PART 3 LINKAGE EDITOR

A program may consist of a (large) number of modules. After having been assembled or compiled, these modules must be processed by the Linkage Editor to match existing external references in the modules with entry points in these modules, or in modules in a user or system library (e.g. Fortran system modules), to create an executable program.

If the program is so large that it occupies most of the memory available during execution, it is advisable to restructure the program and divide it into segments with the overlay technique. This is necessary if the program is too big to fit in the available memory.

The Linkage Editor can handle both segmented and non-segmented programs.

The overlay technique makes it possible to load into memory only those parts of the program which are required at a certain moment. They will be overlaid when their presence is no longer necessary.

Compared to a non-segmented program, a segmented program requires some extra words which are added to the load module. At execution time, however, a considerable memory space is gained.

The size of the Linkage Editor is approximately 5 pages of 2K words for the longest path of the Linkage Editor, plus the remaining free memory (up to 32K words) for tables.

Programs may consist of more than one object module for one or more of the following reasons:

- Modular programming techniques have been used.
- Part of the program has been written in Fortran and part of it in Assembly Language.
- An overlay program has been developed.

In all the above cases each source module has been assembled or compiled separately. Linking such a multi-module program involves:

- Resolving inter-module references. EXTRN symbols in a module are replaced by the addresses of ENTRY symbols in other modules.
- Resolving any remaining unresolved references by scanning either the system or user library, or both, to find the modules having ENTRY symbols corresponding to the unresolved EXTRN symbols. The modules selected by this scan are automatically included by the Linkage Editor in the load module, and each EXTRN symbol which caused the scan is replaced by the address of the ENTRY symbol in the module selected by the scan.
- Positioning COMN areas in the load module. Each labelled common is placed at the end of the segment containing the first module referring to it. All blank commons are treated as the same area, and the Linkage Editor places the largest one encountered at the end of the load module.

  A blank common block can be implicitly allocated behind the longest path of the overlay structure, or explicitly at an absolute address.
- Segmenting an overlay program according to user commands, using either disc-resident or memory-resident overlay segments. The Linkage Editor may also link several small modules into one larger one, which will be re-input at a later date.

This chapter contains a general description of the overlay technique and gives some definitions of terms.

The overlay technique is a programming technique which allows to reduce the memory space needed for program execution.

The program's object modules must be linked and organised in such a way that modules are loaded only when their presence is required. To this end the program is divided into segments.

To obtain this goal, a overlay tree structure can be designed which is the graphical representation of the program's organisation.

Segment

A segment is a part of the program and consists of one or more modules. Each segment is separated from another segment on the LKE input file by a NOD record.

A program may consist of up to 128 segments. The order in which they are placed on the Linkage Editor input file is determined by the user, and depends on the program and the references the segments may make among each other.

Disc-Resident Overlay

A disc-resident overlay is a segment kept on the disc as part of the generated load module. It is loaded into the program area when required, and is in turn overwritten by the next overlay segment when no longer needed.

Memory-Resident Overlay

A memory-resident overlay (ROV) is a segment which is loaded at machine definition time and then remains in memory. The MMU Page Table is manipulated by the MAS system in order to bring the segment into the program-visible area, or to remove it when no longer required. Each ROV segment of a program forms one secondary load module.

If a ROV segment is re-entrant, it may be used by more than one program simultaneously.

Root

An overlay structure has as its basis a root, which is that part of the program that will always be in memory as it exercises the control of the program. It must be the first segment on the LKE input file.

Path

The branches of the tree, called paths, constitute, together with the root, the way along which the program is executed. Each path of the tree consists of one or more segments. A path is terminated when, while scanning the input file:
- the end of the object input file occurs, or
- a NOD command is read which has the same name as a preceding NOD record.

No path may comprise segments whose cumulative size exceeds 32K words.

Node

Each segment on the input file, except for the root, commences with a node, i.e. a NOD record. The NOD record has the following format:

NOD <name>[,{<ROV segment name> | *}[,<absolute address>]]

where <name> may consist of up to 6 ASCII characters of which the first one must be a letter, specifying the name given to this node. Different segments may use the same node-name.

<ROV segment name>, if present, specifies explicitly the name of the ROV segment being started.

* (if present) tells the Linkage Editor to allocate an implicit name to the ROV segment.

<absolute address>, if present, specifies the absolute address at which the ROV segment will be loaded. Doing this makes it possible for more than one program to share the segment.

If only the <name> parameter is present, the node defines the start point of a disc-resident overlay.

Level

The level of a segment in a path is the number of nodes between that segment and the root in a path.

Ascendant

In a path, segments with a lower level are called ascendants.

Descendant

In a path, segments with a higher level are called descendants.

Exclusive

A segment located in another path is called exclusive.

Example

A program consists of 10 modules which we will label A through J. Of these modules, A and B form the root (segment 0).

In the program we can distinguish 6 paths and 9 segments (root included). Segments 0, 3 and 8 are built of more than one module. The paths are:

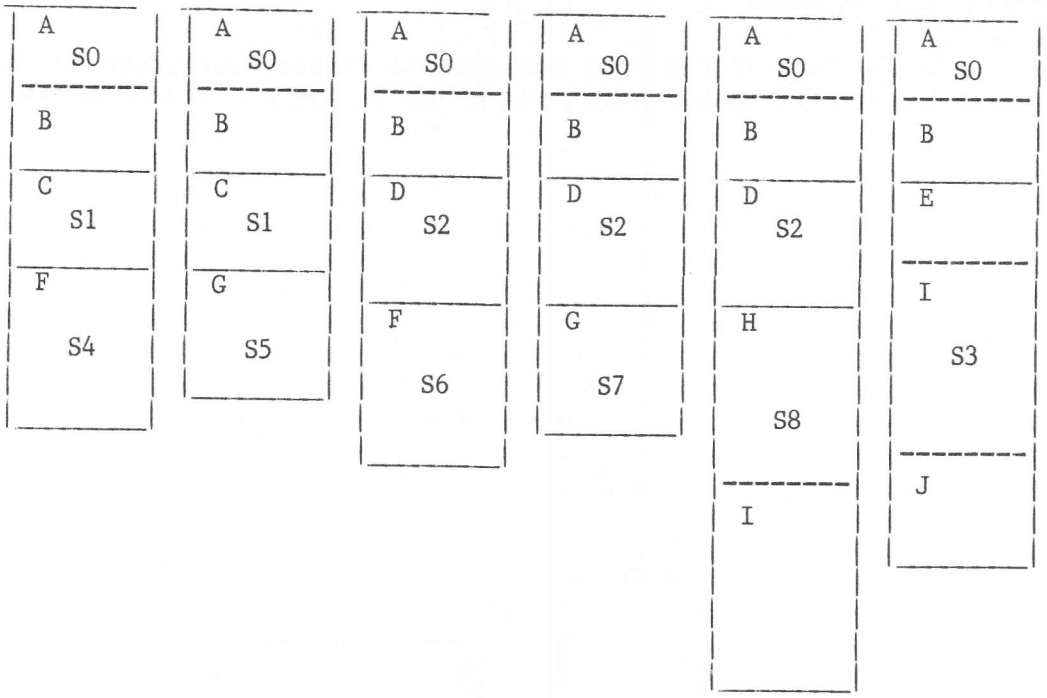


Fig. 3-1. Example of Paths

From these paths the following overlay tree may be built:

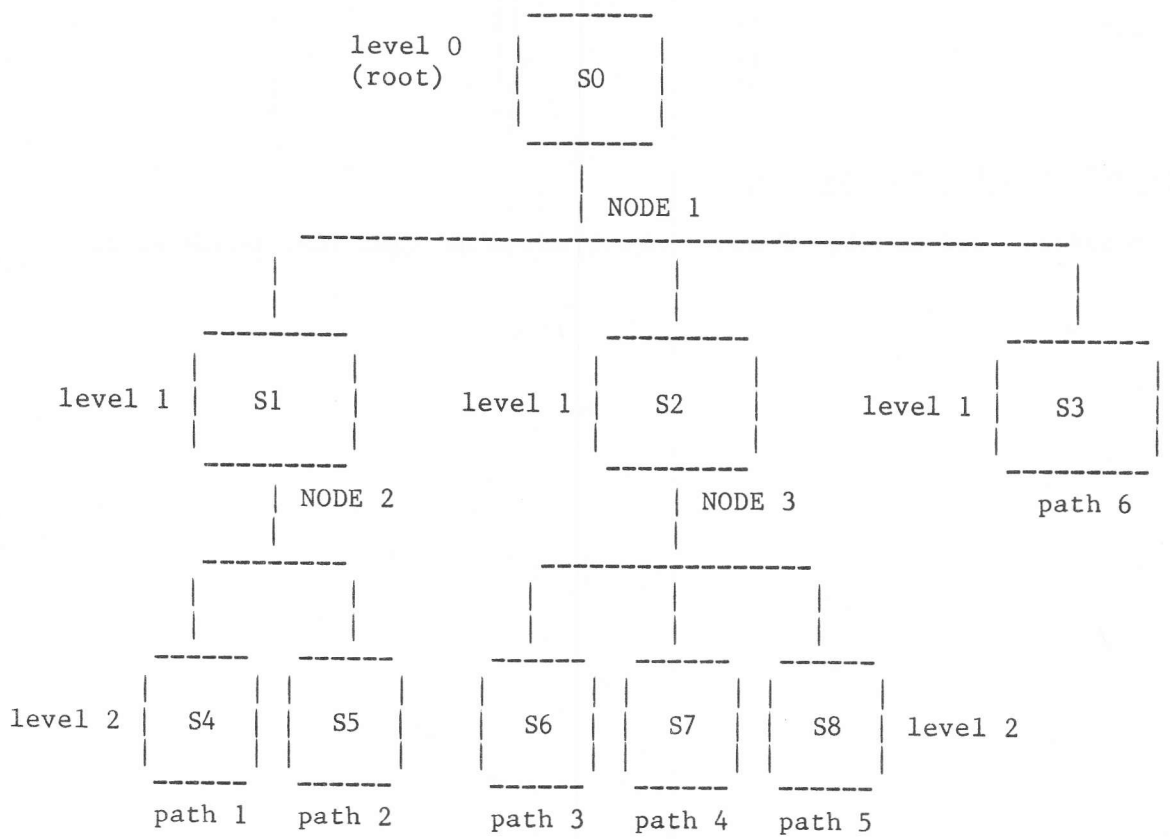


Fig. 3-2. Example of Overlay Tree

In this example, path 1 will be executed first, and then path 2. Of the latter, Segments 0 and 1 are already in memory; Segment 5 overlays Segment 4. In path 3 Segments 2 and 6 will overlay Segments 1 and 5, and so on.

Example Showing ROV-Segments

The commands down the left of the page produce the overlay structure on the right, which is built up from the same modules as in the previous example:

```
INC A
INC B
NOD N1
INC C
NOD N2,F
INC F
NOD N2,G
INC G
NOD N1
INC D
NOD N3,F
INC F
NOD N3,G
INC G
NOD N3
INC H
INC I
NOD N1
INC E
INC I
INC J
LKE
OPT
```

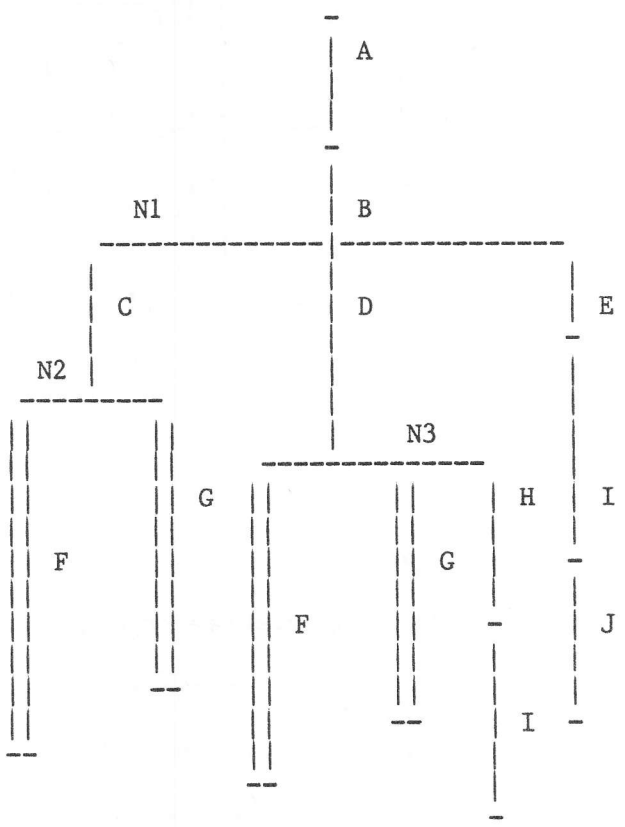


Fig. 3.3 Example with ROV Segments

The secondary load modules F and G are treated as identical wherever they occur.

Though no specific programming requirements are necessary to have a successful linking and production of a segmented program, the following rules should be obeyed:

- Segmented programs are not re-entrant, as segments will be overlaid during a run. Only the root and blank common are never overlaid.
- Avoid references to exclusive segments.
- Avoid excessive loading, overlaying and reloading of segments; otherwise the program will run very slowly.
- Use common areas as often as possible for inter-module communication.
- Overlay programs to be run in a foreground machine must not be declared by an FCL RON command. In other words, they may be middle-ground, memory-resident (declared by FCL LOD or FCL REP) or swappable (declared by FCL SWP).
- It is not possible to return to the interrupted program part, by means of an RTN instruction or RETURN statement, from an exclusive segment if that segment has been called by a CF instruction or CALL statement. The reason is that the stack may not contain the right information for a proper return, as the segment is overlaid by the exclusive segment.

  The segment causing the overlay (by a CF instruction) is not reloaded at the return from the exclusive segment.
- A block data subprogram must be in the segment with the highest level using the common.

The input file for the Linkage Editor can be created using the standard processors ASM or FRT, and with the BCL INC and NOD commands. The first time the object file is used in a Job it is created in the area for the :JOB DAD and Userid, and is assigned to filecode /D5. Subsequent object modules will be added to the end of the file by BCL INC commands, or ASM or FRT processor calls.

The object file is removed by the BCP from the background machine filecode table when the Job ends. This will also free the granules on the DAD.

Since the object modules on the file may make references to other modules in the user or system library or to both, these libraries have to be scanned by the processor to look for missing references. To facilitate this scanning, an object library directory is created and kept up to date each time a module is kept with a KOM command or deleted by a DOB command. The directory is placed in the object module library by the system.

SEGMENTED PROGRAM

To produce a segmented load module, the program modules must be placed on the object file according to the overlay design. The first segment on the file must be the root. Next, the following segments must be loaded, as follows:

- When several (exclusive) segments have the same immediate ascendant, their common beginning location is called a node. To define the node on the object file a NOD command must be given, specifying the name of the node. The name is recorded as an ASCII record occupying one sector in the object file and is used by the Linkage Editor, but it is not included in the load module.
- When several modules form one segment, as many ASM, INC or FRT commands may be given as necessary, up to the next NOD command.

In this way up to 128 segments may be specified for one run. At the end of this Chapter, examples are given of how to proceed.

When all segments are on the temporary object file, the Linkage Editor must be called with the BCL command input from the device assigned to /E0 or /EE:

LKE [DUMP={ALL | PROG | NO}][,SIZE={MAX | <n>}]

The parameter DUMP indicates whether a dump must be made after an abort or an exit (LKM 3).

ALL Dump the Monitor and the background machine in case of an abort, or if bit 8 is set in register A7 for LKM 3.
PROG Dump only the background machine.
NO No dump required. This is the Default.

The parameter SIZE reserves a work area of n pages of 2K words for the Linkage Editor. It is only useful when the parameter SIZE is not specified in the 'Declare Batch Processing Machine' command BCP.

MAX The system will reserve 32K words (16 pages) of work area for the Linkage Editor.
n A number, ranging from 0 through 16, specifying the number of additional pages required as work area. Default = 0 pages.

If the LKE command is accepted and /E0 is assigned to an interactive device such as console keyboard or display, the message:

LKE:

is output to that device. The user may now input the OPT command described in this chapter.

If the LKE command is rejected, then:

- If /E0 is assigned to an interactive device, a message explaining the error is output on this device, followed by the message:
  LKE:

  requesting the user to enter the correct LKE command.

- If /E0 is assigned to a non-interactive device (such as a card reader), an error message is output on the ERR device, followed by the message:
  LKE:

  requesting the user to input the correct command from the device assigned to ERR.
- If /E0 is assigned to a non-interactive device and there is no ERR device defined for this Job, the error message is output to the device assigned to /02 and the Linkage Editor exits. The BCP is reloaded and reads all subsequent commands on /E0 until one of the commands :EOJ, :EOB or :STP is encountered.

Error Messages:

PARAM. NOT VALID	The parameter is 1) erroneous 2) <n> greater than 16
PROCESSOR NOT CATALOGUED	
DAD ASSIGN ERROR	
I/O ERROR	
SEARCH DIRECT. NOT POSSIBLE	

LKE OPT STATEMENT

The OPT control statement must be given immediately after having called the Linkage Editor with the LKE command, and must be present on the device assigned to /E0 or /EE.

```
OPT [STAD=<name>][,CBLK=<addr>][,CREF={YES | NO}][,MAP={YES | NO}]
    [,SLIB={YES | NO | <name>}][,ULIB={YES | NO | <name>}][,CATL=<name>]
    [,DBUG={ENTR | STAB | NONE}][,GENE={LM | OB}][,KEEP=(<ident list>)]
    [,FRGT=(<ident list>)][,ONAM=<name>][,DLST=<name>][,INTC=(<ident list>)]
    [,ROVP=<name>][,CROV=(<ident list>)]
```

STAD= <name> is the start address of the load module, which must be an entry point in the root. Default = last start address encountered in the root.

This option must be supplied if GENE=OB and neither KEEP nor FRGT is specified.

CBLK= <addr> is the absolute address of a blank common. The blank common is loaded at the specified address. Only applies if GENE=LM. Default = last region address. (See also Map and Symbol Table.)

CREF= YES: a cross reference listing is printed.
NO: no cross reference listing is printed. (Default.)

MAP= YES: a MAP listing will be output.
NO: no MAP listing will be output. (Default.)

SLIB= YES: The system library is scanned to resolve external references. Only applies if GENE=LM.
NO: the system library does not need to be scanned. (Default.)
<name>: specifies the name of the system library to be scanned.

ULIB= YES: the user library "USRLIB" must be scanned to resolve external references. Only applies if GENE=LM.
NO: no user library needs to be scanned. (Default.)
<name>: specifies the name of the user library to be scanned.

CATL= <name>: the name under which the load module is catalogued. <name> must consist of 1 through 6 alphanumeric characters. Only applies if GENE=LM. Default = the file is not catalogued.

DBUG= This debugging parameter is only useful when a load module must be created which will be debugged. In a non-segmented program, all entry points and commons are placed in the table which will be added to the module. In a segmented program, only the entry points of the root are saved. The option only applies if GENE=LM. Default = NONE (no table).

ENTR: a symbol table containing only entry point names is generated.
STAB: a symbol table, containing all symbols originally kept by the Assembler, is generated.

GENE= specifies whether a load module (LM) or an object module (OB) is to be generated. Default = LM.

KEEP= specifies a list of entry names which have been resolved, but are to be kept by the Linkage Editor. Only applies if GENE = OB; may not be used if the FRGT option is present.
Default = only the start address is kept - see the STAD option.

FRGT= specifies a list of entry names which have been resolved, and may be forgotten by the Linkage Editor. Entry names not in the list will be kept. Only applies if GENE = OB; may not be used if the KEEP option is present.
Default = only the start address is kept - see the STAD option.

ONAM= specifies the name of the generated object module, up to six alphanumeric characters. Only applies if GENE = OB. Default = 'NONAME'.

DLST= specifies the name of the generated internal symbol table, containing the names of resolved entry points which are no longer known externally. Only applies if GENE = OB. The table need only be kept if the final Link Edit for this generated object module will use the DBUG option. Default = no table is kept.

INTC= specifies the names of labelled common blocks which will be made internal to the generated object module. Only applies if GENE = OB.

ROVP= specifies a prefix of one or two letters for implicit names of ROV segments. For example, if the prefix is 'XX', the Linkage Editor assigns 'XX00' to the first ROV segment, 'XX01' to the second, and so on; ROV-segments whose names are given explicitly in NOD statements are not counted. This option only applies if GENE = LM.

CROV= specifies a list of ROV-segment names; each ROV segment is catalogued under the current <userid> with the specified name, which then becomes a secondary load module name. This option only applies if GENE = LM.

Error Messages

When the OPT control statement is processed, one of the following error messages may be printed on the error message listing device:

```
I/O ERROR <filecode> <status>
KEYWORD OCCURRENCE ERROR
INCORRECT /D6 ASSIGNMENT        1) no source file could be assigned;
                                2) disc overflow
OPTION STATEMENT MISSING
INVALID KEYWORD
TWICE THE SAME KEYWORD
= NOT FOLLOWING THE KEYWORD
, NOT FOLLOWING THE PARAMETER
INVALID BLANK COMMON BASE
INVALID START ADDRESS NAME
INVALID USER LIBRARY NAME
INVALID SYSTEM LIBRARY NAME
INVALID USER LIBRARY ASSIGNMENT
INVALID SYSTEM LIBRARY ASSIGNMENT
INVALID DBUG KEYWORD VALUE
INCORRECT /D5 ASSIGNMENT        no object file could be assigned.
PARAMETER VALUE MISSING
INVALID INTEGER
/D5 NOT AN OBJECT FILE
YES OR NO NOT FOUND
NO OBJECT CODE FILE
ERROR WHEN WRITING AN EOF ON /D5
INVALID OPTION STATEMENT
INVALID PARAMETER FOR CATL OPTION
```

See also Fatal and Non-Fatal Errors.

If an erroneous OPT control statement is given, the user may input the correct one on the device assigned by ERR or on /E0.

PROCESSING

The processor starts reading the whole input file, storing relevant information in tables. All external references encountered during reading are placed in a symbol table, at the same time indicating whether the reference is absolute or relative. The external references may consist of entry points and labelled commons.

The Linkage Editor now tries to match the references according to the overlay structure of the program, obeying the following rules:

- References are first looked for in the segment. If they cannot be found in the segment, the ascendants are searched and next the descendants. If a double definition was given in an ascendant, the first one encountered is taken and a non-fatal error message is printed. If a reference is made to one or more descending segments, the reference is defined the first time it is encountered. The external reference in that case is not replaced by the entry point's address, but by the address of a link block which points to the segment loader.

- Depending on whether the user has specified the relevant parameters or uses the default, the processor starts looking for the missing references in the user library, the system library, or in both (user library first).

  Such a library has a directory containing all relevant information as given in clusters 2, 5, 6 and 7 concerning the entry points, externals or commons, in each module in the library file.

  If the entry point is found in a library, the module which contains the entry point is included in the program. If the module in which the entry point appears is referred to by more than one segment, the module is included in the segment with the lowest level (i.e. nearest the root).

- If the external reference is not yet found after scanning one or both libraries, the Linkage Editor will look for it in the exclusives. As the referencing to exclusives may cause stack problems, a warning message is output.

  When the reference is found in an exclusive segment, the external is not replaced by the address of the entry point but by the address of a link block. Should the entry point be present in more than one exclusive segment, the first time the entry point is encountered is taken as the definition. If the external reference is not resolved at all, an error message is output.

Processing of Commons

Commons may be labelled or blank. The Linkage Editor processes them in different ways:

Labelled Common

Labelled commons have a fixed length. They are allocated by the processor at the end of the segment in which they are referenced. Consequently, they can be overlaid during a program run, but the initial values, given by a block data subprogram, are reloaded each time the segment is loaded.

When a reference is made to a labelled common whose label is used in several segments, the common is allocated to the segment with the lowest level.

Blank Common

The largest blank common encountered in the program is placed at the end of the program, and is never overlaid. The user must, however, take care not to destroy this area when he is using a Get Buffer request. The beginning address of this buffer must point to a location after the last address of the blank common. See also the example in Map and Symbol Table.

When the user has given an absolute address to a blank common, the blank common is located at the address specified.

LOAD MODULE

At the end of the processing a load module, segmented or not, is built. When the load module is generated, the object code is taken from the modules on the input file and relocatable words and external references are replaced by their real addresses.

When a segmented program must be generated, the Linkage Editor adds, each time a segment is created, a segment load block to the root. This block contains information on where the segment is to be loaded, its length and the sector of the disc where the segment can be found.

The last segment load block is followed by a segment loader, which at program execution time controls the loading of segments not already in memory.

NON-SEGMENTED PROGRAM

If the user wishes to produce a non-segmented program, the Linkage Editor operates as if all modules placed on /D5 by ASM, INC and/or FRT are a segment of level 0. In fact, it should be considered to have one path of one segment. No NOD commands are used, and no segment loader or link block is added to the load module.

OBJECT MODULE OUTPUT

The user may wish to combine several smaller object modules into one larger object module, which will be re-input to the Linkage Editor later on. The Linkage Editor combines all the object modules placed on the /D5 input file by ASM, FRT or INC commands. The NOD statement is meaningless and should not be present.

The user may specify which ENTRY symbols, already resolved by EXTRNs, should be kept in the entry point table for the generated object module, and which may be forgotten.

The output of the Linkage Editor consists of:

- a load module
- a MAP (optional)
- a Symbol Table (optional)
- error messages.

LOAD MODULE

The load module is an executable program in object format. The module is output on the load module file /D6. Each sector of the file contains 188 code words and a 12-word relocation table (RTB), of which:

- bit 0 of word 0 = 1 if the first word is relocatable, or
  = 0 if the first word is absolute.
- bit 1 of word 0 is associated with the second code word
  :
- bit 0 of word 1 is associated with the 17th code word
  :
  etc.

The first six code words of the load module, which are stored in the first locations of the program, have the following meaning(s):

- Word 0
  for non-segmented programs:
  Program start address.
  for segemented programs:
  Program start address increased by 1, pointing to a location in the root.

- Word 2
  for non-segmented programs:
  Number of sectors on the disc occupied by the load module.

  for segmented programs:
  Number of sectors on the disc occupied by the root.

- Word 4
  Effective length of the load module (blank common, if not given an absolute address, included).

- Word 6
  Symbol Table address.

- Word 8
  for non-segmented programs:
  First code word of the first module of the program.
  for segmented programs:
  Length of the program area (which may be longer than the effective program length; see example in MAP and Symbol Table).

- Word A
  for segmented programs only:
  The number of segments of the program, the root not included. The following n * 4 words contain n segment load blocks. The last 4-word item is followed by the Segment Loader.

MAP AND SYMBOL TABLE

An example of a MAP is given at the end of this Chapter; it contains the following items:

START Start address of the program.
LENGTH Length, in characters, of the longest path.
REGION The length of the longest path as loaded from n sectors. Since the segments are loaded one disc sector at a time, an entire sector may therefore have been loaded without that sector being completely filled with the contents of a segment.

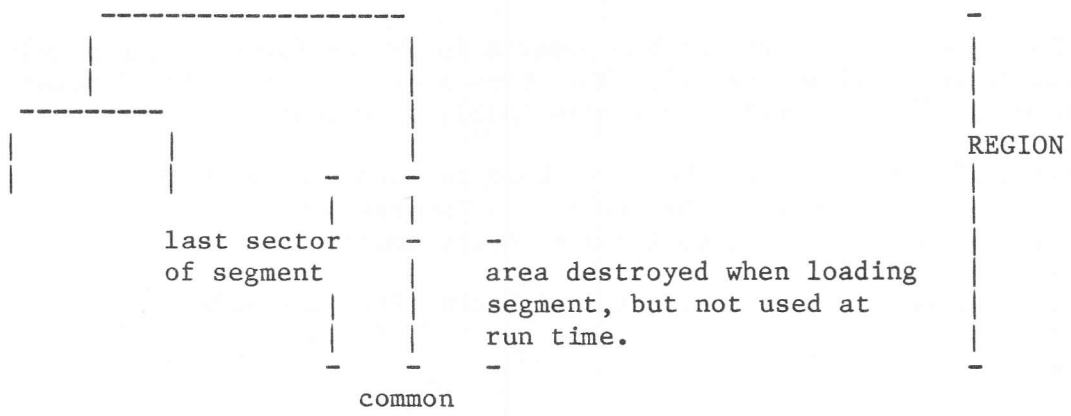
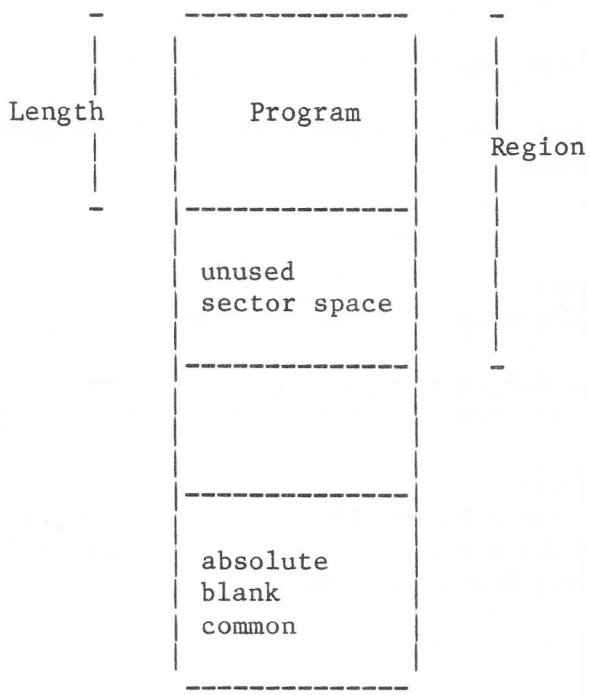


Fig. 3.4 Example of Region

Example:

The following examples explain the difference between the length and the region. The first program contains an absolute blank common and in the other example the program contains a relocatable blank common.

Absolute Blank Common



Here Length < Region, but it may happen that Length = Region if all sectors were entirely used.

Relocatable Blank Common

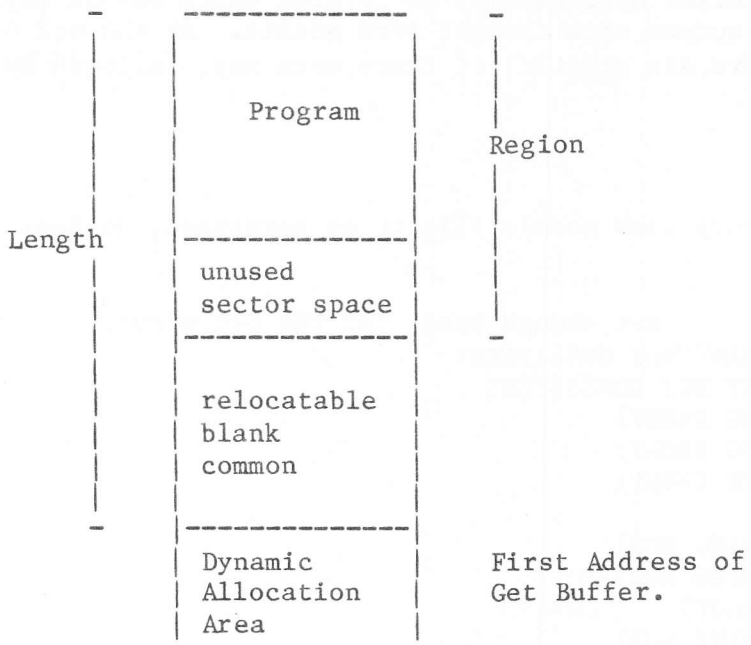


Fig. 3.5 Examples of Length and Region

Then the segments are printed in ascending level number order, where:

SEGMENT # number of the segment in the overlay tree
ADDRESS # address of the segment in memory
SECTOR # number of the sector in which the segment is written
ASCENDANT # number of the segment's immediate ascendant. For the root this is always /FF.

Then the IDENT of each module in the segment is printed, with the module address.

The list of segments is followed by the Symbol Table. All entry points and common blocks belonging to a segment are listed in alphabetical order.

The symbol table is built of the following items over 4 columns:

TYPE SEGMENT ADDRESS NAME

where:

TYPE A = absolute entry point
B = absolute address of blank common as given in the OPT statement
C = relocatable blank or labelled common
D = symbol table entry point
E = relocatable entry point.

SEGMENT number of the segment in which the symbol appears.

ADDRESS address of the symbol.

NAME entry point name. If the same name is defined in several segments, the name is printed each time it is encountered, but with a different segment number.

ERROR MESSAGES

During and after processing, error messages may be printed which may or may not influence the processing and output of a correct load module. At the end of processing the number of errors are printed, if there were any, followed by the error message(s).

Fatal Errors

Fatal errors cause the temporary load module file to be scratched; no load module is produced.

CORE OVERFLOW not enough space for the processor.
DIRECTORY AND SYSTEM LIBRARY NOT CONSISTENT
DIRECTORY AND USER LIBRARY NOT CONSISTENT
END MISSING (EOF FOLLOWING IDENT)
END MISSING (NOD FOLLOWING IDENT)
END MISSING (2 CONSECUTIVE IDENT)
FIRST ROOT RECORD IS EOF
IDENT MISSING (END FOLLOWING NOD)
IDENT MISSING (EOF FOLLOWING NOD)
IDENT MISSING (FIRST OF ROOT)
IDENT MISSING (2 CONSECUTIVE NOD)
IDENT OR NOD RECORD MISSING OR INVALID
I/O ERROR <ECB0> <ECB8>
NOD NOT ALLOWED BEFORE THE ROOT
PROGRAM LENGTH EXCEEDS 32K
2 CONSECUTIVE NOD
KEYWORD OCCURRENCE ERROR
INCORRECT /D6 ASSIGNMENT
OPTION STATEMENT MISSING
INVALID KEYWORD
TWICE THE SAME KEYWORD
= NOT FOLLOWING THE KEYWORD
, NOT FOLLOWING THE PARAMETER
INVALID BLANK COMMON BASE
INVALID START ADDRESS NAME
INVALID USER LIBRARY NAME
INVALID USER LIBRARY ASSIGNMENT
INVALID SYSTEM LIBRARY NAME
INVALID SYSTEM LIBRARY ASSIGNMENT
INVALID DBUG KEYWORD VALUE
INCORRECT /D5 ASSIGNMENT
PARAMETER VALUE MISSING
INVALID INTEGER
/D5 NOT ASSIGNED TO AN OBJECT FILE
YES OR NO NOT FOUND
NO OBJECT CODE FILE
ERROR WHEN WRITING AN EOF ON /D5
INVALID OPTION STATEMENT
INVALID PARAMETER FOR CATL OPTION
NO PREFIX SPECIFIED FOR ROV SEGMENT
ROV SEGMENT IS NOT A LEAF
INVALID ROVP KEY WORD VALUE
IMPOSSIBLE TO ASSIGN FILECODE /D1 FOR SECONDARY LOAD MODULE
TEMPORARY FILECODE /D1 FOR SECONDARY LOAD MODULE CANNOT BE DELETED

Non-Fatal Errors

These errors are, in fact, warning messages for the user. The Linkage Editor continues processing, but the produced load module may or may not be executable. The hexadecimal number of errors is printed on the listing device assigned to /02 and on the operator's console, after processing.

ABSOLUTE ADDRESS IN MODULE <name> SEGMENT <number>
ABSOLUTE START ADDRESS IN MODULE <name>
DOUBLE DEFINITION ON <name> The first one is taken.
ERROR IN MODULE <name>
EXCLUSIVE REFERENCE FROM SEGMENT <no> TO <name> IN SEGMENT <no>
NO START ADDRESS
REF. TO UNSATISFIED EXTERNAL <name> IN SEGMENT <no> AT ADDRESS <no>
    The address is relative to the beginning of the segment.
UNDEFINED START ADDRESS NAME
    The name specified in the OPT statement is not defined in the root.
<no> UNSATISFIED EXTERNAL REFERENCE
    <no> is in hexadecimal. The symbol table indicates which external references could not be matched, by printing an asterisk.
ROV ADDRESS IS NOT A PAGE BOUNDARY

SEVERITY CODES

A severity code is output on the listing device when the Linkage Editor's processing is terminated.

0 Normal exit.

/10 The load file has been produced, but some minor errors, such as unresolved external references, occurred.

/30 The load file has not been produced; an error message from the Linkage Editor has been output to filecode /02.

/40 The Linkage Editor did not start processing because of an assignment error or invalid OPT statement. An error message has been output to filecode /01.

Examples

Two examples are given of how to proceed when a non-segmented or a segmented program must be produced. Each example is followed by a MAP and by a Symbol Table. The source modules are catalogued modules in MWSC:1 of DAD SSDOC2.

```
DCB 16
FCD /E0,CR06
FCD /1,TY10
FCD /C0                    Batch Machine declaration
FCD /C1
FCD /C2
FCD /C3
FCD /2,LP07
FCD /E2,PR20
FCD /F0,/C0,SUPERV
FCD /F2,/C0,SSDOC1
FCD /F3,/C0,SSDOC2
DEN
BYE BATCH
```

followed by the :JOB command:

```
:JOB USID=MWSC:1,DAD=/F3
```

Next the commands for ASM and OPT are given:

```
ASM
OPT PROG=M:CMDS,LIST=NO
NOD LEV0
ASM
OPT PROG=M:CASS,LIST=NO
ASM
OPT PROG=M:CA,LIST=NO
ASM
OPT PROG=M:CC,LIST=NO
NOD LEV1
ASM
OPT PROG=M:MESS,LIST=NO
NOD LEV2
ASM
OPT PROG=M:GO,LIST=NO
ASM
OPT PROG=M:PRNT,LIST=NO
ASM
OPT PROG=M:LIST,LIST=NO
ASM
OPT PROG=M:EDIT,LIST=NO
NOD LEV3
ASM
OPT PROG=M:SCRT,LIST=NO
ASM
OPT PROG=M:TERM,LIST=NO
```

The assembled source modules are now on the object file, which is the input for the Linkage Editor.

```
LKE
OPT MAP=YES,SLIB=NO,ULIB=NO,CATL=EDITING
```

The MAP is:

```
                START = 024C     LENGTH =5F60     REGION =6014
                     ***   OVERLAY STRUCTURE   ***

                          ***   LEVEL #    0   ***

    SEGMENT # 00       ADDRESS = 00B8   SECTOR # 0000  ASCENDANT # FF
M:CMDS  00B8

                          ***   LEVEL #    1   ***

    SEGMENT # 01       ADDRESS = 0468   SECTOR # 0003  ASCENDANT # 00
M:CASS  0468       M:CA    04E8       M:CC    056A

                          ***   LEVEL #    2   ***

    SEGMENT # 02       ADDRESS = 0660   SECTOR # 0005  ASCENDANT # 01
M:MESS  0660

                          ***   LEVEL #    3   ***

    SEGMENT # 03       ADDRESS = 096A   SECTOR # 0008  ASCENDANT # 02
M:GO    096A       M:PRNT  0A4E       M:LIST  1026       M:EDIT  5938

                          ***   LEVEL #    4   ***

    SEGMENT # 04       ADDRESS = 5E7E   SECTOR # 0042  ASCENDANT # 03
M:SCRT  5E7E       M:TERM  5F0E
```

and the Symbol Table is:

```
                          ***   SYMBOL TABLE   ***

E 00 0332 BELL2    E 00 033E BELL4    E 00 0138 BUF2     E 00 018A BUF3
E 03 4C20 BUFCH1   E 03 4C54 BUFCH2   E 00 0312 CASS2    E 01 04B2 CHKEOV
E 00 034A CHKTAB   E 03 5A70 CKCMDS   E 00 00E6 CMDBUF   E 00 00B8 CMDCAR
E 03 5142 COMPRE   E 03 09B8 COPYA1   E 00 01DA E:BUF1   E 00 01E6 E:RED1
E 00 01F2 E:WRT1   E 03 5938 EDTFLG   E 00 031E EOFREW   E 01 0468 FLAGCT
E 03 096A FLAGGO   E 03 1184 INDXBF   E 03 0A5A LINEBF   E 03 0A4E LINFLG
E 03 0E84 M:AD1    E 03 0E8A M:AD24   E 01 04E8 M:CA     E 01 046A M:CASS
E 01 056A M:CC     E 00 024C M:CMDS   E 03 5A48 M:EDIT   E 03 096C M:GO
E 03 4CDA M:LIST   E 02 0920 M:MESS   E 03 0B4A M:PRNT   E 04 5E92 M:SCR2
E 04 5EA6 M:SCR3   E 04 5EBA M:SCR4   E 04 5ECE M:SCR5   E 04 5EE4 M:SCR6
E 04 5EFA M:SCR7   E 04 5E7E M:SCRT   E 04 5F0E M:TERM   E 00 0350 M:TOP
E 00 00D0 NOSIGN   E 03 0A52 NUMBUF   E 03 10CE PAGBUF   E 03 0B06 PRTBUF
E 03 0B1E PRTTY    E 03 0AB2 REBUF1   E 03 0ADA REBUF2   E 03 0B70 SEARCH
E 03 0B02 STRNG    E 03 0B04 STRNG2   D 00 0368 SYMB1    D 01 04CE SYMB2
D 03 5546 SYMB3    D 03 0EBE SYMB4    D 02 0942 SYMB5    D 03 0A1E SYMB6
D 01 0618 SYMB7    D 01 0544 SYMB8    E 00 0306 VOLUME
```

If the same modules are Link-Edited without NOD commands, the following MAP and Symbol Table are output:

```
                START = 5D4A     LENGTH =5D9C     REGION =5E1E
                      ***   OVERLAY STRUCTURE   ***

                           ***   LEVEL #    0   ***

    SEGMENT # 00        ADDRESS = 0008   SECTOR # 0000  ASCENDANT # FF
M:CMDS  0008        M:CASS  0318        M:CA    0398        M:CC    041A
M:MESS  0500        M:GO    080A        M:PRNT  08EE        M:LIST  0EC6
M:EDIT  57D8        M:SCRT  5CBA        M:TERM  5D4A


                           ***   SYMBOL TABLE   ***

E 00 0282 BELL2   E 00 028E BELL4   E 00 0088 BUF2    E 00 00DA BUF3
E 00 4AC0 BUFCH1  E 00 4AF4 BUFCH2  E 00 0262 CASS2   E 00 0362 CHKEOV
E 00 029A CHKTAB  E 00 5910 CKCMDS  E 00 0036 CMDBUF  E 00 0008 CMDCAR
E 00 4FE2 COMPRE  E 00 0858 COPYA1  E 00 012A E:BUF1  E 00 0136 E:RED1
E 00 0142 E:WRT1  E 00 57D8 EDTFLG  E 00 026E EOFREW  E 00 0318 FLAGCT
E 00 080A FLAGGO  E 00 1024 INDXBF  E 00 08FA LINEBF  E 00 08EE LINFLG
E 00 0D24 M:AD1   E 00 0D2A M:AD24  E 00 0398 M:CA    E 00 031A M:CASS
E 00 041A M:CC    E 00 019C M:CMDS  E 00 58E8 M:EDIT  E 00 080C M:GO
E 00 4B7A M:LIST  E 00 07C0 M:MESS  E 00 09EA M:PRNT  E 00 5CCE M:SCR2
E 00 5CE2 M:SCR3  E 00 5CF6 M:SCR4  E 00 5D0A M:SCR5  E 00 5D20 M:SCR6
E 00 5D36 M:SCR7  E 00 5CBA M:SCRT  E 00 5D4A M:TERM  E 00 02A0 M:TOP
E 00 0020 NOSIGN  E 00 08F2 NUMBUF  E 00 0F6E PAGBUF  E 00 09A6 PRTBUF
E 00 09BE PRTTY   E 00 0952 REBUF1  E 00 097A REBUF2  E 00 0A10 SEARCH
E 00 09A2 STRNG   E 00 09A4 STRNG2  D 00 02B8 SYMB1   D 00 037E SYMB2
D 00 53E6 SYMB3   D 00 0D5E SYMB4   D 00 07E2 SYMB5   D 00 08BE SYMB6
D 00 04C8 SYMB7   D 00 03F4 SYMB8   E 00 0256 VOLUME
```