60471200 F

020

2.0
(2.1 = G)

CONTROL DATA CORPORATION

CYBER CROSS SYSTEM
VERSION 1
BUILD UTILITIES
REFERENCE MANUAL

CONTROL DATA®
NOS 1
NOS/BE 1

60471200

CONTROL DATA CORPORATION

CYBER CROSS SYSTEM VERSION 1 BUILD UTILITIES REFERENCE MANUAL

CONTROL DATA®
NOS 1
NOS/BE 1

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released
(4/76)	
B	Manual Update (ECO 06420)
(6/76)	
C	Corrections on pages 3-2 and 3-11
(11/77)	
D	Manual revised to incorporate CYBER Cross NOS R6.
(8/79)	
E	Manual revised to change above references from R6 to R5.
(2/80)	
F	Manual revised to incorporate on-line console removal and all PSRs to level 528.
(10/80)	Manual title changed from CYBER Cross System, Version 1 Link Editor and Library
	Maintenance Programs Reference Manual. This revision obsoletes all previous editions.

Publication No.
60471200

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

Printed in the United States of America

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. Box 4380-P
Anaheim, California 92803

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	REV	PAGE	REV	PAGE	REV	PAGE	REV
Cover	-						
Title Page	-						
ii thru ix	-						
1-1 thru 1-3	F						
2-1 thru 2-6	F						
3-1 thru 3-3	F						
4-1 thru 4-11	F						
5-1 thru 5-10	F						
6-1 thru 6-10	F						
A-1/A-2	F						
B-1 thru B-12	F						
C-1 thru C-5	F						
D-1 thru D-3	F						
E-1/E-2	F						
F-1 thru F-4	F						
G-1 thru G-6	F						
H-1 thru H-4	F						
I-1 thru I-7	F						
Index-1 thru Index-7	F						
Comment Sheet	-						
Mailer	-						
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 five CYBER Cross System build utilities which aid in generating load files for a CONTROL DATA 255X Network Processor Unit (NPU). Such a load file contains the on-line system for an 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.

These utilities are run on a CYBER host (6000, CYBER 70 series, or CYBER 170 series) under the NOS or NOS/BE operating systems. It is assumed that the reader is familiar with the command structure of these operating systems.

The utilities described in this manual are:

- The library maintenance utility (MPLIB) used with CCI that 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 Expand utility used with CCP that has two functions:

  (1) It allows a user to generate a variant input for the application programs and hardware of NPU. (See the glossary for the use of the term application.)

  (2) It provides variant inputs for the NOS load-file generator utility.
- The Autolink utility used with CCP generates input directives for the Link utility.
- The Link utility (MPLINK) 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) 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 these utilities 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 macroassembler
- The CYBER Cross microassembler

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

- Installation procedures using a NOS or NOS/BE operating system.
- The CYBER Cross version of the PASCAL compiler.
- The CYBER Cross macroassemblers and microassemblers.

CONVENTIONS USED

Throughout this manual, the following conventions are used in the presentation of statement formats, commands and request formats, operator type-ins, and diagnostic messages:

ALN	Uppercase letters indicate words, acronyms, keywords, or mnemonics required by the network software as input to it, or produced as output.
aln	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 (exception: 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.
___	An underscore indicates a control character (examples: carriage return is CR, line feed is LF, and space is SP). If a space is required in a command, it is indicated as SP. Spaces shown in a command are normally placed there only for convenience in separating the parameters; they are not required.

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

RELATED MANUALS

Additional information on the CYBER Cross system and the CCP and CCI installation procedures can be found in the following documents that are available from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota, 55103.

Publication Title	Publication Number
NOS Version 1 Reference Manual, Volume 1 of 2	60435400
NOS Version 1 Installation Handbook	60435700
Update Version 1 Reference Manual	60449900
NOS Version 1 System Maintenance Reference Manual	60455380
Network Products NAM Version 1 Network Definition Language Reference Manual	60480000
NOS/BE Version 1 Reference Manual	60493800
NOS/BE Version 1 Installation Handbook	60494300
CYBER Cross System PASCAL Compiler Reference Manual	96836100
CYBER Cross System Microassembler Reference Manual	96836400
CYBER Cross System Macroassembler Reference Manual	96836500

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

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

This manual describes five 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 normally used in conjuction with the standard CCP or 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 program (MPLIB) used only with CCI
- A system variant handler (Expand) used only with CCP
- An Autolink auxiliary for the Link utility used only with CCP
- A Link utility (MPLINK) used with both CCP and CCI
- An Editor utility (MPEDIT) used with both CCP and CCI

All the utilities execute on a CYBER host computer operating under 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 (MPLIB)

This utility uses object code 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 and a directory to all the modules on the libraries.

SYSTEM VARIANT GENERATOR (EXPAND)

Several hardware and software variables must be specified to customize a CCP load file for a given NPU configuration in a network. Expand has two parts: a variant definition section, and a section that aids the NOS load file generator utility in building the CCP load file.

As shown in figure 1-1, the variant definition section operates with a user-supplied file, called USERBPS, to generate a temporary new program library and a set of updated autolink directives.

The load file generator section supplies variant definitions to the NOS load file generator utility. That utility converts the initialized CCP memory image load file module produced by the link Editor into a CCP load file that can be downline loaded from the host into an NPU. As shown in figure 1-1, the initialized load module file is one of the inputs used to produce the load file for CCP.

AUTOLINK

Autolink uses a set of input directives to generate MPLINK input directives for CCP. Autolink simplifies module assignment and maximizes the amount of space that is assigned to message processing buffers.

Autolink also gives information about the amount of buffer space that would be available for various combinations of application packages (such as combinations of TIPs) located in an NPU with a given memory size.

NOTE

Application programs are defined for CCP and CCI as well as for the host. In this manual, an application program (called an application) executes in the NPU. (See the glossary for the use of the term application.)

LINKING UTILITY (MPLINK)

MPLINK assigns space, and links together all the modules that are to used in a build operation, that is, 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.

EDITOR (MPEDIT)

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

INPUTS TO THE UTILITIES

There are two types of inputs to the utilities:

- Directive files. These directives are normally arranged into a batch input file.
- Other files. The most important of these files consist of object code modules which are built into the on-line CCP or CCI system.

GENERAL COMMAND FORMAT FOR THE UTILITIES

The command inputs to the utilites are in the form of directives. The general form of any directive is a command identifier (keyword) followed by a set of parameters:

KEYWORD, param 1, param 2, ..., param n

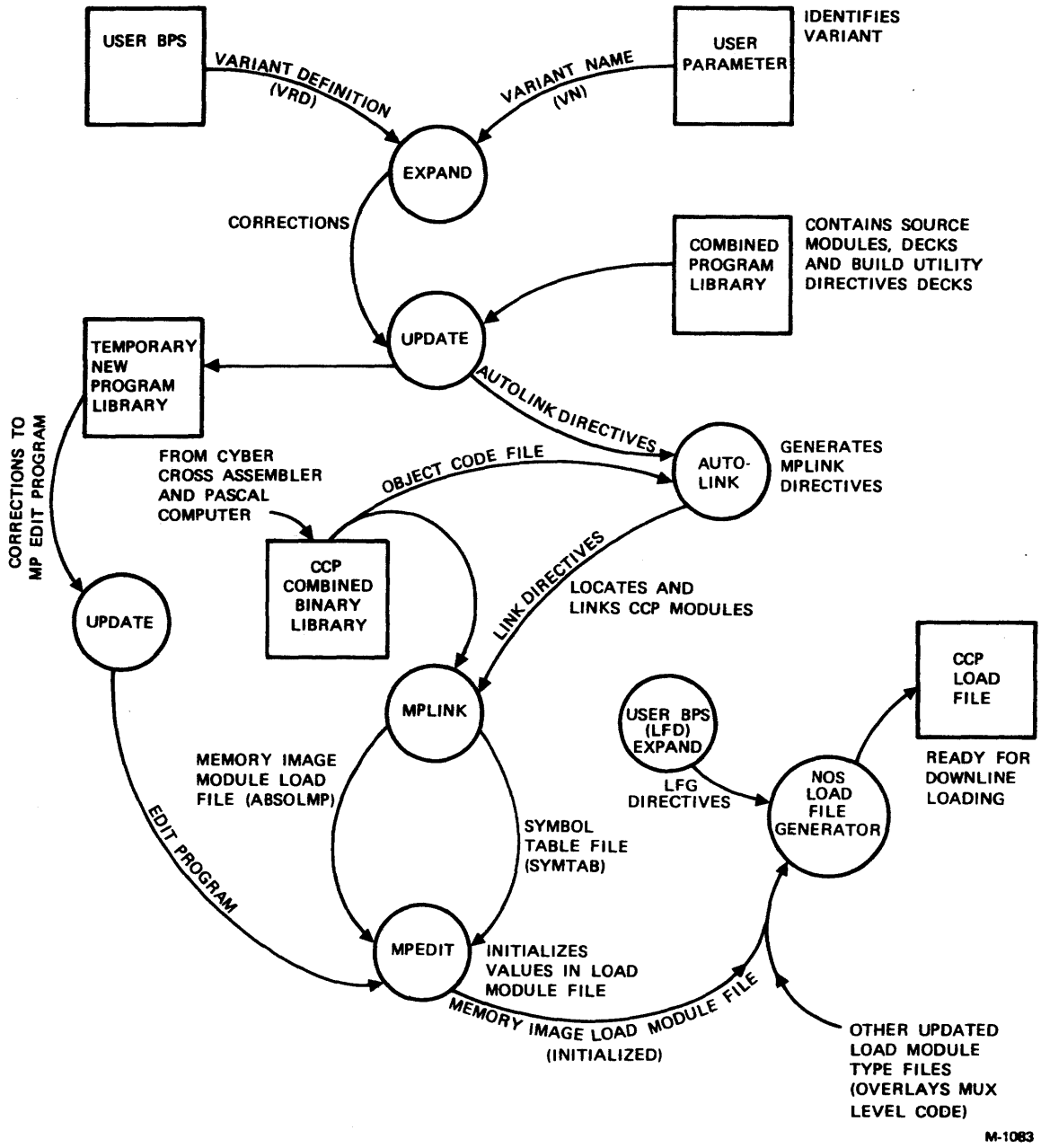


Figure 1-1. Producing a CCP Downline Load File

Parameters are separated by commas. Parameters usually specify values (such as the size of an 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 macroassembler, or by the CYBER Cross microassembler. At the time of use, 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, or selected records read from a program library.

OUTPUTS FROM THE UTILITIES

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

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

Optional outputs from various utilities include load maps, listings of all modules, buffer space reports, 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 directory to the programs on the library, together with information about these programs.

INTRODUCTION

The library maintenance program, MPLIB, can be used with CCP or 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. Two files are normally used:

- An object code file which contains the object code for modules that are to be added to the new library, or which are to replace existing object code modules by the same name which are already on the old library. This file is required. If there is no old library, the modules on the file create a library.
- The old library file. This file, though optional, is normally always present.

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

MPLIB INPUTS

The library maintenance program requires two files: a directives file and the object code file which contains the modules used to generate the library. If a library has already been built, MPLIB also requires the old library file.

There are also several calling parameters that 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 in which they are to be executed. The directives are of three types:

- Those which cause modules to be selected for the library, or deleted from an existing library.
- A directive that allows the operator to order a listing of the new and/or the old library.
- A directive to 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:

- Are used to construct 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. Note that once a library file has been created, it cannot be modified by MPLIB. The method of modifying a library file is to create 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 provides two output files: the new library file and an optional listing of the new and/or 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 which the directives have deleted.

Default name of the new library is NEWLIB.

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 a 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.

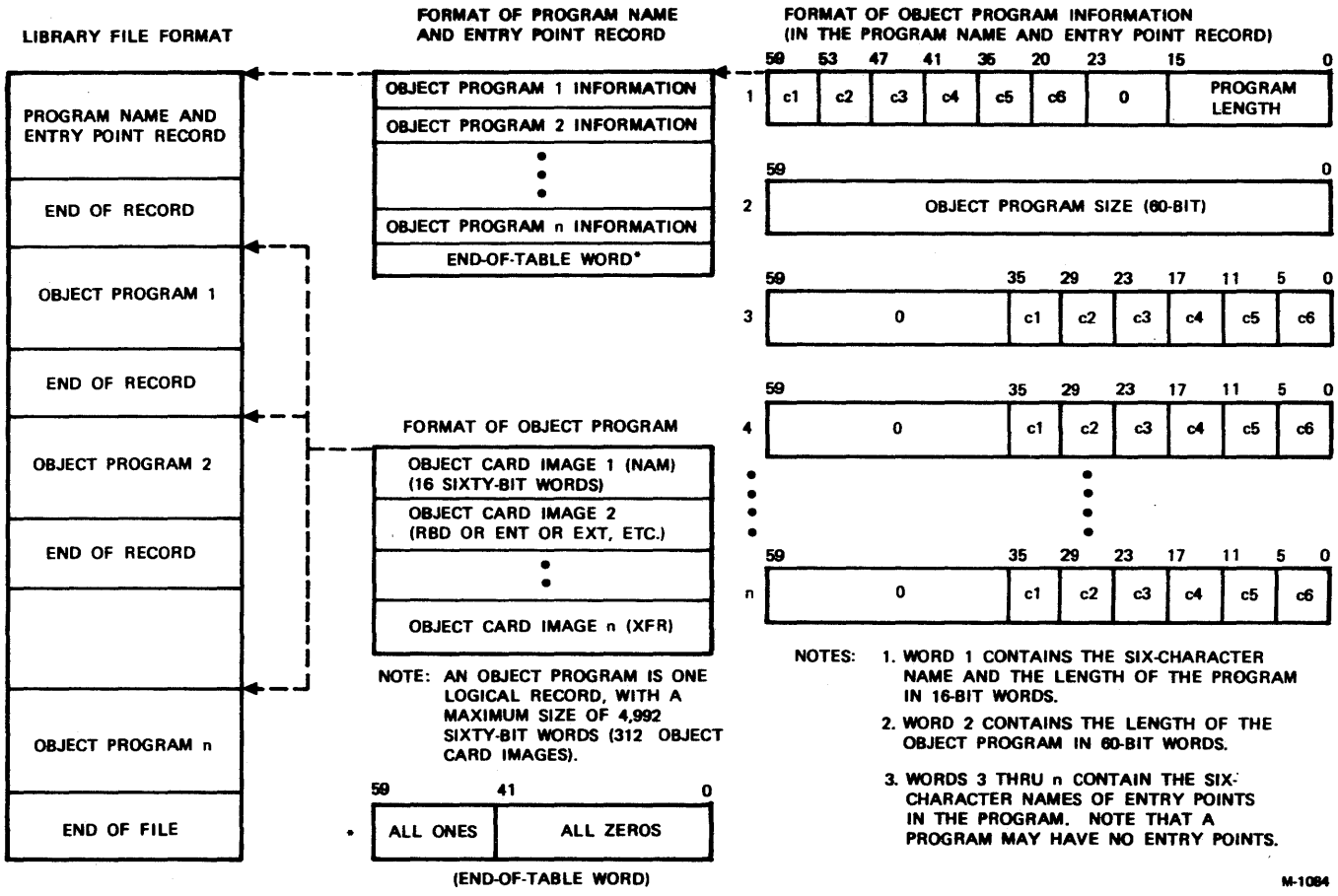


Figure 2-1. Format of an MPLIB Library File

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 are associated with the MPLIB name call statement. All of these parameters specify files. 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); and lfn5 is the name of 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. For example:

MPLIB.

This call produces:

- a new library, NEWLIB (lfn5)
- from an old library, OLDLIB (lfn 4), and an object code input file, LGO (lfn1)
- as commanded by the directives file, INPUT (lfn2)
- with any listings output to the listing file, OUTPUT (lfn3).

Example 2:

MPLIB,,,OLDOBJ,NEWOBJ.

This call uses default parameters for the object and directives files, and for the listing file. It names the new and old libraries, however. The call produces:

- a new library, NEWOBJ (lfn5)
- from an old library, OLDOBJ (lfn 4), and an object code input file, LGO (lfn1)
- as commanded by the directives file, INPUT (lfn2)
- with any listings output 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    CC4C    ZEROX          PBPSWI  0013  PBPSWI                    BIV02C
                                                                       BIVC40
BEGINX   C00E    BEGINX         PBPUTP  0017  PBPUTP                    BIV050
                                                                       BIC230
PRINTR   C04C    PRINTR         PBGETP  0011  PBGETP                    BIC250
                                                                       BIC400
JUMPS    CC1C    JUMPS          PBSTPM  0013  PBSTPM                    BIC41C
                                                                       BIC420
ADDRES   C01A    CIPADD         UNLOCK  000F  UNLOCK                    BIC43C
                 CCPLEV                                                BINTXT
                 CCPCYC         OULOCK  0011  OULOCK                    BINCC5
                 ADDRLC                                                BIN110
                 ADDRES         PRRTCD  0CCR  PRRTCD                    BIN14C
                 CCPVER                                                BIN18C
                 ADDRSU         QENTRY  0C4B  QENTRY                    BIN44C
                                                                       BISTID
PBLN00   C029    PBLN00         QEXIT   001E  QEXIT                    BIST8I
                                                                       BISTRS
PBLN01   CC1F    PBLN01         BUFMAI  0C76  PBRELC                    BISTRC
                                              PRREL1
PBLN02   001F    PBLN02                       PBGET1   INSTA7  01A7     BIX10C
                                              PBRELZ                   BIX140
PBLN03   CC1F    PBLN03                                                BIC100
                                 LISTSR  C028  PBLSGE                   BIC21C
PBLN06   002C    PBLN06                       PRLSPU                   BIY35C
                                                                       BICLAS
PBLN08   CC1F    PBLN08         PBSTOP  001C  PBSTOP                    BICCDN
                                                                       BICVER
PBGETC   CC06    PBGETC         PBSLJ   CCC7  PBSLJ                     BIBUTH
                                                                       BITERM
PBPUTC   CCC7    PBPUTC         PTTPIN  002A  PTTPIN                    BIC200
                                                                       BIN230
PBCALL   CCCE    PBCALL         PBSCLA  002F  PBSCLA                    BIN25C
                                                                       BIT10C
PBSTOR   CC22    PBSTOR         MCDMST  00CE  MILT8                     BIX130
                                              MILT9                    BIX15C
PBFILE   002P    PBCF                         MILTO                    BIX160
                 PBEF                         MILT4                    BIX17C
                                              MILT1                    BIX18C
PILPT    CC3C    PILPT                        MILT3                    BIX190
                                              MILT6                    BIX20C
PBAMAS   0012    PBAMAS                       MILTA                    BIX210
                                                                       BIX240
PBLMAS   000C    PBLMAS         ASMLSE  CCCC                           BIX250
                                                                       BIX300
PBCMAS   00CF    PBCMAS         HSPRUI  00CF  HBCC9                    BIX320
                                              HINSPT                   BIX350
PBSMAS   C00C    PBSMAS                       HTERM2                   BIX44C
                                                                       BIX45C
PBBEXI   000F    PBBEXI         HS2PRU  C14C  H2ISPT                    BIERRC
                                                                       BIT001
PBAEXI   000E    PBAEXI         HSPRUT  027D  HXTPT                    BIDC09
                                              HNTTPT
PBSETP   0022    PBSETP                       HIATPT   TPST78  01C1     B27TPT
                                              H2PTPT                   B37TPT
PBCLRP   000C    PBCLRP
                                 HSPMGS  000C  NAKHMS   BSCMSG  002D     ACKCMS
PBCOMP   001F    PBCOMP                       ACKHMS                   ACK1MS
                                                                       EOTBMS
PBRDPG   0010    PBRDPG         INPST7  021D  BISTAR                    WACKBM
                                              BIDISC                   TTDBMS
                                              BIENTR                   DISCBM
                                              BIE010                   ENQBMS
                                                                       NAKBMS
```

Figure 2-2. Sample MPLIB Library Listing

PDEXIT	0C4C	PDEXIT	PNTMLD	0098	PNTMLD
PDSTTR	02B2	PDSTTR	PNCNTL	0056	PNCNTL
PDTERM	0011	PDTERM	PNLCR	0026	PNLCR
CLCIAG	04BF	CLCIAG	PNSTAT	002F	PNSTAT
CLDMUX	0012	CLDMUX	PN1LNS	0077	PN1LNS
PIMSGL	003A	PIMSGL	PN2LNS	005C	PN2LNS
PIMSGA	007D	PIMSGA	PNLNST	00B7	PNLNST
PIDELM	0C4B	PIDELM	PN1TML	009F	PN1TML
PITYML	0007	PITYML	PNTMLS	009A	PNTMLS
PITYTI	054F	PITYTI	PNBRDC	0138	PNBRDC
PITYTC	0CCE	PITYTC	PN1BRD	0CE7	PN1BRD
PNAWAI	001F	PNAWAI	PNCVLD	0118	PNCVLD
PNBIN	00CA	PNBIN	PNCVLC	0099	PNCVLE
PNSMBA	0056	PNSMBA	PNCVLT	001A	PNCVLT
PNLNBA	0C4D	PNLNBA	PNFRCE	0027	PNFRCE
PNRVBS	0021	PNRVBS	PNPSTA	00A2	PNPSTA
PNTCCL	0037	PNTCCL	PNPSTA	0129	PNPSTA
PNCBEL	0035	PNCBEL	PNSMGE	0125	PNSMGE
PNGTCB	0C4C	PNGTCB	CDEBUG	0021	CDEBUG
PNTCBS	00E6	PNTCBS	PBCLR	002F	PBCLR
PNDLTC	005F	PNDLTC	PBDISP	00EE	PBDISP
PNDISC	0009	PNDISC	PBLOAD	0C4A	PBLOAD
PNSMTR	00CA	PNSMTR	PBILL	0007	PBILL
PNSMWL	0171	PNSMWL	PPHALT	0101	PPHALT
PNSMDI	00E4	PNSMDI	PBMEM	00E3	PBMEM
PNCONF	0254	PNCONF	TCTIME	0011	TCTIME
PNLNCN	01AF	PNLNCN	TCSTAR	00CB	TCSTAR
PNTMLC	025C	PNTMLC	TCSTOP	00CB	TCSTOP
PNDELE	00D9	PNDELE	PIDTBL	0081	PIDTBL
PNENAB	006F	PNENAB	MAIN$	0018	MAIN$
PNDISA	01C5	PNDISA			
PNLINE	00F5	PNLINE			

Figure 2-2. Sample MPLIB Library Listing (Contd)

MPLIB DIRECTIVES

There are six MPLIB directives. Four of them are directly concerned with chosing the programs that form the new libary. 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.

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 column 1 of a Hollerith coded card or card image.

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

*ALL, ADD ALL THE OBJECT CODE FILE PROGRAMS TO THE NEW LIBRARY

The *ALL directive causes MPLIB to copy all of 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 package) to an existing library.

Format of the directive is:

*ALL [.]

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

The *PUT directive has two formats:

- One form adds a single program (module) from the object code file to the new library file.
- The other form adds a sequence of programs from the object code file to the new library 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 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.

Note that 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. There are two forms of the directive. 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.)

Note that 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 in conjunction 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. There are two forms of the directive. 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 multiprogram 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.

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

*LST, LIST A LIBRARY

The *LST directive is used to request a listing of the new library or a listing of the old library. Formats for these requests are:

*LST,OLD[.] for the old library listing

*LST,NEW[.] for the new library listing

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

*END, END THE LIBRARY BUILDING OPERATION

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

*END[.]

MPLIB ERROR MESSAGES

Library maintenance error messages are listed in table B-1 of appendix B. The action that should be taken when these messages appear is also given in that table.

INTRODUCTION

Expand is a special purpose utility which provides a simplified user interface to the CCP installation process.

Two directives are used to describe the NPU variants (see glossary) and the CCP downline load file.

The CCP user provides an indirect access permanent file called USERBPS which contains the CCP variant definitions for the system to be installed. This utility expands the USERBPS configuration definitions into a set of directives used by the appropriate build step. Two types of definitions are found on USERBPS:

- VRD, a variant definition. This expands corrections to the update input decks (see figure 1-1 and the NOS Installation Handbook).
- LFD, a load file definition. This expands the Load File Generator utility directives for the load file generation build step (see figure 1-1 and the NOS Installation Handbook).

EXECUTING EXPAND

Expand is called automatically by the appropriate CCP installation build steps. The user must, however, know the form of the variant definitions entered in USERBPS.

The variant parameters file (USERBPS) can contain any number of CCP variant and load file definitions. The user must supply a variant name (see figure 1-1) for each variant to be processed.

During the variant build step, the user specifies the variant name. The Expand utility is automatically called to search USERBPS for the specified variant name. The utility uses the associated definition to generate a set of Update directives. These expanded definitions are used to generate input for the Autolink and MPEDIT utilities, which are described elsewhere in this manual. Note that CCP installation build steps also call Autolink and MPEDIT automatically.

SYNTAX OF EXPAND VARIANT DEFINITION PARAMETERS

The general definition format is:

keyword=type, parameter1=value1,...,parametern=valuen.

Some parameters optionally take multiple values. Multiple values are separated by slashes.

The following rules apply to all USERBPS definitions:

- The keyword identifies the type of definition (VRD or LFD).
- Blanks are ignored.
- All definitions terminate with a period.
- Omitted parameters do not need delimiting commas.
- If a definition requires two or more lines, the first nonblank character of each continuation line is a plus sign (+).
- All values for a parameter must appear in a single line.
- The first nonblank character of a comment line is an asterisk (*).

VRD, DEFINES A CCP VARIANT

Format of the VRD definition in USERBPS follows. Note that the order of the last three parameters must be as shown (NP, CP, and TR).

```
VRD=vn, VT=v1[/v2/v3], SZ=xK, TS=[t1/t2/.../t]n, NL=n,
+   NP=id, CP=id, TR=pa1-id1/pa2-id2/..../pan-idn.
```

VRD identifies the entry as a variant definition and specifies the variant name. The name is a three-character string used by the CCP variant build step to create unique, permanent file names.

VT specifies variant type. One value is required; two others are optional:

- v1 - is required, its values can be F, L or R where
  F indicates a front-end 255X; the unit includes a HIP but no LIP.
  L indicates a local 255X; the unit includes a HIP and a LIP.
  R indicates a remote 255X; the unit includes a LIP but no HIP.
- v2 - is optional; its value is D indicating the variant should include on-line diagnostic support modules.
- v3 - is optional; its value is T indicating the variant is a test build. A test build includes modules for TIPDEBUG, TESTGEN, TUP (the variable TOTUP is set to 1), and CONSOLE.

Examples of VT parameters are:

- VT=L/D/T Specifies a test build for a local NPU with diagnostics.
- VT=F/D Specifies a normal build for a front end NPU with diagnostics.
- VT=R Specifies a normal build for a remote NPU without diagnostics.

SZ specifies the variant memory size. Allowable values are:

65K	65,536 words of memory
81K	81,920 words of memory
96K	98,304 words of memory
128K	131,072 words of memory

The SZ memory size must not conflict with the corresponding NPU definition in the network definition language, NDL (see the Network Definition Language Reference Manual).

TS specifies the TIPs to be included in this variant. TS can assume up to ten independent values.

- A indicates the ASYNC TIP is included.
- E indicates the ASYNC extended option (ASYNCEXT) is included.
- H indicates the HASP TIP is included.
- M indicates the Mode 4 TIP is included.
- X indicates the X.25 TIP with PAD subTIP is included.
- B is reserved for future use.
- 1 indicates a User1 TIP is included (TIP provided by user).
- 2 indicates a User2 TIP is included (TIP provided by user).
- 3 indicates a User3 TIP is included (TIP provided by user).
- 4 indicates a User4 TIP is included (TIP provided by user).

Examples of the TS parameter are:

TS=A/E/M	NPU has the Mode 4 TIP, and the ASYNC TIP with extentions.
TS=H/X	NPU has the HASP TIP and the X.25 TIP with PAD subTIP.
TS=A/E/M/X/1	NPU has both normal and extended versions of the ASYNC TIP, the Mode 4 TIP, the X.25 TIP with the PAD subTIP, and one user-written TIP designated as user TIP 1.

NL specifies maximum number of 255X lines (ports) to be configured for this variant. The associated nl decimal value ranges between 1 and 254.

NP specifies the NPU node ID for this NPU. The associated np decimal value ranges between 2 and 255. The value must not conflict with the corresponding NPU definition in the network configuration file, NCF, which was generated during installation by network definition language directives.

CP specifies the host coupler decimal ID if the variant type is front-end (F) or local (L). CP is omitted for variant type remote (R). The cp value must not conflict with the corresponding coupler statement in the network configuration file, NCF, which was generated during installation by network definition language directives.

TR specifies the trunks if the variant type is local (L) or remote (R). TR is omitted for variant type front-end (F). Two values are required for each trunk: port address (pa) in hexadecimal, and the three-character variant name of the NPU at the other end of the trunk. All trunks are specified by one TR parameter. Value pairs are separated by slashes (/). An NPU can have one to eight trunks. For example:

TR=1-RM1/2-RM2/3-RM3

specifies a local variant with three trunks. These are connected at ports 1, 2, and 3 to remote NPUs RM1, RM2, and RM3 respectively.

EXAMPLES OF VRD DEFINITIONS

Example 1:

VRD=EX1, VT=L/D, SZ=81K, TS=A/M, NL=100, NP=11, CP=2, TR=1-RM1/2-RM2.

This defines:

- A local NPU with 81,920 words of memory having an NPU ID of 11 and a coupler node ID of 2.
- This normal build includes two TIPS (Mode 4 and normal ASYNC) and on-line diagnostics.
- Two remote NPUs are attached. Remote NPU RM1 is connected through trunk port 1. Remote NPU RM2 is connected through trunk port 2.
- No more than 100 lines can be connected to this NPU.

Example 2:

VRD=EX2, VT=R/T, SZ=81K, TS=A/E/H/X, NL=127, NP=23, TR=3-L81.

This defines:

- A remote NPU with 98,304 words of memory having an NPU ID of 23.
- This test build includes four TIPs: HASP, X25 with PAD subTIP, and both normal and extended versions of ASYNC. On-line diagnostics are omitted.
- This remote NPU is connected on trunk port 3 to a local NPU with variant name L81.
- No more than 127 lines can be connected to this NPU.

Example 3:

VRD=EX3, VT=F/D/T, SZ=128K, TS=A/E/M/H/X, NL=127, NP=30, CP=15.

This defines:

- A front-end NPU with 131,072 words of memory having an NPU ID of 30, and a coupler node ID of 15.
- No remote 255X units are connected to this NPU.
- This test build includes all five standard TIPs and on-line diagnostics.
- No more than 127 lines can be connected to this NPU.

LFD, DEFINES A CCP LOAD FILE VARIANT

The format of an LFD statement in USERBPS is:

LFD=gn, LM=vv1-p2lid1/..../vvn-p2lidn.

LFD identifies this entry as a load file definition, and specifies the load file name (gn). The gn value is a three-character string used by build step CCPGLF to create a unique permanent file name for the output file (see the NOS Installation Handbook).

LM specifies the CCP variant load modules to include in this load file. The multiplex subsystem firmware (phase 1) and dump bootstrap load modules are automatically included in every load file. The on-line diagnostics and remote 255X dump/load overlay load modules are automatically included if files ZDGN and/or ZREM are present.

The associated vvi-p2lidi value consists of two parts: (1) vvi is the three-character name of a variant load module (permanent file name = Zvvv) generated by the CCP variant build step; (2) p2lidi is the three character name specified for this variant as the phase 2 load ID in the corresponding NPU statement in the network configuration file.

One vvi-p2lidi value is required for each variant to be included in the load file. Successive vvi-p2lidi values are separated by slashes (/).

EXAMPLES OF CCP LOAD FILE DEFINITIONS

Example 1:

```
LFD=EX4, LM=EX1-N11/EX2-N23/EX3-N30.
```

defines a load file containing the variants created in the three VRD variant definition examples given previously.

NOTE

In LFD examples, p2lidi naming follows the load file generator convention: the name starts with the letter N and is followed by the two-digit NPU node ID.

Example 2:

```
LFD=EX5, LM=EX3-N30.
```

This defines a load file containing the variant shown in the third VRD example given previously.

EXPAND ERROR MESSAGES

All error messages generated by the Expand utility are written to the file specified by the fourth parameter of the Expand name call statement. If no file name is specified in that parameter, error messages are written to the file with the local name, ERMSGF. During a variant build or a load file generation, errors are written to the output file. The messages appear on the listing from that build step.

Each error message indicates the problem type (for example, a USERBPS file error or a macro text file error). In some cases, the erroneous text is also copied into the error message.

An error message has one to three lines:

- The first line is the text line. It contains the text passed to the Expand utility from the build procedure step.
- The second line specifies the primary USERBPS line of a variant or load file definition.
- The third line is an associated variant definition.

Some error messages omit one or more of these lines.

In some cases, an arrow occurs underneath the text line of an error message. Any text to the right of the arrow was not processed by Expand at the time it detected the error. For example:

```
LN = EX3
     ↑
```

In this case, the error occurred either in the symbol LN or in the equal sign. Expand had not processed the EX3 symbol.

The Expand utility error messages are shown in table B-2 of appendix B. The action to be taken when these messages appear is also given in that table.

INTRODUCTION

Autolink generates the MPLINK directives for CCP. It simplifies MPLINK use and provides an optimized MPLINK input directives file. By optimizing the location of executable modules in NPU, Autolink provides the maximum number of buffers. These buffers can be assigned for message processing purposes.

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.

CCP APPLICATION PROGRAM TYPES

A CCP application program (normally called an application) is a group of modules working together to perform some major system function. There are two types of CCP applications:

- Base applications that must be included in every CCP system. Most of these applications are located in an unpaged area in main memory. One of them, the service module, is located in the paged area of main memory. Parts of the service module that cannot fit on this page are located in the paged area of main memory.
- Terminal interface packages (TIPs). This category also includes the host interface package (HIP) and the link interface package (LIP). These applications are normally located in the paged area of extended memory, although individual application modules can be included in paged or unpaged main memory.

Table 4-1 relates the standard CCP applications names to the common names for these applications

The NPU memory is paged. Current page length for CCP is 8192 words. CCP uses one page located at address 2000_{16}; all other pages are imaged at this page, but are in fact located in extended memory (addresses higher than 65,536). Having all modules of an application located on the same computer page speeds processing. Autolink allows the user to specify that a module is to be loaded on the same page as its application; Autolink can locate other modules elsewhere to optimize buffer space or to avoid page overflow. Paging and the associated addressing problems caused by paging are discussed in detail in the Link utility section of this manual.

TABLE 4-1. CCP APPLICATION NAMES

Name	Common Name/Function
ASYNC	Asynchronous Terminal Interface Package
ASYNCEXT	Extended Asynchronous Terminal Interface Package
BASESYS	Base System Modules
CONSOL	NPU Console Package
HASPTIP	HASP Terminal Interface Package
HIP	Host Interface Package
LIP	Link Interface Package
INITIAL	Initialization Routines
IVT	Interactive Virtual Terminal Common Routines
MODE4	Mode 4 Terminal Interface Package
OLDSYS	On-line Diagnostics
PIBUF2	Buffer Space
PIDTBL	ID Table (this is always the last application specified)
SVMODULE	Service Module
TUP	Test Utilities Package
X25L2	X.25 TIP Level 2 (Frame Handling)
X25L3	X.25 TIP Level 3 (Packet Handling)
X25PAD	PAD SubTIP for the X.25 TIP
X25TIP	X.25 Terminal Interface Package

AUTOLINK INPUTS

There are two types of Autolink input files: directives and object code files for the directives to act upon. These are shown in figure 4-1.

Autolink Input Directives

The user specifies the linking operations to be performed by a series of input directives to Autolink (this is automatic if the standard system is used). The directives are contained on an input file. Four types of directives exist:

- Directives specifying the size and paging characteristics of the NPU memory. Reserved areas can also be specified (the characteristics of a reserved area are discussed later).
- Directives defining the applications and associated modules to be used in the build. These directives also specify whether the module must be loaded on the same page as the other modules of the application.

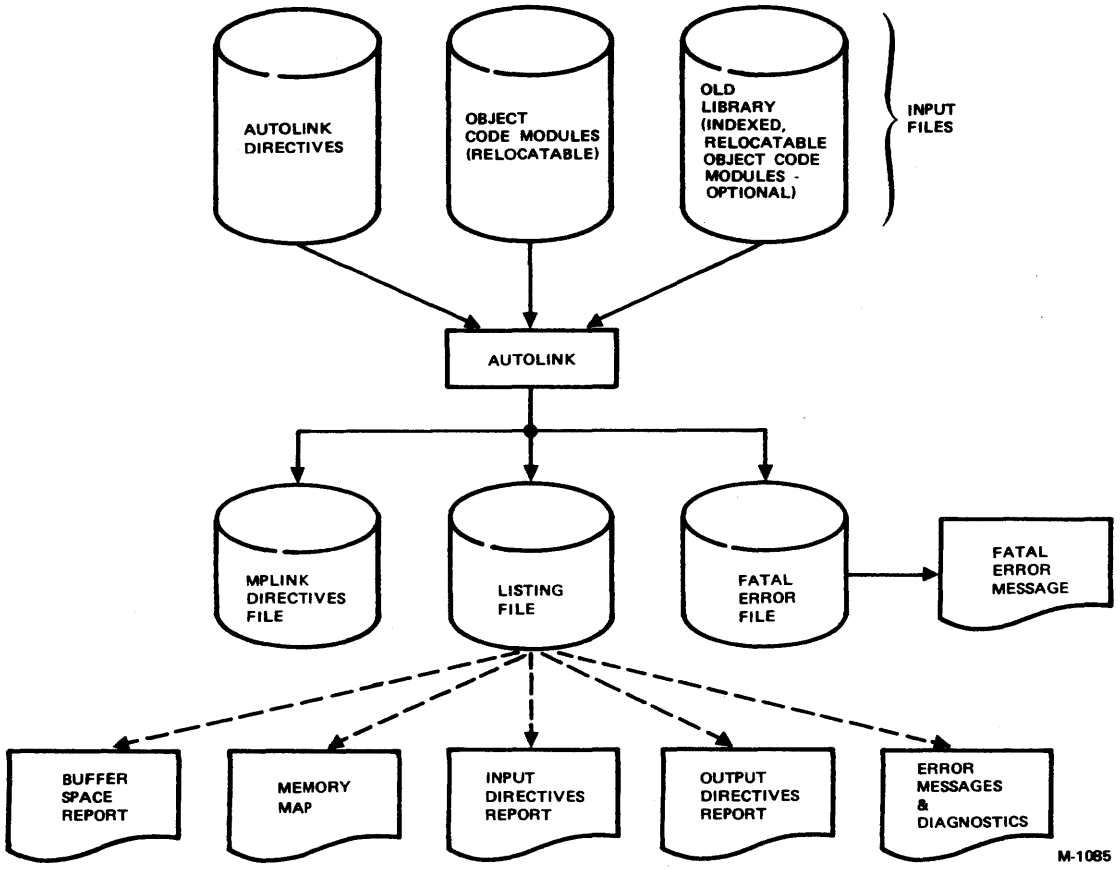


Figure 4-1. Autolink Logical Flow

- Directives defining the optional Autolink reports.
- Passive MPLINK directives. These directives are not processed by Autolink, but are saved and are copied into the output file Autolink generates for MPLINK.

Autolink Input Modules

Modules are input to Autolink from special object code files (see figure 1-1). These files can be the direct output of a Cross assembler or compiler, or a library file created by the CCP installation procedures. In either form, the object code is relocatable. The format of this relocatable code is given in appendix F.

OUTPUTS FROM AUTOLINK

The principal Autolink output is a file of directives for MPLINK. The other outputs are the Autolink reports and a fatal error file (see figure 2-1). The reports are:

- Buffer space report
- Output directives report
- A memory map
- An input directives report

EXECUTING AUTOLINK

The CYBER Cross System must be installed for Autolink to be executed. Autolink's name call statement is of the form:

```
ALKOVL(files)
```

If the user follows the installation procedures given in the NOS Installation Handbook, the input files necessary for Autolink's proper operation should be available at Autolink execution time.

AUTOLINK INPUT FILES

Inputs are presented to Autolink on three input files:

- The input directives file (required)
- An object code file which contains modules (required)
- An object code library file (optional)

INPUT DIRECTIVES FILE

This file contains the directives necessary to generate the MPLINK directives file. The MPLINK directives contain all control information needed by the Link utility to produce the memory image load module (ABSOLMP) file and the symbol table (SYMTAB) file. These two files are the required inputs to the Edit utility. See figure 1-1.

OBJECT CODE MODULES FILE

This file of object code modules contains all the modules that are to be linked. These modules are the output of a CYBER Cross System compiler or assembler.

LIBRARY FILE

This optional file contains the object code for modules that are grouped into a library. The function of this file is identical to that of the object code module file; the library provides a second source of object modules.

AUTOLINK OUTPUT FILES

There are four types of Autolink output files:

- Output directives file
- Fatal error message file
- Listings files
- Temporary file for saving passive MPLINK directives

The output directives file is normally left as an on-line mass storage file so that it can be used as the input to the MPLINK utility.

Information generated by fatal error processing is stored in the fatal error message file. If no fatal error occurs, this file is empty.

The user can select optional reports, the output directives file, or a listing file contain one or more reports.

A temporary file is used to save the passive MPLINK directives until they are needed in producing the output directives file. The contents of the file are not available to the user.

OUTPUT DIRECTIVES FILE

This file contains the MPLINK directives that are generated by the Autolink processing its own input directives. The passive directives that were used as inputs to Autolink are copied to the MPLINK directives file. The Autolink output file is used as the directives input to MPLINK.

Any fatal error prevents generation of the output directives file.

LISTING FILE

This file contains any operator-requested reports, as well as any informative messages generated during Autolink processing. Informative messages are either warnings that indicate non-fatal input directive error, or provide diagnostic information about the fatal error.

Fatal errors that occur during the MPLINK directives generation prevent reports; fatal errors that occur during report generation can result in a partial report. The partial report information is not reliable.

FATAL ERROR MESSAGE FILE

This file's message specifies the cause of the fatal error and contains diagnostic comments. Standard CCP installation procedures copy the fatal error file to the output file and normally terminate the variant build step.

EXECUTING AUTOLINK

Autolink is executed through the name call statement: ALKOVL(files). If the user follows the installation procedures given in the NOS Installation Handbook, the input files necessary for Autolink's proper operation should be available, and rewound at Autolink execution time.

Autolink's name call statement contains eight file names. The names must appear in the order given. If a name is omitted, its delimiting commas must remain. Format of the Autolink name call statement is:

ALKOVL(lfn1,lfn2,lfn3,lfn4,lfn5,lfn6,lfn7,lfn8)

where the file names are defined as follows:

lfn1 - the name of the file containing the input directives to Autolink. Default is TAPE1.
lfn2 - the name of the file containing the main object code input. Default is TAPE2.
lfn3 - the name of the file containing the object code library. This is the secondary object code input and is optional. Default is TAPE3.
lfn4 - the name of the output directives file. Default is TAPE4.
lfn5 - the name of the reports file. This file contains all reports and informative messages. Default is TAPE5.
lfn6 - the name of the provisional file that stores the passive MPLINK directives. Default is TAPE6.
lfn7 - the name of the fatal error file. The file is generated only in the case of a fatal error. Default is TAPE7.
lfn8 - the name of the error file. Default is TAPE8.

AUTOLINK DIRECTIVES

Autolink directives are of two types:

- Passive MPLINK directives that are not used by Autolink but are copied into the output directives file.
- Active Autolink directives, which are processed by Autolink to form an output directives file or a report, or both. Some of these directives specify the modules to be selected from the input object code files and how they are to be used in the build. Other directives specify the report or reports which Autolink is to generate.

PASSIVE MPLINK DIRECTIVES

Passive MPLINK directives allow the user to insert MPLINK directives directly; they are copied unaltered into the output directives file.

The group of passive MPLINK directives which Autolink can use is:

- *COM which defines the blank common area
- *COR which defines the size of the NPU memory to be used for the build
- *DAT which defines the labeled common area for PASCAL tables
- *DSTK which defines the PASCAL reentrant/recursive routines stack area used by MPLINK
- *DMP which orders a hexadecimal dump of the load file image
- *DVAR which defines the dynamic area for PASCAL variables
- *ENT which defines an entry point for a module
- *LIB which defines the library file used to resolve unsatisfied externals
- *SYN which defines a synonym for a module name or entry point
- *SYSID which defines the system name and header record for the load file

These directives are described in detail in the Link utility section of this manual. Other MPLINK directives cannot be used by Autolink.

ACTIVE AUTOLINK DIRECTIVES

Active Autolink directives are processed by Autolink and, in most cases, generate MPLINK directives for the output file. A directive consists of a directive name followed an equal sign and a parameter list:

DIRNAME = parameter list

Blanks are ignored.

Active Autolink directives are summarized in table 4-2.

The order of the directives is significant:

- APPL directives must be entered before MOD directives. APPL directives define the applications to be used in the build. The order of entering APPL directives with respect to one another has an effect which is explained in the Special Considerations for Using Autolink subsection. The last application must be PIDTBL.
- RPT=INFO turns on the input directives report. Any input directives entered ahead of this directive are omitted from the report; all directives following the RPT=INFO directive are included.
- MOD directives must follow all other directives. The order of entering MOD directives with respect to one another has as effect as explained in the Special Considerations for Using Autolink subsection.

TABLE 4-2. SUMMARY OF AUTOLINK DIRECTIVES

Directive	Description
APPL	Defines names of all applications that can be used in any system build.
BUFSPSIZE	Specifies the memory sizes to be used in the buffer space report; that is, several trial computer sizes can be used.
CORESIZE	Specifies the memory size of the NPU which is to receive the build.
DEF	Specifies the applications to be used in this build.
DEFBASE	Specifies base applications which must be used in every build.
MOD	Defines the module name, and associates the module with one or more applications.
PAGESIZES	Specifies the size of a page (currently, CCP cannot use more than one size).
PAGEREG	Specifies page register number.
RESERVE	Specifies reserved areas of main memory.
RPT	Specifies reports to be generated as part of Autolink run. rpx=BUFSP specifies the buffer space report. rpx=DIR specifies the output directives report. rpx=MAP specifies a main memory map. rpx=INFO specifies a list of the input directives and application lengths.

NOTE

If the order of the MOD directives is the same as the order of the modules in the object code input file, the execution speed of MPLINK is greatly increased.

Comments for Directives

Comments can appear anywhere in the input directives file. Comments are delimited in the usual PASCAL syntax fashion:

- The comment starts with an ASCII underscore (_). This is the same as the bent arrow (↵) in CDC graphics.

- The comment ends with an ASCII question mark (?). This is the same as the down arrow (↓) in CDC graphics.

APPL, Specifies Applications

The APPL directives identify the applications that comprise the system. An application consists of a group of modules performing a set of related functions. The group is identified by the single application name.

Only one application name is used in each application directive. Every application referenced by a MOD directive must have an APPL directive. No more than 60 applications can be defined.

The format of an APPL directive is:

APPL=appl [(ADDR = $nnnn)]

where appl is the user supplied name of the application. It consists of one to six letters and digits. ADDR is an optional parameter specifying a starting address ($nnnn) for the application in hexadecimal.

The last application directive to be entered must be PIDTBL. The order of entering other APPL directives with respect to one another is explained in the Special Considerations for Using Autolink subsection.

CORESIZE, Specifies the Memory Size of the Variant Build

This directive specifies the memory size of the variant build for the NPU. The format of the directive is:

CORESIZE=n

where n is the memory size. Allowable values are: 64K (65,536 words of memory), 80K (81,920 words of memory), 96K (98,304 words of memory), and 128K (131,072 words of memory). Default value is 128K.

DEF, Specifies the Applications to be Included in the Build

This directive allows the user to select applications for a variant build from the full set of applications specified by APPL directives. Any application specified by an APPL that is not included in a DEF directive is ignored. Format of the directive is:

DEF=appl

where appl is an application name. It must be the same as the name for the application used in an APPL directive.

DEFBASE, Specifies Base Applications that Must be Included in Every Build

CCP requires some applications in every build. These base applications must not appear in DEF directives. Base applications are not included as optional applications in the buffer space report; however, the size of DEFBASE applications is calculated during that report's generation. Format of the directive is:

DEFBASE = appl

where appl is an application name. It must be the same as the name used for the application in the APPL directive.

MOD, Specifies Where a Module can be Located during a Build

This directive specifies the type of location available to a module during a variant build. Autolink handles up to 600 MOD directives.

MOD directives must follow all other directives. Each MOD directive associates a module with at least one application. All modules used in the build must have a MOD directive.

Autolink generates an informative message if a MOD directive is issued for a module not available in one of the object code input files. Autolink generates a fatal error message if a MOD directive is issued for such a module belonging to a defined application.

Note that a module directive can be carried on two or more physical lines as defined in the syntax rules given previously.

The MOD directive consists of a name followed by a parameter list enclosed by parentheses. The parameters are separated by commas, and can appear in any order. Format of the MOD directive is:

```
MOD=modname(P=p,ADDR=addr,FILL,TH=tophat,
APPL=appl1/appl2/.../appln)
```

where modname specifies the name of the module as it appears in the object code file. Only the first six characters of the name are significant.

P - Specifies the type of paging to be used:
- P - indicates a pageable module. If a module is (1) pageable and (2) not specifically assigned a load address (ADDR parameter), then the module is located at the next available page address of the associated application, if possible. Otherwise, the module is located at the next available main memory address of its application.
- NP - indicates a non-pageable module. NP is the default value for the P parameter.
- F - indicates a module that is forced to reside with the paged portion of its associate application. An application cannot be assigned to a given page unless all its F-designated modules can fit on that page.
- R - indicates a reverse-loaded module; that is, space is reserved for the module based on a fixed ending address rather than a fixed beginning address. Unless otherwise specified, the ending address of the first R module in the directives file is the highest available main memory address ($FFFE_{16}$). Ending addresses of subsequent R modules start at a address one less that the beginning address of previous R module.

ADDR - specifies the module is to located at an absolute address.
- addr is the absolute hexadecimal ($aaaa) address. For R modules, addr is the ending address; for all other modules, addr is the starting address. A fatal error results if one ADDR module overflows into the region assigned to another ADDR module. If the ADDR parameter is omitted, Autolink allocates a location according the requirements of other parameters.

FILL - indicates that the module need not be loaded in association with its application; rather, the module can be used to fill empty locations throughout main memory or paged memory that result from specifying reserved areas and absolute addresses, after assigning space for other some other types of modules (the filling operation is perfomed after several other types of processing have already been performed). Only if P=P can a FILL module be used to fill empty space on a page.

TH - indicates a tophat module. A tophat module is normally a module that is called by several other modules. To minimize the code required to locate a tophat module's entry point, a small auxiliary piece of code is compiled 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.

tophat - is the name of the pageable module as it appears in the object text file. Only the first six characters are significant.

APPL - specifies the names of all associated applications. There must be at least one APPL in every MOD directive.

appl - is the name of an application as it appears in the APPL directive.

An example of a series of MOD directives is given in figure 4-2.

PAGEREG, Specifies Page Register Number

If specified, this directive indicates the decimal page register number to be used for extended memory addresses. If no PAGEREG directive is included in the set of autolink directives, no page registers are explicitly included in the MPLINK directives. Format of the directive is:

PAGEREG = n

where n is the decimal page register number. For the CCP system, page register 0 is used, and it designates the page starting at 2000_{16}.

PAGESIZES, Specifies the Size of a Page

This directive specifies the size of each page in pageable memory. Currently, CCP requires that all pages be of the same size. The format of the directive is:

PAGESIZES=s1

where s1 is the selected page size in K words (K=1024). The allowable values are 2K, 4K, 8K, and 16K. Default value is 8K.

RESERVE, Specifies a Reserved Area of Memory

This directive prohibits Autolink automatically locating modules in the specified area. However, Autolink can locate modules in a reserved area if an ADDR parameter in a MOD directive specifies that the module should be located in a reserved area. Each reserved area requires a separate RESERVE directive. Format of the directive is:

RESERVE=addrb,addre

where addrb is the lower boundary of the area in hexadecimal ($aaaa); and addre is the upper boundary of the area in hexadecimal ($aaaa).

BUFSPSIZE, Specifies Memory Sizes for the Buffer Space Report

This directive specifies the NPU memory size, or sizes, to be used in the buffer space report (see the form of the report in the Autolink Reports subsection). If BUFSPSIZE is not specified, the report is generated only for the memory size specified by the CORSIZE directive. Format of the directive is:

BUFSPSIZE=s1,s2,...sn

where si is the memory size to be included in the report. Allowable values are: 64K (65,536 words of memory), 80K (81,920 words of memory), 96K (98,304 words of memory), and 128K (131,072 words of memory).

RPT, Specifies an Autolink Report

This directive specifies a report to be generated by Autolink. Four reports are possible. Format of the directive is:

RPT=rpt

where rpt specifies one of the reports: BUFSP, DIR, MAP, or INFO. BUFSP specifies the buffer space report, DIR - specifies the output directives report, MAP - specifies a memory map report, and INFO - specifies the input directives report. The position of this card determines the content of the report: only directives that follow the card are included in the report.

AUTOLINK REPORTS

Autolink generates four optional reports, each of which can be selected by an RPT directive. If the BUFSPSIZE directive is used, it determines the size of memory for the buffer space report.

BUFSP, BUFFER SPACE REPORT

This report indicates the amount of main memory space available for assigning to buffers for a given NPU build. Buffer space is used for processing message traffic.

The report format is governed by the BUFSPSIZE directive. BUFSPSIZE specifies all the memory sizes that are to be used while generating the report. A report includes all combinations of CCP applications that can fit into a permitted build configuration. For each of these combinations and memory sizes, the report gives the amount of space available for buffers. Note that the base applications do not explicitly appear in the report; however, the space used by these applications is calculated in generating the report data.

Since the report requires considerable computation time, it should not be generated as a routine matter. A partial buffer space report is shown in figure 4-3.

```
MOD = PNSWML(P = F, ADDR = $2000, APPL = $VMODULE)
MOD = PNAWAIT (P = P, APPL = SVMODULE)
    .
    .
    .
MOD = PNDSTAT(P = NP, APPL = SVMODULE)
    .
    .
    .
MOD = PGDSTAT(P = P, FILL, TH = PNDSTAT, APPL = SVMODULE/BASESYS)
    .
    .
    .
MOD = PIDTBL(P = NP, APPL = PIDTBL)
MOD = GLOBL$(P = NP, ADDR = $0DA0, APPL = BASESYS)
    .
    .
    .
MOD = PIBUF2(P = R, ADDR = $FFFE, APPL = PIBUF2)
MOD = PBREAD(P = R, APPL = CONSOL)
    .
    .
    .
MOD = MAIN$(P = R, APPL = INITIAL)
    .
    .
    .
MOD = R4M4IN(P = NP, FILL, APPL = MODE 4)
MOD = R4M4CC(P = NP, FILL, APPL = MODE 4)
MOD = R4M4TP(P = P, APPL = MODE4)
    .
    .
    .
MOD = PTMD4TIP(P = F, APPL = MODE 4)
    .
    .
    .
MOD = PTASNOPS(P = F, APPL = ASYNC/ASYNCEXT)
MOD = R4ASYT(P = P, APPL = ASYNC/ASYNCEXT)
    .
    .
    .
MOD = R4ASYI(P = P, APPL = ASYNC/ASYNCEXT)
MOD = ASYMSG(P = P, APPL = ASYNC/ASYNCEXT)
    .
    .
    .
```

The first set of modules specified (which is also the order of the modules on the object code input file) is from the service module. The leading module specifies (P = F) that this module must reside on the main memory page (starting address = 2000_{16}) with other F designated service module routines. Since this is the first MOD statement, the beginning of this module is located at the beginning of the specified page. The second service module program can be paged anywhere.

PNDSTAT, also a part of the service module, is not pageable; therefore, it must be located in main memory other than at the main memory page (locations 2000_{16} through $3FFF_{16}$). The last service module program specified in the example is PGDSTAT. It can be paged, and must be vectored (using a tophat) to PNDSTAT. Since it is tied to PNDSTAT, that module cannot be pageable. The module can be used to fill holes in main memory, or on any page. Note that PGDSTAT is also a part of the base system application.

PIDTBL is the module belonging to the last application. It is nonpageable. PIDTBL must be loaded as the assigned space above the last main memory sequential application; that is, normally, it is the application just below the beginning of the buffer area.

GLOBL$ is a module that is assigned a specific address ($0DA0_{16}$). It cannot be paged, and is a part of the base system.

PIBUF2 begins the modules in the reverse loading sequence. The sequence ends with MAIN$.

PIBUF2 defines the vector to the area used for buffers. The vector is at the upper end of main memory. The actual buffer space exists below the reverse-loaded modules, and normally above the last of the sequentially loaded modules (the last application in this sequence is PIDTBL). The application name is PIBUF2.

PBREAD is also reverse-loaded. It is part of the console application package. The last address of PBREAD is one less that the first buffer word address.

MAIN$ is the last reverse-loaded module. Some of the reverse-loaded programs are not used during the normal on-line CCP processing, but are part of the initialization needed when CCP is configured after the downline load file is sent to the NPU. The buffer area is not determined until Autolink finishes its execution.

The Mode 4 TIP has both pageable and nonpageable portions. It starts with two nonpageable modules, R4MRIN and R4M4CC. Both of these can be used to fill holes in main memory. The first pageable Mode 4 module, R4M4TP, preferably will be located at the beginning of its own page, since it is the first nonservice module specified by a P = P MOD directive. Only one of the Mode 4 modules must be located on the same page as its other modules (P = F), that is, the Mode 4 worklist handler, PTMD4TIP.

The pageable Asynchronous and Asynchronous Extended TIP modules must be linked together on the same page, or in main memory. The TIP entry routine, PTASNOPS, indicates that is must reside on the same page as the other F modules (P = F) for these related TIPs. Some of the remaining modules are designated as pageable; others are required to be nonpageable.

Figure 4-2. Example of MOD Directives

```
*****************************************************************************
*       *       *       *       *       **       *       *       *       *
*MODE4 *HASPTI*HLIP   *ASYNC *ASYNC **    65   *    80   *    96   *   128   *
*       *       *       *       *       **       *       *       *       *
*****************************************************************************
*   X   *   X   *   X   *   X   *   X   **  ---   *  ---   * 13136 * 30974 *
*       *   X   *   X   *   X   *   X   **  ---   *  5454  * 21687 * 31825 *
*   X   *       *   X   *   X   *   X   **  ---   *  3610  * 19852 * 31183 *
*       *       *   X   *   X   *   X   **  ---   * 11921  * 27450 * 32034 *
*   X   *   X   *       *   X   *   X   **  ---   *  ---   * 18482 * 31280 *
*       *   X   *       *   X   *   X   **  ---   * 10531  * 26764 * 32131 *
*   X   *       *       *   X   *   X   **  ---   *  8645  * 24929 * 31489 *
*       *       *       *   X   *   X   **  ---   * 16998  * 32340 * 32340 *
*   X   *   X   *   X   *       *   X   **  ---   *  ---   * 14487 * 32325 *
*       *   X   *   X   *       *   X   **  ---   *  6805  * 23038 * 33176 *
*   X   *       *   X   *       *   X   **  ---   *  4961  * 21203 * 32534 *
*       *       *   X   *       *   X   **  ---   * 13272  * 28801 * 33385 *
*   X   *   X   *       *       *   X   **  ---   *  3365  * 19833 * 32631 *
*       *   X   *       *       *   X   **  ---   * 11882  * 28115 * 33482 *
*   X   *       *       *       *   X   **  ---   *  9996  * 26280 * 32840 *
*       *       *       *       *   X   **  ---   * 18349  * 33691 * 33691 *
*   X   *   X   *   X   *   X   *       **  ---   *  ---   * 13136 * 30974 *
*       *   X   *   X   *   X   *       **  ---   *  5454  * 21687 * 31825 *
*   X   *       *   X   *   X   *       **  ---   *  3610  * 19852 * 31183 *
*       *       *   X   *   X   *       **  ---   * 11921  * 27450 * 32034 *
*   X   *   X   *       *   X   *       **  ---   *  ---   * 18482 * 31280 *
*       *   X   *       *   X   *       **  ---   * 10531  * 26764 * 32131 *
*   X   *       *       *   X   *       **  ---   *  8645  * 24929 * 31489 *
*       *       *       *   X   *       **  ---   * 16998  * 32340 * 32340 *
*   X   *   X   *   X   *       *       **  ---   *  7739  * 23801 * 33534 *
*       *   X   *   X   *       *       **  ---   * 15937  * 29801 * 34385 *
*   X   *       *   X   *       *       **  ---   * 14240  * 29159 * 33743 *
*       *       *   X   *       *       **  ---   * 22653  * 34594 * 34594 *
*   X   *   X   *       *       *       **  ---   * 12850  * 28878 * 33840 *
*       *   X   *       *       *       **  ---   * 21283  * 34691 * 34691 *
*   X   *       *       *       *       **  ---   * 19317  * 34049 * 34049 *
*****************************************************************************

   --- == INSUFFICIENT BUFFER SPACE

LINKGEN COMPLETED
```

Figure 4-3. Sample Buffer Space Report for a CCP Run (partial report only)

DIR, OUTPUT DIRECTIVES REPORT

This report generates a copy of all the directives that Autolink generates for MPLINK. The directives are in the order in which they appear in the MPLINK input file. An output directives report is shown in appendix H as the MPLINK input directive file.

MAP, MEMORY ADDRESS MAP REPORT

This report produces two listings:

- One listing gives the names and starting addresses of all modules in ascending address order.
- The other listing gives the names and starting addresses of all modules in alphabetical module name order.

This report is similar to the maps produced by MPLINK, and is a convenience when the user does not plan to run MPLINK immediately following Autolink. A sample report is shown in figure 4-4.

INFO, INPUT DIRECTIVES REPORT

This report lists input directives and application lengths. Only those directives that follow the RPT=INFO directive in the input file are included in this report. A sample report is shown in appendix G.

SPECIAL CONSIDERATIONS FOR USING AUTOLINK

This subsection discusses four aspects of using Autolink:

- The interrelationship of the APPL, DEF, and MOD directives
- Requirements imposed on modules used by two or more applications
- A method for minimizing the number of MPLINK directives that are generated
- Autolink's method of locating modules in a build

CYBER MINI CROSS SYSTEM - LINK EDITOR -

MODULE MEMORY MAP - SORTED BY MODULE ADDRESS

MODULE	*ADDRESS*	*MODULE*	*ADDRESS*	*MODULE*	*ADDRESS*	*MODULE*	*ADDRESS*
ZEROX	0000	PNLNBA	4148	PBSETP	5393	PN2DLT	5F8D
PBINTR	0100	PNSGAT	4195	PBCLRP	5385	PN2FUL	6005
JUMPS	0140	PNCEFI	42AE	PBCOMP	53C1	PNROUT	6051
ADDRES	0150	PNCECN	4448	PBRDPG	53E0	PNDIRA	60AF
GLOBLS	0DA0	PNSTOR	444E	PBPSWI	53F1	PNDIRD	6134
ISPOLD	1F58	PTAFOU	4470	PBPUTP	5408	PNQUEU	617E
PBCALL	1F60	PTARET	44C8	PBGETP	5422	PNDEQU	61D4
PBLN00	1F98	R4ASYI	4404	PBSTPM	5433	PBFMAH	6235
PBLN01	1FC4	ASYMSG	47A8	UNLOCK	5446	PBTQAH	6320
PBAMAS	1FED	ASYLEM	4781	QULOCK	5455	P1TQAH	6387
PNSMWL	2000	PTAPSP	473E	PBRTCD	5466	PBTQAD	640F
PNAWAI	2194	AASCST	43BC	QENTRY	5471	PMMLFH	6470
PNRTN	21B7	AAEBCD	49EC	QEXIT	548C	PTMSCA	6494
PNSMBA	21C1	AACAPL	493C	BUFMAI	54DA	PBCLKI	6514
PNRVRS	2217	ATAPLA	497C	LISTSR	5550	PB100M	652F
PNTECO	2238	ABAPLA	493C	PBSTOP	5578	PBDELE	653E
PNQREL	227A	AASTAP	49EC	PTTPIN	5588	PBINSE	657D
PNGTCB	2294	AASBAP	443C	MODMST	5592	PBUPDA	66F3
PNTCBS	22F0	ASTDAS	447C	PBLN04	5699	PBTMRS	6751
PNDLTC	2346	AEBCDA	44EC	PBLN07	56A5	PBTICK	6818
PNRCWA	2432	AEAPLA	487C	PBLN09	56B1	PBTOQU	6883
PNDISC	2458	AAEAPL	43EC	PBLN10	56BD	PBTOSR	68DD
PNSMTO	24E1	ACAPLA	4C3C	PBLN11	56C9	PBTODE	6956
PNSMTR	2520	A7T06P	4C3C	PBLN12	56D5	PBTIMA	69A2
PNSMOT	2566	PBLN06	4CFC	PBLN13	56E1	PB16AD	69E9
PNLLCN	2650	PTCTCH	4D34	PBLN14	56ED	PBLCBT	6A21
PNLNCN	27C2	PTRETO	4D55	PBLN15	56F9	PBTIMO	6B05
PNTMLC	2A24	PTSV1L	4D5A	PBMEMR	5705	PBLLEN	6B8F
PNDELS	2F7D	PTSV2L	4D80	PBLCBP	5732	PBLLRM	6BD0
PNENAB	3077	PTRT1L	4D96	PBMIN	575F	PNBMPS	6C37
PNDISA	3113	PTRT2L	4D9E	PBMAX	577F	PMCOIN	6C59
PNLINE	32CA	PTREGL	4DA6	PBSTRI	579F	PBCOIN	6C8E
PNTMLD	34A5	PBTWLE	4E36	PBCOPY	57EE	PMCDRV	6CDE
PN1LLS	35C8	PBQBLK	4E4F	PBPUTY	5878	PBSCLA	7297
PNLLST	3644	PBQ1BL	4E78	PBDLTX	58C8	PMWOLP	72C6
PNCNTL	372E	PTINIT	4F81	PBPAGE	5928	PTCLAS	74A5
PNLCR	376D	PTIVTC	4EC2	PBXFER	5988	PMT1SE	7664
PNSTAT	3796	PBSLJ	4ED8	PB1RAD	598D	PTLMUX	7689
PN1LNS	37D6	PGDSTA	4E52	PB188I	5A2A	PBSWLE	76C0
PN2LNS	386E	PBMLIA	5077	PB18CO	5AD8	PBINTP	76C8
PNLNST	391E	PGSWIT	5184	PB8FAV	5C05	PBUPAB	76D4
PN1TML	39F5	PBLN02	52C8	PBADJU	5C7C	PBDNAB	76D7
PNTMLS	3AFA	PBLN03	5257	PN1SRC	5D6D	PBLOST	76F5
PNBRDC	3BFE	PBLN08	5306	PN1GTP	5DA4	PBHDRB	76FF
PN18RD	3D62	PBFILE	5325	PN1ADD	5DD8	PBPIPO	774D
PNOVLO	3E7C	PBLMAS	5350	PN1DLT	5E28	PBIIPO	7758
PNERCE	3F9A	PBOMAS	535D	PN1FUL	5E8A	PTBACK	779F
PNOVLT	3FC1	PBSMAS	536B	PN2SRC	5EB2	PTBREA	77ED
PNOVLD	4000	PBBEXI	5377	PN2GTI	5EDE	PTSTRT	784F
PNPSTA	4089	PBAEXI	5385	PN2ADD	5F38	PTSTOP	7892

Figure 4-4. Sample Memory Map Report for a CCP Run

INTERRELATIONSHIP OF THE APPL, DEF, AND MOD DIRECTIVES

APPL directives specify all the applications that can be used in a NPU build.

DEF directives specify all the applications that are to be included in a given build. To specify a different build, it is necessary to change only the DEF directives in the input directives file.

Each application used in a build must have at least one module defined for it by a MOD directive; that is, at least one MOD directive specifies that application in its APPL = appl parameter. If an application is not included in a given build (that is, it is not defined by a DEF directive), that application does not need to have any MOD directives specifying it.

DUPLICATE MODULES

When a module is used by two or more applications, that module's MOD directive must specify all the applications. Using the same MOD name in two MOD directives causes a fatal error.

A module is loaded for one application only; it is not duplicated for other applications. If a module with a tophat is located on a page, the tophat (relocation vector) is retained. If a module with a relocation vector is located in a nonpaged area, the module name is included in the DELETED TOPHAT MODULE LIST, which is one of Autolink's informative messages.

MINIMIZING THE NUMBER OF OUTPUT DIRECTIVES

Whenever possible, Autolink creates an output directive to MPLINK in the form *L,modi-modj. This statement directs MPLINK to start linking the module named modi following the previously linked module, and to continue linking consecutive modules as they appear on the input object code file until the module called modj is linked. This requires that both the Autolink and the Link utilities use the same input object code file. Otherwise, a nonfatal error occurs.

AUTOLINK'S METHOD OF SELECTING A LOCATION FOR A MODULE

Autolink uses the following algorithm to maximize the amount of buffer space:

- Paged modules are assigned space first.
- Next, modules with main memory addresses and their associated modules are assigned space.
- Next, modules that are to be reverse-loaded are assigned space starting at the upper end of assignable main memory (location $FFFE_{16}$), or at the address specified.
- The remaining applications are assigned space in the largest free area in main memory.
- FILL modules are located in the main memory holes and in page holes.
- The last application (which must be PIDTBL) is located at the end of the sequential applications.
- Buffer space is computed.

The specific steps used by Autolink in each of these phases are described below.

Figure 4-5 shows this sequence.

Phase 1, Assigning Space to Paged Modules

First, Autolink locates modules assigned to a page by the MOD directive's P=P parameter. If an application is the only user of this module, that application is located on the same page.

Next, any remaining applications with an NC parameter are assigned locations on free pages. Applications which are non-exclusive users of a P=P module are located on free or nonreserved pages.

Then Autolink calculates the length of any applications which have not yet been located on a page. If an application will not fit on any one page, part of it is placed on the page with the largest assignable space. The rest of it is assigned to main memory.

Finally, Autolink sorts the pageable filler modules (Mod parameters P=P and FILL) modules by length. Starting with the largest module, Autolink attempts to locate that module in the largest free space in paged memory. This process continues until all the modules are either located or put aside because paged space was not large enough to hold them.

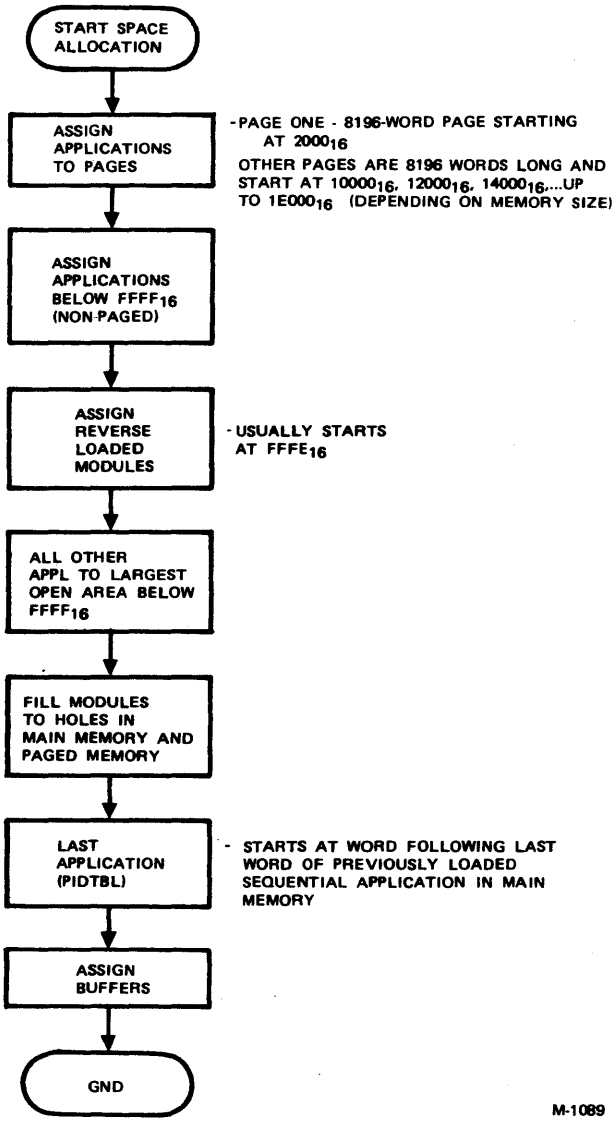


Figure 4-5. Autolink Sequence of Locating and Link Modules

Phase 2, Assigning Space in Main Memory

This main memory space assignment phase occurs after page space assignment is completed.

Applications with specific addresses (ADDR = addr) are assigned space in the order in which they appear in the APPL directives. A module associated with two or more applications is located with the first application encountered, not in the order of the MOD directive's APPL parameter. Within an application, modules are located in the order in which they appear in the input directives file.

A module with the MOD parameter FILL is not located during this phase.

If a module overlaps into space already assigned to a module (ADDR parameter is specified) during this phase, a fatal error occurs.

Phase 3, Assigning Space for Reverse-Loaded Modules

Reverse-loaded modules are specified by MOD parameter P=R. Space is assigned to these modules in the same order in which they are encountered in the input directives file. The ending address of the first module encountered is located at a fixed or a specified main memory address. The next module's ending address is assigned to the word preceding the previous reverse-loaded module's starting address. This process continues until all the reverse-located modules are given space in one contiguous group.

The ending address given to the set of reverse-loaded modules is determined in one of two mutually exclusive ways:

- The MOD directive for the first reverse-loaded module in the directives input file is given a specific address in the ADDR parameter (example: PIBUF2 is the first reverse loaded module; its directive specifies ADDR = $FFFE). This address is the ending address of the module rather than the starting address.

- No address is specified for the first reverse-loaded module in the directives file. In this case, the highest assignable main memory address is used ($FFFE_{16}$) as the ending address of the module.

If one reverse-loaded module overlaps another (as could be the case if a reverse-loaded module is given an ADDR parameter), a fatal error occurs.

Phase 4, Assigning Space to Sequence Applications in Main Memory

This phase assigns space for all those modules belonging to applications specified in a DEF directive, but which (1) are not assigned addresses, (2) are not FILL modules, and (3) are not exclusively associated with the last application.

The first application in the input directives file is assigned space starting at the beginning of the largest unassigned area in main memory. All the modules associated with that application are located contiguously in the order in which the MOD directives appeared in the input directives file. When the first application is finished, space is assigned for the next application in the same manner, starting at the next word in main memory. This process continues until all modules in all applications (except the last application, PIDTBL) are assigned space.

At the end of this stage, a sequential set of applications fits into the area selected. During this phase, if any module overflows the end of that area, a fatal error occurs.

Phase 5, Assigning Space for FILL Modules

At this stage of the process only two types of module remain to be located: FILL modules (designated by the FILL parameter of the MOD directive) and modules that are exclusively associated with the last application. FILL modules are assigned space in main memory in any unassigned area large enough to accommodate them. If all the FILL modules cannot be located, a fatal error occurs.

Phase 6, Assigning Space for the Last Application

The only remaining modules are those exclusively associated with the last application, PIDTBL. Its module lengths are added to calculate a single application length. Its address is set to the word following the ending address of last forward-loaded module.

AUTOLINK MESSAGES

Two types of messages are generated during Autolink operations:

- Certain nonfatal errors generate an informative message specifying events of interest to the user, but events which do not stop Autolink processing. These messages are copied to the listing file.

- Fatal errors generate a fatal error message; it is copied to the fatal error file. A fatal error also generates informative messages that are copied to the listing file.

INFORMATIVE MESSAGES

Informative messages are copied to the output listing file. Such messages are preceded by the directive that was being processed when the error or processing step occurred. The general format of these messages in the listing file is:

Directive
Program scan pointer address
Message text

The informative and nonfatal error messages are listed in table B-3 of appendix B. The type of action that should be taken when the messages appear are also given in that table.

FATAL ERRORS

A fatal error generates the message FATAL ERROR for the fatal error file. Autolink processing stops except to prepare the diagnostic information that is copied to the output listing file. This consists of two parts:

- Diagnostic comments

- A message specifying the fatal error type. Most fatal error messages are concerned with a directive error. This type of error message is treated the same as an informative message: the fatal error message is preceded by the directive which was being processed when the error occurred. The general format of these fatal error messages in the listing file is:

  Directive
  Program scan pointer address
  Fatal error message

The fatal error messages are given in table B-4 of appendix B. The type of action that should be taken when the messages appear is also given in that table.

INTRODUCTION

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

- 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 an 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 CCP or 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
- Generated by Autolink procedures during a standard CCP 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 (131,072) 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 - $FFFF_{16}$.
- In the page address mode, the page registers are used to achieve an effective 18-bit memory address in the range 0000 - $3FFFF_{16}$ (only the range 0000-$1FFF_{16}$ is used).

PAGE ADDRESSING MODE

In page address 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 - $7FF_{16}$) 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 - $7F_{16}$) 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 - $1F_{16}$). This 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 5-1 correlates the 18-bit address to page register and page displacement.

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

- Page displacement
- Page number
- Page register
- Page register set

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.

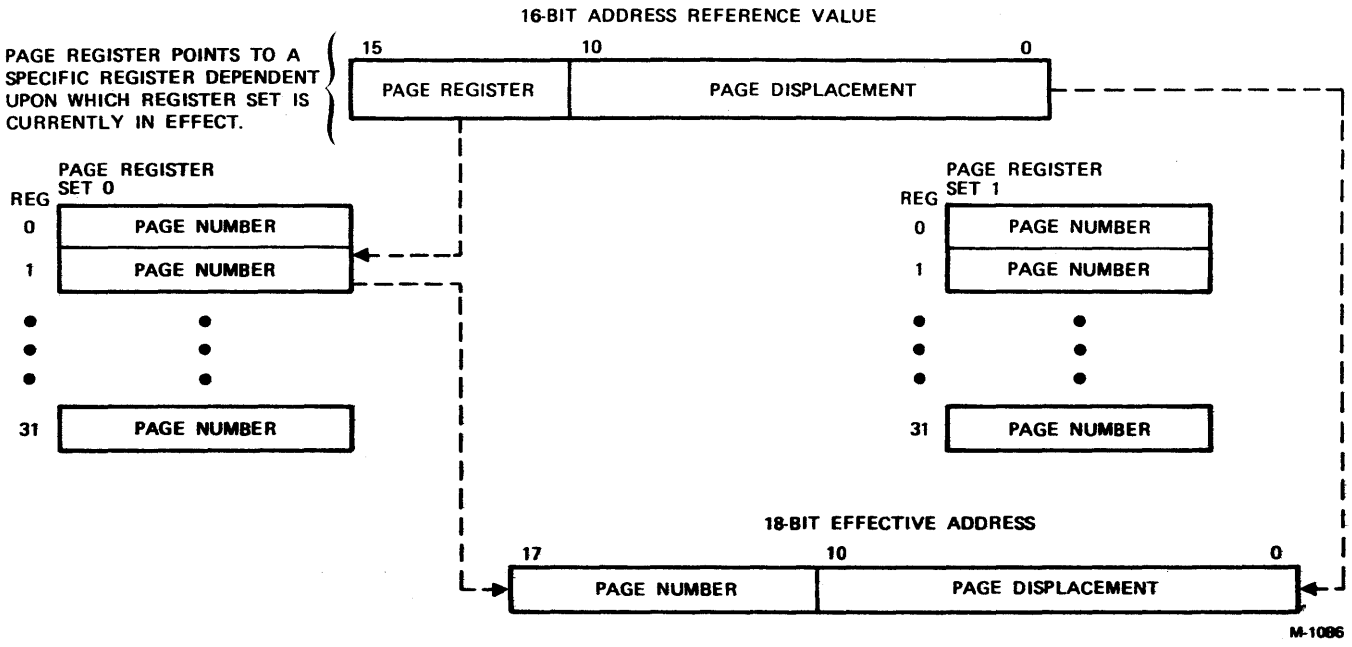


Figure 5-1. Page Register Selection

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

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 one 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 name identifies a module, an entry point, or an overlay. The allowable keywords are:

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:/PREG(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.
  The page register set to be used is 1.

- $1A45:/PREG(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:PREG((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 which 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 INPUTS

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 5-2. Typical MPLINK use is given in the examples in appendix H.

MPLINK DIRECTIVES FILE

This file can be generated in one of three ways.

- CCP: The simplest way is to use the Autolink utility. Autolink solves many of space assignment problems that the user would otherwise have to solve by repeated trials.

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

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

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. If Autolink was used, the same object code file must be presented to Autolink and to MPLINK.

These modules in this file were previous put in this form by a CYBER Cross assembler or compiler (macro-assember, microassember, 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.

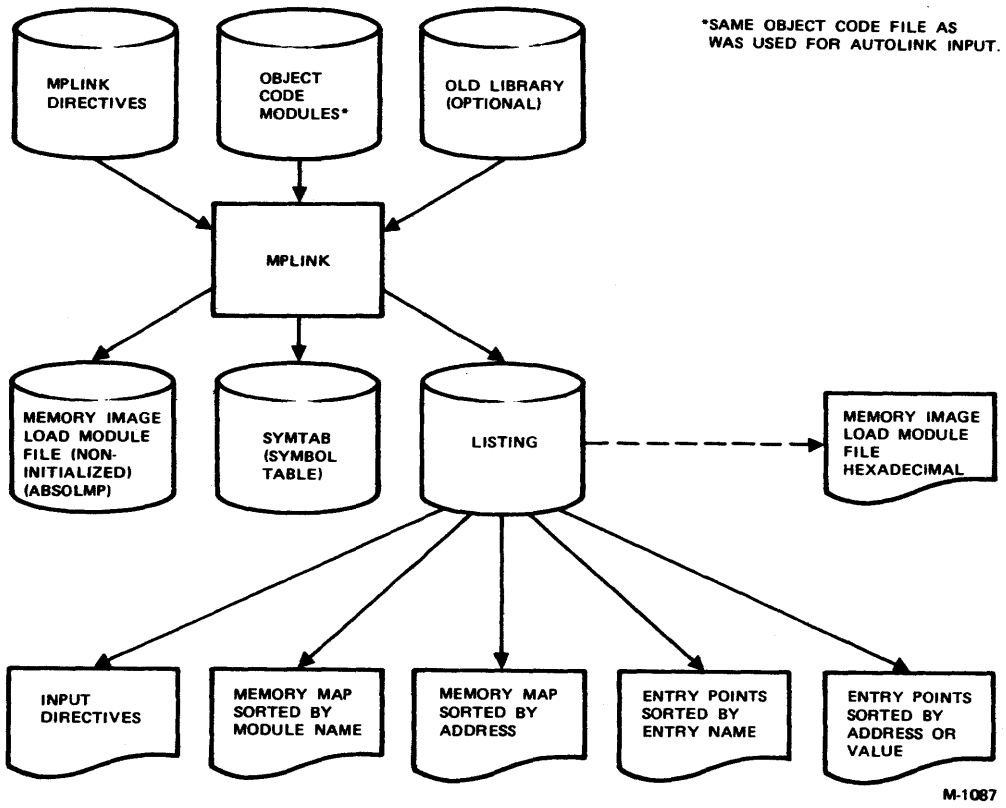


Figure 5-2. MPLINK Procedural Flow

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 (SYMTAB) FILE

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 5-3.
- A module memory map sorted by module address.
- An entry symbol list sorted by entry name. A sample partial listing is shown in figure 5-4.
- 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	4BFC	PBCOPY	57EE	PBPROP	7A55	PINWIN	F724
AAEBCD	48FC	PBDELE	653E	PBPSWI	53F1	PIPROT	F67F
AASBAP	4A3C	PBDLTX	58C8	PBPUTP	5408	PISIZC	F6D5
AASCST	48BC	PBDNAB	76D7	PBPUTY	5878	PITMRS	FE05
AASTAP	49FC	PBFILE	5325	PBQBLK	4E4F	PIWLIN	FBF8
AdAPLA	498C	PBFMAD	7F84	PBQ18L	4E78	PLCBIN	1605C:205C
ACAPLA	4C3C	PBFMAH	6295	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	PBHDRR	76FF	PBSLJ	4EDB	PLREAD	160DD:20DD
ASCF29	1535C:335C	PBIIPO	7758	PBSMAS	5368	PLTKOP	16000:2000
ASTDAS	4A7C	PBILL	803F	PBSTOP	5578	PMCDRV	6CDE
ASYFRR	11729:3729	PBINSE	657D	PBSTPM	5433	PMCOIN	6C59
ASYLFM	4781	PBINTP	76CB	PBSTRI	579F	PMMLEH	6470
ASYMSG	47A8	PBINTR	0100	PBSWLE	76C0	PMT1SE	7664
ATAPLA	497C	PBIOPO	7956	PBTICK	6818	PMWOLP	72C6
A128EB	153DC:33DC	PBLCBP	5732	PBTIMA	69A2	PNAWAI	2198
A7TO6P	4C8C	PBLCBT	6A21	PBTIMO	6805	PNBMPS	6C37
BEGINX	F671	PBLLEN	6B8F	PBTMRS	6751	PNBROC	3BFE
BUFMAI	54DA	PBLLRM	6BD0	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	78A5	PBTOQU	68B3	PNCONF	15902:3902
E29ASC	1541C:341C	PBLNO0	1F9B	PBTOSR	68DD	PNDELE	2F7D
FCSRCB	1559C:359C	PBLNO1	1FC4	PBTWLE	4E36	PNDEQU	61D4
GLOBL$	0DA0	PBLNO2	52C8	PBUPAB	76D4	PNDIRA	60AF
HASPMS	858E	PBLNO3	52E7	PBUPDA	66F3	PNDIRD	6134
HSPTCB	1558E:358E	PBLNO4	5699	PBXFER	5988	PNDISA	3118
HSR4IP	85CA	PBLNO6	4CFC	PB100M	652F	PNDISC	2458
HSR4IT	14F6B:2F6B	PBLNO7	56A5	PB16AD	69E9	PNDLTC	2346
HSR4TP	150E4:30E4	PBLNO8	5306	PB18AD	598D	PNENAB	3077
IC	16010:2010	PBLNO9	5631	PB188I	5A2A	PNFRCE	3F9A
ISPOLD	1F5B	PBLN10	568D	PB18CO	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	PBLN14	56ED	PGSWIT	5184	PNLLIN	7ADA
MODMST	55B2	PBLN15	56F9	PIAPPS	FC77	PNLLLI	7E66
PBADJU	5C7C	PBLOAD	8045	PIBUF1	F822	PNLLLO	7EC8
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	5385	PBOMAS	535D	PILINI	FD63	PNLNCN	27E2
PBCLR	801D	PBPAGE	592B	PILMT	F6E8	PNLNST	391E
PBCOIN	6C8E	PBPIPO	774D	PIMLIA	FE97	PNOVLD	4000

Figure 5-3. Sample MPLINK (Partial) Memory Map Sorted by Module Name

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.

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

MPLINK(D=infile,R=outfile,CSET=cset)

where D is the logical file which presents the input directives to MPLINK. Default is INPUT. R is the logical file that receives the listings. 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.

Appendix H 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

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	488C		AEBLS	R	45B4		ALARM2	A	0002	
AAEAPL	R	48EC		AECHR1	R	466D		ALARM3	A	0003	
AAERCD	R	48EC		AECIN1	R	46DA		ALCAPL	R	4C7C	
AANREA	A	0007		AECIN2	R	46DC		ALCORA	R	4ABC	
AAOUTP	A	0003		AECIN3	R	46E7		ALEAPL	R	488C	
AAREAD	A	0004		AECIN4	R	46E9		ALEBCD	R	483C	
AASBAP	R	483C		AECKMD	R	45A8		ASASCI	A	0022	
AASTAP	R	49EC		AECOD1	R	45C0		ASAUTO	A	0018	
ABAPLA	R	498C		AECOD2	R	46A0		ASCE26	R	1539C:339C	
ACAPL	R	12BE0:36E0		AECSE	R	116D7:36D7		ASCE29	R	1535C:335C	
ACARTD	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	11687:3687		AEEIN1	R	46F5		ASSOL	A	0001	
ACCORR	A	0C01		AEEIN2	R	46F7		ASXPT	A	0002	
ACDELM	A	0002		AEEIN3	R	4702		ASXSOI	A	0005	
ACEAPL	R	0007		AEEIN4	R	4704		ASYNCE	R	153A	
ACFBCD	A	0001		AEELL	R	45C5		AS2741	A	0020	
ACFLL	A	0020		AEELT1	R	45CE		ATAPLA	R	497C	
ACSPL	A	0001		AEELT3	R	45E0		ATELL	A	0000	F:7
ACXMSG	R	85BE		AEEPL	R	45F1		ATEPL	A	0000	7:7
ACXPTR	R	1D69		AEEPT1	R	45F8		ATPDI2	R	11318:3318	
ACLIH2	A	3D69		AEESL1	R	4780		ATPDI5	R	1133F:333F	
ACLIM1	A	0A00		AEESL2	R	4782		ATPDI6	R	11363:3363	
ACPBLI	A	0001		AEINPT	R	4583		ATPDI8	R	11387:3387	
ACPBOB	A	0C12		AEINT	R	458C		ATPPR1	R	112D3:32D3	
ACPCLR	A	0C0C		AEMBT	R	455F		ATPPR4	R	112F2:32F2	
ACPEVE	A	0000		AESED1	R	473C		AUCAPL	R	4C3C	
ACPICS	A	3C50		AESLL	R	4570		AUCORA	R	4A7C	
ACPIOW	A	0C60		AES110	R	4658		AUEAPL	R	487C	
ACPLR1	R	1C00		AES150	R	4668		AUEBCD	R	4AFC	
ACPOBL	A	3058		AES151	R	4662		AVASCE	R	11729:3729	
ACPOMA	A	0C6C		AES300	R	4674		AVASCP	R	1E5D	
ACPONS	A	0C48		AES301	R	467A		AVB7TO	R	1E74	
ACPRMA	A	0C10		AEX8LS	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		AE4XOL	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	

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 this 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 5-4. Sample MPLINK Memory Map Sorted by Entry Name (Partial)

where An * indicates the beginning of a MPLINK directive. dirname is the name of the directive. paraml 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.

An example of an MPLINK input directives file is given in appendix H.

SUMMARY OF MPLINK DIRECTIVES

Table 5-1 summarizes the MPLINK directives.

MPLINK DIRECTIVE PARAMETERS

There are three general types of paramters:

- A name; it can be the the 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 identification
- A memory address

MPLINK Directive Parameter Names

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

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

MPLINK Directive Overlay Identifier Parameter

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

MPLINK Memory Address Parameters

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

*L, SPECIFIES MODULES TO BE LINKED

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

*L,mod,addr.

TABLE 5-1. 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 causes MPLINK to locate object code module, mod, at starting address, addr. As it is located, other linking operations also occur: addresses are absolutized 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,modl-mod2,addr.

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

- *L,modl-mod2.

  Same as the previous form, except modl 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,modl-mod2,addr.

  Links a series of modules on the object code input file, starting with modl 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 modl) are linked. If either modl 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. 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 (1) a *L has already been used or (2) 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. 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. 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.

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 a *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.

Format of the *OVLY directive is:

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

where name is the overlay name (normally six letters or numbers). Note that the overlay name 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 which 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.

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.

Format of the directive is:

```
*SYN,name1,name2.
```

where name1 is the name of a declared entry point or a defined module; and name2 is the name to be associate with name1. At every occurence of name1 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. Format of the directive is:

```
*COR,addr.
```

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

```
$FFFF   -   65,536 words
$13FFF  -   81,920 words
$17FFF  -   93,302 words
$1FFFF  -   131,072 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 which MPLINK uses to resolve unsatisfied externals during the linking process. 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. 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:

```
nam1
constant
nam1+constant
nam1+nam2
nam1-constant
nam1-nam2
```

nam1 and nam2 can be local variables or entry points which are absolute, or have been previously absolutized.

*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 which has not completely finished processing. 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 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 PASCAL Compiler Reference Manual).

Format of the directive is:

*DVAR,addrb,addre.

where addrb is the beginning 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 be used for other purposes (buffers, programs, globals, or other reserved areas).

*COM, DEFINES A BLANK COMMON AREA FOR MACROASSEMBLER PROGRAMS

This directive allocates a blank common area that is referenced by macroassembler modules. 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 macroassembler programs can reference this area. 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 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. Format of the directive is:

*DMP.

*END, LAST MPLINK DIRECTIVE

This directive must end 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. Format of the directive is:

*END,addr.

where addr is a hexadecimal number or an entry point name. 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-5 in appendix B lists the MPLINK error messages, and the action which the user should take in response to the message.

INTRODUCTION

The Edit utility is used to initialize values in specified variables of the CCP or CCI absolutized modules.

The MPEDIT utility requires three inputs:

- The non-initialized 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 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).

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 I.

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. 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).

Three optional parameters are available with the MPEDIT call statement:

```
MPEDIT(D=infile,R=outfile,CSET=cset)
```

where D is the local file which presents the input directives to MPEDIT. Default is INPUT. R is the local file that receives the listings. 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.

Appendix I shows examples of executing MPEDIT.

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

MPEDIT PROGRAM SYNTAX

The statements which comprise an MPEDIT program are similar to PASCAL statements (see the CYBER Cross Compiler Reference Manual) with some restrictions and extentions. A typical extention allows an expression on the left side of a VALUE statement. The evaluted expression specifies the address which receives the assigned value. Comments are permitted as in PASCAL statements. Appendix I show selected sections of an MPEDIT program.

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

- A definition/declaration section
- An assignment (initialization) section

The program ends with a terminator.

The sections of the program are shown in figure 6-1. The MPEDIT flow is shown in figure 6-2.

The definition/declaration section consists of three parts that 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).

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

MPEDIT SYNTAX

MPEDIT program 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

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) Composite statement; Memory Resident Partition.

Figure 6-1. MPEDIT Program Format

- Local symbols used for equating constant locally, or specifiying a local variable
- External symbols in MPEDIT always have an array attribute
- Literals
- Address functions
- Expressions

MPEDIT KEYWORDS

The following keywords are reserved for MPEDIT controls:

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

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

MPEDIT RESERVED WORDS

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

/TRACE /DMP$ /ESL$ /NAM$

MPEDIT assigns specific address meanings to the following reserved symbols:

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

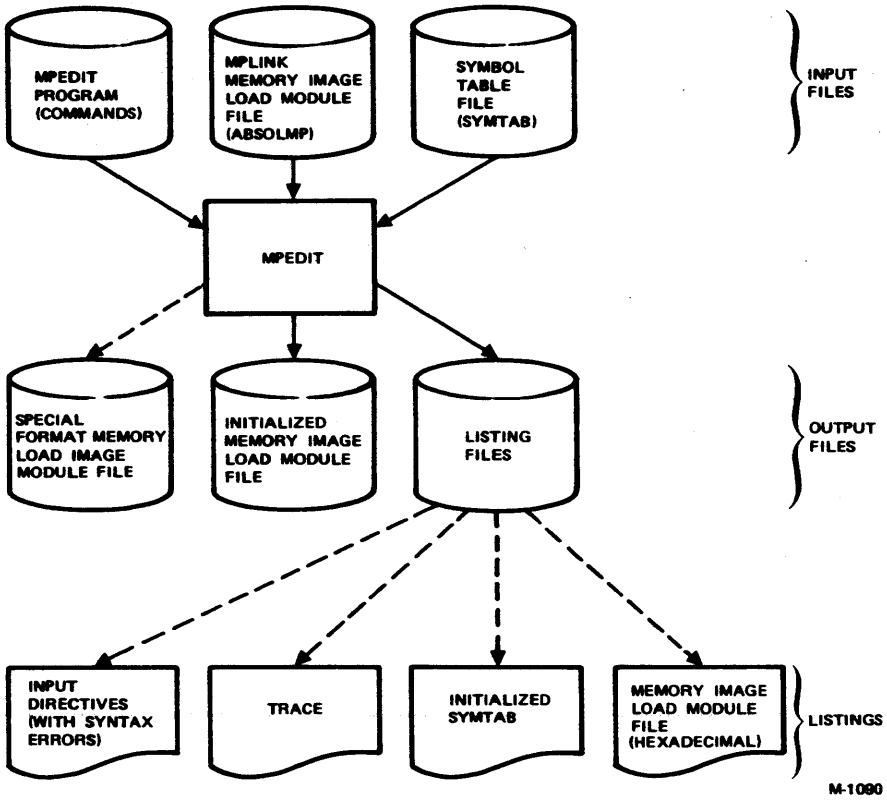


Figure 6-2. MPEDIT Program Flow

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

MPEDIT 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.

MPEDIT 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$  UTOPARAM (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 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.

MPEDIT 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
```

MPEDIT 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):

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

In addition, MPEDIT defines four more address functions:

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

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

```
/xxxxx (name)
```

/START, Field Start Address Function

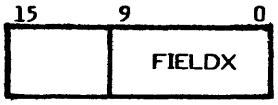
The /START function has the format:

```
/START(external)
```

The function returns the start position in bits (range 15-0) 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 form:

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

Example: to find the terminal class field (BSTCLASS) start position and length in the base terminal control block (TCB) descriptor table, the following MPEDIT statements are used:

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

/ENTRY, Entry Point Address Function

This function accepts a module name as a parameter, and generates the address of the module's associated entry point. The /ENTRY function has the form:

```
/ENTRY(external)
```

Example: To locate the entry point and the page number of the service module in CCP, the following MPEDIT statements include address functions:

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

/VFD, Variable Field Definition Address Function

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

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 arithmentic 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:

```
GLOBL$
(/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.

Examples:

- A

  This is an address expression unless A was declared in an array. In that case, the lack of subscripts indicates 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.

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

- 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

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 that is 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 equal sign, and a literal, or a previously defined, constant. The declaration terminates with a semicolon. Literal and previously defined constants can be signed.

It is possible to define a constant value as an expression which itself is a mixture of constants, previously declared local constants, and entry symbols which appear in SYMTAB.

Examples are given in figure 6-3.

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. 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, 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.

VARIABLE DECLARATION PART

The next part of an MPEDIT program contains variable declarations; this section is also optional. It 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.

Example of a variable declaractions are given in figure 6-3.

```
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;
   /BFLCD0  = /BUFSZ0*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 6-3. Examples of MPEDIT Constant, Variable, and Array Declarations

ARRAY DECLARATION PART

The next part of an MPEDIT program contains array declarations; this section is also optional. Array declarations allow programmers to create external symbols as arrays so that elements can be referenced by an index. Any external symbol that is 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 6-3.

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

- Local assignment statements
- Address assignment statements
- FOR loop statement

- An embedded composite statement
- Empty statement

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 I.

Local Assignment Section

A local assignment section 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 Section

An address assignment statement has the form:

```
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 Statement

The FOR statement in MPEDIT is entirely analagous 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. The basic form of the FOR 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.

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 Statement

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. The format of the statement is:

```
BEGIN
      statement;
      statement;
        .
        .
        .
      statement
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 which are to be acted upon as a single statement.

The example in figure 6-4 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 6-4. Methods of Packing an NPU Array

Empty Statement

The empty statement is analagous 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 which 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 16-bit word after the initializing value is inserted
- The line number of the MPEDIT program which caused initialization of the field

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. 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 6-5.

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. 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 6-6. The listing was requested by the pseudovariable declaration:

```
/$ESL := 2;
```

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 a MPEDIT run.

MPEDIT DIAGNOSTICS

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. Any of the following types of statements fail:

- statement has an undefined identifier

TRACE LIST

```
578  VFD($0016A, $F, $F) := 0096;   0000 → 0096
579  VFD($00168, $F, $F) := 000F;   0000 → 000F
580  VFD($00167, $F, $F) := 0032;   0000 → 0032
581  VFD($00168, $F, $F) := 0000;   0000 → 0000
582  VFD($00169, $F, $F) := 0000;   0000 → 0000
595  VFD($0F67E, $F, $F) := F659;   0000 → F659
600  VFD($01080, $F, $F) := 0020;   0000 → 0020
601  VFD($01081, $F, $F) := 0005;   0000 → 0005
608  VFD($011A1, $F, $F) := 8093;   0000 → 8093
609  VFD($011A0, $8, $8) := 0010;   0000 → 0010
610  VFD($0119E, $F, $4) := 0009;   0000 → 4800
611  VFD($011A0, $E, $5) := 0001;   0010 → 0210
612  VFD($0119D, $7, $7) := 0002;   0000 → 0002
613  VFD($011A0, $F, $0) := 0001;   0210 → 8210
617  VFD($011A9, $F, $F) := 76D2;   0000 → 76D2
618  VFD($011A8, $8, $8) := 000E;   0000 → 000E
619  VFD($011A6, $F, $4) := 000A;   0000 → 5000
620  VFD($011A8, $E, $5) := 0004;   000E → 080E
621  VFD($011A5, $7, $7) := 0002;   0000 → 0002
622  VFD($011A8, $F, $0) := 0001;   080E → 880E
626  VFD($011B1, $F, $F) := 5179;   0000 → 5179
627  VFD($011B0, $8, $8) := 000A;   0000 → 000A
628  VFD($011AE, $F, $4) := 0008;   0000 → 5800
629  VFD($011B0, $E, $5) := 000A;   000A → 140A
630  VFD($011AD, $7, $7) := 0005;   0000 → 0005
631  VFD($011B0, $F, $0) := 0001;   140A → 940A
635  VFD($01189, $F, $F) := 2189;   0000 → 2189
636  VFD($011B8, $8, $8) := 0004;   0000 → 0004
637  VFD($011B6, $F, $4) := 000C;   0000 → 6000
638  VFD($011B8, $E, $5) := 0001;   0004 → 0204
639  VFD($011B5, $7, $7) := 0004;   0000 → 0004
640  VFD($011B8, $F, $0) := 0001;   0204 → 8204
644  VFD($011C1, $F, $F) := 69E5;   0000 → 69E5
645  VFD($011C0, $8, $8) := 000D;   0000 → 000D
646  VFD($011BF, $F, $4) := 000D;   0000 → 6800
647  VFD($011C0, $E, $5) := 0001;   000D → 020D
648  VFD($011BD, $7, $7) := 0001;   0000 → 0001
649  VFD($011C0, $F, $0) := 0001;   020D → 820D
653  VFD($011C9, $F, $F) := 8093;   0000 → 8093
654  VFD($011C8, $8, $8) := 0010;   0000 → 0010
655  VFD($011C6, $F, $4) := 000E;   0000 → 7000
656  VFD($011C8, $E, $5) := 0001;   0010 → 0210
657  VFD($011C5, $7, $7) := 0006;   0000 → 0006
658  VFD($011C8, $F, $0) := 0001;   0210 → 8210
662  VFD($011D1, $F, $F) := 3D39;   0000 → 3D39
663  VFD($011D0, $8, $8) := 002F;   0000 → 002F
664  VFD($011CE, $F, $4) := 000F;   0000 → 7800
665  VFD($011D0, $E, $5) := 0001;   002F → 022F
666  VFD($011CD, $7, $7) := 0003;   0000 → 0003
667  VFD($011D0, $F, $0) := 0001;   022F → 822F
671  VFD($01231, $F, $F) := 653C;   0000 → 653C
672  VFD($01230, $8, $8) := 000C;   0000 → 000C
673  VFD($0122E, $F, $4) := 001B;   0000 → D800
674  VFD($01230, $E, $5) := 0001;   000C → 020C
675  VFD($0122D, $7, $7) := 0001;   0000 → 0001
676  VFD($01230, $F, $0) := 0001;   020C → 820C
680  VFD($0115D, $7, $7) := 0003;   0000 → 0003
681  VFD($01165, $7, $7) := 0001;   0000 → 0001
682  VFD($0116D, $7, $7) := 0003;   0000 → 0003
683  VFD($01175, $7, $7) := 0002;   0000 → 0002
684  VFD($0117D, $7, $7) := 0001;   0000 → 0001
685  VFD($01185, $7, $7) := 0001;   0000 → 0001
686  VFD($0118D, $7, $7) := 0001;   0000 → 0001
688  VFD($0153D, $F, $F) := 0004;   0000 → 0004
694  VFD($01774, $F, $F) := 0002;   0000 → 0002
695  VFD($01775, $F, $F) := 0001;   0000 → 0001
696  VFD($01776, $F, $F) := 000D;   0000 → 000D
697  VFD($01777, $F, $F) := 6AFD;   0000 → 6AFD
698  VFD($01778, $F, $F) := 00F0;   0000 → 00F0
699  VFD($01779, $F, $F) := 00F0;   0000 → 00F0
700  VFD($0177B, $F, $F) := 5D65;   0000 → 5D65
701  VFD($0177A, $F, $F) := 0008;   0000 → 0008
702  VFD($0177C, $F, $F) := 0002;   0000 → 0002
703  VFD($0177D, $F, $F) := 0001;   0000 → 0001
704  VFD($0177E, $F, $F) := 0010;   0000 → 0010
705  VFD($0177F, $F, $F) := 8093;   0000 → 8093
706  VFD($01780, $F, $F) := 0002;   0000 → 0002
707  VFD($01781, $F, $F) := 0002;   0000 → 0002
708  VFD($01782, $F, $F) := 000D;   0000 → 000D
709  VFD($01783, $F, $F) := 6B89;   0000 → 6B89
710  VFD($01788, $F, $F) := 0078;   0000 → 0078
711  VFD($01789, $F, $F) := 0078;   0000 → 0078
712  VFD($0178A, $F, $F) := 0008;   0000 → 0008
713  VFD($0178B, $F, $F) := 413F;   0000 → 413F
714  VFD($0178C, $F, $F) := 0002;   0000 → 0002
715  VFD($0178D, $F, $F) := 0001;   0000 → 0001
716  VFD($0178E, $F, $F) := 0010;   0000 → 0010
717  VFD($0178F, $F, $F) := 8093;   0000 → 8093
718  VFD($01784, $F, $F) := 0002;   0000 → 0002
719  VFD($01785, $F, $F) := 0002;   0000 → 0002
720  VFD($01786, $F, $F) := 000E;   0000 → 000E
721  VFD($01787, $F, $F) := 7682;   0000 → 7682
722  VFD($01790, $F, $F) := 0014;   0000 → 0014
723  VFD($01791, $F, $F) := 0014;   0000 → 0014
724  VFD($01792, $F, $F) := 0008;   0000 → 0008
725  VFD($01793, $F, $F) := 444C;   0000 → 444C
726  VFD($017A4, $F, $F) := 0000;   0000 → 0000
727  VFD($017A5, $F, $F) := 0000;   0000 → 0000
728  VFD($017A7, $F, $F) := 0000;   0000 → 0000
739  VFD($01859, $D, $1) := 0001;   0000 → 1000
740  VFD($01859, $4, $4) := 0008;   1000 → 1008
744  VFD($01867, $D, $1) := 0001;   0000 → 1000
745  VFD($01867, $4, $4) := 0009;   1000 → 1009
746  VFD($01867, $7, $2) := 0002;   1009 → 1049
750  VFD($01883, $4, $4) := 000F;   0000 → 000F
751  VFD($0188B, $F, $F) := 000E;   0000 → 000E
```

Figure 6-5. Partial MPEDIT Trace Listing

- statement causes an attempt to assign a nonexistent memory location
- statement has a bad field specification (overflows word or has an illegal format)
- statement has 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
```

CYBER NINI 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	0C00		AEAPL	R	11709:3709		AINPLB	A	000E	
AACORR	R	48BC		AEASCI	R	46A7		AINPSB	A	000D	
AAEAPL	R	48EC		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	0303		AEAUT2	R	464E		AISP4F	R	44FA	
AAREAD	A	0C04		AEAUT3	R	4654		ALARM1	A	0001	
AASBAP	R	4A3C		AEBLS	R	4584		ALARM2	A	0002	
AASTAP	R	49EC		AECHR1	R	466D		ALARM3	A	0003	
ABAPLA	R	49BC		AECIN1	R	46DA		ALCAPL	R	4C7C	
ACAPL	R	116ED:35ED		AECIN2	R	46DC		ALCCRA	R	4ABC	
ACARTO	A	0078		AECIN3	R	46E7		ALEAPL	R	48BC	
ACAUTO	A	0001		AECIN4	R	46E9		ALEBCD	R	483C	
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	001B	
ACCORR	L	0003		AECSLL	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	0C00		AEEIN4	R	4704		ASXPT	A	0002	
ACEPL	A	0001		AEELL	R	45C5		ASXSOI	A	0005	
ACKMSG	R	858E		AEELT1	R	45CE		ASYNCE	R	153A	
ACKPTR	R	1068		AEELT3	R	45F0		AS2741	A	0020	
ACLIM2	A	0000		AEEPL	R	45F1		ATAPLA	R	497C	
ACLIM1	A	0A00		AEEPT1	R	45FA		ATELL	A	0000	F:7
ACPBLI	A	0001		AEESL1	R	4780		ATEPL	A	0000	7:7
ACPBOB	A	0002		AEESL2	R	4782		ATPDI2	R	11318:3318	
ACPCLR	A	000C		AEINPT	R	4583		ATPDI5	R	1133F:333F	
ACPEVF	A	0000		AEIN1	R	458C		ATPDI6	R	11363:3363	
ACPICS	A	0050		AEMBT	R	455F		ATPDI8	R	11387:3387	
ACPIOW	A	0060		AESEOI	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		AES130	R	4668		AUCCRA	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 6-6. Partial MPEDIT SYMTAB Listing (Sorted by Entry Name)

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.

MPEDIT ERROR MESSAGES

Table B-6 in appendix B lists the MPEDIT error messages, the message meaning, and the action which 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	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	ASCII Code	CDC Graphic	ASCII Graphic Subset	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	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							
W	W	27	0-6	26	0-6	127	∧	&	67	0-8-7	37	12	046
X	X	30	0-7	27	0-7	130	↑	' (apostrophe)	70	11-8-5	55	8-5	047
Y	Y	31	0-8	30	0-8	131	↓	?	71	11-8-6	56	0-8-7	077
Z	Z	32	0-9	31	0-9	132	<	<	72	12-0 or 12-8-2†††	72	12-8-4 or 12-0†††	074
0	0	33	0	12	0	060							
1	1	34	1	01	1	061	>	>	73	11-8-7	57	0-8-6	076
2	2	35	2	02	2	062	≤	@	74	8-5	15	8-4	100
3	3	36	3	03	3	063	≥	\	75	12-8-5	75	0-8-2	134
4	4	37	4	04	4	064	≠	^(circumflex)	76	12-8-6	76	11-8-7	136
5	5	40	5	05	5	065	; (semicolon)	; (semicolon)	77	12-8-7	77	11-8-6	073

†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 63-graphic set, display code 00 has no associated graphic or card code; 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.

M-184

UTILITY 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	Expand
B-3	Autolink - Informational messages
B-4	Autolink - Fatal errors
B-5	Link
B-6	Edit

TABLE B-1. LIBRARY MAINTENANCE UTILITY ERROR MESSAGES

Message Text	Meaning/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.
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.
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 preceeds 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.

TABLE B-1. LIBRARY MAINTENANCE UTILITY ERROR MESSAGES (Contd)

Message Text	Meaning/User Action
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 preceeds 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. EXPAND UTILITY ERROR MESSAGES

Message Text	Meaning/User Action
EMPTY MACRO CALL FILE	An empty file was passed to Expand from the build procedures./ Notify a NOS system analyst.
ERROR IN ASSOCIATED VARIANT DEFINITION	/The user should correct the error in the USERBPS file, and rerun the build step.
ERROR IN MACRO CALL FILE	An erroneous file was passed to Expand from the build procedures./ Notify a NOS system analyst.
ERROR IN MACRO TEXT FILE	Probably a section of expected text was not found in the file. The file is generated from the EXPTEXT deck on the CCP program library./ Notify a NOS system analyst.
ERROR IN USERBPS FILE	/The user should correct the error in the USERBPS file and rerun the build step.
INCOMPLETE SYMBOL	An end-of-line was found before the delimiter in the USERBPS./ The user should correct the error in the USERBPS file, and rerun the build step.
LINE TOO LONG AFTER SUBSTITUTION	When the symbol was substituted in the text line, an overflow condition occurred. The file is generated from the EXPTEXT deck on the CCP program library./ Notify a NOS system analyst.
NIL SYMBOL	No characters were found before the delimiter in the USERBPS./ The user should correct the error in the USERBPS file, and rerun the build step.
SYMBOL TOO BIG	The character string had more than ten characters in the USERBPS./ The user should correct the error in the USERBPS file, and rerun the build step.

TABLE B-3. AUTOLINK INFORMATIVE MESSAGES

NOTE:

Some Autolink messages are prefaced with one of two messages:
***** ERROR *****
***** WARNING *****

Message	Meaning/Action to be Taken
AUTOLINK COMPLETED	Informative message only./ No action is required.
AUTOLINK OUTPUT DIRECTIVES LIST	Informative message only. This message is followed by the directives which serve as input to MPLINK./ No action is required.
DELETED TOPHAT MODULE LIST	The listed modules had a tophat program that was deleted, since the modules were located in main memory rather than in paged memory./ No action; informative only.
EXCEEDS THE 18 COLUMN REPORT LIMITATION	A BUFSP report has columns for all applications to be included in the build report, and for all the memory sizes to be tested. A maximum of 18 columns can be fit on the output report page. If the combination of APPL directives and BUFSPSIZE sizes exceeds 18, this message is generated./ Reduce the number of applications in the build, or reduce the number of memory sizes for the report.
INVALID ADDRESS SPECIFICATION	The address parameter in a MOD or RESERVE directive is not valid./ Use a legal address: addresses should be within memory size, and be expressed as a four-digit hexadecimal value starting with a dollar sign ($).
INVALID (C/NC/ADDRESS) ATTRIBUTE	While parsing an APPL directive, Autolink found an invalid address parameter, or the right parenthesis was missing./ Correct the invalid parameter.
INVALID LIST FILE	The list file is not named correctly, the first character of the name is not a letter, or the file is missing./ Correct the naming error, or make the file available.
INVAID MOD TERMINATOR	In a MOD directive, the terminator should be a comma except after modname (an opening parenthesis), or after the last appl value (a closing parenthesis)./ Correct the error.
INVALID OR MISSING LGO FILE	The load-and-go file is not named correctly, the first character of the name is not a letter, or the file is missing./ Correct the naming error, or make the file available.
INVALID P= PARAMETER	In the MOD directive, the P parameter must take the value of P, NP, F, or R./ Correct the error.
INVALID SECONDARY BINARY FILE	The secondary binary file is not named correctly, the first character of the name is not a letter, or the file is missing./ Correct the naming error or make the file available.
MORE THAN 60 APPLICATIONS DEFINED	No more than 60 APPL directives are allowed./ Eliminate the unnecessary APPL directives.
MORE THAN 60 DEF PARAMETERS	The combined number of DEF and DEFBASE directives exceeds 60./ Eliminate the unnecessary directives.
MORE THAN 64 BUFSPSIZE PARAMETERS	No more than 64 parameters can be associated with the BUFSPSIZE directive./ Eliminate the unnecessary parameters.

TABLE B-3. AUTOLINK INFORMATIVE MESSAGES (Contd)

Message	Meaning/Action to be Taken
MULTIPLE CORESIZE DIRECTIVES ENTERED	Only one CORESIZE directive can be entered with an input directives file./ Select the correct CORESIZE directive for the file, and eliminate the others.
NO APPLICATIONS GIVEN FOR THESE MODS:	The list names the object code modules which have no corresponding application./ Correct the error.
NO MOD DIRECTIVES FOR THESE OBJECT PGMS:	The listed object code programs were found on the input object code file, but were not named in a MOD directive./ Check the names, and add MOD (and other) directives if these modules should be a part of the variant build.
NON-NUMERIC (HEX) DIGIT	A character is not recognized as a legitimate hexadecimal digit in one of the following directives: BUFSPSIZE, CORESIZE, PAGEREG, PAGESIZES, or RESERVE./ Correct the error and reenter the directive file.
PAGE OVERFLOW FORCED BY USER	The F parameter in the MOD directive forces an application length that requires more space than is allocated to the page./ Change F modules so that fewer are forced to the same page as their applications.
REPORTS REQUESTED BUT NO LIST FILE GIVEN	The output file for reports was not specified in the program name-call statement./ Correct the name-call statement to specify an output file for the reports.
UNDEFINED APPLICATION NAME	The appl name specified in the APPL parameter of the MOD directive was not specified in any APPL directive./ Use a correct APPL parameter, or enter the correct APPL directive.

TABLE B-4. AUTOLINK FATAL ERROR MESSAGES

NOTE:

Fatal error messages are prefaced with the message
EXECUTION TERMINATED DUE TO ERRORS
************FATAL ERRORS************

Message	Meaning/Action to be Taken
CORSIZE GREATER THAN 128	Autolink detected a value greater than 128 in a CORESIZE or BUFSPSIZE directive./ Correct the invalid memory size parameter.
EOF ENCOUNTERED, CONTINUATION EXPECTED	Additional MOD parameters were expected (APPL is required; TH is required if P=P). Instead, an EOF was detected in the directive./ Add the required parameter, or parameters, to the directive.
INVALID DIRECTIVE TERMINATOR	In a MOD directive, a comma or blank followed an application name, or a nonblank character followed the right parenthesis terminator./ Correct the error, and reenter the directive file.
INVALID INPUT DIRECTIVE	Format of input directive is not valid./ Correct the input directive, and reenter the directive file.

TABLE B-4. AUTOLINK FATAL ERROR MESSAGES (Contd)

Message	Meaning/Action to be Taken
INVALID MOD NAME	While parsing a MOD directive, Autolink could not parse the modname parameter. A modname consists of six (or more) characters starting with a letter./ Correct the error.
INVALID MOD SUBPARAMETER	In one of the parameters in a MOD directive, the parameter name is not correct (must be P, ADDR, FILL, TH, or APPL), or the = is missing, or a value is illegal./ Correct the error.
INVALID OR MISSING INPUT DIRECTIVES FILE	The input directives file is not named correctly, the first character of the name is not a letter, or the file is missing./ Correct the naming error, or make the file available.
INVALID OUTPUT DIRECTIVES FILE	The output directives file is not named correctly, the first character of the name is not a letter, or the file is missing./ Correct the naming error, or make the file available.
INVALID REPORT TYPE REQUEST	An invalid report name was used in the RPT directive./ Use a valid name (BUFSP, DIR, MAP, or INFO).
INVALID RESERVE ADDRESS	The beginning and ending reserve addresses should be separated by a comma./ Correct the error.
MAIN MEMORY EXCEEDED	The variant requires more then 64K words of main memory, or addresses are assigned which prevent the variant from being located within main memory./ Check applications that have preassigned addresses. Delete applications as necessary.
MODULE=xxxxx OVERLAPS ANOTHER MODULE or MODULE OVERLAP ERROR	When locating modules, Autolink detected that addressed module space overlapped into nonaddressed module space./ Adjust the lowest module address.
MORE THAN 32 PAGE SIZES	More than 32 parameters were used in one PAGESIZES directive./ Use only one value per directive.
NO MODS DEFINED FOR THESE APPLICATIONS	This messages lists the APPL directives which have no MOD directives naming them in an APPL parameter./ Delete the APPL directives, or add MOD directives with these applications specified.
NO OBJECT TEXT FOR THESE MOD DIRECTIVES: or NO OBJECT TEXT FOR THESE REQUIRED MODS:	This message lists modules which should have been present in the input object code file because they were specified by MOD statements./ Remove the MOD directives or include the object code for the modules in the input object code file.
NON-MOD DIRECTIVE FOLLOWS MOD DIRECTIVE	At this stage in the input directives file, there should be only MOD directives; all other directives should have occurred earlier in the file./ If the directive is valid, place it before the MOD directives.
PAGE REGISTER GREATER THAN 31	The page register parameter value in a PAGEREG directive is greater than 31./ Change the parameter to a legal value (range 0 through 31).
PAGE SIZE GREATER THAN 64	The pagesize parameter in a PAGESIZES directive is greater than 64./ Change the parameter to a legal value (2K, 4K, 8K, or 16K).

TABLE B-4. AUTOLINK FATAL ERROR MESSAGES (Contd)

Message	Meaning/Action to be Taken
********WHILE LOADING FILLER MODULES nnn MODULES REMAIN UNLOADED xxxxxx . . . xxxxxx INDEX J = iiii MODCNTR = iiii NEXTADR = hhhhh MAXADDR = hhhhh	Either more modules are being loaded than are specififed by the MOD directive (index J is greater than the module counter), or more main memory is needed (next address is greater than maximum address). Names of the modules are given; nnn is a decimal number; hhhhh addresses are given in hexadecimal./ Correct the problem by retrying. If the build attempt still fails, contact a systems analyst.

TABLE B-5. MPLINK ERROR MESSAGES

Message	Meaning/User Action
*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 directives.
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.
ASSIGNEMENT OPERATOR EXPECTED	The expression evalauator expected the := operator in the *VE directive but did not find one./ The users 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.
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.

TABLE B-5. MPLINK ERROR MESSAGES (Contd)

Message	Meaning/User Action
CURRENT UPPER LIMIT EXCEEDED	During a 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 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.
EXPRESSON 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 been previously absolutized./ The user should alter the *VE expression, or the order in which directives are added, so that the name is absolutized by the time MPLINK encounters the *VE directive.
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.
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.
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).

TABLE B-5. MPLINK ERROR MESSAGES (Contd)

Message	Meaning/User Action
INCORRECT GROUP SPECIFICATION	The module names specified by the mod1-mod2 parameter in a *L or *RL expression are not correct; that is, mod1, 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 (mod1-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 mod1-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 evalauator expected an 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.
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.

TABLE B-5. MPLINK ERROR MESSAGES (Contd)

Message	Meaning/User Action
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 INDENT	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.

TABLE B-6. MPEDIT ERROR MESSAGES

Message	Meaning/Programmer Action
(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 an array statement has 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 has 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.
**** 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 a defining operand./ The programmer should revise the 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.
= EXPECTED	MPEDIT syntax requires that the next element of the constant definition be an equal sign./ 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.

TABLE B-6. MPEDIT ERROR MESSAGES (Contd)

Message	Meaning/Programmer Action
ARRAY TABLE OVERFLOW	The number of arrays declared in the ARRAY section exceeds the capacity of MPEDIT./ The programmer show 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.
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 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: CONST VAR ARRAY BEGIN END OVERLAY FOR TO DO OF CHAR DIV MOD The keyword found 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.
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.
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.

TABLE B-6. MPEDIT ERROR MESSAGES (Contd)

Message	Meaning/Programmer Action
NO ASSOCIATED LEFT PARENTHESIS	The expression evaluator found a right parenthesis that was not preceeded 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 has the form: ARRAY size OF x. The array statement lacks OF following the array size specification./ The programmer should revise the composite statement.
OPERAND EXPECTED	The expression evaluator expected an 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.

GLOSSARY AND MNEMONICS C

Absolute Address (NPU) -
The location of a word in NPU memory measured by its displacement from word 0.

Application (CCP or CCI) -
A set of modules which are designed to be executed together to perform a special function. In CCP and CCI, applications include a HIP, a LIP (CCP only), any of the standard or user-written TIPs, the base modules, initialization routines, interactive virtual terminal common routines (CCP only), on-line diagnostics, the buffer space, the service module, and the test utilities.

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

Autolink -
A CYBER Cross utility program which generates an input directives file for the MPLINK utility.

Base -
A set of programs that is used in every CCP or CCI build.

Blank Common -
A common area used by macroassembler programs.

Buffer Space Report -
An optional autolink output. It reports the amount of buffer space available as a function of the main memory size, and the TIPs selected for a build.

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.

Directives Report -
An autolink report which lists the directives in the file that is sent to MPLINK.

Dump Memory -
A hexadecimal listing 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) which 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 which 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 which 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 that halt the utility, and nonfatal errors that are noted, but allow the utility to continue processing.

Expand -
The build utility that expands variant definitions into directives for the CCP variant build and load file generator installation steps.

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

Fatal Error File -
An autolink file containing the fatal error message generated during an autolink processing phase.

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.

Fill Module -
A module that can be assigned to fill holes in main memory, or (in some cases) to fill a hole on a page.

Forced Page Module -
A module that must reside on the same page as the paged portion of its application.

Information Report -
An autolink report listing the input directives and application lengths.

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.

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.

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

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 an an input to the autolink, MPLINK, MPEDIT and MPLIB utilities. The MPLIB utility generates a new library.

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

Library Maintenance -
The function performed by 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) which links object code modules into 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 (1) locating (assigning space) for object code modules on a memory image load module file, and (2) 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, edited, absolutized object code modules. 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 absolutized code 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 autolink or MPLINK report showing the main memory location of every module in the build.

MOD Directive -
An autolink directive which identifies a module to be included in an application package.

Modname -
The name of a module. Autolink uses the first six characters of the name.

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 except the expand utility.

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

Output Directives -
An autolink output file containing the MPLINK directives used to link the system modules during an MPLINK run.

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.

Package (CCP or CCI) -
A special class of applications that handle terminal, host or link interfaces; for example, the terminal interface packages (TIPs).

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

Pageable Module -
A module that can be located on the application page, or in main memory if the application's page is full.

Passive MPLINK Directives -
Directives not processed by autolink, but passed directly to MPLINK in the output directives file. These are : *COM, *COR, *DAT, *DSTK, *DMP, *DVAR, *ENT, *LIB, *SYN, and *SYSID.

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

Report -
One of the reports that is associated with the Autolink, 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 Package (TIP) -
An application that handles the interface between the NPU and a type of terminal, such as TTY terminals, or Mode 4 terminals.

Tophat -
Indicates a tophat module. A tophat module is normally a module that is called by several other modules. To minimize the code required to locate a tophat module's entry point, a small auxiliary piece of code is compiled 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.

User Build Parameters File -
A user file (USERBPS) which contains the CCP variant load module definitions and the CCP load file definitions. Expand uses the file to generate Update directives based on the variant definition.

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

*ALL	Copy all LGO files to new library - MPLIB directive
*CB	Upper boundary declaration - MPLINK
*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 stack 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
*LST	List the library - MPLIB directive
*OVLY	Define overlay-area directive - MPLINK
*PUT	Insert/replace module in library - MPLIB directive
*RL	Reverse-linking directive - MPLINK
*SUP	Suppresses copying programs from the LGO to the library - MPLIB
*SYN	Define external synonym - MPLINK directive
*SYSID	System identification - MPLINK directive
*UL	Upper limit directive - MPLINK
*VE	Directive which assigns a value to a local variable - MPLINK
/DMP$	List the load tape - MPEDIT statement
/ENTRY	Address entry function - MPEDIT statement
/ESL$	List SYMTAB - MPEDIT statement
/LENGTH	Field length function - MPEDIT statement
/NAM$	Generate the NPU load tape - MPEDIT statement
/PGDISP	Page displacement function - MPLINK/ MPEDIT statement
/PGNUM	Page resister number function - MPLINK/MPEDIT statement
/PGSET	Page register set function - MPLINK/MPEDIT statement
/START	Field state location function - MPEDIT statement
/TRACE	Trace of edit operations - MPEDIT statement
/VFD	Variable field definition - MPEDIT statement
ABSOLMP	Absolute memory image load file
ARRAY	Array declaration command - MPEDIT
BEGIN	Begin statement - MPEDIT
BIP	Block interface package
BUFSIZE	Buffer space report directive - Autolink
BUFSP	Buffer space report - Autolink
CCP	Communications Control Program
CHAR	Character mode, array declaration - MPEDIT
CONST	Constant declaration - MPEDIT
CORESIZE	NPU memory size directive - Autolink
CSET	CDC code set variable
D	Input file parameter - MPLINK directive
DEF	Define applications - Autolink directive
DEFBASE	Define applications variant - Autolink directive
DIR	Output directives report - Autolink directive
DIV	Division operator - MPEDIT
DO	Part of MPEDIT loop directive (FOR x TO y DO...)
END	End statement of a composite statement - MPEDIT
FOR	Part of MPEDIT loop directive (FOR x TO y DO...)
GLOBL$	CCP/CCI data base area
HIP	Host interface package
INFO	Input directives and application lengths report
INPUT	Default input file
LFG	Load file generator - Expand
LGO	Load-and-go file
LIP	Link interface package

LOADER	First record on a memory image load module tape
MAP	Main memory map report - Autolink
MOD	Module directive - Autolink
MOD	Modulus operator - MPEDIT
MPEDIT	Edit utility
MPLIB	Library maintenance utility
MPLINK	Link utility
NEWLIB	New library file - MPLIB
OF	Part of array declaration - MPEDIT
OUTPUT	Default output file
OVERLAY	Overlay identifier - MPEDIT
PAGESIZES	Page size directive - Autolink
RESERVE	Reserve main memory - Autolink directive
RPT	Report generator directive - Autolink
SVM	Service module
SYMTAB	Symbol table file
TIP	Terminal interface package
TO	Part of MPEDIT loop directive (FOR x TO y DO...)
USERBPS	User build parameters file - Expand
VAR	Variable declaration - MPEDIT
VRD	CCP variant definition - Expand
X.25	A TIP

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. 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 non-initialized 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 macroassembler, microassembler, 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.

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. 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).

In this 30-word record the fields are:

- 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. 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 a *L or *RL directive.

In this 30-word record the fields are:

- 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 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$

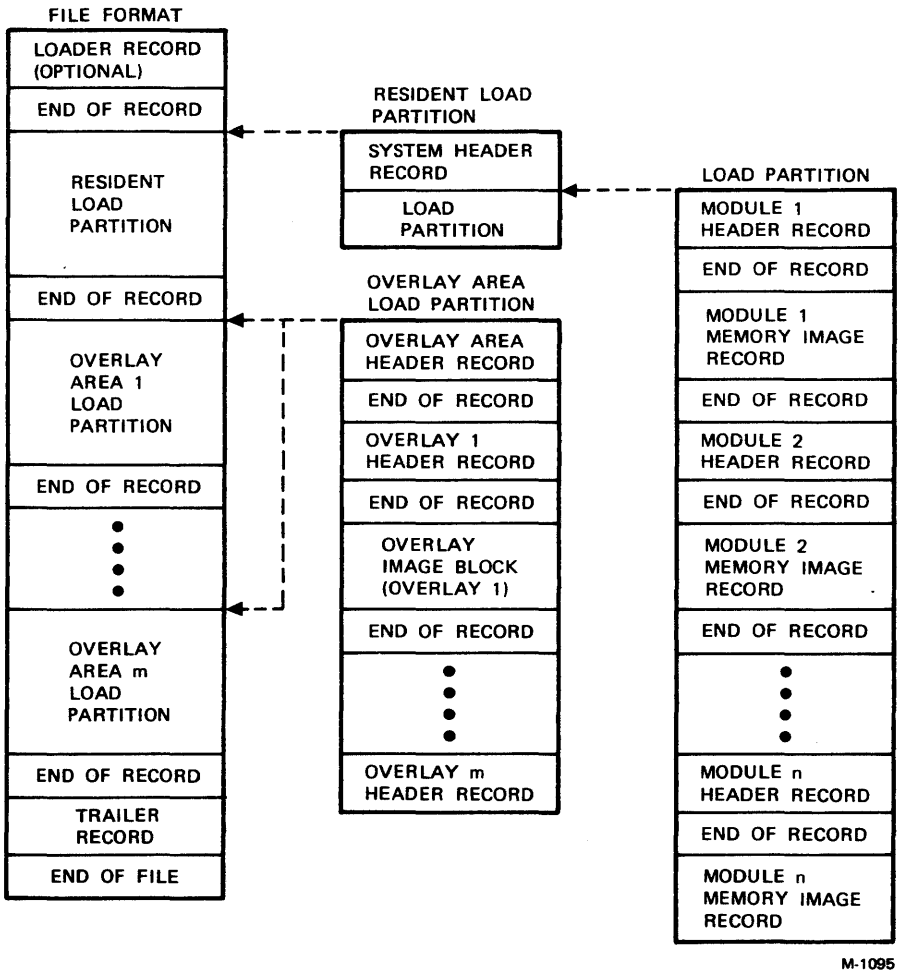


Figure D-1. Format of an MPLINK or MPEDIT Output Load File

- The name associated with a PASCAL-compiled program
- The name specified on the NAM card of a macroassembled program
- The name of a microassembled program

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

- 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).

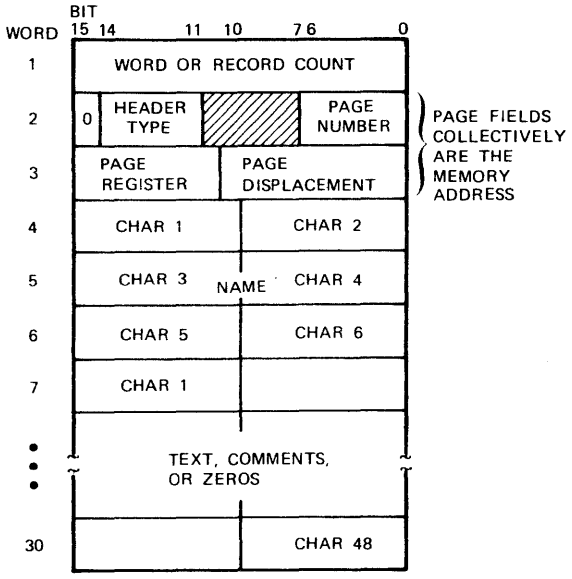


Figure D-2. Format of Load File Header Record

- 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 a *L or *RL directive following an overlay declaration.

In this 30-word record the fields are:

- 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).

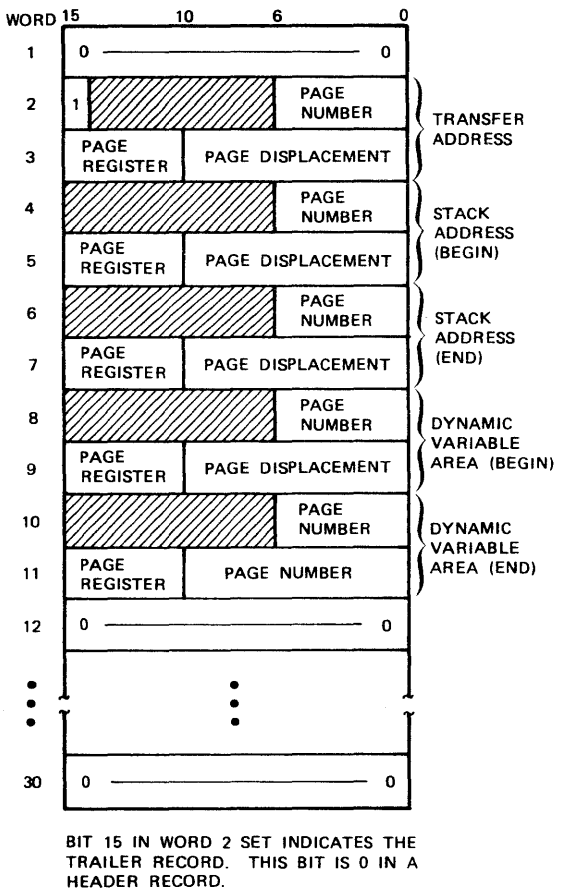


Figure D-3. Format of Load File Trailer Record

- 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.

OPTIONAL MEMORY IMAGE LOAD MODULE FILE FORMAT E

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 15 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.

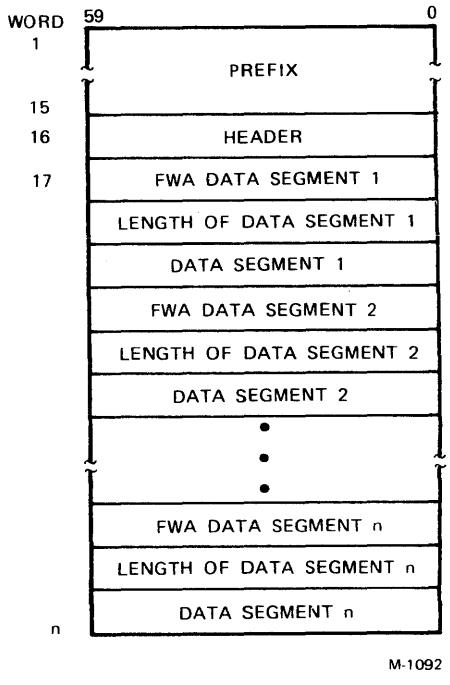


Figure E-1. Optional Load Module Record Format

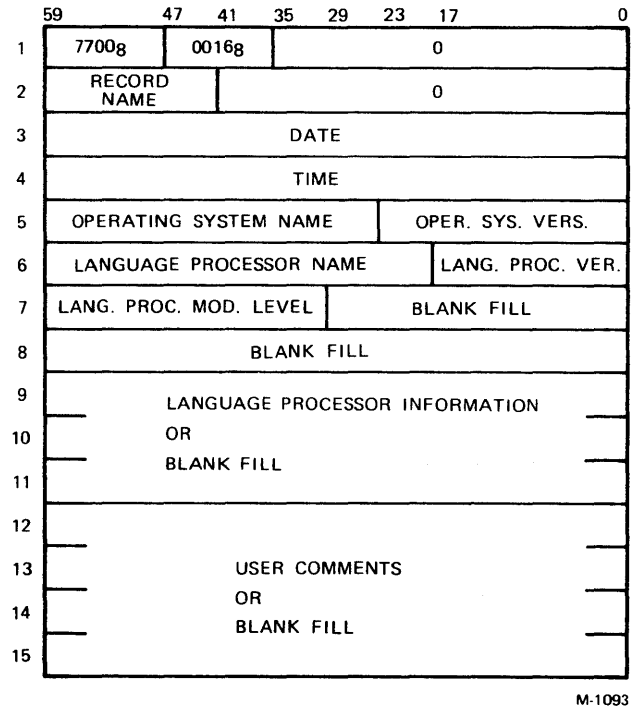


Figure E-2. Optional Memory Image Record Prefix Format

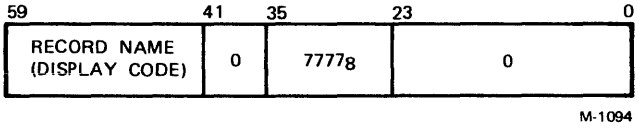


Figure E-3. Optional Load Module Record Header Format

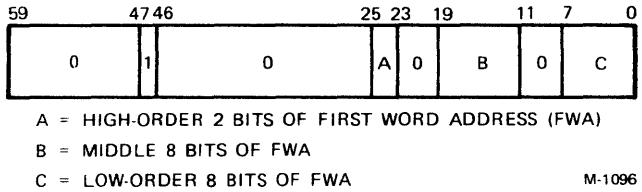


Figure E-4. Optional Memory Image Record First Word Address Format

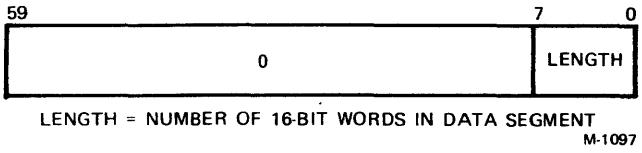


Figure E-5. Optional Memory Image Record Length Format

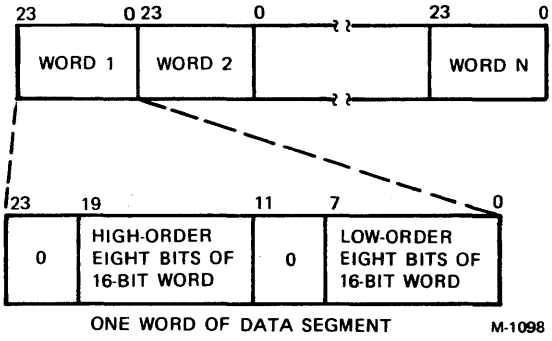


Figure E-6. Optional Load Module Record Segment Format

RECORD BLOCK

The object text input to the autolink or 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 of 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 text block

Word 3 + n + 1 = the checksum

A record block will not exceed one card image, thus the length of an object text 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; and if the data portion is less than 57 words, the record block is padded to fill a complete 80-column card image.

OBJECT TEXT BLOCK

The object text 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 text word. The following object text block types are defined:

Type	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 1, and all other bits set equal 0.

A module's object text 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 text 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.

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.

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

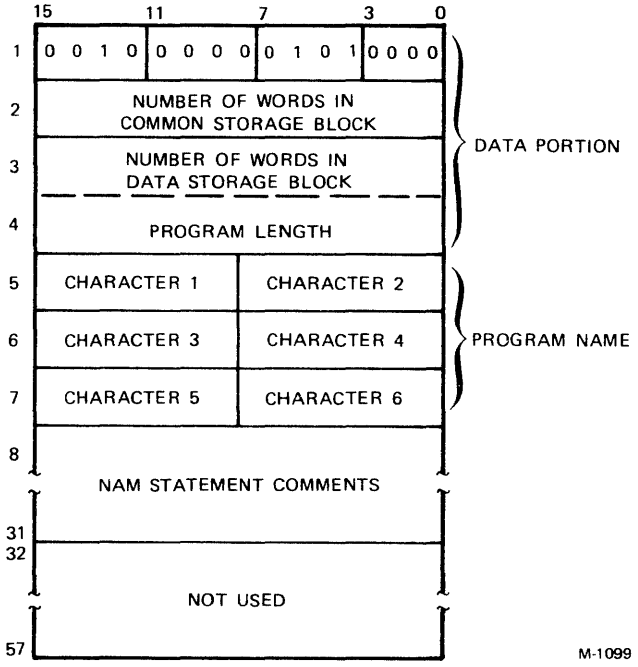


Figure F-1. NAM Block Image

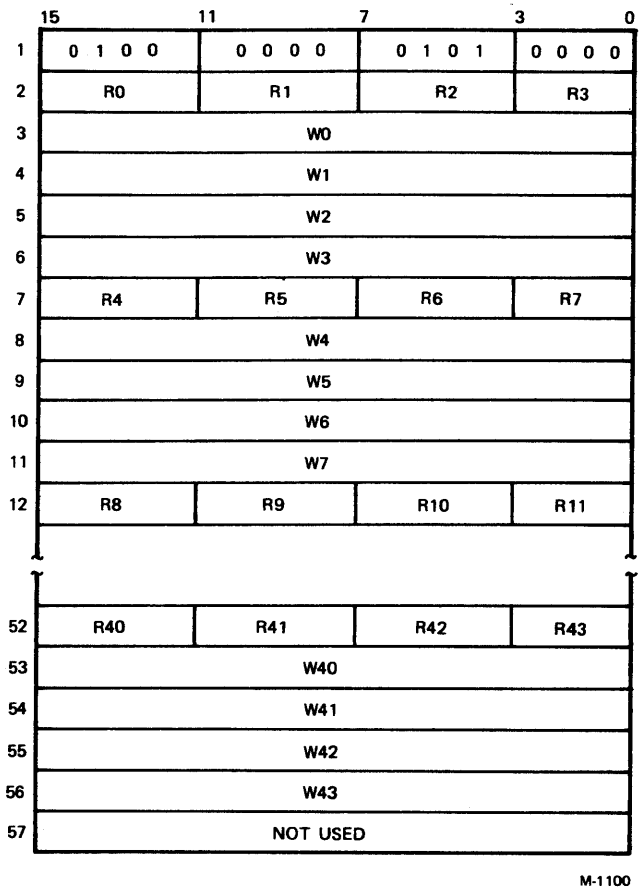


Figure F-2. RBD Block Image

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

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, 0011 is assumed. Zero is the leading bit for all but the last relocation byte 1 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.

where A is the starting address; S is the size of the area reserved by BZS; R is the relocation of the starting address; An is the starting address of the nth entry; Sn is the size of the BZS reservation for the nth entry; and Rn is the relocation byte of the nth entry.

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

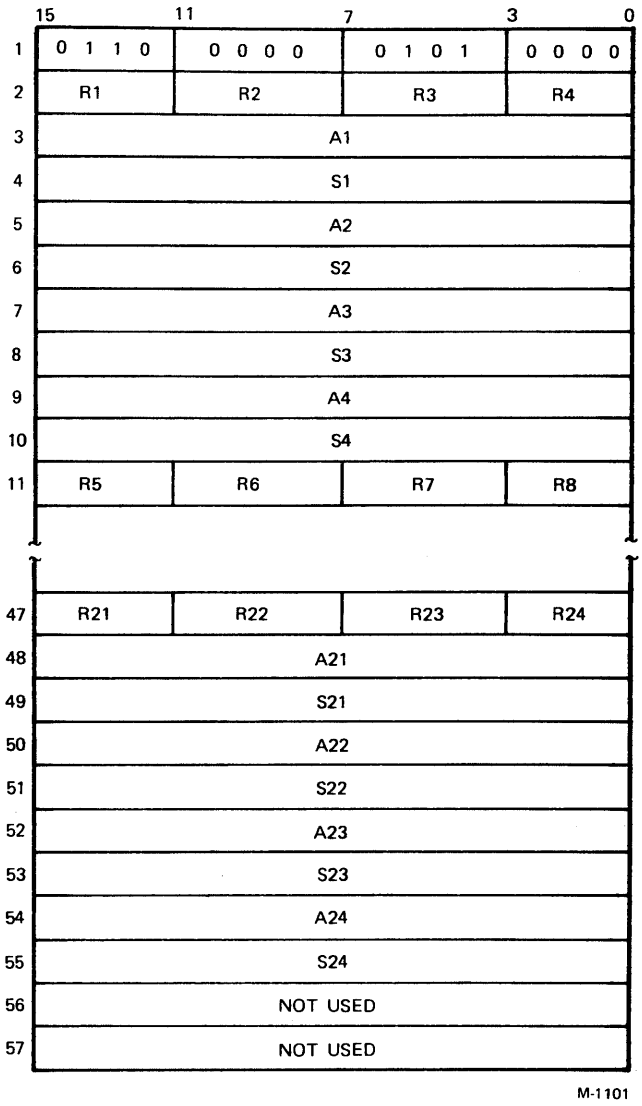


Figure F-3. BZS Block Image

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.

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

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

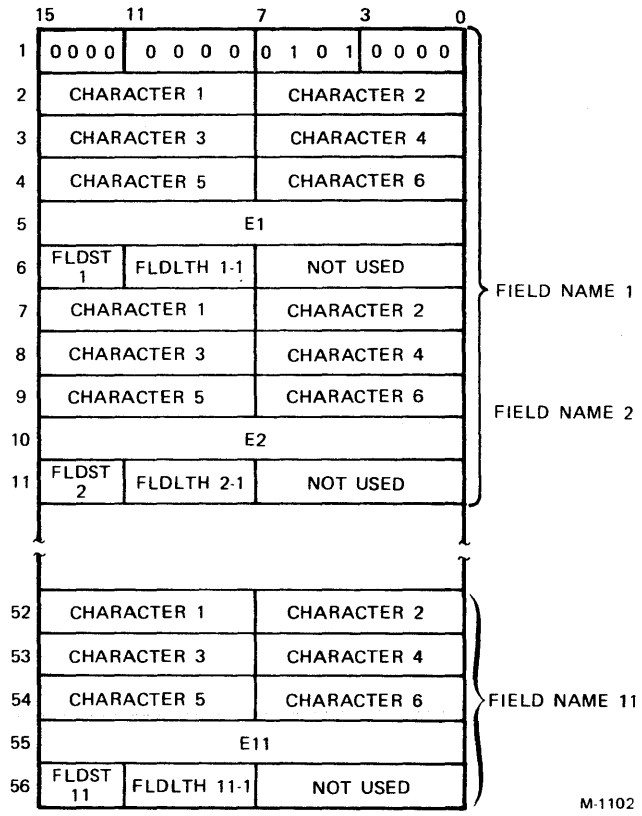


Figure F-4. ENF Block Image

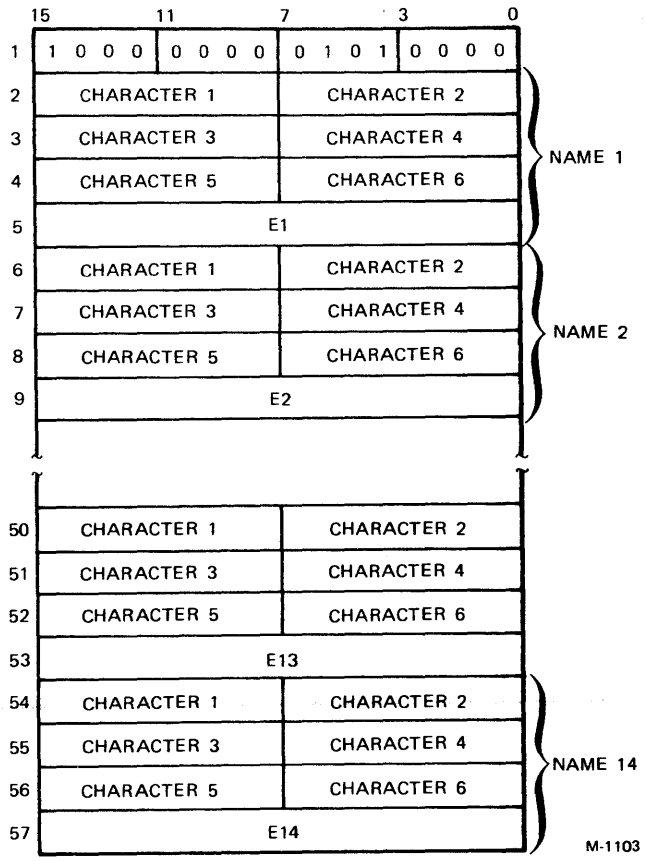


Figure F-5. ENT Block Image

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.

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

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), 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.

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

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 which contains no references within the modules object text is indicated by a $8000 in the link address.

EXT BLOCK

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

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

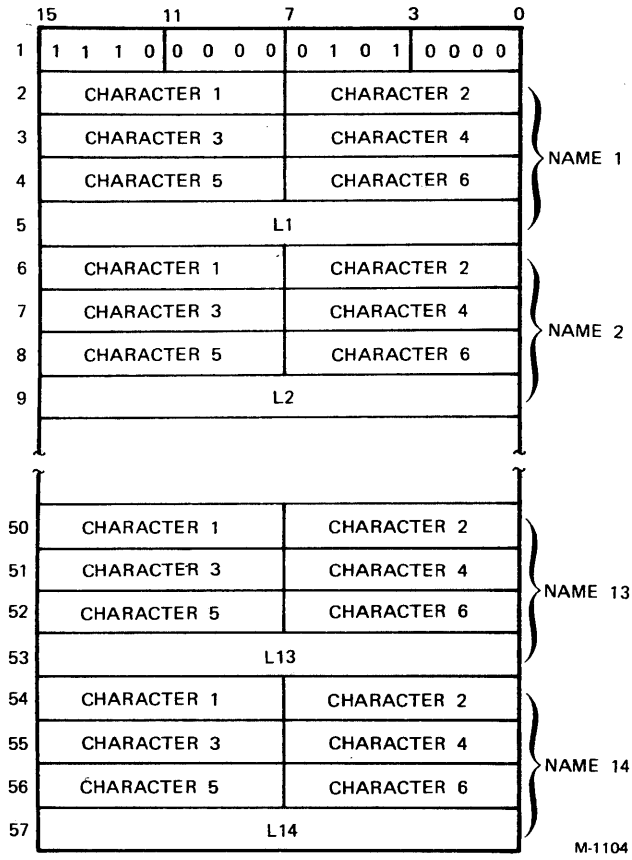


Figure F-6. EXF Block Image

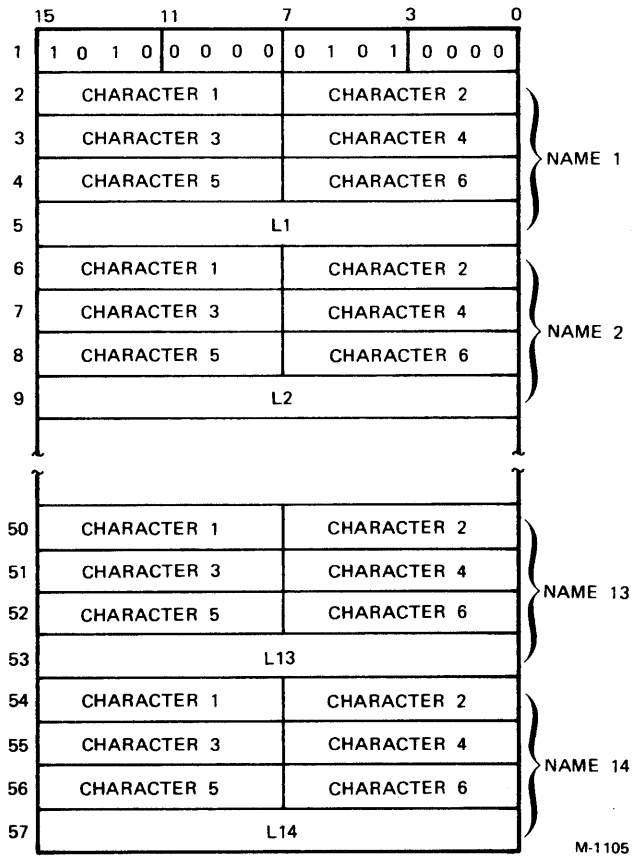


Figure F-7. EXT Block Image

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.

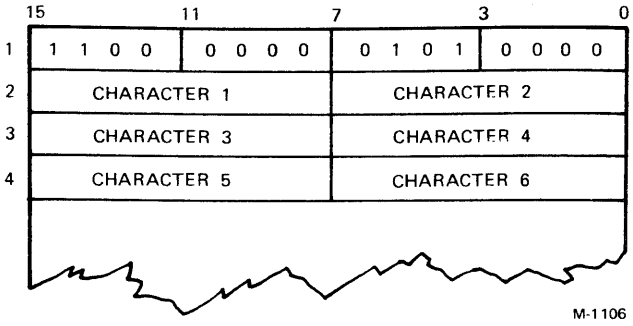


Figure F-8. XFR Block Image

AUTOLINK UTILITIES EXAMPLES G

This appendix gives examples of Autolink files and messages. The examples are:

- A partial Autolink directives file (figure G-1)
- A list of the applications included in the build generated by the directives file of figure G-1 (figure G-2)
- An informative message noting the object programs in the input object code file which did not have an associated MOD directive (figure G-3)
- An informative message noting the relocation vectors (tophats) that were deleted during the space assignment of tophat modules (figure G-4)
- An informative message giving the buffer area for the build, both test area and user buffer areas (figure G-4)

```
LINKGEN INPUT DIRECTIVES LIST


*
*-  AUTO-LINK INPUT DIRECTIVES,  THIS DECK AND SUPPORTING COMDECKS
*-  PROVIDE THE AUTO-LINK PROGRAM WITH INFORMATION NECESSARY TO
*-  PRE-PROCESS THE CCP OBJECT FILE (FULL COMPILE OPTION) FOR
*-  DETERMINING THE OPTIMUM MEMORY ALLOCATION.
*
*SYSID,R96,CCP VARIANT LOAD MODULE.
*
*-  PAGE DEFINITIONS, 4 PAGE REGISTERS, PAGESIZE IS 8K.
*
PAGEREG = 4
PAGESIZES = 8
*
*-  STACK RELATED DIRECTIVES, STACK IN LOCATIONS 40 THRU FC.
*
*ENT,DSTKFW,$0040.
*ENT,DSTKLW,$00FC.
*ENT,DVARFW,$0000.
*ENT,DVARLW,$0000.
*DSTK,$0040,$00FC.
*
*-  MEMORY RESERVE AREA, OVERLAY AREA IS LOCATIONS 170 THRU D9F.
*-  LOCATIONS 0 THRU 16F ALSO RESERVED FOR MISC PURPOSES.
*
RESERVE=$0170,$0D9F
RESERVE=$0000,$016F
*
*-  MEMORY SIZE IS SET ACCORDING TO UPDATE *DEFINE DIRECTIVE.
*-  POSSIBLE SIZES ARE 65K, 81K, 96K OR 128K.
*
CORESIZE = 96
*
*-   SET UP SYNONYM NAMES FOR MODE4 TRANSLATIONS TABLES
*
*SYN,VI4AASC,VB4AASC.
*SYN,VI4ABTA,VB4ABTA.
*
*-  THE FOLLOWING DIRECTIVES SPECIFY THE APPLICATIONS THAT ARE
*-  IN THE CCP OBJECT FILE.  FOUR OF THE APPLICATIONS ARE CONDITIONALLY
*-  SPECIFIED IN THAT THEY MAY OR MAY NOT BE IN THE OBJECT FILE.
*-       (1) TESTGEN - NONSTANDARD USED ONLY BY CDC DEVELOPMENT.
*-       (2) UTOPIA - NONSTANDARD USED ONLY BY CDC DEVELOPMENT.
*-       (3) HLIP - DEPENDS ON PURCHASE OF REMOTE CONCENTRATOR.
*-       (4) OLDSYS - DEPENDS OF PURCHASE OF ON-LINE DIAGNOSTICS.
*-
*-  NOTE- THIS SET OF DIRECTIVES SPECIFY WHICH APPLICATIONS ARE ON
*-  THE OBJECT FILE. THEY DO NOT CONTROL WHICH APPLICATIONS ARE
*-  INCLUDED IN THE VARIANT BUILD.  THE VARIANT BUILD IS CONTROLLED
*-  BY THE DIRECTIVES IN COMDECK ALDEFS.
*
APPL = SVMODULE
APPL = ASYNCEXT
APPL = ASYNC
APPL = MODE4
APPL = HASPTIP
APPL = HIP
APPL = HLIP
APPL = OLDSYS
APPL = IVT
APPL = CONSOL
APPL = BASESYS
APPL = PIBUF2
APPL = TUP
APPL = INITIAL
*
     .
     .
     .
*-  GENERATE SYNONYMS FOR THOSE APPLICATIONS THAT ARE NOT INCLUDED
*-  IN THIS VARIANT BUILD.
*
*SYN,PTHIPI,PBILL.
*SYN,PTHIPQ,PBILL.
*SYN,PBDEBUG,PBILL.
*SYN,UTOPIA,PBILL.
*SYN,PBBREAKPOI,PBILL.
*SYN,PBWRAPSNAP,PBILL.
*SYN,PB2SNAP,PBILL.
     .
     .
     .
```

Figure G-1. Autolink Input Directives File (Partial)

```
*
*-  THE FOLLOWING DIRECTIVES CALL THE SUPPORTING AUTO-LINK COMDECKS,
*-  (1) SETUP REPORT DEFINITIONS,
*-  (2) DEFINE APPLICATIONS TO INCLUDED IN THIS VARIANT BUILD,
*-  (3) DEFINE WHICH OBJECT MODULES BELONG TO WHICH APPLICATIONS.
*-  NOTE- MOD DIRECTIVES MUST BE LAST ON AUTO-LINK'S INPUT FILE.
+
*
*-                  COMDECK ALRPT
*-
*-  THIS COMDECK SETS UP REPORT DIRECTIVES FOR AUTO-LINK ACCORDING
*-  *DEFINES ( BUFSRPT,DIRECTRPT,MEMAPRPT,TOPHATRPT ).
+
*
*-                       COMDECK ALDEFS
*-
*-  THE DIRECTIVES IN THIS COMDECK CONTROL WHICH APPLICATIONS ARE
*-  INCLUDED IN A VARIANT BUILD.  SOME APPLICATIONS ARE ALWAYS
*-  INCLUDED WHILE OTHERS ARE ONLY INCLUDED IF THEY ARE SPECIFIED
*-  BY UPDATE DIRECTIVES *DEFINE.
*-
+
DEF = SVMODULE
DEF = IVT
DEF = BASESYS
DEF = INITIAL
DEF = PIBUF2
DEF = PIDTBL
DEF = ASYNCEXT
DEF = MODE4
DEF = HASPTIP
DEF = HLIP
DEF = OLDSYS
*
*-                       COMDECK ALSMOD
*-
*-
*-  THIS APPLICATION IS PAGEABLE    (P     = P) AND
*-  NON-PAGEABLE    (P     = NP)
+
MOD = PNSMWL
              (P     = F,
              ADDR = $2000,
              APPL = SVMODULE)
MOD = PNAWAIT
              (P     = P,
              APPL = SVMODULE)
MOD = PNRTN
              (P     = P,
              APPL = SVMODULE)
    ●
    ●
    ●
*
*-  THE FOLLOWING APPLICATIONS IS LINKED INTO ANY
*-  PAGE.
+
MOD = PNCONFIGURE
              (P     = P, FILL,
              APPL = SVMODULE)
MOD = PGDSTAT
              (P     = P, FILL,
              TH   = PNDSTAT,
              APPL = SVMODULE/BASESYS)
MOD = PNSMGEN
              (P     = P, FILL,
              APPL = SVMODULE)
    ●
    ●
    ●
*-  THIS APPLICATION IS PAGEABLE.
+
MOD = PTIVTPRSR
              (P     = P, FILL,
              APPL = IVT)
MOD = PGIVTCMD
              (P     = P, FILL,
              TH   = PTIVTCMD,
              APPL = IVT/BASESYS)
    ●
    ●
    ●
```

Figure G-1. Autolink Input Directives File (Partial) (Contd)

```
*
*-  THE REST OF THIS APPLICATION IS NON-PAGEABLE  (P     =NP)
*-  AND MUST RESIDE IN THE BASE.
*-
*-     NOTE THAT PIDTBL IS A SEPERATE APPLICATION.
*-    THIS IS NECESSARY TO CONTROL THE LINKING.
*-     PIDTBL IS THE LAST MODULE LINKED IN THE
*-     FORWARD DIRECTION.
*
MOD = PIDTBL
              (P     = NP,
              APPL  = PIDTBL)
MOD = GLOBL$
              (P     = NP,
              ADDR  = $00A0,
              APPL  = BASESYS)
MOD = ZEROX
              (P     = NP,
    ●
    ●
    ●
*-
*-  THIS APPLICATION IS LINKED IN REVERSE ORDER.
*-
*-  APPLICATION TUP IS ALWAYS INCLUDED IN A VARIANT BUILD. HOWEVER,
*-  TUP IS OVERLAYED BY BUFFERS UNLESS THE VARIANT TYPE IS SPECIFIED
*-  AS TYPE T (TEST BUILD) IN WHICH CASE THE VARIABLE TOTUP IS SET
*-  TO 1.
*-
*
MOD = PBBREAKPOI
              (P     = R,
              APPL  = TUP)
MOD = PBWRAPSNAP
              (P     = R,
              APPL  = TUP)
MOD = PB2SNAP
              (P     = R,
              APPL  = TUP)
MOD = PB1SNAP
              (P     = R,
              APPL  = TUP)
    ●
    ●
    ●

*-  THIS APPLICATION IS LINKED IN REVERSE ORDER.
*-  PIMLIA RESIDES IN HIGHEST CORE AND MAIN$ RESIDES IN
*-  LOWEST CORE.
*
MOD = PIMLIA
              (P     = R,
              APPL  = INITIAL)
MOD = PITMRS
              (P     = R,
              APPL  = INITIAL)
MOD = PILININIT
              (P     = R,
              APPL  = INITIAL)
MOD = PIAPPS
              (P     = R,
              APPL  = INITIAL)
    ●
    ●
    ●
*
*-                    COMDECK ALMODE4
*-
*-  THIS APPLICATION IS NON-PAGEABLE AND
*-  PAGEABLE.
*-
*
MOD = R4M4IN
              (P     = NP, FILL,
              APPL  = MODE4)
MOD = R4M4CC
              (P     = NP, FILL,
              APPL  = MODE4)
MOD = R4M4TP
              (P     = P,
    ●
    ●
    ●
```

Figure G-1. Autolink Input Directives File (Partial) (Contd)

```
                (P     = NP, FILL,
                 APPL  = X25CF1/ X25CF2)
MOD = PXPAD3
                (P     = P, FILL,
                 APPL  = L2DEBUG/ L3DEBUG)
MOD = PX2MUX
                (P     = F,
                 APPL  = X25L2/ X25CF1/ X25CF2)
MOD = PX2OPS
                (P     = F,
                 APPL  = X25L2/ X25CF1/ X25CF2)
MOD = PX2POBT
                (P     = F,
                 APPL  = X25L2/ X25CF1/ X25CF2)
*
*-  END OF ALINPDIR DECK,  START OF SUPPORTING COMDECKS
*

 ************************************************

 FOR TEST PURPOSES THE BUFFER AREA OF THIS BUILD IS
      DECIMAL = 18338          HEXADECIMAL = $47A2

 THE USER BUFFER AREA FOR THIS BUILD IS
      DECIMAL = 28453          HEXADECIMAL = $6F25

 ************************************************
```

Figure G-1. Autolink Input Directives File (Partial) (Contd)

APPLICATION NAME	PAGED LENGTH DEC	PAGED LENGTH HEX	MAIN LENGTH DEC	MAIN LENGTH HEX
SYMODU	7729	$1E31	2229	$08B5
ASYNCE	8155	$1FDB	2257	$08D1
MODE4	7477	$1D35	851	$0353
HASPTI	6402	$1902	209	$00D1
HIP	1443	$05A3	0	$0000
HLIP	4784	$12B0	306	$0132
OLDSYS	0	$0000	50	$0032
IVT	0	$0000	410	$019A
CONSOL	0	$0000	3991	$0F97
BASESY	0	$0000	14476	$388C
PIBUF2	0	$0000	0	$0000
TUP	0	$0000	3605	$0E15
INITIA	0	$0000	2415	$096F
PIDTNL	0	$0000	129	$0081

LINKGEN COMPLETED

Figure G-2. Applications Used in the Build from Directives File (figure G-1)

```
*****WARNING*****  NO MOD DIRECTIVE FOR THESE OBJECT PGMS:
        PTDUMP
        PTIRES
        PTIBAC
        PTISTU
        RANDOM
        PTIVT1
        PTIEJE
        PTBPAT
        PTBCOD
        PTBVT1
        PTECHO
        PTUPLI
        PBDEBU
```

Figure G-3. Informative Message, Missing MOD Directives

```
DELETED TOPHAT MODULE LIST

        PNDSTA
        PBHALT
        PTIVTC
```

Figure G-4. Deleted Tophat List and Buffer Area Message

LINK UTILITY EXAMPLES H

This appendix gives the following examples:

- Two examples of calling MPLINK independently of the CCP or CCI build procedures (figures H-1 and H-2). Note that in the ordinary case, this information is not necessary since the build procedures automatically supply all the calling procedures.
- An MPLINK directives file (figure H-3). This file was produced for CCP by Autolink using the Autolink directives files shown in appendix G.

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(0,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 H-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 H-2. MPLINK Execution Example 2

CYBER MINI CROSS SYSTEM - LINK EDITOR -

LINK DIRECTIVES

```
1     *SYSID,R96,CCP VARIANT LOAD MODULE.
2     *ENT,DSTKFW,$0040.
3     *ENT,DSTKLW,$00FC.
4     *ENT,DVARFW,$0000.
5     *ENT,DVARLW,$0000.
6     *DSTK,$0040,$00FC.
7     *SYN,VI4AASC,VB4AASC.
8     *SYN,VI4ABTA,VB4ABTA.
9     *SYN,PNCONTML,PBILL.
10    *SYN,PNTSMTR,PBILL.
11    *SYN,PN8LDX25,PBILL.
12    *SYN,PNDLX25,PBILL.
13    *SYN,PN5SRCH,PBILL.
14    *SYN,PNTSMON,PBILL.
15    *SYN,PTHIPI,PBILL.
16    *SYN,PTHIPO,PBILL.
17    *SYN,PBDEBUG,PBILL.
18    *SYN,UTOPIA,PBILL.
19    *SYN,PBBREAKPOI,PBILL.
20    *SYN,PBWRAPSNAP,PBILL.
21    *SYN,PB2SNAP,PBILL.
22    *SYN,PB1SNAP,PBILL.
23    *SYN,PBOPSHLT,PBILL.
24    *SYN,PBTUPBREAK,PBILL.
25    *SYN,PBTUP,PBILL.
26    *SYN,PBDECODE,PBILL.
27    *SYN,PBPERFORM,PBILL.
28    *SYN,PBTIPDBG,PBILL.
29    *SYN,PBOMPREG,PBILL.
30    *SYN,PBTUPDUMP,PBILL.
31    *SYN,PBREAD,PBILL.
32    *SYN,PBWRIT,PBILL.
33    *SYN,PBTEST,PBILL.
34    *SYN,PBQUICK,PBILL.
35    *SYN,PBTMED,PBILL.
36    *SYN,PBIOSE,PBILL.
37    *SYN,PBORCD,PBILL.
38    *SYN,PBTTYS,PBILL.
39    *SYN,PBIOTM,PBILL.
40    *SYN,PBQCIN,PBILL.
41    *SYN,PBDISP,PBILL.
42    *SYN,PBTTYINT,PBILL.
43    *SYN,PBGETC,PBILL.
44    *SYN,PBSTARTIO,PBILL.
45    *SYN,PBSUPMSG,PBILL.
46    *SYN,PBDEMT,PBILL.
47    *SYN,PBCONSOLE,PBILL.
48    *SYN,PBIEMT,PBILL.
49    *COR,$17FFF.
50    *SYN,PNDSTA,PGDSTA.
51    *SYN,PBSWIT,PGSWIT.
52    *L,ZEROX ,$00000.
53    *L,PBINTR-ADDRES,$00100.
54    *L,GLOBL$,$00DA0.
55    *L,ISPOLD.
56    *L,PBCALL.
57    *L,PBLN00-PBLN01.
58    *L,PBAMAS.
59    *L,PNSMWL,$02000:$04.
60    *L,PNAWAI-PNSMBA.
61    *L,PNRVRS-PNSMTR.
62    *L,PNSMDI-PNLLCN.
63    *L,PNLNCN-PNLNST.
64    *L,PNITML-PNOVLO.
65    *L,PNERCE.
66    *L,PNOVLT.
67    *L,PNOVLD,$04000.
68    *L,PNPSTA.
69    *L,PNLNBA.
70    *L,PNSGAT-PNCECN.
71    *L,PNSTOR.
72    *L,PTAFOU.
73    *L,PTARET.
74    *L,R4ASYI-ASYMSG.
75    *L,ASYLEM.
76    *L,PTAPSP.
77    *L,AASCST.
78    *L,AAEBCD.
79    *L,AACAPL.
80    *L,ATAPLA-AASBAP.
81    *L,ASTDAS.
82    *L,AEBCDA.
83    *L,AEAPLA-ACAPLA.
84    *L,A7T06P.
85    *L,PBLN06.
86    *L,PTCTCH-PBQ1BL.
87    *L,PTINIT.
88    *L,PTIVTC.
89    *L,PBSLJ .
90    *L,PGDSTA.
91    *L,PBMLIA.
92    *L,PGSWIT.
93    *L,PBLN02-PBLN03.
94    *L,PBLN08.
95    *L,PBFILE.
96    *L,PBLMAS-PBSTOP.
97    *L,PTTPIN.
98    *L,MODMST.
99    *L,PBLN04-PBLN15.
100   *L,PBMEMB-PNDEQU.
101   *L,PBEMAH-PBTOAD.
102   *L,PMMLEH.
103   *L,PTMSCA.
104   *L,PBCLKI-PNBMPS.
105   *L,PMCOIN-PMCDRV.
106   *L,PBSCLA.
107   *L,PMWOLP-PMTISE.
108   *L,PTLMUX.
109   *L,PBSWLE-PBINTP.
110   *L,PBUPAB-PTBACK.
111   *L,PTBREA.
112   *L,PTSTRT-PTSTOP.
113   *L,PTICMD.
114   *L,PBIOPO-PNLLTC.
115   *L,PBEMAD.
116   *L,QDEBUG-PBCLR .
117   *L,PBLOAD-PBILL .
118   *L,PBHALT.
119   *L,PBMON -TOSTOP.
120   *L,R4M4IN-R4M4CC.
121   *L,LIPSMA.
122   *L,PNTNKS.
123   *L,HASPMS.
124   *L,HSR4IP.
125   *L,PIDTBL.
126   *L,MAIN$ ,$0F659.
127   *L,BEGINX.
128   *L,PIPROT-PISIZC.
129   *L,PILMT .
130   *L,PINWIN.
131   *L,PILCBS-PIGETA.
132   *L,PIFR1 .
133   *L,PIBUF1.
134   *L,PINIT .
```

Figure H-3. Sample MPLINK Directives File for a CCP Run

```
135     *L,PIWLIN-PIAPPS.
136     *L,PILINI-PTTMRS.
137     *L,PIMLIA.
138     *L,PIBUF2.
139     *L,PTASNO,$10000:$04.
140     *L,R4ASYT-R4A274.
141     *L,ASYERR.
142     *L,PTMSQU.
143     *L,PTDELM-PTASNM.
144     *L,PTAREC-PTAFAL.
145     *L,PTAPBU-PTAFSC.
146     *L,PTAPI .
147     *L,PTMD4T,$12000:$04.
148     *L,R4M4TP.
149     *L,PTSTAC-PT4TEX.
150     *L,PT4TCB.
151     *L,PNSMGE.
152     *L,PTHSOP,$14000:$04.
153     *L,HSR4IT-FCSRCB.
154     *L,PTHSMU.
155     *L,HSPTCB-PTTPHA.
156     *L,PNCONF.
157     *L,PGIVTC.
158     *L,PLTKOP,$16000:$04.
159     *L,PLIPTC.
160     *L,IC    .
161     *L,PLCBIN.
162     *L,PLREAD.
163     *L,PLIPML.
164     *L,CLEANU-PLIP  .
165     *L,PTIVTP.
166     *L,PTLINI.
167     *L,PGHALT.
168     *END.
```

Figure H-3. Sample MPLINK Directives File for a CCP Run (Contd)

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 I-1.
- A partial listing of the memory image load module file (Figure I-2).

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

```
*
*****************************************
*                                       *
*    COPYRIGHT CONTROL DATA CORP. 1975,  *
*    1976, 1977, 1978, 1979, 1980       *
*                                       *
*****************************************
```

DEFINITION/DECLARATION SECTION

CONSTANT DECLARATION PART

```
*
*****************************************
*                                       *
*        CONSTANTS                      *
*                                       *
*****************************************
CONST
      /TRUE   =  1;
      /FALSE  =  0;
      /NAME   = 'E2';                  * GENERATE MPPPU FILE

*
***********************************
*                                 *
*        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
      /LPIO;
      /TCBCONS ;
      /TCBMLIA ;
       /BOS32;
      /IDTBL;           * NETWORK DEFINITION TABLE WORK POINTER
*************************************
*                                   *
*  THE FOLLOWING IS THE SOURCE INPUT *
*  FOR APPLICATION UNIQUE VARIABLES  *
*  DEFINITION FOR THIS SYSTEM        *
*                                   *
*************************************
```

Figure I-1. Partial MPEDIT Program

```
      BUFLCOC(BOSO../BOS3] OF 1;
      BUFMASKS[/BOSO../BOS3] OF 1;
      BECTLBK[/BOSO../BOS3] OF /SIZBFCTLBK;
      NICTCT [ NOBOO..NODIAG ] OF 1;
      NJICT [ NOTMLIA..NOTDIAG ] OF 15;
      BFTHRESH [BOTI..BOTHMUX] OF 1;        * BUFFER THRESHOLDS        *
              ●
              ●
              ●
      AVTC4LF [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
*
      AVUCCODFT[/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
*
      HSTCBAT[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 := $000F;
      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;
      BEGINV + $0 := MAIN$;                 * SYSTEM INIT              *
*************************************
*     SET UP POST MORTEM DELIMITERS *
*************************************
*
      CENTLIM := $20;
      CCENTLIM:= $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 ].BYWLREQ   := /TRUE;
```

ASSIGNMENT SECTION

PSEUDO VARIABLE DECLARATION PART (PART OF ASSIGNMENT SECTION)

Figure I-1. Partial MPEDIT Program (Contd)

```
*+
      *************************************
      *  SERVICE MODULE LOCAL VARIABLES   *
      *************************************+
  /BZOWNER;
  /BZLNSPD;
  /BZN2;
  /BZ1PKTLNTH;
  /BZ2PKTLNTH;
  /BZK;
  /BZ1LPVC;
  /BZ2LPVC;
  /BZDCE;
  /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;
      /COSMBESZE;
*************************************************************************+
**END APPLICATION UNIQUE VARIABLE SOURCE*+
*************************************************************************+
*                                                                        ARRARY DECLARATION
*****************************************
*                                       *
*     ARRAY DECLARATIONS - BASE AND CCP *
*     TABLES                            *
*                                       *
*****************************************
+
ARRAY
      BJTIPTYPT [NOHDLC..N1USR1] OF /SIZTIPTYPT;
      BWWLENTRY [1..17] OF /J1WLMAX;
      CBTIMTBLE [OLCBTMSCN..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;
      J1TUPBERS [1..64] OF 1;
      NBLTYT [NOLDIAG..NOLAST,1..NKRC3OUT] OF 2;
      NBLTP  [NOLDIAG..NOLAST] OF 1;
      BUFLENGTH[/BOSO../BOS3] OF 1;
      TCBLENGTH[/BOTSO../BOTS7] OF 1;
```

Figure I-1. Partial MPEDIT Program (Contd)

```
%
%              * *  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                                          FOR..TO LOOP
        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 ] :=      $D;        JKTMASK[4 ] :=      $D;
      JKMASK[5 ] :=     $1D;        JKTMASK[5 ] :=     $1D;
      JKMASK[6 ] :=     $3D;        JKTMASK[6 ] :=     $3D;
      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] :=  $FFFF;
                ●
                ●
                ●
%*************************************************************************  NESTED FOR..TO LOOPS
*                  INITIALIZATION OF LINE TYPE TABLES                   *
*************************************************************************
;
%;
   FOR /I:=NOLDIAG TO NOLAST DO
   FOR /II := 3 TO NKRC3OUT DO                % PRIME ALL SECOND WDS     ;
   NBLTYT [/I,/II].NBINT2 :=$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)        ;
                ●
                ●
                ●
```

Figure I-1. Partial MPEDIT Program (Contd)

```
*************************************************************************
          ** 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 := $0208;
/L := /L + 1;  /IDTBL + /L := $030C;
/L := /L + 1;  /IDTBL + /L := $0400;
/L := /L + 1;  /IDTBL + /L := /ENDE;
/L := /L + 1;  /IDTBL + /L := $0008;
/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 := $0000;
/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           *
      *                                                      *
      ********************************************************

      DDTCBFDT := DGTCBFDT;

      DGTCBFDT[0].DDNUMENT := 50;           *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 I-1. Partial MPEDIT Program (Contd)

```
/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 I-1. Partial MPEDIT Program (Contd)

```
                                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  205A   4552  4F20:]
ZEROX
  0000    [:1400  F671
  0010
  0020
  0030                                                                                              :]

HEADER
  0000    [:000F  401E  F671  4745   4749  4F53  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  005C   0402  CC00        0403   C003  0040  0404  C003         1400  F659:]

HEADER
  0000    [:0040  4000  0100  5042   494E  5452  4E54  4552   5255  5054  2054  5241   5053  2054  4142  4C45
  0010      2025  4C4F  4144  2041   5420  2431  3030  2920   2020  2020  2020  2049   4E54  5241:]
PRINTR
  0100    [:        1400  1F98               1400  1FC4               1400  52C3               1400  52E7
  0110              1400  56A3               5400  4CFC  0E14         5400  4CFC  0E18         1400  56AF
  0120              1400  5306               1400  568B               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  45F3  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         1A60        0032                0096  000F              :]
```

Figure I-2. Sample Memory Image Load Module Hexadecimal

INDEX

COMMENT SHEET

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

PUBLICATION NO.: 60471200 **REVISION:** F

NAME:_____

COMPANY:_____

STREET ADDRESS:_____

CITY:_____ **STATE:**_____ **ZIP CODE:**_____

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 below (please include page number references).

CUT ALONG LINE

AA3419 REV. 4/79 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE

TAPE

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
P. O. Box 4380-P
Anaheim, California 92803

CUT ALONG LINE

FOLD

FOLD

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A

CONTROL DATA CORPORATION