, the second address becomes the new boundary value. If a *CB,0 directive is used, the boundary is removed.

Programs that are prevented from being linked by the *CB,addr directive are subsequently linked by the link all (*L) directive unless an *L has already been used or there is no *L directive in the input directives file. In either of these cases, the unlinked modules are linked following the last program linking.

*LL - Defines a Lower Limit for Linked Modules

This directive prohibits any module from being located below a given address in memory. If a module's starting address is less than the specified address, processing is halted and a fatal error message is generated. The format of the directive is:

```
*LL,addr.
```

Since *LL is positional (that is, it applies only to linking directives that follow it in the directives file), more than one *LL directive can be included in the file. In this case, if *LL,addr2 follows *LL,addr1, the specified lower threshold is changed to addr2 for the remaining directives. To cancel a lower limit, the user enters the directive:

```
*LL,0.
```

*UL - Defines an Upper Limit for Linked Modules

This directive prohibits any part of any module from being located above a given address in memory. If a module's ending address is greater than the specified address, processing is halted and a fatal error message is generated. The format of the directive is:

```
*UL,addr.
```

Since *UL is positional (that is, it applies only to linking directives that follow it in the directives file), more than one UL directive can be included in the file. In this case, if *UL,addr2 follows *UL,addr1, the specified upper limit is changed to addr2 for the remaining directives. To cancel an upper limit, the user enters the directive:

```
*UL,0.
```

*SYSID - Identifies the System Load File

This directive establishes a user-supplied name for the load file. The system name and the optional text are placed in the memory resident header record of the load file.

The format of the directive is:

```
*SYSID,name[,text].
```

If the SYSID is not specified in a directive, the load file will not have a memory resident header record. In such a case, if an *L directive includes a module with the name LOADER, that module is placed at the head of the load file.

The optional text is any string of letters or numbers up to a total of 48 characters.

*OVLY - Specifies Overlay Areas and the Modules in an Overlay

This directive has two purposes:

It defines the limits of an overlay area.

It specifies the modules that are to be a part of the overlay. An overlay consists of all modules defined by *L directives which follow one overlay directive and which precede the next overlay directive or the *END directive.

The format of the *OVLY directive is:

```
*OVLY,name,ovlyid,addrb,addre.
```

where name is the overlay name (note that the overlay name is six letters or numbers and has the attributes of a defined entry point); ovlyid is the two-letter overlay identifier; addrb is the beginning address of the overlay area; and addre is the ending address of the overlay area.

In using the *OVLY directive, the user should observe these rules:

A memory image load module file can have no more than ten overlay areas.

An *L directive that specifies the overlay area name as its starting address designates its first module as the start of that overlay.

Overlay areas cannot overlap.

*ENT - Defines Entry Points

This directive assigns a four-component memory address to a user-assigned name. The name is used to resolve like-named external references. The name must not be the same as an entry point in an already linked module.

The format of the directive is:

```
*ENT,name,addr.
```

*SYN - Defines External Synonyms

This directive equates an arbitrary name to a declared entry point name or defined module name. The equated name is to be used for resolving external references.

The format of the directive is:

```
    *SYN,namel,name2.
```

where namel is the name of a declared entry point or a defined module; and name2 is the name to be associated with namel. At every occurrence of namel in the object code, name2 is substituted.

*COR - Defines NPU Memory Size

This directive defines the size of the NPU memory for which the load file is being generated. The format of the directive is:

```
    *COR,addr.
```

where addr specifies one of the four legal CCP or CCI memory sizes. These are:

$FFFF	65536 words
$13FFF	81920 words
$17FFF	93302 words
$1FFFF	131072 words

If addr is omitted, a default value of $FFFF is used. This is equivalent to an address specification of $FFFF:$1F:0; that is, $FFFF_{16}$ is the last address of memory, memory page 31 holds that address, and address register set 0 is used for that range of addresses.

*LIB - Specifies Library File

This directive specifies the library file that MPLINK uses to resolve unsatisfied externals during the linking process. The format of the directive is:

```
    *LIB.
```

The library file is always presented to MPLINK with the local file name of NEWLIB.

*VE - Equates a Variable to an Expression

This directive assigns the value of the specified expression to the named variable. The format of the directive is:

```
    *VE,nam:=exp.
```

where nam creates a local variable of that name; exp can have any of the following formats:

```
    naml

    constant

    naml+constant

    naml+nam2

    naml-constant

    naml-nam2
```

naml and nam2 can be local variables or entry points that are absolute or have been previously made absolute.

*DSTK - Allocates a Stack Area for Recursive/Reentrant PASCAL Programs

This directive allocates the area that is used by all reentrant and recursive PASCAL programs to save processing parameters when a call is made to a program that has not completely finished processing. The format of the directive is:

```
    *DSTK,addrb,addre.
```

where addrb is the starting address of the area and addre is the ending address.

The programmer can choose a starting address in any part of main memory that is not to be used for other purposes (buffers, programs, globals, or other reserved areas).

*DVAR - Allocates a Dynamic Variable Area for PASCAL Programs

This directive allocates the dynamic variable area used by all PASCAL programs. The area is accessed by the PASCAL standard procedure NEW. NEW is a variable space allocation routine; it automatically allocates space for a variable based on the type of variable (see the CYBER Cross System PASCAL Compiler Reference Manual).

The format of the directive is:

```
    *DVAR,addrb,addre.
```

where addrb is the starting address of the area and addre is the ending address.

The programmer can choose a starting address in any part of main memory that is not to be used for other purposes (buffers, programs, globals, or other reserved areas).

*COM - Defines a Blank Common Area for Macro Assembler Programs

This directive allocates a blank common area that is referenced by macro assembler modules. The format of the directive is:

```
    *COM,addrb,addre.
```

where addrb is the starting address of the area and addre is the ending address.

*DAT - Defines the Labeled Common Area

This directive defines the labeled common area. PASCAL global variables are assigned to this area, and macro assembler programs can reference this area. The format of the directive is:

```
    *DAT,addrb,addre.
```

where addrb is the starting address of the area and addre is the ending address.

The programmer can choose a starting address in any part of main memory that is not to be used for other purposes (buffers, programs, globals, or other reserved areas).

PASCAL global variables are defined in an object code module named GLOBL$. The appearance of GLOBL$ in an *L directive takes precedence over a *DAT directive.

*DMP - Generates the Memory Image Load Module File Hexadecimal Listing

This directive causes MPLINK to generate an output file consisting of a hexadecimal dump of the memory image load module file. The listing is sent to the file named OUTPUT. The format of the directive is:

```
    *DMP.
```

*END - Ends MPLINK Directive Input File

This directive ends the MPLINK input directives file. It also specifies the address of the first instruction to be executed after the load file is downline-loaded into the NPU. The format of the directive is:

```
    *END,addr.
```

where addr is a hexadecimal number or an entry point name. The default for address is location 0.

MPLINK ERROR MESSAGES

If an error occurs during an MPLINK run, an error message is delivered to the output file. The messages are preceded by a leading-up arrow. If the error is a recognized syntax error, the up-arrow is followed by the character that was being processed when the error occurred. Table B-2 in appendix B lists the MPLINK utility error messages, the significance of the message, and the action that the user should take in response to the message.

EDIT UTILITY

INTRODUCTION

The edit utility is used to initialize values in specified variables of the Communications Control Program (CCP) or the Communications Control INTERCOM (CCI) modules which have been made absolute.

The MPEDIT utility requires three inputs:

The noninitialized memory image load module file (APSOLMP) output of MPLINK. This is the file to be initialized.

The symbol table file (SYMTAB) output of MPLINK. This is used to locate the modules to be initialized.

The MPEDIT directives that control the initialization. This extensive file of directives is called an MPEDIT program throughout this section. The program is similar in format to a CYBER Cross PASCAL program; that is, it consists of a declaration/definition part followed by a group of executed statements. The user should be familiar with PASCAL compiler requirements and syntax (see the CYBER Cross System PASCAL Compiler Reference Manual).

If a standard CCP or CCI build procedure is used, the MPEDIT program is available from the CCP or CCI program library. The program is generated by the build procedures from the release tapes (see figure 1-1 in section 1).

The output of MPEDIT is an initialized version of the memory image load module file. This file can be converted to a downline-load file for an NPU.

MPEDIT also supplies several optional output listings.

The MPEDIT utility section contains the following subsections:

A description of the input files required

A description of the output files: the required memory image load module file, and the optional listings

The method of executing the MPEDIT program

The structure of the MPEDIT program

Error message discussion

Addressing for the MPEDIT utility follows the rules specified in the MPLINK utility section.

MPEDIT INPUTS

MPEDIT requires two files produced by MPLINK:

The memory image load module file (ABSOLMP). The file structure is shown in appendix D.

The symbol table file (SYMTAB). This file contains every entry symbol defined during MPLINK, together with the symbol's absolute location in the memory image file.

MPEDIT also requires the MPEDIT program. The syntax of this program is described in detail in the remainder of this section. A sample of parts of an MPEDIT program is given in appendix H.

MPEDIT OUTPUTS

The MPEDIT utility produces two standard outputs:

A memory image load module file with the specified variables initialized.

NOTE

A variable can take the form of a declared constant, a variable, or a field within an array. Fields in arrays are restricted to 16 bits (one contiguous NPU word) in length.

A listing of the MPEDIT input program. Any syntax errors encountered in this program are indicated on this listing.

Four optional outputs are also supplied:

A specially formatted memory image load module file, tailored for downline loading on an NPU. The format of this load file is given in appendix E. This is not the downline-load file for CCP or CCI, but is a required input to generate that file. That load file itself is generated by the CCP or CCI installation procedures.

A trace listing of the MPEDIT assignments that were made.

A listing of the symbol table (SYMTAB), which includes local symbols that were introduced during the MPEDIT phase.

A hexadecimal listing of the memory image load module.

EXECUTING MPEDIT

MPEDIT is executed by attaching the MPEDIT permanent file and then executing the name call statement MPEDIT (see the appropriate NOS or NOS/BE reference manual).

These optional parameters are available with the MPEDIT call statement:

```
MPEDIT[D=infile,R=outfile,CSET=cset,X=1]
```

where:

D is the local file which presents the input directives to MPEDIT. The default is INPUT.

R is the local file that receives the listings. The default is OUTPUT.

CSET is the host display code set to be used. CSET=63 selects the CDC 63-character display code set; this is the default value. CSET=64 selects the CDC 64-character display code set.

X=1 aborts the job if errors occur. If X=1 is omitted, the default action is not to abort the job, despite errors.

Appendix H shows examples of executing MPEDIT.

Note that the user must rewind the ABSOLMP and SYMTAB files before entering this utility; MPEDIT does not rewind the files before using them.

MPEDIT STATEMENT FORMAT

MPEDIT statements are similar to PASCAL statements (see the CYBER Cross Compiler Reference Manual) with some restrictions and extensions. A typical extension allows an expression on the left side of a VALUE statement. The evaluated expression specifies the address that receives the assigned value. Comments are permitted as in PASCAL statements. Appendix H shows selected sections of an MPEDIT program.

As in a PASCAL program, the MPEDIT utility has two parts, which occur in the order given:

A definition/declaration section

An assignment (initialization) section

The utility ends with a terminator.

The sections of the MPEDIT utility are shown in figure 4-1. The MPEDIT flow is shown in figure 4-2.

The definition/declaration section consists of three parts, which must occur in the order given:

A constant definition part

A variable definition part

An array definition part

The assignment section consists of one or more composite statements. One composite assignment statement is required for the memory resident portion of CCP or CCI; one additional composite assignment statement is required for each overlay to be edited. If overlay statements are present, they must precede the memory resident statement. The memory resident statement must be present even if it is an empty statement (empty statements are defined later).

PROGRAM STRUCTURE

```
CONST                                               (1)
         Constant Definition Part

VAR                                                 (1)
         Variable Declaration Part

ARRAY                                               (1)
         Array Declaration Part

OVERLAY      overlay identifier                     (2)
         BEGIN
                  Assignment Section
         END;

BEGIN                                               (3)
         Assignment Section
END.
```

(1) Optional.

(2) Optional composite statement; can be repeated for every overlay that requires editing up to the maximum number defined for the link edit.

(3) Required composite statement; Memory Resident Partition.

Figure 4-1. MPEDIT Program Format

The MPEDIT terminator is a period (.) immediately following the END statement of the memory resident assignment statement.

MPEDIT SYNTAX

The MPEDIT utility uses the following syntax elements:

Keywords that designate the part of the program or operation to be performed by the assignment section

Reserved symbols used to order the optional outputs

Local symbols used for equating constants locally, or specifying a local variable

External symbols in MPEDIT that always have an array attribute

Literals

Address functions

Expressions

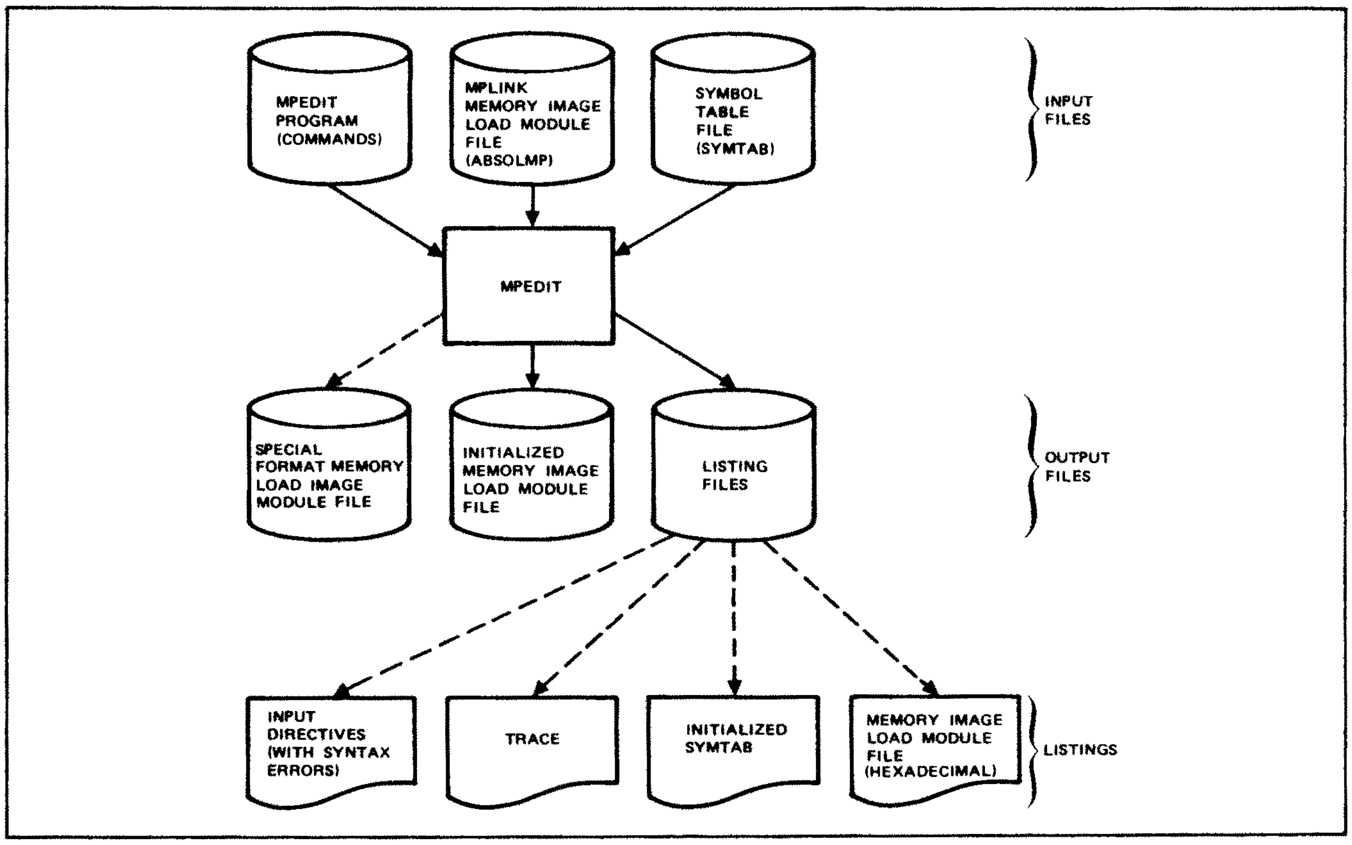

MPEDIT PROGRAM (COMMANDS)
MPLINK MEMORY IMAGE LOAD MODULE FILE (ABSOLMP)
SYMBOL TABLE FILE (SYMTAB)
INPUT FILES
MPEDIT
SPECIAL FORMAT MEMORY LOAD IMAGE MODULE FILE
INITIALIZED MEMORY IMAGE LOAD MODULE FILE
LISTING FILES
OUTPUT FILES
INPUT DIRECTIVES (WITH SYNTAX ERRORS)
TRACE
INITIALIZED SYMTAB
MEMORY IMAGE LOAD MODULE FILE (HEXADECIMAL)
LISTINGS

Figure 4-2. MPEDIT Program Flow

KEYWORDS

The following keywords are reserved for MPEDIT controls:

ARRAY
BEGIN
CHAR
CONST
DIV
DO
END
FOR
MOD
OF
OVERLAY
TO
VAR

These control words have the same definitions in the PASCAL compiler.

RESERVED WORDS

MPEDIT assigns specific output option request meanings to the following reserved symbols:

/DMP$
/ESL$
/NAM$
/TRACE

MPEDIT assigns specific address meanings to the following reserved symbols:

/ENTRY
/LENGTH
/PGDISP
/PGNUM
/PGREG
/PGSET
/START
/VFD

These symbols must be used only to perform the desired MPEDIT functions. The functions are discussed later in this section.

LOCAL SYMBOLS

Local symbols are used to equate a constant in the constant definition part of the program or to declare a local variable.

A local symbol is defined by a slash (/) followed by one to six letters and/or digits. The first character must be a letter. The dollar sign ($) is considered to be a digit. The following are valid local symbols:

/ABCDEF
/A6
/MAIN$4
/I

A local symbol can have more than six letters or digits. MPEDIT, however, truncates the symbol at the seventh character and discards that character and all that follow. The user cannot, therefore, define two local symbols such as /ABCDEFG and /ABCDEFH. MPEDIT treats both of these as /ABCDEF.

EXTERNAL SYMBOLS

External symbols are used during array processing. The symbols refer to arrays in the SYMTAB load file produced by MPLINK. An external symbol consists of one to six letters and/or digits. The first character must be a letter. The dollar sign ($) is considered to be a digit. An external symbol cannot be one of the keywords defined earlier. The following are valid external symbols:

```
A36F
MAIN$
GLOBL$
UPTOPARAM (treated as UTOPAR)
```

An external symbol can have more than six letters or digits. MPEDIT, however, truncates the symbol at the seventh character and discards that character and all that follow. The user cannot, therefore, define two external symbols such as /ABCDEFG and /ABCDEFH. MPEDIT treats both of these as /ABCDEF.

External symbols can be qualified by other external symbols. To do this, the user separates the external symbols by a period. A single external symbol can be progressively qualified by additional external symbols, as shown in the examples:

```
A36F.FIELD

GLOBL$.RECORD.FIELD
```

where:

The first external symbol specifies a location on the memory image file.

Intermediate external symbols (if any) specify a displacement from the previous location (for instance, the start of a record in the global variables).

The final external symbol specifies a field as:

A displacement from the start of the previous external symbol

A start bit position for a field

A field length (in bits)

This method of qualification is identical to that used in the PASCAL syntax.

LITERALS

A literal can be a decimal number (a sequence of decimal digits), or a hexadecimal number (a sequence of hexadecimal digits preceded by a $). Internally, literals are represented as 16-bit quantities. Larger quantities are illegal.

Signed literals are allowed (the leftmost bit is a sign bit). A negative literal forces the complement of the 16-bit quantity. If a literal is represented by less than 16 bits, the unused left bits are packed with binary zeros. Examples of legal literals are:

```
123
$147
-$F
```

ADDRESS FUNCTIONS

MPEDIT provides nine functions that can be used within operand or address expressions to generate an address. The functions are executed by a reserved word in the form /xxxxx. Five of these functions are also available with MPLINK (see address functions in MPLINK):

```
/OVID
/PGDISP
/PGNUM
/PGREG
/PGSET
```

In addition, MPEDIT defines four more address functions:

```
/ENTRY
/LENGTH
/START
/VFD
```

As in the MPLINK case, the function takes the format:

```
/xxxxx(name)
```

/START - Field Start Address Function

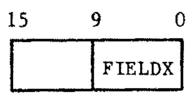
The /START function has the format:

```
/START(external)
```

The function returns the start position in bits (range 15 to 0) of a field relative to the start of a variable word. For example:

```
/START(FIELDX)
```

generates a bit position of 9 as the start of FIELDX:



/LENGTH - Field Length Address Function

The /LENGTH function has the format:

```
/LENGTH(external)
```

The function returns the value of the field length (in bits) minus 1. A single bit field has a value of zero; a full word field has a value of 15. PASCAL fields cannot exceed word length, nor can a field start in one word and overflow into the next.

Example 1: The length of FIELDX in the example above is requested with /LENGTH(FIELDX). The address function would return a value of 10.

Example 2: The terminal class field (BSTCLASS) start position and length in the base terminal control block (TCB) descriptor table are found by using the following MPEDIT statements:

```
DGTCBFDT[5].DDFSTRT  :=/START(BSTCLASS);

DGTCBFDT[5].DDFLNTH  :=/LENGTH(BSTCLASS);
```

/ENTRY - Entry Point Address Function

The /ENTRY function has the format:

```
/ENTRY(external)
```

This function accepts a module name as a parameter and generates the address of the module's associated entry point.

Example: The entry point and the page number of the service module in CCP are located by using the following MPEDIT statements that include address functions:

```
BYWLCB [BOSMWL] .BYPRADDR :=/ENTRY(PNSWML);

BYWLCB [BOSMWL] .BYPAGE   :=/PGNUM(PNSWML);
```

/VFD - Variable Field Definition Address Function

The format of the /VFD specification is:

```
/VFD(addr/disp,fldstrt,fldlngth)
```

where:

addr/disp defines the absolute address of the word holding the field in the memory image load file.

fldstrt defines the start bit position of the field within the word (range 15 to 0).

fldlngth defines the length of the field (in bits) minus 1 (range 0 to 15).

The /VFD function uses three parameters (address/displacement, field start, and field length) to generate the location and length of a field in the memory image load module file.

The address expression A, where A is an external, is equivalent to the expression:

```
/VFD(A,/START(A),/LENGTH(A))
```

or more completely:

```
/VFD(A:PGREG(A):/PGSET(A),/START(A),/LENGTH(A))
```

MPEDIT EXPRESSIONS

Two types of expressions are used:

Operand expressions

Address expressions

Operand Expressions

An operand expression produces a single 16-bit binary value. An expression is a valid combination of:

Constants (which are interpreted as 16-bit integers)

Local variables

Functions

Unqualified external symbols

These are joined together by arithmetic operators (+, -, *, DIV, and MOD) and are grouped within parentheses to specify the order in which the operations are to be performed. An evaluated external symbol is represented by its address/displacement value. Qualified address values can be accessed using the address evaluation functions given in MPLINK and MPEDIT, above.

In an operand expression, an external symbol cannot be subscripted, even if it is declared in an array. The NPU performs all arithmetic in one's complement mode so that results are unique.

Examples of operand expressions are:

```
GLOBAL$

(/PGREG(MOD)+$F21)DIV/START(GLOBL$)
```

Address Expressions

An address expression has one of two forms:

A /VFD address function call.

A single external symbol. The symbol can be qualified. If this form is used, symbols that are declared in arrays must be properly subscripted; that is, the subscript expressions must be operand expressions with values that fall within the expected range.

Example 1:

A

This is an address expression unless A was declared in an array. In that case, the lack of subscripts indicates that it is an operand expression. It could also be an operand expression if its usage forced that conclusion; that is, it is on the righthand side of an assignment statement.

Example 2:

```
    A[2]
    /VFD(MAIN$,0,0)
```

Example 3:

```
    ARRAY A[1...5,1...5] OF 24;
          C[1...10] OF CHAR;

    VAR/I; /J
       .
       .
       .
    A[/I,(/START(Q)-2)*/J].B.C[3]'address
    expression'
```

MPEDIT PROGRAM STRUCTURE

An MPEDIT program consists of four parts:

Constant declaration part

Variable declaration part

Array declaration part

Assignment section

The first three parts are optional, but the assignment section is required.

CONSTANT DECLARATION PART

The first part of an MPEDIT program contains constant declarations; this section is optional. Constant declarations allow programmers to create synonyms for literals. A local symbol defined as a constant behaves as a true constant; its appearance is legitimate wherever a literal is expected.

If a constant declaration part is present, it is preceded by the keyword CONST. The complete list of constant declarations must follow that word.

Each declaration has the following format:

```
    /symbol=expression;
```

That is, the declaration consists of a local symbol, an equals sign, and a literal, or a previously defined constant. The declaration terminates with a semicolon. Literal and previously defined constants can be signed.

A constant value can be defined as an expression that itself is a mixture of constants, previously declared local constants, and entry symbols that appear in SYMTAB.

Examples are given in figure 4-3.

VARIABLE DECLARATION PART

The next part of an MPEDIT program contains variable declarations. This section, also optional, allows programmers to create local symbols for local variables. These symbols exist only during the MPEDIT phase. All such variables are 16-bit quantities that can be used in one's complement arithmetic.

If the variable declaration part is present, it is preceded by the keyword VAR. That word is followed by the complete list of local variable declarations.

Each declaration has the following format:

```
    /symbol;
```

That is, the declaration consists of a local symbol followed by a semicolon terminator.

Examples of variable declarations are given in figure 4-3.

ARRAY DECLARATION PART

The next part of an MPEDIT program contains array declarations. This section, also optional, allows programmers to create external symbols as arrays so that elements can be referenced by an index. Any external symbol to be indexed must be declared as an array.

If the array declaration part is present, it is preceded by the keyword ARRAY. That word is followed by the complete list of array declarations.

A declaration can have either of two formats:

```
    name[index] OF number;

    name[index] OF CHAR;
```

In each case, the name is an external symbol, the index is a range of numbers in PASCAL notation (number..number), and the declaration is terminated with a semicolon. Note that number itself can be an expression. If the CHAR format is used, the array corresponds to a PASCAL packed array; that is, there are two characters packed per NPU word.

Examples of array declarations are shown in figure 4-3.

ASSIGNMENT SECTION

There are two general types of assignment sections: resident assignments and overlay assignments. All overlay assignments must precede the resident assignment section. The two types are identical except that each overlay assignment section begins with:

```
    OVERLAY overlay identifier
```

An assignment section consists of a single composite statement that is delimited by the keywords BEGIN and END. The composite statement consists of zero or more statements that direct the MPEDIT actions to be performed. There are five types of assignment statements:

Local

Address

FOR loop

Composite

Empty

```
CONSTANT DECLARATIONS

CONST
   /TRUE    = 1;                                      Constants as decimal numbers
   /FALSE   = 0;
   /COUPLER = $0000;                                  Constants as hexadecimal numbers
   /PORT01  = $0100;
   /PORT02  = $0200;
   /BOS1    = 1;                                      Constants as previously defined constants
   /BOS16   = /BOS1;
   /BFLCDO  = /BUFSZO*2 - /J1LSTPAD;                  Constants as arithmetic expressions
   /DBFSZE  = /BECTLBK + (3*/SIZBECTLBK);

VARIABLE DECLARATIONS

VAR
   /I                                                 General loop index
   /I3                                                General use variable
   /P                                                 Work pointer for program
   /IDTBL                                             Table work pointer
   /BZOWNER                                           Local variables
   /BZLNSPD
   /BSCN
   /BSPGWAIT

ARRAY DECLARATIONS

ARRAY

   JZOPSBASE [BOCHWL..BODUMMY] OF 2;                  Sequence of elements defined by symbols;
                                                        numerically defined size
   DBPFCTBLE [1..DBLAST] OF 2;                        Sequence of elements defined by symbols;
                                                        constant defined size
   CGTCBS [0..C4TCM1] OF /SIZTCB                      Combination of both of the above
   VATCBAT [1..40] OF 3;                                sequence of elements defined by
                                                        numbers; numerically defined size
   JGTESTABLE  /FALSE../TRUE,/FALSE../TRUE,/FALSE../TRUE] OF 1;   Sequence of element sets defined by
                                                        symbols; numerically defined size
   NAMEN [1..20] OF CHAR;                             Sequence of elements defined by numbers;
                                                        elements packed two per NPU word.
```

Figure 4-3. Examples of MPEDIT Constant, Variable, and Array Declarations

Each statement (except the last) is terminated by a semicolon.

An MPEDIT program must include an assignment section for the memory resident programs, even if that section consists of only one empty statement.

Selected portions of an assignment section for a CCP MPEDIT program are given in appendix H.

Local Assignment Statement

A local assignment statement has the following format:

    local variable:=operand expression

where := is the assignment operator.

MPEDIT evaluates the operand expression to find the value and places that value in the named local variable. Examples of local address assignment statements are:

```
VAR /I; /J; /K; /L;
    .
    .
    .
/I:=/I+1;  /J:=0;  /K:=/LENGTH(X);
/L:=/VALUE(A.B)+C;
```

Address Assignment Statement

An address assignment statement has the format:

    address expression:=operand expression

An address expression can take the form of an operand expression (that is, the operand expression can appear on the lefthand-side of the assignment operator). In this case, a /VFD with full-word attributes is implied.

Semantically, MPEDIT evaluates the righthand-side operand expression and replaces the value in the memory image location specified on the lefthand side with this new value.

If a 16-bit value is assigned to a smaller than 16-bit field, the higher order bits are truncated.

As mentioned above, an address assignment statement can have an operand expression on the lefthand side. For example:

```
/I+1:=0 is interpreted as /VFD(/I+1,15,15) :=0
```

MPEDIT is instructed to zero the full 16-bit word that appears at location /I+1. Similarly, 1:=0 would zero the full word at memory image location 1.

Example:

```
VAR/I;
  .
  .
  .
/I:=0
```

This sets the local variable /I to zero. To zero the word at memory location /I, the lefthand side of the assignment must be forced to look like an expression. This could be done in any of the following three ways:

```
+/I:=0

(/I):=0

/VFD(/I,15,15):=0
```

FOR Assignment Statement

The format of the FOR assignment statement is:

```
FOR control variable:=initial operand expression
    TO final operand expression
    DO statement
```

The FOR ... TO statement assigns values for the control variable in increasing order. The control variable must be a local variable.

The FOR assignment statement in MPEDIT is entirely analogous to the FOR ... TO statement in PASCAL. The statement causes the indicated statement to be repeated, while a progression of values is assigned to a control variable.

In the following example, the FOR statement causes MPEDIT to set 256 successive locations, beginning at the external symbol GLOBL$, with the value of the preceding memory location's address:

```
VAR /I
  .
  .
  .
FOR /I:=GLOBL$
    TO GLOBL$+$FF
    DO   (/I):=/I-1
```

Composite Assignment Statement

The format of the composite assignment statement is:

```
BEGIN
    statement;
    statement;
      .
      .
      .
    statement
END.
```

A composite statement is a sequence of statements (which can include embedded composite statements) that are to be executed in the order specified. A composite statement is delimited by BEGIN and END.

A composite statement is interpreted syntactically as a single statement. It is used to delimit the entire assignment section. It is also useful for specifying several statements that are to be acted upon as a single statement.

The example in figure 4-4 gives alternative ways of packing an array.

METHOD 1

```
VAR /I;
    .
    .
    .
FOR /I := 0 TO 49 DO

BEGIN
    /VFD(GLOBL$+/I,15,7) := $4D;
    /VFD(GLOBL$+/I,7,7) := /I
END;
```

METHOD 2

```
VAR /I; /J;  /K;

ARRAY   GLOBL$  [0..99]   OF CHAR;
    .
    .
    .
/K := $4D;

FOR /I := 0 to 49 DO

BEGIN
    /J := 2*/I;
    GLOBL$  [/J] := /K;
    GLOBL$  [/J+1] := /I
END;
```

This example packs an array of 50 NPU (16-bit) words starting at location GLOBL$. Value $4D is placed in the upper half word, and the word count (0 through 49) in the lower half word.

Figure 4-4. Methods of Packing an NPU Array

Empty Assignment Statement

The empty assignment statement is analogous to the empty PASCAL statement. It contains no information; it can be used anywhere that a statement is appropriate. The empty statement exists so that a syntax error is not generated if the user inadvertently enters a semicolon. This most frequently occurs after the statement that preceded the END statement in a composite statement.

COMMENTS

Comments can be introduced in any position within a statement that does not violate a keyword or a symbol. Comments are delimited at the beginning by an ASCII underscore (this appears as a broken arrow in display code) and at the end by an ASCII question mark (this appears as a down-arrow in display code). Any character can be used within a comment except the delimiters.

REQUESTING A TRACE OPERATION

The pseudovariable /TRACE is used in MPEDIT to request a trace listing. The trace listing presents the following information:

The address or field that is initialized

The value to be inserted into the field

The previous contents of the full 16-bit word holding the field (the field can be all or only a part of that word)

The current contents of the full 6-bit word after the initializing value is inserted

The line number of the MPEDIT program that caused initialization of the field

The format of the pseudovariable requesting a trace listing is:

/TRACE:=x;

If the value for /TRACE is 2 or greater, the listing is produced. If /TRACE is assigned a value of 0 or 1, the trace report for that /TRACE entry is not produced. The default value for /TRACE is 2. The pseudovariable can appear anywhere in the assignment section; however, only those assignment statements that appear after the trace request will be included in the listing. For this reason it is customary to define the pseudovariables at the beginning of the assignment section.

A partial trace listing is shown in figure 4-5.

REQUESTING THE SYMTAB LISTING

The pseudovariable /ESL$ is used in MPEDIT to request a listing of the symbol table (SYMTAB). This report includes the local symbols. The report is generated at the end of an MPEDIT run. The format of the request is:

/ESL$:=x;

If the value of x is 2 or greater, the SYMTAB listing is produced; if the value is 0 or 1, the listing is suppressed. The default value for /ESL$ is 0.

The request can appear anywhere in the assignment section of the program. A sample partial SYMTAB listing is shown in figure 4-6. The listing was requested by the pseudovariable declaration:

/$ESL:=2;

```
                                   CYBER MINI CROSS SYSTEM - LINK EDITOR -

                                                 TRACE LIST

 44  VFD($00002, $7, $7) := 0000;   0000 → 0000          114  VFD($C0020, $8, $0) := 0000;   1000 → 1000
 45  VFD($00003, $F, $F) := 1A81;   0000 → 1A81          115  VFD($00020, $7, $7) := 004D;   1000 → 104D
 53  VFD($C0004, $F, $3) := 0000;   0000 → 0000          117  VFD($C0021, $F, $F) := 4943;   0000 → 4943
 54  VFD($C0004, $8, $2) := 0004;   0000 → 0800          122  VFD($C0024, $F, $3) := 0002;   0000 → 2000
 55  VFD($C0004, $8, $0) := 0000;   0800 → 0800          126  VFD($C0028, $F, $3) := 00C1;   0000 → 1000
 56  VFD($C0006, $7, $7) := 0001;   0000 → 0001          127  VFD($00028, $B, $2) := 0000;   1000 → 1000
 58  VFD($C0007, $F, $F) := 7FFF;   0000 → 7FFF          128  VFD($C0028, $8, $0) := 0000;   1000 → 1000
 63  VFD($00008, $F, $3) := 0006;   0000 → 6000          129  VFD($C0028, $7, $7) := 004E;   1000 → 104E
 67  VFD($0000C, $F, $3) := C001;   0000 → 1000          131  VFD($00029, $F, $F) := 3135;   0000 → 3135
 68  VFD($0000C, $B, $2) := 0000;   1000 → 1000          133  VFD($0002A, $F, $F) := 4C4D;   0000 → 4C4D
 69  VFD($0000C, $8, $0) := 000C;   1000 → 1000          135  VFD($0002B, $F, $7) := 0020;   0000 → 2000
 70  VFD($0000C, $7, $7) := 0044;   1000 → 1044          140  VFD($C002C, $F, $3) := 0005;   0000 → 5000
 72  VFD($0000D, $F, $F) := 4253;   0000 → 4253          141  VFD($C002C, $B, $2) := C000;   5000 → 5000
 77  VFD($00010, $F, $3) := 0002;   0000 → 2000          142  VFD($C002C, $8, $0) := C000;   5000 → 5000
 81  VFD($00014, $F, $3) := 0000;   0000 → 0000          143  VFD($0002C, $7, $7) := 0000;   5000 → 5000
 82  VFD($00014, $8, $2) := 00C1;   0000 → 0200          144  VFD($0002D, $F, $F) := 04BC;   0000 → 04BC
 83  VFD($C0014, $8, $0) := 0000;   0200 → 0200          146  VFD($C002E, $7, $7) := 0000;   0000 → 0000
 84  VFD($00014, $7, $7) := C000;   0200 → 0200          147  VFD($0002F, $F, $F) := B808;   0000 → B808
 85  VFD($00015, $F, $F) := 5200;   0000 → 5200          152  VFD($00030, $F, $3) := 0004;   0000 → 4000
 87  VFD($00016, $7, $7) := 0000;   0000 → 0000          156  VFD($C0034, $F, $3) := 000F;   0000 → F000
 88  VFD($00017, $F, $F) := 5300;   0000 → 5300          163  VFD($C0038, $F, $3) := 0001;   0000 → 1000
 93  VFD($00018, $F, $3) := 0000;   0000 → 0000          164  VFD($C0038, $B, $2) := C00C;   1000 → 1000
 94  VFD($00018, $8, $2) := 0002;   0000 → 0400          165  VFD($00038, $8, $0) := 0000;   1000 → 1000
 95  VFD($00018, $8, $0) := 0000;   0400 → 0400          166  VFD($00038, $7, $7) := 0053;   1000 → 1053
 96  VFD($C0018, $7, $7) := 0000;   0400 → 0400          168  VFD($00039, $F, $F) := 414D;   0000 → 414D
 97  VFD($C0019, $F, $F) := 5301;   0000 → 5301          173  VFD($0003C, $F, $3) := 0004;   0000 → 4000
 99  VFD($C001A, $7, $7) := 0000;   0000 → 0000          177  VFD($00040, $F, $3) := C00F;   0000 → F000
100  VFD($0001B, $F, $F) := 5321;   0000 → 5321          186  VFD($0004D, $F, $F) := 0000;   1C01 → 0000
105  VFD($C001C, $F, $3) := 000F;   0000 → F000          187  VFD($0004E, $F, $F) := 0000;   0800 → 0000
112  VFD($00020, $F, $3) := 0001;   0000 → 1000          188  VFD($0004F, $F, $F) := 0000;   18FD → 0000
113  VFD($C0020, $B, $2) := 0000;   1000 → 1000
```

Figure 4-5. Partial MPEDIT Trace Listing

```
                                   CYBER MINI CROSS SYSTEM - LINK EDITOR -

                                  ENTRY SYMBOL LIST - SORTED BY ENTRY NAME

*ENTRY**R/A**ADDRESS/VALUE**BIT S/L*         *ENTRY**R/A**ADDRESS/VALUE**BIT S/L*         *ENTRY**R/A**ADDRESS/VALUE**BIT S/L*

AACAPL   R        493C                       ADST2    A        0002                       AIDLET   A        0003
AACDAD   A        0000                       AEABLS   R        474A                       AIDLE    A        0001
AACDPT   A        0000                       AEAPL    R     11709:3709                    AINPLB   A        000E
AACORR   R        48BC                       AEASCI   R        46A7                       AINPSB   A        000D
AAEAPL   R        48FC                       AEATTN   R        4769                       AISPT1   R        44D5
AAEBCD   R        48FC                       AEATT1   R        4775                       AISPT6   R        452A
AANREA   A        0007                       AEAUT1   R        4648                       AISP4C   R        4512
AAOUTP   A        0003                       AEAUT2   R        464E                       AISP4E   R        44FA
AAREAD   A        0004                       AEAUT3   R        4654                       ALARM1   A        0001
AASBAP   R        4A3C                       AEBLS    R        45B4                       ALARM2   A        0002
AASTAP   R        49FC                       AECHR1   R        466D                       ALARM3   A        0003
ABAPLA   R        498C                       AECIN1   R        46DA                       ALCAPL   R        4C7C
ACAPL    R     116ED:36ED                    AECIN2   R        46DC                       ALCORA   R        4ABC
ACARTO   A        0078                       AECIN3   R        46E7                       ALEAPL   R        4BBC
ACAUTO   A        0001                       AECIN4   R        46E9                       ALEBCD   R        4B3C
ACCAPL   A        0004                       AECKMD   R        45A8                       ALF      L        0002
ACCAPL   L        0004                       AECOD1   R        46C0                       ANIL     L        0000
ACCASE   R     11687:3687                    AECOD2   R        46A0                       ASASCI   A        0022
ACCORR   A        0003                       AECSE    R     116D7:36D7                    ASAUTO   A        0018
ACCORR   L        0003                       ACESLL   R        475F                       ASCE26   R     1539C:339C
ACDELM   A        0002                       AECSL1   R        477A                       ASCE29   R     1535C:335C
ACEAPL   L        0002                       AECSL2   R        477C                       ASCINT   R        176C
ACEAPL   A        0002                       AEEIN1   R        46F5                       ASDISC   A        0003
ACEBCD   A        0001                       AEEIN2   R        46F7                       ASSLL    A        0004
ACEBCD   L        0001                       AEEIN3   R        4702                       ASSOL    A        0001
ACELL    A        0000                       AEEIN4   R        4704                       ASXPT    A        0002
ACEPL    A        0001                       AEELL    R        45C5                       ASXSOI   A        0005
ACKMSG   R        85BE                       AEELT1   R        45CE                       ASYNCE   R        153A
ACKPTR   R        1D6B                       AEELT3   R        45E0                       AS2741   A        0020
ACLIH2   A        0D00                       AEEPL    R        45F1                       ATAPLA   R        497C
ACLIM1   A        0A00                       AEEPT1   R        45F8                       ATELL    A        0000            F:7
ACPBLI   A        0001                       AEESL1   R        4780                       ATEPL    A        0000            7:7
ACPBOB   A        0002                       AEESL2   R        4782                       ATPDI2   R     1131B:331B
ACPCLR   A        000C                       AEINPT   R        4583                       ATPDI5   R     1133F:333F
ACPEVE   A        0000                       AEIN1    R        458C                       ATPDI6   R     11363:3363
ACPICS   A        0050                       AEMBT    R        455F                       ATPDI8   R     11387:3387
ACPIOW   A        0060                       AESEDI   R        473C                       ATPPR1   R     112D3:32D3
ACPLR1   R        1C0D                       AESLL    R        4570                       ATPPR4   R     112F2:32F2
ACPOBL   A        0058                       AES110   R        465B                       AUCAPL   R        4C3C
ACPOMA   A        006C                       AES150   R        4668                       AUCORA   R        4A7C
ACPONS   A        0048                       AES151   R        4662                       AUEAPL   R        487C
ACPRMA   A        0010                       AES300   R        4674                       AUEBCD   R        4AFC
ACRITT   A        000A                       AES301   R        467A                       AVASCE   R     11729:3729
ACRLF    L        0003                       AEXBLS   R        4635                       AVASCP   R        1E5D
ACR      L        0001                       AEXDLM   R        463A                       AVB7TO   R        1E74
ADDRES   R        0150                       AEXDTA   R        462A                       AVCNTR   R        1E3D
ADDRLC   R        015F                       AEXPTO   R        462F                       AVCORE   R     1173A:373A
ADDRSU   R        0160                       AEXSOI   R        4604                       AVCORR   R        1E5E
ADEADT   A        0014                       AE4XDL   R        47A7                       AVCRLF   R        1E52
ADST1    A        0001                       AE4XIN   R        4792                       AVCRMS   R        1E50
```

Figure 4-6. Partial MPEDIT SYMTAB Listing (Sorted by Entry Name)

REQUESTING THE INITIALIZED LOAD MODULE FILE LISTING

The pseudovariable /DMP$ is used in MPEDIT to request a listing of the initialized memory image load module file. Format of the request is:

/DMP$:=x;

If the value of x is 2 or greater, the listing is produced; if the value is 0 or 1, the listing is suppressed. Default value for /DMP$ is 2. The request can appear anywhere in the assignment part of the program.

The memory image load module file values are listed in hexadecimal; the listing is generated at the end of an MPEDIT run.

REQUESTING THE OPTIONAL FORM OF THE INITIALIZED LOAD MODULE FILE

The pseudoconstant /NAM$ is used in MPEDIT to request the optional form of the initialized memory image load file. That load file is especially formatted for downline-loading in the NPU. The format of the file is shown in appendix E.

Any three-character identifier can be assigned to the /NAM$ definition. The identifier specified is placed in the heading of the load file. An example of the /NAM$ definition is:

/NAM$:=OE2

The definition can appear anywhere in the CONST definition part of the program.

NOTE

This alternate form is not the final form of the load file that is downline-loaded into an NPU to provide the on-line CCP or CCI. Instead, as shown in figure 1-1 in section 1, the CCP or CCI installation procedures use a load file generating utility to process this optional form of the memory image load module file along with other load files. After processing by the host's load file generating utility, the reformatted and combined load file can be downline-loaded into an NPU.

MPEDIT DIAGNOSTICS AND ERROR MESSAGES

Some types of statement faults cause errors from the programmer's standpoint but do not generate an error message. Others can generate one or more error messages. Statements with the following types of errors will fail:

An undefined identifier

An attempt to assign a nonexistent memory location

A bad field specification (overflows word or has an illegal format)

An out-of-range subscript

A failed statement behaves functionally like a null statement. The following examples show failed statements.

Example 1:

```
U:=0
```

where U is an undefined identifier. This acts as an empty statement.

Example 2:

```
VAR/I;
    .
    .
    .
FOR /I:=$100 TO $202 DO
       (/I):=0
```

where only addresses $100 through $1FF are defined in the load file. In this case, 259 assignment statements are executed. The first 256 are valid; the last three fail.

If a failed statement or other error causes an error message, the error message is delivered to the output file. The messages are preceded by an up-arrow. If the error is a recognized syntax error, the up-arrow is followed by the character that was being processed when the error occurred.

Table B-3 in appendix B lists the MPEDIT error messages, the significance of the message, and the action that the operator or programmer should take in response to the message.

CHARACTER SET

The CYBER host uses one of two character sets to the CYBER Cross Build Utilities:

63-character ASCII

64-character ASCII

These code sets are shown in table A-1.

TABLE A-1. 63/64 CHARACTER ASCII CODE

CDC Graphic	ASCII Graphic Subset	6-bit Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	7-bit ASCII Code	CDC Graphic	ASCII Graphic Subset	6-bit Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	7-bit ASCII Code
:†	:	00††	8-2	00	8-2	072	6	6	41	6	06	6	066
A	A	01	12-1	61	12-1	101	7	7	42	7	07	7	067
B	B	02	12-2	62	12-2	102	8	8	43	8	10	8	070
C	C	03	12-3	63	12-3	103	9	9	44	9	11	9	071
D	D	04	12-4	64	12-4	104	+	+	45	12	60	12-8-6	053
E	E	05	12-5	65	12-5	105	-	-	46	11	40	11	055
F	F	06	12-6	66	12-6	106	*	*	47	11-8-4	54	11-8-4	052
G	G	07	12-7	67	12-7	107	/	/	50	0-1	21	0-1	057
H	H	10	12-8	70	12-8	110	(	(	51	0-8-4	34	12-8-5	050
I	I	11	12-9	71	12-9	111	)	)	52	12-8-4	74	11-8-5	051
J	J	12	11-1	41	11-1	112	$	$	53	11-8-3	53	11-8-3	044
K	K	13	11-2	42	11-2	113	=	=	54	8-3	13	8-6	075
L	L	14	11-3	43	11-3	114	blank	blank	55	no punch	20	no punch	040
M	M	15	11-4	44	11-4	115	, (comma)	, (comma)	56	0-8-3	33	0-8-3	054
N	N	16	11-5	45	11-5	116	. (period)	. (period)	57	12-8-3	73	12-8-3	056
O	O	17	11-6	46	11-6	117	≡	#	60	0-8-6	36	8-3	043
P	P	20	11-7	47	11-7	120	[	[	61	8-7	17	12-8-2	133
Q	Q	21	11-8	50	11-8	121	]	]	62	0-8-2	32	11-8-2	135
R	R	22	11-9	51	11-9	122	%	%	63††	8-6	16	0-8-4	045
S	S	23	0-2	22	0-2	123	≠	" (quote)	64	8-4	14	8-7	042
T	T	24	0-3	23	0-3	124	→	_ (underline)	65	0-8-5	35	0-8-5	137
U	U	25	0-4	24	0-4	125	∨	!	66	11-0 or 11-8-2†††	52	12-8-7 or 11-0†††	041
V	V	26	0-5	25	0-5	126	∧	&	67	0-8-7	37	12	046
W	W	27	0-6	26	0-6	127	↑	' (apostrophe)	70	11-8-5	55	8-5	047
X	X	30	0-7	27	0-7	130	↓	?	71	11-8-6	56	0-8-7	077
Y	Y	31	0-8	30	0-8	131	<	<	72	12-0 or 12-8-2†††	72	12-8-4 or 12-0†††	074
Z	Z	32	0-9	31	0-9	132	>	>	73	11-8-7	57	0-8-6	076
0	0	33	0	12	0	060	≤	@	74	8-5	15	8-4	100
1	1	34	1	01	1	061	≥	\	75	12-8-5	75	0-8-2	134
2	2	35	2	02	2	062	¬	^(circumflex)	76	12-8-6	76	11-8-7	136
3	3	36	3	03	3	063	; (semicolon)	; (semicolon)	77	12-8-7	77	11-8-6	073
4	4	37	4	04	4	064							
5	5	40	5	05	5	065							

†Twelve or more zero bits at the end of a 60-bit word are interpreted as end-of-line mark rather than two colons. End-of-line mark is converted to external BCD 1632.

††In installations using a graphic 63-character set, display code 00 has no associated graphic or card code; 6-bit display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations from ASCII/EBCDIC % yield a blank (55_8).

†††The alternate Hollerith (026) and ASCII (029) punches are accepted for input only.

DIAGNOSTIC MESSAGES

Each of the utilities described in this manual generates a set of error and (in some cases) informational messages. Some of these messages are sent to the output file (error file), and others are sent to a special fatal error file.

The error messages in this appendix are arranged by utility type. The tables are:

Table	Utility
B-1	Library Maintenance
B-2	Link
B-3	Edit

TABLE B-1. MPLIB ERROR MESSAGES

Message Text	Significance	User Action
AN ATTEMPT WAS MADE TO WRITE TOO MANY PROGRAMS TO THE NEW LIBRARY FILE	The library file is limited to 425 programs.	Delete nonused programs.
DEL DIRECTIVE DOES NOT HAVE A MATCH ON THE OLD LIBRARY FILE	The specified program (or the first program of a group) does not have a match on the old library file.	Check the object code module names against the directive parameter names.
I/O ERROR - READING INTERMEDIATE LIBRARY FILE	The intermediate file could not be read.	Try again. If error persists, call a system analyst.
I/O ERROR - READING LGO FILE	The LGO file is not in the proper format, or has been damaged.	Generate a new LGO file.
I/O ERROR - READING OLD LIBRARY FILE	The old library file is not in the proper format, or has been damaged. The old library cannot be used.	Try again. If error persists, call a system analyst.
I/O ERROR - WRITING INTERMEDIATE LIBRARY FILE	The intermediate library file could not be written. The space for temporary files was exceeded, or there was an error in writing the file.	Return unused local files and try again; if second attempt fails, allocate more space for temporary files.
I/O ERROR - WRITING NEW LIBRARY FILE	The new library file could not be written. The file may be write-protected, or it could exceed the user's allocated file size, or there may be an error in writing the file.	Try again after checking protection of file. If error persists, call a system analyst.
NAME FIELD ON DIRECTIVE CARD IS NOT RECOGNIZABLE	Program names consist of one to six letters, numbers, or $.	Check the directive name. Correct as appropriate.
NEW LIBRARY ENTRY POINT TABLE OVERFLOW	The total number of entry points plus programs (times 2) cannot exceed 4000.	Rewrite the programs to have fewer entry points or larger programs.
NO END-OF-TABLE WORD FOR ENTRY POINT TABLE RECORD ON LIBRARY FILE	The old library file is not in the proper format or has been damaged. The old library cannot be used.	Try again. If error persists, call a system analyst.

TABLE B-1. MPLIB ERROR MESSAGES (Contd)

Message Text	Significance	User Action
NO XFR BLOCK FOR PROGRAM ON RANDOM LGO FILE	The LGO file is not in the proper format.	Try again. If error persists, call a system analyst.
NON-ASCII (NOT $20-$5F) CHARACTER IN PROGRAM NAME OR ENTRY POINT ON THE LGO FILE	The LGO file is not in the proper format, or has been damaged.	Correct any format error; then try again.
PUT OR SUP DIRECTIVE DOES NOT HAVE A MATCH ON LGO FILE	The specified program (or programs) does not have a match on the LGO file.	Check the object code module names against the directive parameter names.
PUT OR SUP DIRECTIVE SECOND NAME DOES NOT HAVE A MATCH ON LGO FILE	The name of the second program (mod2) in a mod1-mod2 parameter does not exist on the LGO file or it precedes the first program name (mod1).	Check the order of modules in the LGO file. Use a different range of modules, or reverse the names in the parameter.
PUT OR SUP SECOND NAME DOES NOT HAVE A MATCH ON OLD LIBRARY FILE	The name of the second program (mod2) in a mod1-mod2 parameter does not exist on the old library file or it precedes the first program name (mod1).	Check the order of modules in the library. Use a different range of modules, or reverse the names in the parameter.
TOO MANY PROGRAMS ON LGO FILE	The LGO file is limited to 425 programs.	Delete nonused programs.
UNRECOGNIZABLE DIRECTIVE	The directive name is not *ALL, *PUT, *SUP, *DEL, or *LST.	Check the directives file, and enter a proper directive name.

TABLE B-2. MPLINK ERROR MESSAGES

Message	Significance	User Action
ADDRESS TABLE OVERFLOW	The combined number of addresses specified in all the directives exceeds the maximum number permitted.	The operator should revise the directives, perhaps using directives with a range of items in the parameters such as *L,mod1-mod2, addr, rather than individual linking or reverse linking directives. This can require more than one library building operation, or a rearrangement of modules on the input object code file.
ASSIGNMENT OPERATOR EXPECTED	The expression evaluator expected the := operator in the *VE directive but did not find one.	The user should correct the directive.
BAD LGO OR NEWLIB FILE	Either the input object code or the NEWLIB file is improperly formatted. An improper file may have been attached.	If this is not the case, the user may need to generate another input object code or NEWLIB file.
COMMA EXPECTED	The expression evaluator encountered a directive with fewer required parameters than expected.	The user should check the expression to assure that all the required parameters are present, and are separated from the previous parameter or directive name by a comma.

TABLE B-2. MPLINK ERROR MESSAGES (Contd)

Message	Significance	User Action
COMMON AREA EXCEEDED	MPLINK found a blank common specification in an object code module that exceeded the area allocated by the *COM directive.	The user should increase the size of the blank common area assigned by the *COM directive.
CURRENT LOWER LIMIT EXCEEDED	During linking caused by a *RL directive, MPLINK attempted to use a memory location below the word specified by the *LL directive.	The user should change the *LL boundary, or insert other directives to relocate the module (or group of modules) that crossed the boundary.
CURRENT UPPER LIMIT EXCEEDED	During normal loading caused by a *L directive, MPLINK attempted to use a memory location above that specified by the *UL directive.	The user should change the *UL boundary, or insert other directives to relocate the module (or group of modules) that crossed the boundary.
DATA AREA EXCEEDED	MPLINK found a named, common specification in an object code module that exceeded the area allocated by the *DAT directive.	The user should increase the size of the common area assigned by the *DAT directive, or decrease the number of applications used.
DIRECTIVE TABLE OVERFLOW	MPLINK encountered too many directives in the input directives file.	The user should consolidate directives (for instance, by using the directives with a range-type parameter such as mod1-mod2).
DUPLICATE ENTRY POINT	The entry point or module name (or at least the first six characters of it) have been used in a previous entry point or module name.	The user should rename one of the two expressions to obtain a unique, six-character name.
DUPLICATE LOADER MODULE	MPLINK found at least two modules with the name LOADER.	The user should eliminate duplicate LOADER modules; he should retain only that module to be used at the head of the load file.
ENTRY POINT TABLE OVERFLOW	The combined number of entry point names, module names, and synonyms exceeded the entry-point table capacity.	The user could rewrite his programs to consolidate modules, or to use fewer *SYN directives.
EXPRESSION OPERAND STACK OVERFLOW	The expression evaluator found too many operands during its processing.	The user should restate the expression with additional, nested parenthesis groupings.
EXPRESSION TABLE OVERFLOW	The number of expressions appearing in the *VE directives exceeds the allowable maximum.	The user should use fewer *VE directives, and revise the input modules accordingly.
EXPRESSION VAL EXCEEDS $3FFF	The value encountered in a *VE directive is too large (value range is 0 to $3FFF).	The user should change the directive.
EXT IN EXPR NOT ABSOLUTIZED	An entry-point name (nam1 or nam2) appearing in a *VE directive has not previously been made absolute.	The user should alter the *VE expression, or the order in which directives are added, so that the name is made absolute by the time MPLINK encounters the *VE directive.
IDENTIFIER EXPECTED	A valid name is expected in a *ENT, *OVLY, or *SYN expression, and it was not supplied.	The user should insert a valid name (a letter followed by a string of letters and/or digits) at the appropriate place in the directive.

TABLE B-2. MPLINK ERROR MESSAGES (Contd)

Message	Significance	User Action
ILLEGAL DIRECTIVE	The keyword that names the directive is incorrect.	The user should reenter the directive with the correct keyword.
ILLEGAL EOF ENCOUNTERED	A temporary MPLINK work file encountered an unexpected end-of-file. This is an internal error.	Rerun MPLINK.
ILLEGAL SYMBOL	The expression evaluator found a symbol it could not interpret.	The user should check the directive for symbols that are not in the 63 or 64-character display code set (as appropriate).
INCORRECT GROUP SPECIFICATION	The module names specified by the modl-mod2 parameter in a *L or *RL expression are not correct; that is, modl, mod2, or both are incorrectly specified.	The user should reenter the directive with the correct module names as they appear on the input object code or NEWLIB files. Alternatively, an incorrect module name on those files should be changed.
INVALID ADDRESS (COMPONENT)	A directive has an invalid addr, or some component of an addr is in error.	The user should correct the address and reenter the directive.
LOGICAL ADDRESS EXCEEDS $FFFF	An object code module is longer in 64K words and cannot be fitted in the NPU.	The module should be divided into smaller modules, or rewritten to reduce the number of words.
MAXIMUM GROUP SIZE EXCEEDED	In a range-type parameter (modl-mod2), the number of modules, or the number of words in all of the modules, exceeds MPLINK's ability to process the group.	The user should split the range parameter between two or more directives, each with a smaller modl-mod2 size.
MEMORY OVERFLOW	MPLINK assigned memory locations that do not exist in the NPU.	The user should check the memory size assigned by the *COR directive. If it is correct, the user can try reassigning the link start locations. It is possible that there is not enough memory for all the planned applications. The user could remove some applications; alternatively, additional memory should be purchased.
MISSING LEFT OR RIGHT PAREN	The expression evaluator found one of two errors: a left parenthesis not followed by a right parenthesis, or a right parenthesis that was not preceded by a left parenthesis.	In either case, the user should modify the expression to include the missing parenthesis, or to delete the unwanted parenthesis.
MOD ON LINK DIR NOT FOUND	The module specified by the mod parameter in a *L or *RL directive could not be found on the input object code or library file.	The user should check the module name and correctly specify it, or add the missing module to the appropriate file.
MODULE TOO LARGE	The specified module exceeds the size of MPLINK's external buffer.	The user should recode the module to compress it, or divide the module into two or more modules.
OPERAND EXPECTED	The expression evaluator expected a numeric value operand.	The user should revise the statement.
OVERLAY AREA LEN LESS THAN 0	In an OVLY directive, the ending address (addre) parameter is smaller than the beginning address (addrb) parameter.	The user should check the limits of the desired overlay size, and enter the proper limits.

TABLE B-2. MPLINK ERROR MESSAGES (Contd)

Message	Significance	User Action
OVERLAY AREA TABLE OVERFLOW	Only ten overlay areas can be declared with *OVLY directives.	The user should rearrange his applications so that no more than ten of them use overlays.
PERIOD EXPECTED	The expression evaluator expected the directive to be terminated with a period but found another * instead.	User should check to assure the directive is properly terminated with a period.
PLUS, MINUS, OR PERIOD EXPECTED	In a variable directive (*VE), the expression evaluator expected a plus, a minus, or a period in the exp parameter. None of these was found.	The user should enter the correct expression in the directive.
SYNONYM TABLE OVERFLOW	Too many *SYN directives were entered.	The user should revise his programs so he will have to declare fewer of these name-equating operations.
TOO MANY LOCAL VARIABLES	Too many variables were declared by *VE directives.	The user should recode to use fewer local variables.
UNDEFINED OR ILLEGAL IDENT	The name of an overlay is being illegally defined, or the referenced name of an entry point in a *SYN or *ENT directive is missing.	The user should use a correct overlay identifier (range AA through ZZ), or should enter the correct entry-point name on the *SYN or *ENT directive.
UNSATISFIED EXT TABLE OVERFLOW	MPLINK has encountered too many unsatisfied external references, or forward external references.	The user should rearrange module sequencing to minimize forward references, or he should delete some of the unsatisfied external references.
*DAT AND GLOBL$ CONFLICT	MPLINK encountered both a *DAT directive and an object code module called GLOBL$. The assigned GLOBL$ area exceeds the area assigned by the *DAT directive.	The user can remove the *DAT directive from the directives file, or he can increase the area assigned by the directive.
*RL FORCES MOD BELOW ADDR 0	The *RL directive causes MPLINK to locate some part of a module below the start of main memory.	The user should change the *RL directive.

TABLE B-3. MPEDIT ERROR MESSAGES

Message	Significance	Programmer Action
ARRAY TABLE OVERFLOW	The number of arrays declared in the ARRAY section exceeds the capacity of MPEDIT.	The programmer should revise his program to include fewer arrays.
BEGIN EXPECTED	MPEDIT syntax requires that a composite statement begin with the keyword BEGIN.	The programmer should revise the composite statement.
COMMA EXPECTED	MPEDIT syntax requires that parameters in /VFD expressions and array expressions be separated by commas.	The programmer should revise the statement.

TABLE B-3. MPEDIT ERROR MESSAGES (Contd)

Message	Significance	Programmer Action
DIMENSION TABLE OVERFLOW	The total number of dimensions declared for all arrays within the ARRAY section exceeds the capacity of MPEDIT.	The programmer should revise his programs to include fewer arrays, or reduce the dimensions in the existing arrays.
DO EXPECTED	MPEDIT syntax requires that a FOR - TO statement be followed with the keyword DO.	The programmer should revise the statement.
END EXPECTED	MPEDIT syntax requires that a composite statement terminate with the END keyword.	The programmer should revise the composite statement.
EXPRESSION OPERAND STACK OVERFLOW	The expression evaluator encountered too many operands during evaluation.	The programmer can usually avoid this problem by revising the statement with additional, nested parenthetical groupings.
ILLEGAL ARITHMETIC OPERATOR	The expression evaluator expected one of the five legal arithmetic operators: +, -, *, DIV, or MOD.	The programmer should revise the statement.
ILLEGAL DIMENSION COUNT	The number of index values specified for an array in the assignment section does not agree with the number of values declared in the array declaration section.	The programmer should revise the program to make the index values agree.
ILLEGAL END	The keyword END is not permitted where it was found.	The programmer should eliminate END from this position in the program.
ILLEGAL KEYWORD	The only keywords recognized by MPEDIT are: ARRAY DO OF BEGIN END OVERLAY CHAR FOR TO CONST MOD VAR DIV The keyword found in the program is not one of these.	The programmer should change the program to use the desired legal keyword.
ILLEGAL SYMBOL IN EXPRESSION	The expression evaluator encountered a symbol that was not in the CDC 63 or CDC 64 display code set (as appropriate).	The programmer should remove the undefined symbol, and substitute a symbol defined in the proper character set.
MISSING RIGHT PARENTHESIS	The expression evaluator found a left parenthesis that was not followed by a right parenthesis.	The programmer should modify the expression to include a right parenthesis, or to delete the left parenthesis.
NO ASSOCIATED LEFT PARENTHESIS	The expression evaluator found a right parenthesis that was not preceded by a left parenthesis.	The programmer should modify the expression to include a left parenthesis, or to delete the right parenthesis.
NOT A LOCAL VARIABLE	The indicated local identifier was not previously specified in the VAR section.	The programmer should either declare the variable in the VAR section, or specify an already declared variable.
OF EXPECTED	MPEDIT syntax requires that an array statement have the form: ARRAY size OF x. The array statement lacks OF following the array size specification.	The programmer should revise the composite statement.

TABLE B-3. MPEDIT ERROR MESSAGES (Contd)

Message	Significance	Programmer Action
OPERAND EXPECTED	The expression evaluator expected a numeric value operand.	The programmer should revise the statement.
OUT OF RANGE	The array index is not within the range declared for the array in the ARRAY section.	The programmer should either increase the declared range, or revise the erroneous statement.
PERIOD EXPECTED	The program ends with a period after the final END statement. None was found.	The programmer should revise the composite statement.
STUFF TABLE OVERFLOW	The number of editing values exceeds the internal capacity of MPEDIT.	The programmer can present the editing information in successive runs, recalling MPEDIT for each new section of the editing operation.
TO EXPECTED	MPEDIT syntax requires that a FOR statement be followed with the keyword TO.	The programmer should revise the composite statement.
:= EXPECTED	MPEDIT syntax requires that the next element of the statement be a defining operand.	The programmer should revise the statement.
**** OUT OF RANGE	This occurs only in a trace listing. The previous statement referenced an address that does not exist in the load file.	The programmer should revise the statement.
(EXPECTED	MPEDIT syntax requires that the next element of the statement be an opening parenthesis.	The programmer should revise the statement.
) EXPECTED	MPEDIT syntax requires that the next element of the statement be a closing parenthesis.	The programmer should revise the statement.
= EXPECTED	MPEDIT syntax requires that the next element of the constant definition be an equals sign.	The programmer should revise the statement.
[EXPECTED	MPEDIT syntax requires that an array statement have the form: ARRAY size OF x. The array statement lacks the opening square bracket around the size parameter.	The programmer should revise the composite statement.
] EXPECTED	MPEDIT syntax requires that an array statement have the form: ARRAY size OF x. The array statement lacks the closing square bracket around the size parameter.	The programmer should revise the composite statement.
; EXPECTED	MPEDIT syntax requires that this statement or declaration be separated from the previous statement or declaration by a semicolon.	The programmer should revise the statement.
xxxxxx MULTIPLE ENTRY DEFINITION	A local symbol, array name, or overlay name has been defined more than once. xxxxxx is the multiply defined name.	The programmer should redefine one of the names, keeping in mind that only the first six characters of the name are unique as far as MPEDIT is concerned.

GLOSSARY

TERMS

Following are terms unique to the description of the software presented in this manual.

Absolute Address -
An address that is permanently assigned to a storage location.

Assignment Section -
The part of the MPEDIT program that contains the statements that assign values to variables or fields.

Blank Common -
A common area used by macro assembler programs.

Build -
The procedure of converting individual source code modules into a linked set of object code modules in the form of a load tape.

Directive -
A utility input statement specifying some utility operation.

Dump Memory -
A hexadecimal listing, or memory dump, of the memory image load module file produced by MPLINK.

Dynamic Variable Area -
An area used by the PASCAL NEW procedure. See the PASCAL Compiler reference manual.

Edit Utility -
The utility (MPEDIT) that allows the user to initialize values in the memory image load file. The initialized memory image load file produced by the edit utility can be converted into an NPU load file.

Entry Point -
A labeled statement in a module that other modules can reference. In some cases, another program can activate a module at the entry point.

Error Messages -
Messages generated by a utility specifying operations that the utility could not perform. The failure could be due to a syntax error, an overflow condition, or other fault. Error messages are usually sent to the output file. Error messages are of two types: fatal errors, which halt the utility, and nonfatal errors, which are noted but allow the utility to continue processing.

External Synonyms -
Statements equating module names and entry points with local names.

Field -
A sequence of continuous bits consistently used to record similar information. For CCP and CCI, fields range from 1 to 16 bits in length and cannot cross word boundaries.

Initialized Load File -
The load file that is generated by MPEDIT. It has the same format as the MPLINK load file; however, selected fields and variables have initial values.

Input Directives File -
A file containing the directives necessary to execute the MPLINK, MPEDIT, or MPLIB utilities.

Installation Procedures -
The CCP or CCI procedures that generate a load file, which can be immediately downloaded into an NPU to form the on-line system of that NPU.

Keyword -
A reserved word used by a utility for a specific operation.

LGO File -
The load-and-go file. The file with this local name contains relocatable object code modules.

Library -
A group of object code modules, together with an index for those modules. The old library can be used as an input to the MPLINK, MPEDIT, and MPLIB utilities. The MPLIB utility generates a new library.

Library File -
A file created by the MPLIB utility. The file contains object code for all modules in the library, plus an index to the modules.

Library Maintenance -
The function performed by the MPLIB utility; that is, generating a new library from a set of object code modules or generating a new library from the old library, together with selected new object code modules.

Link Utility -
The utility (MPLINK) that links object code modules into a memory image load module. MPLINK also produces a symbol table file. Both of these files are used as input for the edit utility.

Linking -
The process of locating (assigning space) for object code modules on a memory image load module file, and resolving external calls in those modules with entry points in other load file modules.

Listing File -
A utility output file. In most cases, it contains user-requested reports.

Load File -
The host file holding a set of linked and edited object code modules which have been made absolute. The file can be downline-loaded into a specific NPU to form the on-line system for that NPU. A different load file (variant) is needed for each NPU to be loaded.

Memory Image Load Module File -
A file produced by the MPLINK utility. The load module file contains the code which has been made absolute for all programs to be used in the CCP or CCI build. MPLINK's version of this file is not initialized; MPEDIT initializes the file. A host load file generator converts the initialized memory image load module into a load file for CCP or CCI.

Memory Map -
An MPLINK report showing the main memory location of every module in the build.

Module -
(1) An integral part of an application that has a name and at least one entry point (a module is sometimes called a routine or a program). Any module can be selected to be used as part of an NPU build. (2) See memory image load module file.

MPEDIT -
The editing utility that assigns values to variables in the memory image load module file generated by MPLINK.

MPLINK -
The utility that assigns space to modules on a memory image load module file and links the modules together by equating external calls in one module to the comparable entry point in another module.

Name Call Statement-
The statement that is executed by the host's operating system to pass control of the computer to the program (or utility) associated with that statement.

Object Code Input Files -
Input files containing modules in object code format. Such files are used in all the utilities.

Object File -
A utility input or output file containing object code for modules.

Overlay -
A set of modules (application) that is not normally resident in the NPU. When the overlay is to be executed, it is loaded into a specific overlay area. The modules which normally use that area cannot be used until the overlay is ejected.

Overlay Area -
The part of an NPU that can be used to execute overlay programs.

Page Addressing -
The method of using an 18-bit address to locate a module that is assigned to a pageable area of memory. All modules assigned to the region above 65K are accessed in page-addressing mode. Some of the area below 65K is also page-addressed. In particular, in CCP and CCI, an 8K page starts at address 2000_{16}. All modules paged above 65K are imaged at this page.

Page (logical) -
An 8196-word section of CCP or CCI memory. All memory is paged. Memory up to 65K is executable at the address given; memory above 65K is imaged at the page beginning at 2000_{16}.

Page (physical) -
A 2K (2048) word section of NPU memory.

Page Register -
A register that indexes one of the NPU physical pages.

Program -
A module or a group of modules with related functions.

Report -
One of the reports associated with the link, edit, or library maintenance utility programs.

Reverse Loaded -
A module that is located in main memory by assigning the address given to the last word of the module. Space is then reserved for all other words in the module down to the first word.

Stack Area -
A reserved area in an NPU memory for use by PASCAL recursive/reentrant procedures.

Terminal Interface Program (TIP) -
An application that handles the interface between the NPU and a type of terminal, such as teletypewriter terminals or Mode 4 terminals.

Tophat -
Refers to a module that is called by several other modules. The code required to locate a tophat module's entry point is minimized by compiling a small auxiliary piece of code with the module. This tophat code sets the page registers when other modules call this module. If a tophat module is located in a main memory, this operation is not necessary, so the tophat auxiliary code is discarded. Otherwise, if a tophat module is paged, the tophat code is located in main memory to set the page registers.

Variant -
The definition of a real set of hardware and software for an NPU. The variant for an NPU defines the memory size, the NPU type (local or remote), the TIPs to be included in the build, and the maximum number of lines that can be configured. The variant also identifies the NPU, the host coupler (if any), and any trunks used by the NPU.

MNEMONICS

Following is a list of reserved words, symbols, keywords, and directives unique to the description of the software presented in this manual.

ABSOLMP	Absolute memory image load file
ARRAY	Array declaration command - MPEDIT directive
BEGIN	Begin statement - MPEDIT directive
BIP	Block interface package
CCI	Communications Control INTERCOM
CCP	Communications Control Program
CHAR	Character mode, array declaration - MPEDIT directive
CONST	Constant declaration - MPEDIT directive
CSET	CDC code set variable
D	Input file parameter - MPLINK and MPEDIT call statement
DIV	Division operator - MPEDIT directive
DO	Part of MPEDIT loop directive (FOR x TO y DO...)
END	End statement of a composite statement - MPEDIT directive
FOR	Part of MPEDIT loop directive (FOR x TO y DO...)
GLOBL$	CCP/CCI data base area
HIP	Host interface package
INPUT	Default input file
LGO	Load-and-go file
LIP	Link interface package
LOADER	First record on a memory image load module tape
MOD	Module operator - MPEDIT directive
MPEDIT	Edit utility
MPLIB	Library maintenance utility
MPLINK	Link utility
NEWLIB	New library file - MPLIB call statement parameter
OF	Part of array declaration - MPEDIT directive
OUTPUT	Default output file
OVERLAY	Overlay identifier - MPEDIT directive
SVM	Service module
SYMTAB	Symbol table file
TIP	Terminal interface package
TO	Part of MPEDIT loop directive (FOR x TO y DO...)
VAR	Variable declaration - MPEDIT directive
X.25	A TIP
*ALL	Copy all LGO files to new library - MPLIB directive
*CB	Upper boundary declaration - MPLINK directive
*COM	Define blank common area - MPLINK directive
*COR	Define 255x memory size - MPLINK directive
*DAT	Define labeled common area - MPLINK directive
*DEL	Delete module - MPLIB directive
*DMP	Define labeled common area - MPLINK directive
*DSTK	Define stack area - MPLINK directive
*DVAR	Define dynamic variable area - MPLINK directive
*END	End-of-directive-file directive - all build utilities
*ENT	Define entry point - MPLINK directive
*L	Link modules - MPLINK directive
*LIB	Define library file - MPLINK directive
*LL	Lower boundary declaration - MPLINK directive
*LST	List the library - MPLIB directive
*OVLY	Define overlay-area-directive - MPLINK directive
*PUT	Insert/replace module in library - MPLIB directive
*RL	Reverse-linking - MPLINK directive
*SUP	Suppress copying programs from the LGO to the library - MPLIB directive
*SYN	Define external synonym - MPLINK directive
*SYSID	System identification - MPLINK directive
*UL	Upper limit - MPLINK directive

*VE	Directive that assigns a value to a local variable - MPLINK directive
/DMP$	List the load tape - MPEDIT directive
/ENTRY	Address entry function - MPEDIT directive
/ESL$	List SYMTAB - MPEDIT directive
/LENGTH	Field length function - MPEDIT directive
/NAM$	Generate the NPU load tape - MPEDIT directive
/PGDISP	Page displacement function - MPLINK/ MPEDIT directive
/PGNUM	Page resister number function - MPLINK/ MPEDIT directive
/PGSET	Page register set function - MPLINK/ MPEDIT directive
/START	Field state location function - MPEDIT directive
/TRACE	Trace of edit operations - MPEDIT directive
/VFD	Variable field definition - MPEDIT directive

MEMORY IMAGE LOAD MODULE FILE FORMAT D

This appendix describes the format of the memory image load module file. It is an output of either MPLINK or MPEDIT. The only difference between the two files is the initialization of certain values. The format of the files is identical.

MPLINK or MPEDIT builds the load module file from object code programs and directives. The object code programs can be on an LGO file or a library file, or, for MPEDIT, a noninitialized memory image load module file produced by MPLINK. If the object code programs are on an input object code file, they have been previously produced individually by a CYBER Cross macro assembler, micro assembler, or PASCAL compiler. Libraries, likewise, are composed of object code programs produced by these CYBER Cross compilers/assemblers. A library has a directory for locating the modules easily.

Each program on the load module file has an execution (load) address. This address is either specified, or equated to zero, by the MPLINK utility. External references from all programs can be resolved from a program library. The user can also specify entry-point values (addresses).

FILE FORMAT

Figure D-1 shows the load module file format on the highest level. On this level, the file consists of an optional loader record, a partition for NPU-resident programs, and partitions for each group of overlay modules (assuming there are any optional overlays). The resident load partition includes a system header record followed by a series of record pairs, one pair for each program in the on-line system. If there are overlay partitions, each of these has a format similar to that of the resident partition. The file is terminated with a trailer record.

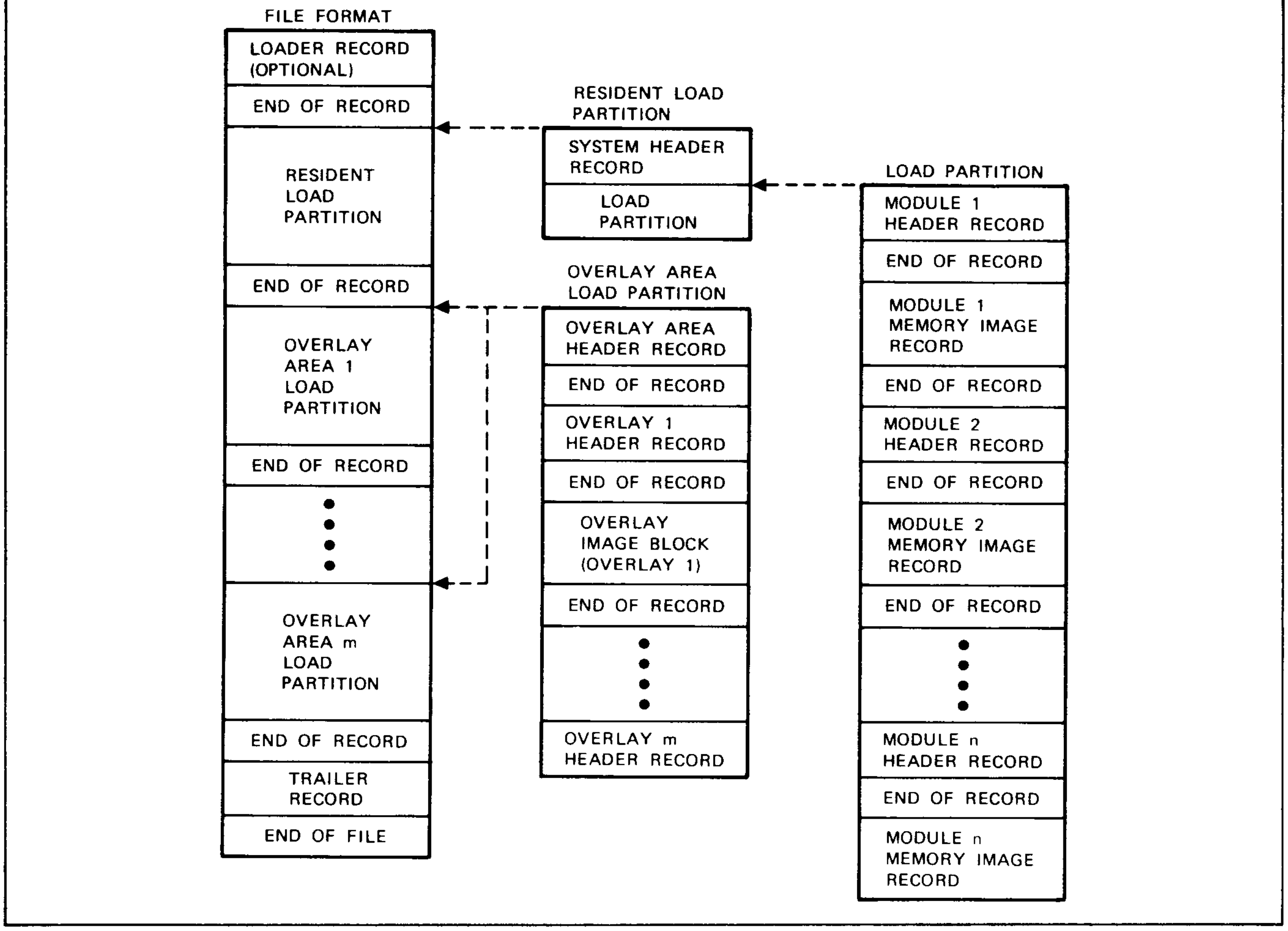


Figure D-1. Format of an MPLINK or MPEDIT Output Load File

OPTIONAL LOADER RECORD

If the LOADER record exists, it is the first record on the load module file. This record is included only if an object text file called LOADER is included in the library or input object code files used as input by MPLINK. The format of this header record is arbitrary.

RESIDENT LOAD PARTITION

This partition contains object code for every module in the on-line NPU system. It does not contain any code for overlay modules.

The partition has a system header, followed by a record pair for every on-line module. The modules occur in the same order in which they occur in NPU memory.

A record pair for a module consists of a module header record followed by a record containing the object code for the module.

System Header Record

Figure D-2 shows the format of the system header record. This record is generated as the direct result of the MPLINK directive: *SYSID,name(,text).

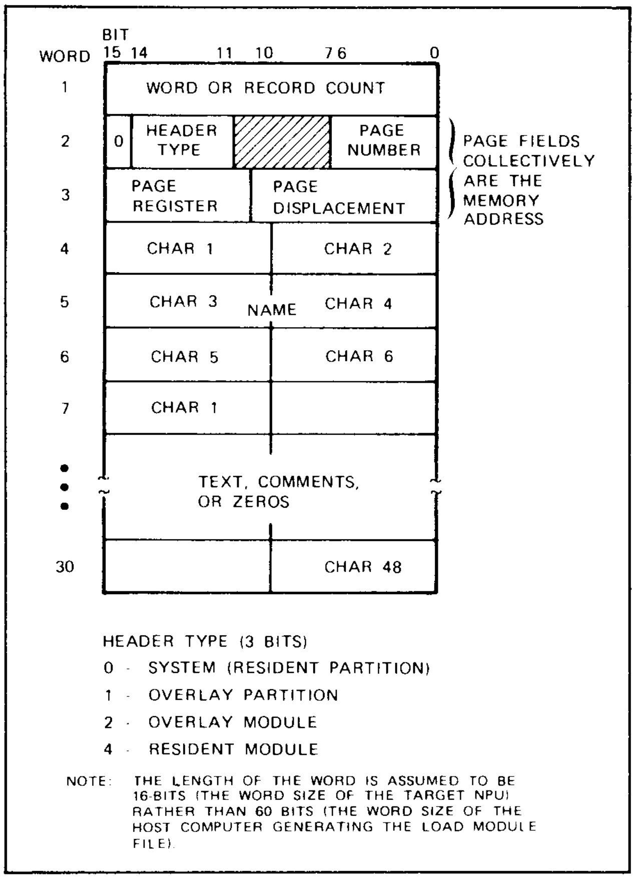

BIT
WORD
15 14 11 10 7 6 0
1 WORD OR RECORD COUNT
2 0 HEADER TYPE PAGE NUMBER
3 PAGE REGISTER PAGE DISPLACEMENT
PAGE FIELDS COLLECTIVELY ARE THE MEMORY ADDRESS
4 CHAR 1 CHAR 2
5 CHAR 3 NAME CHAR 4
6 CHAR 5 CHAR 6
7 CHAR 1
TEXT, COMMENTS, OR ZEROS
30 CHAR 48
HEADER TYPE (3 BITS)
0 - SYSTEM (RESIDENT PARTITION)
1 - OVERLAY PARTITION
2 - OVERLAY MODULE
4 - RESIDENT MODULE
NOTE: THE LENGTH OF THE WORD IS ASSUMED TO BE 16-BITS (THE WORD SIZE OF THE TARGET NPU) RATHER THAN 60 BITS (THE WORD SIZE OF THE HOST COMPUTER GENERATING THE LOAD MODULE FILE).

Figure D-2. Format of Load File Header Record

In this 30-word record the fields are as follows:

RECORD COUNT (word 1) is the number of records in the resident partition. The number of modules in this partition is:

Number(modules) = (records - 1)/2

HEADER TYPE (word 2) is a system header (type = 0).

MEMORY ADDRESS (words 2 and 3) is not used.

NAME (words 4, 5, and 6) is the six-character ASCII identifier specified by the name parameter in the *SYSID directive.

TEXT (words 7 through 30) is specified by the text parameter in the *SYSID directive. The character code is ASCII. Any characters not used are filled with zeros.

Module Header Record

Figure D-2 shows the format of the module header record. This record is generated by MPLINK at the time modules are linked as the result of an *L or *RL directive.

In this 30-word record the fields are as follows:

WORD COUNT (word 1) is the number of 16-bit words of object code in the following record.

HEADER TYPE (word 2) is a module header (type = 4).

MEMORY ADDRESS (words 2 and 3) designates the address of the module's first word. It has three parts (see the Page Addressing description in the MPLINK section):

PAGE NUMBER is the 7-bit logical page number.

PAGE REGISTER is the 5-bit page register ID.

PAGE DISPLACEMENT is the 11-bit displacement (in words) to the first word of the module on the physical page.

NAME (words 4, 5, and 6) is a six-character ASCII identifier. It has one of the following formats:

A PASCAL common area name: MAIN$ or GLOBL$

The name associated with a PASCAL-compiled program

The name specified on the NAM card of a macro assembled program

The name of a micro assembled program

COMMENTS (words 7 through 30) is blank if this is a PASCAL-compiled module. It is the comment (in ASCII characters) on the NAM card if this is a macro assembled module.

Resident Module Record

This record consists of 16-bit words of object code.

OVERLAY LOAD PARTITION

There can be one to ten overlay partitions. Each partition is separately identified and occurs in the same order that the overlay directives were entered in MPLINK.

An overlay partition contains object code for every module in that overlay.

The partition has an overlay header, followed by a record pair for every module in the overlay. The modules occur in the same order in which they occur in NPU memory when the overlay is moved into its execution area.

A record pair for an overlay module consists of an overlay module header record followed by a record containing the object code for the overlay module.

Overlay Area Header Record

Figure D-2 shows the format of the overlay area header record. This record is generated as the direct result of the MPLINK *OVLY directive.

In this 30-word record the fields are as follows:

RECORD COUNT (word 1) is the number of records in this overlay partition. The number of overlay modules in this partition is:

Number(modules) = (records - 1)/2

HEADER TYPE (word 2) is an overlay area header (type = 1).

MEMORY ADDRESS (words 2 and 3) specifies the first word of the overlay area. It has three parts:

PAGE NUMBER is the 7-bit logical page number.

PAGE REGISTER is the 5-bit page register ID.

PAGE DISPLACEMENT is the 11-bit displacement (in words) to the first word of the module on the physical page.

NAME (words 4, 5, and 6) is the six-character ASCII identifier specified by the name parameter in the *SYSID directive.

TEXT (words 7 through 30) is not used. Characters are filled with zeros.

Overlay Module Header Record

Figure D-2 shows the format of an overlay module header record. This record is generated by MPLINK at the time modules are linked as the result of an *L or *RL directive following an overlay declaration.

In this 30-word record the fields are as follows:

WORD COUNT (word 1) is the number of 16-bit words of object code in the following record.

HEADER TYPE (word 2) is an overlay module header (type = 2).

MEMORY ADDRESS (words 2 and 3) specifies the address of the overlay module's first word when it is in NPU memory. The address has three parts:

PAGE NUMBER is the 7-bit logical page number.

PAGE REGISTER is the 5-bit page register ID.

PAGE DISPLACEMENT is the 11-bit displacement (in words) to the first word of the module on the physical page.

NAME (words 4, 5, and 6) is a six-character ASCII identifier. It is the name associated with a PASCAL-compiled program.

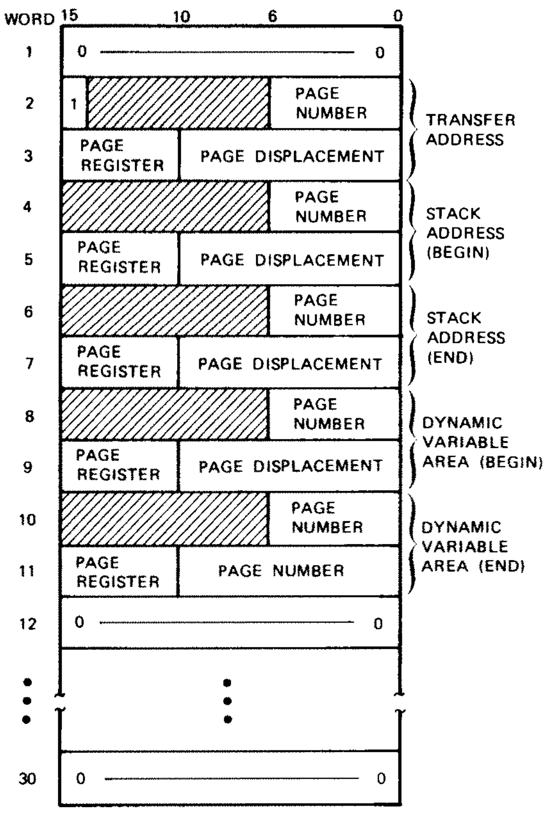
COMMENTS (words 7 through 30) is blank. The characters are filled with zeros.

Overlay Module Record

This record consists of 16-bit words of object code.

TRAILER RECORD

The format of the trailer record is shown in figure D-3. The use of the words is described on that figure.



BIT 15 IN WORD 2 SET INDICATES THE TRAILER RECORD. THIS BIT IS 0 IN A HEADER RECORD.

NOTE: THE LENGTH OF THE WORD IS ASSUMED TO BE 16 BITS (THE WORD SIZE OF THE TARGET NPU) RATHER THAN 60 BITS (THE WORD SIZE OF THE HOST COMPUTER GENERATING THE LOAD MODULE FILE).

Figure D-3. Format of Load File Trailer Record

OPTIONAL MEMORY IMAGE LOAD MODULE FILE FORMAT

The optional memory image load file for an NPU consists of a single record. The file is generated by the pseudoconstant /NAM$ in the MPEDIT utility.

The record begins with a prefix and a header as shown in figure E-1. The data within the record is segmented. Each segment is preceded by the first word address (FWA) for which it is intended. Each segment is also preceded by a length field. The length field indicates the number of 16-bit words in the data segment. The length can never exceed 120 16-bit words.

The prefix is shown in figure E-2. It contains information describing the creation of the record. Except for the first 60-bit word and the binary zero fill in the second 60-bit word, all information in the prefix is in display code, with blank fill, so that it can be printed. The prefix contains exactly fifteen 60-bit words.

The header is one 60-bit word. It contains the record name in display code, in bit positions 59 through 42. The bit pattern of the remainder of the word is shown in figure E-3.

The first word address and the length field formats are shown in figure E-4 and E-5.

The data segment format is shown in figure E-6.

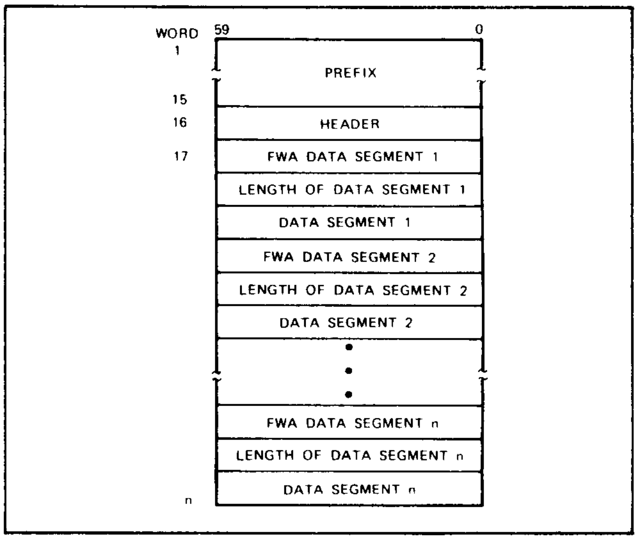


Figure E-1. Optional Load Module Record Format

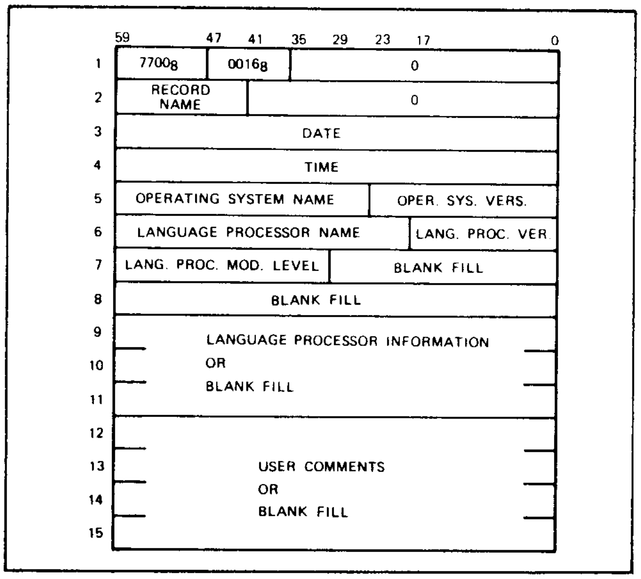

59 47 41 35 29 23 17 0
1 7700₈ 0016₈ 0
2 RECORD NAME 0
3 DATE
4 TIME
5 OPERATING SYSTEM NAME OPER. SYS. VERS.
6 LANGUAGE PROCESSOR NAME LANG. PROC. VER
7 LANG. PROC. MOD. LEVEL BLANK FILL
8 BLANK FILL
9 10 11 LANGUAGE PROCESSOR INFORMATION OR BLANK FILL
12 13 14 15 USER COMMENTS OR BLANK FILL

Figure E-2. Optional Memory Image Record Prefix Format

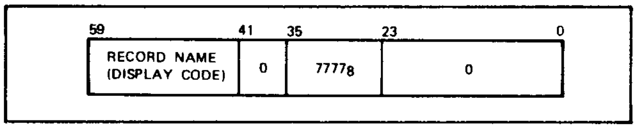


Figure E-3. Optional Load Module Record Header Format

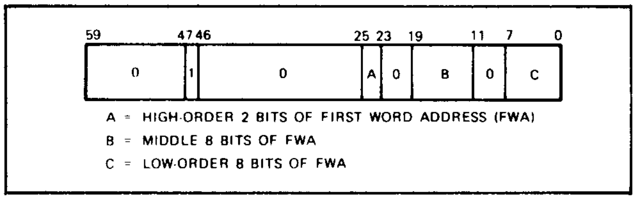

59 47 46 25 23 19 11 7 0
0 1 0 A 0 B 0 C
A = HIGH-ORDER 2 BITS OF FIRST WORD ADDRESS (FWA)
B = MIDDLE 8 BITS OF FWA
C = LOW-ORDER 8 BITS OF FWA

Figure E-4. Optional Memory Image Record First Word Address Format

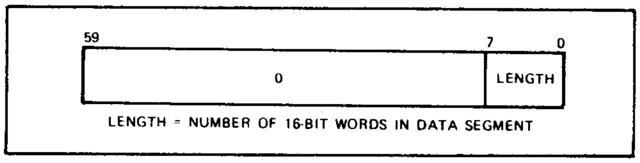


Figure E-5. Optional Memory Image Record Length Format

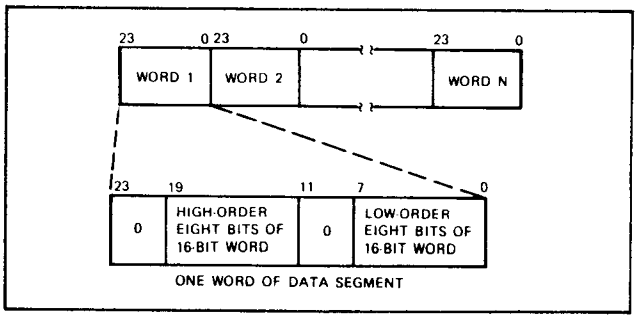

23
0 23
0
23
0
WORD 1
WORD 2
WORD N
23
19
11
7
0
0
HIGH-ORDER EIGHT BITS OF 16-BIT WORD
0
LOW-ORDER EIGHT BITS OF 16-BIT WORD
ONE WORD OF DATA SEGMENT

Figure E-6. Optional Load Module Record Segment Format

RELOCATABLE OBJECT CODE FILE FORMAT F

RECORD BLOCK

The object code input to the library maintenance and link utilities is the relocatable binary code generated by the CYBER Cross System translators: PASCAL compiler or macro assembler. The relocatable binary is represented in record blocks of 960 bits of information: i.e., sixteen 60-bit words or sixty 16-bit words.

The data portion of the record block is formatted accordingly:

Word 1 (16-bit words) -

Bits 15 through 8 = module sequence number

Bits 7 through 4 = 5, the 7/9 binary card indicator

Bits 3 through 0 = 0

Word 2 = the complement of the length of the data portion in 16-bit words

Word 3 to n = the object code block

Word 3 + n + 1 = the checksum

A record block will not exceed one card image; thus the length of an object code block (words 3 to n, where n is the length of the data portion) is 57 words or less. The checksum immediately follows the last data word in the record block; if the data portion is less than 57 words, the record block is padded to fill a complete 80-column card image.

The file of record blocks may be in one of two formats, depending on whether the packed binary (PB) parameter was selected on the translator call card. If the PB parameter was not selected, the file is written with one record block per record. If the PB parameter was selected, the file is written with the record blocks packed such that each record contains one program preceded by a CYBER Loader 7700 table.

OBJECT CODE BLOCK

The object code block, which contains the relocatable binary information, is headed by a type of block indicator field in bits 15 through 13 of the first object code word. The following object code block types are defined:

Type and Indicator	Description
NAM 001	Name block
RBD 010	Command block sequence
BZS 011	Zero storage block
ENT 100	Entry point block
EXT 101	External name block
ENF 000	Entry field block
EXF 111	External field block
XFR 110	Transfer address block

The remainder of this first word contains a constant of bits 6 and 4 set equal to 1, and all other bits set equal to 0.

A module's object code begins with a NAM block and terminates with an XFR block. The ENT and EXF blocks follow the RBD blocks. The RBD, BZS, ENT, and ENF blocks may come in any order.

The following is the format for the eight block types. Note that the word positions indicated are relative to the beginning of an object code block.

NAM BLOCK

The NAM block contains a word count for common and data storage, the module length, and the name of the program. See figure F-1.

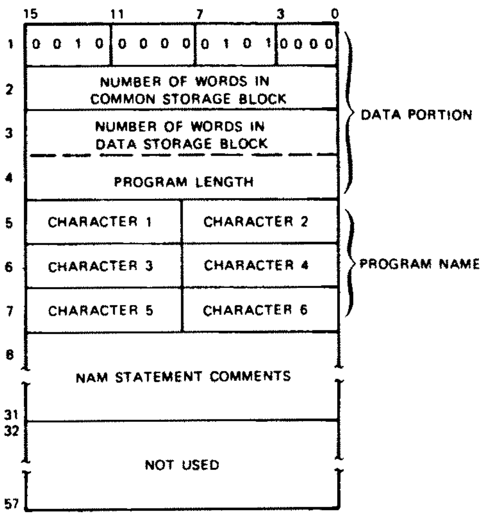
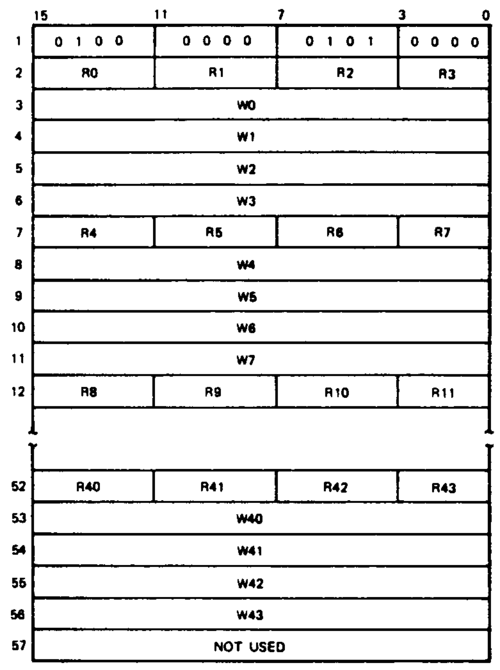


Figure F-1. NAM Block Image

RBD BLOCK

An RBD block contains a portion of the actual command sequence data of the module. See figure F-2. Words 2 through 57 contain the relocation bytes and words for the command sequence input. Each relocation byte is a 4-bit indicator that identifies a word of the command sequence input as an absolute 15-bit address, or as a 15-bit address relative to some relocation base. The relocation base for a word is determined by the particular combination of bit settings within the relocation byte.



Wn	The nth word of the input block (n=1 to 43).
Rn	The relocation byte of the nth word.
W0	The origin address of the input block.
R0	The relocation byte for W0.

Figure F-2. RBD Block Image

The following are the relocation bytes in RBD blocks:

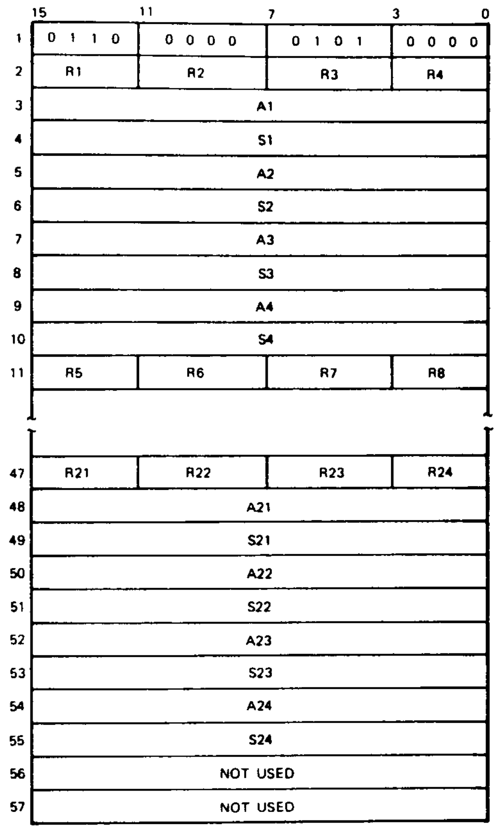
0000	Absolute (no relocation)
0001	Positive program relocation
0101	Negative program relocation
0010	Positive common storage relocation
0110	Negative common storage relocation
0011	Positive data storage relocation
0111	Negative data storage relocation

There is one relocation byte for every word in the command sequence output, and a maximum of 45 entries in the RBD block. The first word is the address relative to the start of the program where the loader begins storing command sequence data. The relocation byte for the first word address (storage address) of an RBD block may be 0000, 0001, or 0011. If the field contains a number larger than 0011, then 0011 is assumed. Zero is the leading bit for all but the last relocation byte; one is the leading bit for the last relocation byte.

BZS BLOCK

A BZS block contains relocation bytes, the starting address, and block sizes for areas of core to be cleared to zeros when the program is loaded. See figure F-3.

The relocation bytes for a starting address may be 0000, 0001, or 0011.

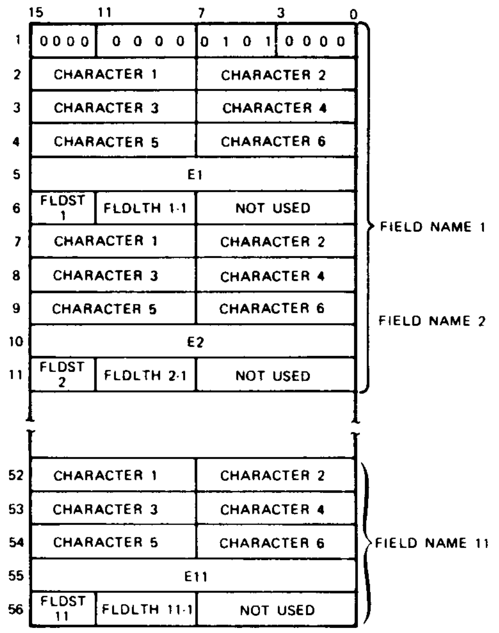


An	Starting address of the nth entry (n=1 to 24).
Sn	Size of the BZS reservation for the nth entry.
Rn	Relocation byte of the nth entry.

Figure F-3. BZS Block Image

ENF BLOCK

Up to 11 entry fields may be specified in an ENF block. See figure F-4. The end of data in this block is identified by zeros. If the sign bit of a word containing the entry point address is 0, the address is program-relocatable. If the sign bit is 1, the address is absolute and in one's complement. Data begins in word 2.

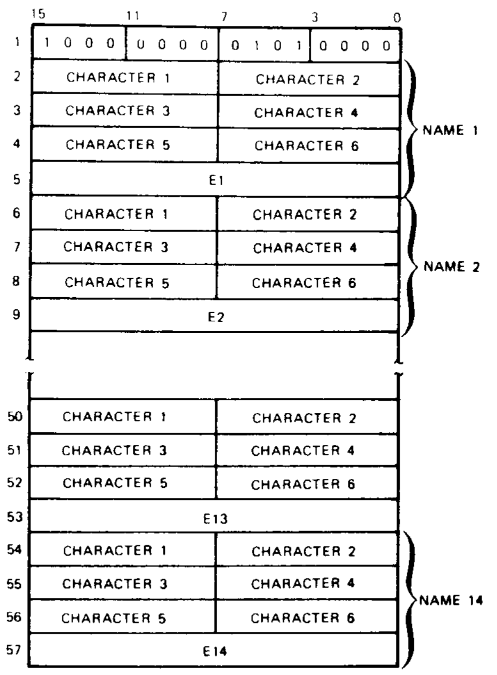


name n	A six-character name of the nth entry in the block.
En	The entry address of the nth field name. En is negative (one's complement) if absolute, and positive if relative.
FLDSTn	The leftmost bit of the nth field: $0 \leq FLDSTn \leq 15$.
FLDLTHn	The length of the nth field: $1 \leq FLDLTHn \leq 16$.

Figure F-4. ENF Block Image

ENT BLOCK

Up to 14 entry point names and addresses may be included in an ENT block. See figure F-5. The end of data in this block is identified by zeros. If the sign bit of a word containing the entry-point address is 0, the address is program-relocatable. If the sign bit of the word is 1, the address is absolute and in one's complement. Data begins in word 2 and extends to word 57.



name n	A six-character name of the nth entry in the block.
En	The entry-point address of the nth name. En is negative (one's complement) if absolute, and positive if program-relocatable.

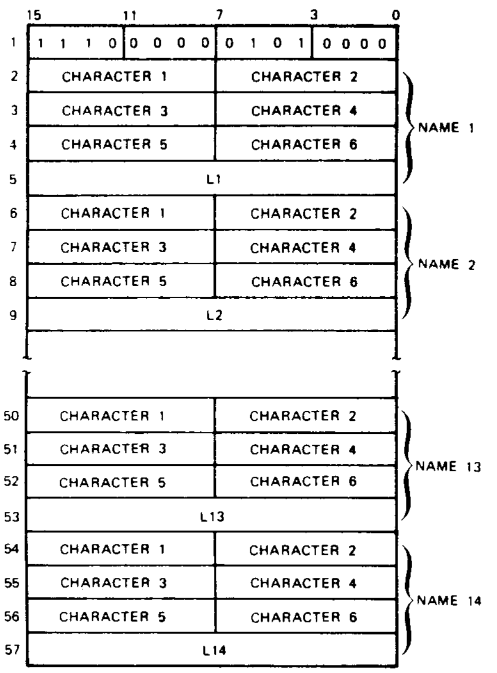
Figure F-5. ENT Block Image

When processing an ENT block, the loader records the entry-point name in its table. The entry-point address is adjusted for relocation (either program or absolute), and then it is recorded in the table of entry points. This procedure is repeated until the end of input is reached (a name equal to 0).

EXF BLOCK

Up to 14 external fields and link addresses may be included in an EXF block. See figure F-6.

The end of the EXF block is indicated by zeros. If the sign bit of the word containing the link address is 0, the address is program-relocatable. If the sign bit is 1, the address is absolute and in one's complement. The format of the data in the block is the same for EXF as for ENT information. Relative external fields are indicated by setting the leftmost bit of the word containing character 1 of the field name. An external name that contains no references within the module's object text is indicated by a $8000 in the link address.



15 11 7 3 0
1 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0
2 CHARACTER 1 CHARACTER 2
3 CHARACTER 3 CHARACTER 4
4 CHARACTER 5 CHARACTER 6
5 L1
NAME 1
6 CHARACTER 1 CHARACTER 2
7 CHARACTER 3 CHARACTER 4
8 CHARACTER 5 CHARACTER 6
9 L2
NAME 2
50 CHARACTER 1 CHARACTER 2
51 CHARACTER 3 CHARACTER 4
52 CHARACTER 5 CHARACTER 6
53 L13
NAME 13
54 CHARACTER 1 CHARACTER 2
55 CHARACTER 3 CHARACTER 4
56 CHARACTER 5 CHARACTER 6
57 L14
NAME 14

name n	A six-character name of the nth entry in the block.
Ln	The link address of the nth name. Ln is negative (one's complement) if absolute, and positive if relative.

Figure F-6. EXF Block Image

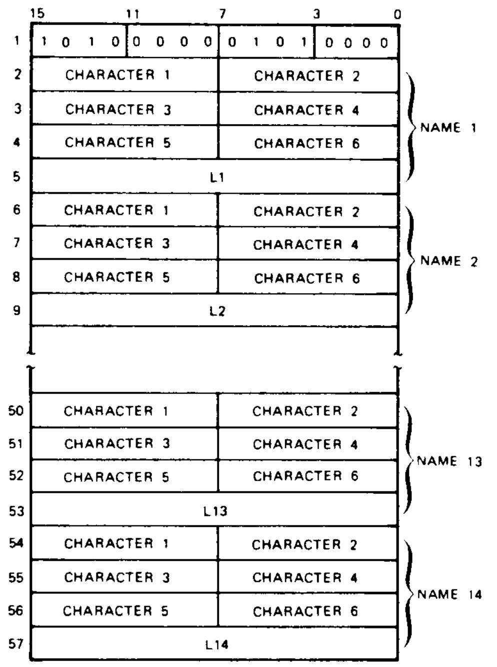
EXT BLOCK

Up to 14 external names and link addresses may be included in an EXT block. See figure F-7.

The end of the EXT block is indicated by zeros. If the sign bit of the word containing the link address is 0, the address is program-relocatable. If the sign bit is 1, the address is absolute and in one's complement. The format of the data in the block is the same for EXT as for ENT information. Relative externals are indicated by setting the leftmost bit of the word containing character 1 of the name. The end-of-link is indicated by a $7FFF.

XFR BLOCK

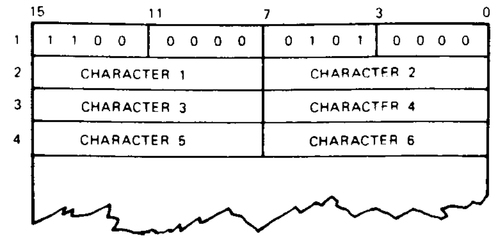
The XFR block contains a transfer address (in words 2 to 4), which is six ASCII characters in length, including trailing spaces. See figure F-8. The transfer address must be an entry point in the program being loaded or in another program loaded during the same load operation.



15 11 7 3 0
1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0
2 CHARACTER 1 CHARACTER 2
3 CHARACTER 3 CHARACTER 4
4 CHARACTER 5 CHARACTER 6
5 L1
NAME 1
6 CHARACTER 1 CHARACTER 2
7 CHARACTER 3 CHARACTER 4
8 CHARACTER 5 CHARACTER 6
9 L2
NAME 2
50 CHARACTER 1 CHARACTER 2
51 CHARACTER 3 CHARACTER 4
52 CHARACTER 5 CHARACTER 6
53 L13
NAME 13
54 CHARACTER 1 CHARACTER 2
55 CHARACTER 3 CHARACTER 4
56 CHARACTER 5 CHARACTER 6
57 L14
NAME 14

name n	A six-character name of the nth entry in the block.
Ln	The link address of the nth name. Ln is negative (one's complement) if absolute, and positive if relative.

Figure F-7. EXT Block Image



15 11 7 3 0
1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0
2 CHARACTER 1 CHARACTER 2
3 CHARACTER 3 CHARACTER 4
4 CHARACTER 5 CHARACTER 6

Figure F-8. XFR Block Image

LINK UTILITY EXAMPLES

This appendix gives two examples of calling MPLINK independently of the CCP or CCI build procedures (figures G-1 and G-2). Note that in the ordinary case, this information is not necessary since the build procedures automatically supply all the calling procedures.

Compile a PASCAL source program and build a load module satisfying external references from an object program library:

```
ABC,CM77000,T77,P4.                             0000,xxxx,xxxxxxxx,SMITH.
REQUEST (ABSOLMP,*PF)                           *Memory Image Load Module File
ATTACH(NEWLIB,OBJPGMLIB03,ID=PT                 *New Library
ATTACH(MPLINK,ID=SCDD)                          *MPLINK Utility
ATTACH(PASCAL,ID=SCDD)                          *PASCAL Compiler
PASCAL(O,CSET=64)                               *List output and use 64 char ASCII
FRMT.
REWIND(LGO)                                     *Reset Object Code Input File
MPLINK(CSET=64)                                 *Call MPLINK
CATALOG(ABSOLMP,LOADMOD01,ID=PT,RP=30)          *Catalog Load Module File
7/8/9
...PASCAL source program...                     *All PASCAL Source Programs
7/8/9

    MPLINK directive file

6/7/8/9
```

NOTES

After all of the object programs on the object code input file have been read and linked, any remaining unsatisfied external references can be resolved using the library if one is supplied.

Figure G-1. MPLINK Execution Example 1

Build a load module from an object program library with editing of the load module file:

```
ABC,CM77000,T77,P4.                             0000,xxxx,xxxxxxxx,SCHOFIELD.
REQUEST(ZAPMP,*PF)
ATTACH(NEWLIB,OBJPGMLIB03,ID=PT)
ATTACH(MPLINK,ID=SCDD)
ATTACH(MPEDIT,ID=SCDD)
MPLINK(CSET=64)
REWIND(ABSOLMP,SYMTAB)                          *Absolute Memory Image Load File and Symbol Table File
MPEDIT(CSET=64)
CATALOG(ZAPMP,LOADMOD02,ID=PT,RP=30)
7/8/9
*REL,NEWLIB.  First MPLINK Directive

    Nest of MPLINK Directive File

6/7/8/9
```

Figure G-2. MPLINK Execution Example 2

EDIT UTILITY EXAMPLES

This appendix gives the following examples:

Selected portions of an MPEDIT program with constant, variable, and array declarations, and a single assignment section for the main CCP programs (that is, no overlay assignment sections). This program is given in figure H-1.

A partial listing of the memory image load module file (figure H-2).

Note that the edit utility is automatically called as a part of the CCP and CCI build procedures.

```
                              CYBER MINI CROSS SYSTEM - LINK EDITOR -

                                        EDIT STATEMENTS

↱
**********************************************
*                                            *
*    COPYRIGHT CONTROL DATA CORP. 1975,      *
*    1976, 1977, 1978, 1979, 1980            *
*                                            *
**********************************************
↓
↱
**********************************************
*                                            *
*        CONSTANTS                           *
*                                            *
**********************************************
↓
CONST
      /TRUE  =  1;
      /FALSE =  0;
      /NAM$     = 0E2;                        ↱ GENERATE MPPPU FILE      ↓
```

DEFINITION/DECLARATION SECTION

CONSTANT DECLARATION PART

```
↱
************************************
*                                  *
*        SYSTEM CONSTANTS          *
*                                  *
************************************
↓
↱
***************************************************************************
              **CORE SIZE**
***************************************************************************
↓
      /CS16K    = $3FFF ;
      /CS32K    = $7FFF ;
      /CS40K    = $9FFF ;
      /CS48K    = $BFFF ;
      /CS56K    = $DFFF ;
      /CS64K    = $FFFE;
      /CS128K   = 1;
      /CS256K   = 2;
↱
***************************************************************************
              **BUFFER CONTROL BLOCK INDECES**
***************************************************************************
↓
      /BOS0     = 0 ;          ↱ SIZE 0 INDEX ↓
      /BOS1     = 1 ;          ↱      1       ↓
      /BOS2     = 2 ;          ↱      2       ↓
      /BOS3     = 3 ;          ↱      3       ↓
↱
                  ●
                  ●
                  ●
```

VARIABLE DEFINITION PART

```
VAR
      /I;                   ↱ GENERAL LOOP INDEX                      ↓
      /I1:                  ↱ GENERAL LOOP INDEX                             ↓
      /I2:                  ↱ GENERAL LOOP INDEX                             ↓
      /I3;                  ↱ GENERAL USE VARIABLE                           ↓
      /P:                   ↱ WORK POINTER FOR MODEM STATE PROGRAMS          ↓
      /J:                   ↱ GENERAL INDEX LOOP                             ↓
      /L:                   ↱WORK-POINTER FOR TIP-INPUT-STATES↓
      /LSAVE:               ↱ SAVE WORK PTR IN TEXT PROCR HDR STATE PROGRAM↓
      /K;                   ↱VARIABLE INDEX KQCNTRTBL                        ↓
      /CHSMLIA :             ↱ SUB PORT LCB TABLE POINTERS - MLIA            ↓
      /CHSCONS :             ↱                               CONSOLE         ↓
      /CHSCOUPLER ;          ↱                               COUPLER         ↓
      /ICHSUBLCB;                ↱ INDEX - CHSUBLCB                          ↓
      /LPID:
      /TCBCONS :
      /TCBMLIA ;
       /BOS32:
      /IDTBL;                ↱ NETWORK DEFINITION TABLE WORK POINTER         ↓
↱*************************************
*                                   *
*  THE FOLLOWING IS THE SOURCE INPUT *
*  FOR APPLICATION UNIQUE VARIABLES  *
*  DEFINITION FOR THIS SYSTEM        *
*                                   *
*************************************↓
```

Figure H-1. Partial MPEDIT Program (Sheet 1 of 6)

```
        BUFLCD[/BOSO../BOS3] OF 1;
        BUFMASKS[/BOSO../BOS3] OF 1:
        BFCTLBK[/BOSO../BOS3] OF /SIZBFCTLBK;
        NICTCT [ NOROO..NODIAG ] OF 1:
        NOTECT [ NOTMLIA..NOTDIAG ] OF 15;
        BFTHRESH [BOTI1..BOTHMUX] OF 1:          → BUFFER THRESHOLDS        ↓
                ●
                ●
                ●
        AVTC4LE [0..4]                  OF 1:
        ***********************************
        *                                 *
        *           ASYNC ARRAYS          *
        *                                 *
        ***********************************↓
→↓
        AVTCBAT  [1..40] OF 3;                   →TCB ACTION TABLE          ↓
        AVLCBAT  [1..3]  OF 3;                   → LCB ACTION TABLE         ↓
        AVTCBFDT [0..0] OF 1;                    →TCB FIELD DESCRIPTOR TABLE↓
        AVLCBFDT [0..1]  OF 1;                   →LCB FIELD DESCRIPTOR TABLE↓
→↓
→
              ASYNC CODE TABLE ADDRESS ARRAYS
↓
        AVUCCODETE[/ACEBCD../ACCAPL] OF 1;
        AVINTABLE [/ACEBCD../ACCAPL,/NOASYASC../NOIBM2741] OF 1;
        AVOUTTABLE [/ACEBCD../ACCAPL,/NOASYASC../NOIBM2741] OF 1;
→
*           INITIALIZATION OF SVM-ARRAYS FOR HASP-TIP
↓
        HSTCBATE[1..40] OF 3;                    →TCB ACTION TABLE          ↓
        HSLCBAT  [1..2]  OF 3:                   → LCB ACTION TABLE         ↓
        HSTCBFDT [0..0] OF 1;
        HSLCBFDT [0..0] OF 1;
→↓
→***********************************************************************↓
→*END APPLICATION UNIQUE ARRAY SOURCE*↓
→***********************************************************************↓
→
****************************************
*                                      *
*       DEFINE PSEUDO VARIABLES        *
*                                      *
****************************************
↓
BEGIN
        VARID := $96;
        VARID+1 := $COOF;
        CCPVER     := $32;                       →CCP VERSION - CSD REPLACES↓
        CCPCYC     :=  0;                        →CCP CYCLE   -USER REPLACES↓
        CCPLEV     :=  0;                        →CCP LEVEL   - CSD REPLACES↓
        /TRACE   := 2 ;
        /ESLS    := 2;                           → EXTERNAL SYM TBL LIST ON ↓
→
**********************************************
*                                            *
*       DEFINE AS MANY POINTERS              *
*       AS POSSIBLE                          *
*                                            *
**********************************************
↓
→↓
        /BOS32 := /BOS16 + 1;
        BEGINY + $D := MAINE;                    → SYSTEM INIT              ↓
→************************************
*       SET UP POST MORTEM DELIMITERS *
*************************************
↓
        CENTLIM := $20;
        COFNTLIM:= $5;
→
*                * *  INITIALIZE WORKLISTS FOR OPS LEVEL PROGRAMS  * *
↓
→

→
*                * *  INTERNAL PROCESS  * *
↓
        BYWLCB[BOINWL ].BYPRADDR   := /ENTRY(PBINTPROC);
        BYWLCB[BOINWL ].BYPAGE     := /PGNUM(PBINTPROC);
        BYWLCB[BOINWL ].BYWLINDEX := BOINWL;
        BYWLCB[BOINWL ].BYMAXCNT   := 4;
        BYWLCB[BOINWL ].BYINC      := 2;
        BYWLCB[BOINWL ].BYWLRFO    := /TRUE;
```

ASSIGNMENT SECTION

PSEUDO VARIABLE DECLARATION PART (PART OF ASSIGNMENT SECTION)

Figure H-1. Partial MPEDIT Program (Sheet 2 of 6)

```
→↓
      →**********************************
      *  SERVICE MODULE LOCAL VARIABLES  *
      **********************************↓
  /BZOWNER;
  /BZLNSPD;
  /BZN2;
  /BZ1PKTLNTH;
  /BZ2PKTLNTH;
  /BZK;
  /BZ1LPVC;
  /BZ2LPVC;
  /BZDCF;
  /BZTRANSTYPE;
  /BZ1LSVC;
  /BZ2LSVC;
  /BZLAPB;
  /BZT1;
  /LCW;
  /BSTCLASS;
  /BSOWNER;
  /BSCN;
  /DN;
  /SN;
  /BSNBL;
  /BSIPRI;
  /BSPGWIDTH;
  /BSPGLENGTH;
  /BSCANCHAR;
  /BSBSCHAR;
  /BSCNTRLCHAR;
  /BSCRIDLES;
  /BSLFIDLES;
  /BSCRCALC;
  /BSLFCALC;
  /BSSPEDIT;
  /BSXPARENT;
  /BSXCHM;
  /BSXCHL;
  /BSXCHAR;
  /BSXID;
  /BSINDEV;
  /BSOUTDEV;
  /BSECHOPLX;
  /BSPGWAIT;
  /BSPARITY;
  /BSABTLINE;
  /BSUSR1;
  /BSUSR2;
  /BSCODE;
 /BSXCHRON;
  /BSCHARREC;
      /LCBVARIANT;                        →FIRST VARIANT LCB INDEX   ↓
      /TCBVARIANT;                        →FIRST VARIANT TCB INDEX   ↓
      /DOLCBEDT;
      /DOTCBEDT;
      /DOSMBFSZ ;
→*****************************************************************↓
→*END APPLICATION CONTROL VARIABLE SOURCE*↓
→*****************************************************************↓
→
*****************************************
*                                       *
*     ARRAY DECLARATIONS - BASE AND CCP *
*     TABLES                            *
*                                       *
*****************************************
↓
ARRAY
      BJTIPTYPT [NOHDLC..NIUSR1] OF /SIZTIPTYPT;
      BWWLENTRY [1..17] OF /J1WLMAX;
      CBTIMTBL[ COLCBTMSCN..COSPARE] OF 4;
      CGTCBS [0..C4TCM1] OF /SIZTCB;      → FIXED TCBS.              ↓
      JACT [/TTY../LP1742] OF /SIZCT;
      JAIOWL [/FALSE../TRUE] OF 1;
      JGTESTABLE [/FALSE../TRUE,/FALSE../TRUE,/FALSE../TRUE] OF 1;
      JZOPSBASE [BOCHWL..BODUMMY] OF 2;    → OPS PROGRAM ARRAY        ↓
      JKMASK [1..17] OF 1;
      JKTMASK[1..17] OF 1;
      JSWLADDR [0..17] OF 1;
      J1TOPREFS [1..64] OF 1;
      NBLTYT [NOLDIAG..NOLAST,1..NKPCBOUT] OF 2;
      NELTP  [NOLDIAG..NOLAST] OF 1;
      BUFLENGTH[/BOSC../BOS3] OF 1;
      TCBLENGTH[/BOTS0../BOTS7] OF 1;
```

ARRAY DECLARATION

Figure H-1. Partial MPEDIT Program (Sheet 3 of 6)

```
↑
*              * *  MLIA INTERRUPT HANDLER  * *
↓
      BYWLCB[BOMLWL ].BYPRADDR  := /ENTRY(PBMLIAOPS);
      BYWLCB[BOMLWL ].BYPAGE    := /PGNUM(PBMLIAOPS);
      BYWLCB[BOMLWL ].BYWLINDEX := BOMLWL;
      BYWLCB[BOMLWL ].BYMAXCNT  := 10;
      BYWLCB[BOMLWL ].BYINC     := 5;
      BYWLCB[BOMLWL ].BYWLREQ   := /TRUE;
↑
*              * *  SERVICE MODULE  * *
↓
      BYWLCB[BOSMWL ].BYPRADDR  := /ENTRY(PNSMWL);
      BYWLCB[BOSMWL ].BYPAGE    := /PGNUM(PNSMWL);
      BYWLCB[BOSMWL ].BYWLINDEX := BOSMWL;
      BYWLCB[BOSMWL ].BYMAXCNT  := 1;
      BYWLCB[BOSMWL ].BYINC     := 4;
      BYWLCB[BOSMWL ].BYWLREQ   := /TRUE;
↑
               ●
               ●
               ●
↑↓
↑↓
      FOR /I := 1 TO 17 DO
        BEGIN
          JSWLADDR[/I] := BWWLENTRY+/J1WLMAX*(/I-1);
        END;
↑     THE VALUES IN JKTMASK ARE THE SAME AS THOSE IN JKMASK          ↓
↑↓
      JKMASK[1 ] :=      0;        JKTMASK[1 ] :=      0;
      JKMASK[2 ] :=      1;        JKTMASK[2 ] :=      1;
      JKMASK[3 ] :=      5;        JKTMASK[3 ] :=      5;
      JKMASK[4 ] :=     $0;        JKTMASK[4 ] :=     $0;
      JKMASK[5 ] :=    $10;        JKTMASK[5 ] :=    $10;
      JKMASK[6 ] :=    $30;        JKTMASK[6 ] :=    $30;
      JKMASK[7 ] :=    $1F;        JKTMASK[7 ] :=    $1F;
      JKMASK[8 ] :=    $1F;        JKTMASK[8 ] :=    $1F;
      JKMASK[9 ] :=    $FF;        JKTMASK[9 ] :=    $FF;
      JKMASK[10] :=   $1FF;        JKTMASK[10] :=   $1FF;
      JKMASK[11] :=   $3FF;        JKTMASK[11] :=   $3FF;
      JKMASK[12] :=   $7FF;        JKTMASK[12] :=   $7FF;
      JKMASK[13] :=   $FFF;        JKTMASK[13] :=   $FFF;
      JKMASK[14] :=  $1FFF;        JKTMASK[14] :=  $1FFF;
      JKMASK[15] :=  $3FFF;        JKTMASK[15] :=  $3FFF;
      JKMASK[16] := $7FEF;         JKTMASK[16] := $7FEF;
      JKMASK[17] :=  $FFFF;        JKTMASK[17] :=  $FFFF;
               ●
               ●
               ●
↑*************************************************************************
*                  INITIALIZATION OF LINE TYPE TABLES                    *
**************************************************************************
↓
↑↓
   FOR /I:=NOLDIAG TO NOLAST DO
   FOR /I1 := 3 TO NKRC3OUT DO                ↑ PRIME ALL SECOND WDS     ↓
   NBLTYT [/I,/I1].NBINT2 :=$FFFF;
↑
*              ---- SET UP CLEAR AND DISABLE COMMANDS ----
↓
   FOR /I := NOLDIAG TO NOLAST DO
   BEGIN
     NBLTYT[/I,NKCLRL].NBINT2 := $0400;      ↑SET THE TERMINAL BUSY BIT ↓
     NBLTYT[/I,NKDISL].NBINT2 := $0400;      ↑TO BUSY OUT THE MODEM     ↓
   END;
↑↓
   ↑LINE TYPE 0 (NOLDIAG) USED FOR  ON LINE DIAGNOSTICS ONLY.LINE       ↓
   ↑CHARACTERISTICS ARE TAYLORED DINAMICALLY DURING EXECUTION           ↓
↑↓
   NBLTYT [NOLDIA,2      ].NBINT1:=MILTO; ↑ MODEM STATES PTR. TABLE ADDR↓
   NBLTYT [NOLDIA,NKINIL].NBINT1:=$0020; ↑ INIT.     SET (ISON)         ↓
   NBLTYT [NOLDIA,NKENBL].NBINT1:=$8840; ↑ ENABLE    SET (DTR, RTS, ISR)↓
   NBLTYT [NOLDIA,NKINPT].NBINT1:=$0200; ↑ INPUT     SET (ION)          ↓
   NBLTYT [NOLDIA,NKDOUT].NBINT1:=$0100; ↑ DIR.OUT   SET (OON)          ↓
   NBLTYT [NOLDIA,NKOBT ].NBINT1:=$8800; ↑ OBT       SET (RTS, DTR)     ↓
   NBLTYT [NOLDIA,NKINOU].NBINT1:=$0100; ↑ IN.AFT.O SET (OON) AND       ↓
   NBLTYT [NOLDIA,NKINOU].NBINT2:=$FDFF; ↑          RESET(ION)          ↓
   NBLTYT [NOLDIA,NKENDI].NBINT2:=$FDFF; ↑ TERM.INP RESET(ION)          ↓
   NBLTYT [NOLDIA,NKENDO].NBINT2:=$FEFF; ↑ TERM.OUT RESET(OON)          ↓
               ●
               ●
               ●
```

FOR..TO LOOP

NESTED FOR..TO LOOPS

Figure H-1. Partial MPEDIT Program (Sheet 4 of 6)

```
→
*****************************************************************************
             ** SET UP BUFFER AREA POINTERS **
*****************************************************************************
↓
→
                              ***  IDTBL ***

          IDTBL DEFINES A NETWORK TO THE CCP. IT MUST BE
          IDENTICALLY INITIALIZED IN EVERY NPU OF A NETWORK.

          IDTBL CONTAINS ONE ENTRY FOR EACH NPU IN THE NETWORK.
          EACH ENTRY IS A VARIABLE NUMBER OF WORDS FOLLOWED BY
          $7FFF AS A TERMINATOR. THE FINAL ENTRY IS FOLLOWED BY
          TWO CONSECUTIVE TERMINATORS.

          THE FIRST WORD OF AN IDTBL ENTRY IS THE NODE ID OF THE
          NPU IN QUESTION. IF THE NPU IS A LOCAL ONE, EACH ID
          ACCESSED VIA COUPLER IS CONTAINED IN FOLLOWING WORDS.
          FINALLY, THERE IS ONE WORD FOR EACH TRUNK CONNECTED
          TO THE NPU: THE LINK-REMOTE-NODE ID IS IN THE RIGHT
          HALF, AND THE PORT NUMBER IS IN THE LEFT HALF.
↓
       IDTBLP := PIDTBL;                    → PASCAL IDTBL POINTER.        ↓
      /IDTBL := PIDTBL - 1;                 → EDITING IDTBL POINTER.       ↓
      /L := 0;


/L := /L + 1;  /IDTBL + /L := $0016;
/L := /L + 1;  /IDTBL + /L := $0209;
/L := /L + 1;  /IDTBL + /L := $030C;
/L := /L + 1;  /IDTBL + /L := $040D;
/L := /L + 1;  /IDTBL + /L := /ENDE;
/L := /L + 1;  /IDTBL + /L := $000B;
/L := /L + 1;  /IDTBL + /L := $0000;
/L := /L + 1;  /IDTBL + /L := $0001;
/L := /L + 1;  /IDTBL + /L := $0003;
/L := /L + 1;  /IDTBL + /L := /ENDE;
/L := /L + 1;  /IDTBL + /L := $000C;
/L := /L + 1;  /IDTBL + /L := $0000;
/L := /L + 1;  /IDTBL + /L := $0001;
/L := /L + 1;  /IDTBL + /L := $0004;
/L := /L + 1;  /IDTBL + /L := /ENDE;
/L := /L + 1;  /IDTBL + /L := $000D;
/L := /L + 1;  /IDTBL + /L := $0000;
/L := /L + 1;  /IDTBL + /L := $0001;
/L := /L + 1;  /IDTBL + /L := $0005;
/L := /L + 1;  /IDTBL + /L := /ENDE;
/L := /L + 1;  /IDTBL + /L := /ENDE;

      B3SBUF := /L + /IDTBL + 1;  →INCREMENT AVAILABLE CORE PTR         ↓
→
                 ●
                 ●
                 ●
→↓
      →*****************************************************
      *                                                   *
      *     SET UP BASE TCB FIELD DESCRIPTOR TABLE        *
      *                                                   *
      *****************************************************↓
→↓
      DOTCBFDT := DGTCBFDT;
→↓
      DGTCBFDT[0].DDNUMENT := 60;          →NO. OF ENTRIES              ↓
→↓
      /BSTCLASS              := 5;
      DGTCBFDT[ 5].DDFSTRT := /START(BSTCLASS);
      DGTCBFDT[ 5].DDFLNTH := /LENGTH(BSTCLASS);
      DGTCBFDT[ 5].DDFDISP := BSTCLASS;
→↓
      /BSOWNER               := 12;
      DGTCBFDT[12].DDFSTRT := /START(BSOWNER);
      DGTCBFDT[12].DDFLNTH := /LENGTH(BSOWNER);
      DGTCBFDT[12].DDFDISP := BSOWNER;
→↓
      /BSCN                  := 13;
      DGTCBFDT[13].DDFSTRT  := /START(BSCN);
      DGTCBFDT[13].DDFLNTH := /LENGTH(BSCN);
      DGTCBFDT[13].DDFDISP := BSCN;
```

Figure H-1. Partial MPEDIT Program (Sheet 5 of 6)

```
*↓
      /DN                  := 14;
      DGTCBFDT[14].DDFSTRT := $F;
      DGTCBFDT[14].DDFLNTH := 7;
      DGTCBFDT[14].DDFDISP := BSLLCB;
*↓
      /SN                  := 15;
      DGTCBFDT[15].DDFSTRT := 7;
      DGTCBFDT[15].DDFLNTH := 7;
      DGTCBFDT[15].DDFDISP := BSLLCB;
*↓
      /BSIPRI              := 19;
      DGTCBFDT[19].DDFSTRT := /START(BSIPRI);
      DGTCBFDT[19].DDFLNTH := /LENGTH(BSIPRI);
      DGTCBFDT[19].DDFDISP := BSIPRI;
                 ●
                 ●
                 ●
      CBTIMTBL[COHLIP].CBINTVAL := 2;
**********************************************************************↓
**END APPLICATION UNIQUE EXECUTION STATEMENT SOURCE**↓
**********************************************************************↓   END OF ASSIGNMENT SECTION
END.
```

Figure H-1. Partial MPEDIT Program (Sheet 6 of 6)

```
                                    CYBER MINI CROSS SYSTEM - LINK EDITOR -

                                           MEMORY IMAGE FILE DUMP

            ***0   ***1   ***2   ***3     ***4   ***5   ***6   ***7     ***8   ***9   ***A   ***B     ***C   ***D   ***E   ***F


HEADER
  0000    [:              0142   5239     3620   2020   4343   5020     5641   5249   414E   5420     4C4F   4144   204D   4F44
  0010      554C   4520   2020   2020     2020   2020   2020   2020     2020   2020   2020   2020     2020 2020:]
R96

HEADER
  0000    [:0040   4000          5A45     524F   5820   2020   2020     2020   2020   2020   2020     2020   2020   2020   2020
  0010      2020   2020   2020   2020     2020   2020   2020   2020     2020   2020   2020   2054     4552 4F20:]
ZEROK
  0000    [:1400   F671
  0010
  0020
  0030                                                                                                                     :]

HEADER
  0000    [:000E   401E   F671   4245     4749   4E58   2020   2020     2020   2020   2020   2020     2020   2020   2020   2020
  0010      2020   2020   2020   2020     2020   2020   2020   2020     2020   2020   2020   2042     4547 494E:]
BEGINX
  F670           [:0401   C000   00FC     0402   C000          0403     C000   0040   0404   C000            1400 F659:]

HEADER
  0000    [:0040   4000   0100   5042     494E   5452   4E54   4552     5255   5054   2054   5241     5053   2054   4142   4C45
  0010      2028   4C4F   4144   2041     5420   2431   3030   2920     2020   2020   2020   2049     4E54 5241:]
PBINTR
  0100    [:       1400   1F9B                   1400   1FC4                   1400   52C9                   1400   52E7
  0110             1400   56A3                   5400   4CFC   0E14            5400   4CFC   0E18            1400   56AF
  0120             1400   5306                   1400   56BB                   1400   56C7                   1400   56D3
  0130             1400   560F                   1400   56EB                   1400   56F7                   1400   5703     :]

HEADER
  0000    [:0010   4000   0140   4A55     4D50   5320   5441   424C     4520   434F   4E54   4149     4E53   204A   554D   5053
  0010      2054   4F20   2020   2020     2020   2020   2020   2020     2020   2020   2020   204A     554D 5053:]
JUMPS
  0140    [:1400          1400   8093     1400   F671                                                                      :]

HEADER
  0000    [:001E   4000   0150   4144     4452   4553   4E20   434F     4E54   4149   4E53   2054     4845   2041   4444   5245
  0010      5353   4553   2F43   4F4E     5445   4E54   5320   4F46     2020   2020   2020   2041     4444 5245:]
ADDRES
  0150    [:115A   1260   1099   109A     1287   0DC4   1774   1296     18C9   0DEC   18E9            0DA7
  0160             17AC   1813   1902            1A6C          0032                   0096   000F              :]
```

Figure H-2. Sample Memory Image Load Module Hexadecimal

INDEX

COMMENT SHEET

MANUAL TITLE: CYBER Cross System Version 1 Build Utilities Reference Manual

PUBLICATION NO.: 60471200

REVISION: G

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments on the back (please include page number references).

_____ Please reply   _____ No reply necessary

FOLD   FOLD

CUT ALONG LINE

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS   PERMIT NO. 8241   MINNEAPOLIS, MN.

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA

Publications and Graphics Division
Mail Stop: SVL104
P.O. Box 3492
Sunnyvale, California 94088-3492

FOLD   FOLD

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE   TAPE