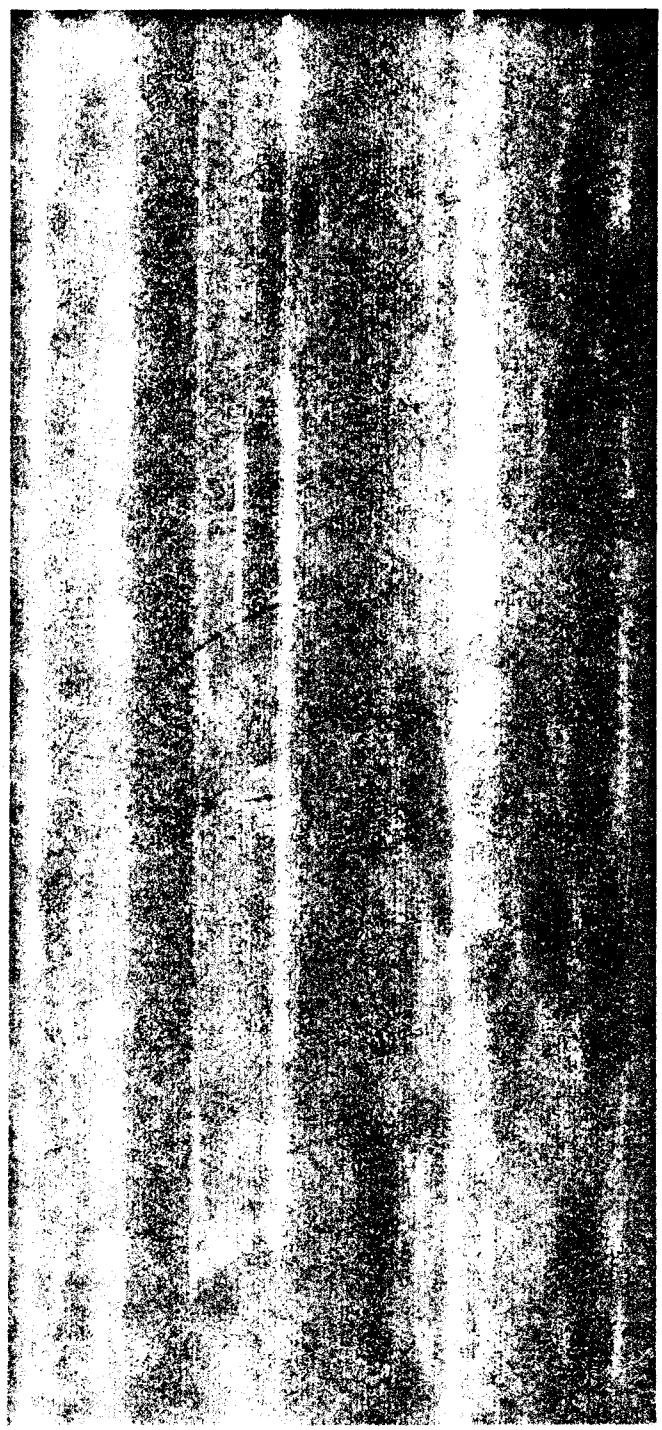
Honeywell Bull

LINK-700 LINKAGE EDITOR

SYSTEM 700

OS/700

SOFTWARE



Ref. : 42-AG08, Rev. 1

Honeywell Bull

LINK-700 LINKAGE EDITOR

SYSTEM 700

OS/700

SUBJECT:

Functions and Commands of the Linkage Editor.

SPECIAL INSTRUCTIONS:

This manual supersedes the previous edition, dated July, 1972. It has been extensively revised and rewritten; therefore, additions and changes are not indicated by change bars and asterisks.

SOFTWARE SUPPORTED:

This manual supports OS/700. See the preface of the OS/700 Systems Manual, Order Number AG02, Addendum A, for information about releases supported by this manual.

DATE:

December 1974

Printed in France

Ref : 61A.2-AG08,Rev.1

PREFACE

This manual describes the LINK-700 Linkage Editor.

Section I describes linkage editor functions.

Section II describes the functional groups of linkage editor commands.

Section III explains linkage editor commands and error messages.

Appendix A describes linkage editor link text.

Appendix B contains a sample program and the resulting linkage map.

Appendix C summarizes linkage editor commands.

The following software manuals are important references for further information:

OS/700 DAP-700 Macro Assembler (Order No. AG17)
OS/700 Operators Guide (Order No. AG14)
System 700 Programmers' Reference Manual (Order No. AC72)

The following symbology is used in this manual:

- Uppercase characters represent reserved words or symbols and must be used or entered exactly as shown.
- Lowercase characters represent a symbolic name or value, the actual value of which is supplied by the user.
- Brackets [ ] indicate that the enclosed entry is optional.
- Braces { } indicate that an enclosed entry must be selected.
- Arrowhead brackets < > are sometimes used to enclose and signify operands.
- (P) represents the control-p character on the console; i.e., P typed with the control key held down.

File No.: 1923

AG08

Dépôt légal: 4e Trimestre 1975

Each section/appendix of this manual is structured according to the heading hierarchy shown below. Each heading indicates the relative level of the text that follows it.

Level	Heading Format
1 (highest)	ALL CAPITAL LETTERS, UNDERLINED
2	Initial Capital Letters, Underlined
3	ALL CAPITAL LETTERS, NOT UNDERLINED
4	Initial Capital Letters, Not Underlined
5 (lowest)	ALL CAPITAL LETTERS FOLLOWED BY COLON: Text begins on the same line

CONTENTS

CONTENTS (cont)

ILLUSTRATIONS

TABLES

SECTION I
LINKAGE EDITOR FUNCTIONS

The LINK-700 Linkage Editor runs as an activity under OS/700. It produces link text modules by linking object modules produced by a language processor (e.g., DAP-700).

OS/700 PROGRAM DEVELOPMENT

OS/700 program development is illustrated in Figure 1-1. The user must have an OS/700 disk operating system (DOS) with a DAP-700 Macro Assembler, LINK-700 Linkage Editor, EDIT-700 Text Editor, FORTRAN-700 FORTRAN Translator, and DOS utilities. DAP-700 and FORTRAN source files must be created on disk by using DOS utilities or the text editor. The FORTRAN Translator converts FORTRAN source files into DAP-700 source files, and produces a FORTRAN listing, if desired. The DAP-700 Macro Assembler assembles the resulting DAP-700 source file and/or a user-coded DAP-700 source file, and creates a DAP-700 object file and a listing file. The linkage editor can link any number of object files to produce link text of the complete program. A link map and symbol table file are produced, if desired. Utilization of link text is determined by the following factors:

- If the program will be disk resident in an OS/700 disk operating system, the load activity (LA) DOS utility command is used to convert the link text to activity format on the disk.
- If the program will be used in an offline environment, or if it will be memory resident at initialization of an OS/700 system, the link text is transferred to an external medium; e.g., magnetic tape or cards, using the DOS transfer media (TM) command. The link text is then read from the external medium into memory using the appropriate link text loader.
- If the program will be stored on an external medium and read into memory using the load activity ($LA) core operating system (COS) command, the link text must be transferred to an external medium as described above. It is then transformed into core-image text using the activity core-image text generator.

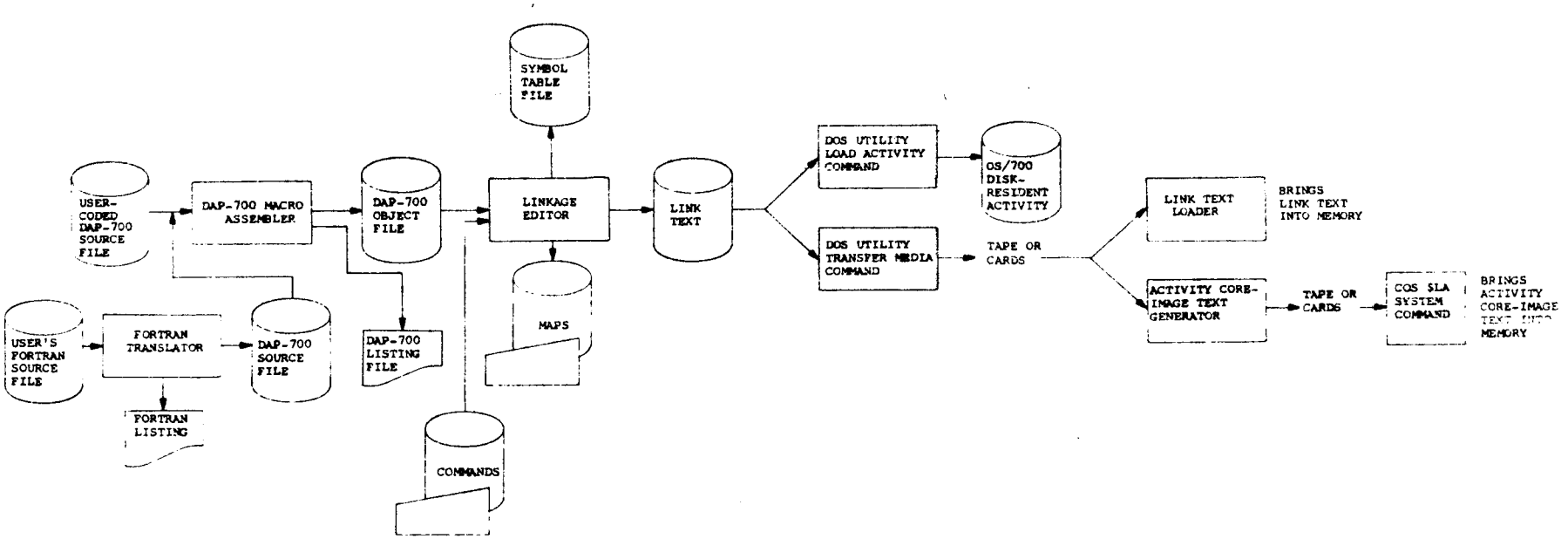


Figure 1-1. OS/700 Program Development

DEFINITION OF TERMS

An understanding of the following terms is necessary for comprehension of this manual.

Base sector - The sector that all memory-reference instructions can access directly. It is usually the same as sector 0 (the first sector of memory), but can be relocated by the base sector relocation (J-base) hardware option.

External symbol name - The name of a data item or instruction defined within one module which can be referenced within another module or by linkage editor commands. An external symbol name is defined within a module as an entry point or by using the linkage editor DEF command.

Gap - Unused area of memory, generated by SECT mode linking, into which a module can later be linked.

Indirect address word - Pointer to a memory location that is used in conjunction with memory-reference instructions to access a location outside the two sectors that a memory-reference instruction can access directly.

Link address - The first even-numbered address[1] beyond the last module linked, unless that module was linked in a gap. The next module is linked at this address unless SECT mode linking occurs or the address is altered by the ADDR command.

Link text module - An absolute memory image of the linked program.

Object module - A sequence of object text blocks prefaced by a text identification block and terminated by an end block. It is the result of a single assembly of a source module with one END pseudo-operation. A symbol table file produced by the linkage editor is also an object module. There can be more than one object module per file.

Overlay - The portion of a program that is loaded into main memory when the functions it performs are required.

Primary base area - Area reserved in the base sector, whether or not it is sector 0, by the BASE command or SETB pseudo-operation. This area is used for desectorizing instructions in all sectors. (See "Desectorization" later in this section.)

Pseudo-operation - The DAP-700 Macro Assembler source code that specifies auxiliary actions to be performed by the assembler and/or the linkage editor.

Relocatable code - Code that can be relocated within memory. Its absolute memory address is assigned by the linkage editor.

Root program - The portion of a program that resides in main memory during the program's execution; i.e., it is not overlaid. The root program is transient code group 0. (See the OS/700 DAP-700 Macro Assembler manual.)

Stream - Linkage editor input or output; represented by a two-character mnemonic. Depending on the user's requirements, input can be object text or commands, and output can be object text, link text, or memory maps. Linkage editor commands can assign OS/700 disk files to streams.

Transient code group - Program code that is part of an activity configured so that it is stored on disk and read into main memory when required for execution. Transient code groups are numbered from 0 to 126; 0 is the root program.

---

[1] For compatibility with earlier Honeywell minicomputers, the link address is always even.

LINKAGE EDITOR TASKS

The linkage editor performs the following tasks which are described later in this section:

- Desectorization
- Module selection and placement
- SECT mode linking
- Relocation of modules
- Resolution of external references
- Allocation of FORTRAN COMMON
- Establishment of overlays

Linkage editor commands provide the user with full control over the linking process. (See Sections II and III.)

Desectorization

The 716 Central Processor's memory is divided into sectors of 512 words each. A memory-reference instruction can directly address any word in the current sector (i.e., the sector containing the instruction), or in the base sector. The base sector is sector 0, unless changed using the base sector relocation option. The word referenced can be either the instruction's operand or an indirect address word. This indirect word points to the instruction's operand directly or via further indirect address words. (See the System 700 Programmers' Reference Manual for a detailed description of addressing.) If an instruction's operand resides in neither the current sector nor the base sector, it can be accessed only by using an indirect address word or by indexing. Desectorization is the process of forming indirect address words so that memory-reference instructions can access operands in any sector of memory.

A program can be desectorized explicitly, using DAC and XAC pseudo-operations. These two pseudo-operations cause the assembler (rather than the linkage editor) to form indirect address words, as in the example below.

```
0001                            *
0002                            *
0003                                    REL
0004                            *
0005                            *
0006 00000  -0 02 00771                 LDA* POINTR  Get data through
0007                            *                    indirect address word.
                                          .
                                          .
                                          .
0011                            *
0012 00771    0 001366          POINTR DAC  DATA     Address of data word.
0013                            *
                                          .
                                          .
                                          .
0017                            *
0018 01366    000005            DATA   DEC  5
0019                            *
0020                            *
0021                                    END

DATA  001366  POINTR 000771
0000 WARNING OR ERROR FLAGS
DAP-700   REV. E    74-05-14
```

DAP-700 Macro Assembly Language allows memory-reference instructions to be coded without regard to sector addressing constraints. The following example illustrates an LDA instruction accessing a data item in another sector. To desectorize this instruction, the linkage editor forms an indirect address word that contains the address of the instruction's operand, stores this word in the sector containing the instruction or in the base sector, and generates an LDA* instruction that references the indirect address word. The user must provide a desectorization area in which the linkage editor can store the indirect address word.

```
0001                            *
0002                            *
0003                                    REL
0004                            *
0005                            *
0006 00000    0 02 01366                LDA  DATA    Let linkage editor
0007                            *                    desectorize instruction.
                                          .
                                          .
                                          .
0011                            *
0012 00771                              BSD  1
0013                            *
                                          .
                                          .
                                          .
0017                            *
0018 01366    000005            DATA   DEC  5
0019                            *
0020                            *
0021                                    END

DATA   001366
0000 WARNING OR ERROR FLAGS
DAP-700   REV. E    74-05-14
```

DESECTORIZATION AREAS

There are two types of desectorization areas: primary and secondary.

The SETB pseudo-operation and the BASE command are used to specify a primary desectorization area. When the linkage editor desectorizes memory-reference instructions, the sector that contains the primary desectorization area is considered the base sector. When the memory-reference instructions are executed, that base sector must be selected.

A secondary desectorization area resides either in the base sector or in the sector containing the instructions to be desectorized (i.e., the current sector). It is specified using the BSD pseudo-operation or the BSD command.

When the linkage editor forms an indirect address word, the area in which it is stored is chosen according to the following priorities:

1. Secondary desectorization area in the current sector
2. Secondary desectorization area in the base sector
3. Primary desectorization area (must be in the base sector)

DESECTORIZATION MODES

The linkage editor supports two desectorization modes: normal and extended. These modes correspond to the two addressing modes of the 716 Central Processor. A program should be linked in the desectorization mode corresponding to the addressing mode in which it will be executed. The proper desectorization mode is essential for desectorizing indexed instructions, handling addresses larger than 16K, and using DAP-700 features related to universal coding: index tags 2 and 3, and the PXA pseudo-operation.

The EXD pseudo-operation and EXD command put the linkage editor into extended desectorization mode, which is the default mode. The LXD pseudo-operation and LXD command put the linkage editor into normal desectorization mode.

Module Selection and Placement

The linkage editor links a module only if one or more of the following conditions apply:

- The module contains an entry point referenced by a previous module but not yet defined.
- The module contains a FRCE pseudo-operation.
- The linkage editor is in FORCE mode. (See Section III, "FORCE Mode Command.")
- The linkage editor is in TOTAL mode. (See Section III, "TOTAL Mode Command.")

If a module does not contain a SECT pseudo-operation, it is linked at the current link address, crossing sector boundaries if necessary. The link address is then advanced to the next even-numbered address beyond the module.

A module is linked in a transient code group only if the module satisfies a previously unresolved reference from that transient code group or transient code group 0.

SECT Mode Linking

With SECT mode linking, if a module is smaller than one sector it is linked entirely within one sector. If a module is larger than one sector it is linked starting at a sector boundary.

A SECT pseudo-operation in a relocatable module directs where the linkage editor should link the module, based on the following rules:

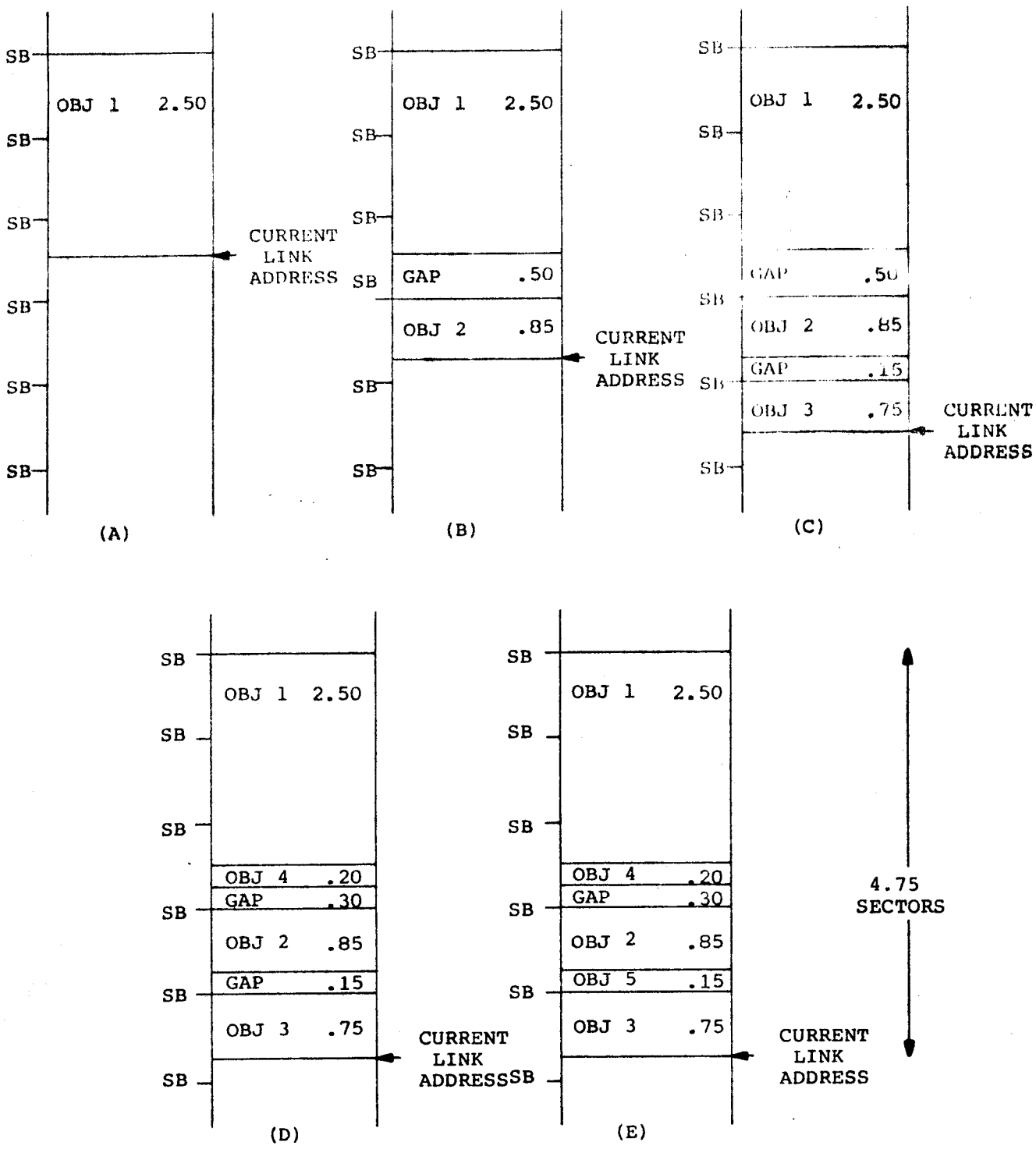
- If a gap resulting from a previous SECT mode operation or MOVE command is large enough to contain the module, the module is linked in that gap.[1] The link address is not altered.
- If the module does not fit into a gap, it is linked at the link address, provided the remainder of the current sector is large enough to contain it. The link address is then advanced to the next even-numbered address beyond the module.
- If the module is larger than the current sector, it is linked at the next sector boundary. The resulting gap is recorded for linking other SECT mode modules. The link address is then advanced to the next even-numbered address beyond the module.

SECT mode linking is most efficient if the modules are linked in descending order by size. Figure 1-2 illustrates SECT mode linking of five object modules of the following sizes:

2.50
.85
.75
.20
.15

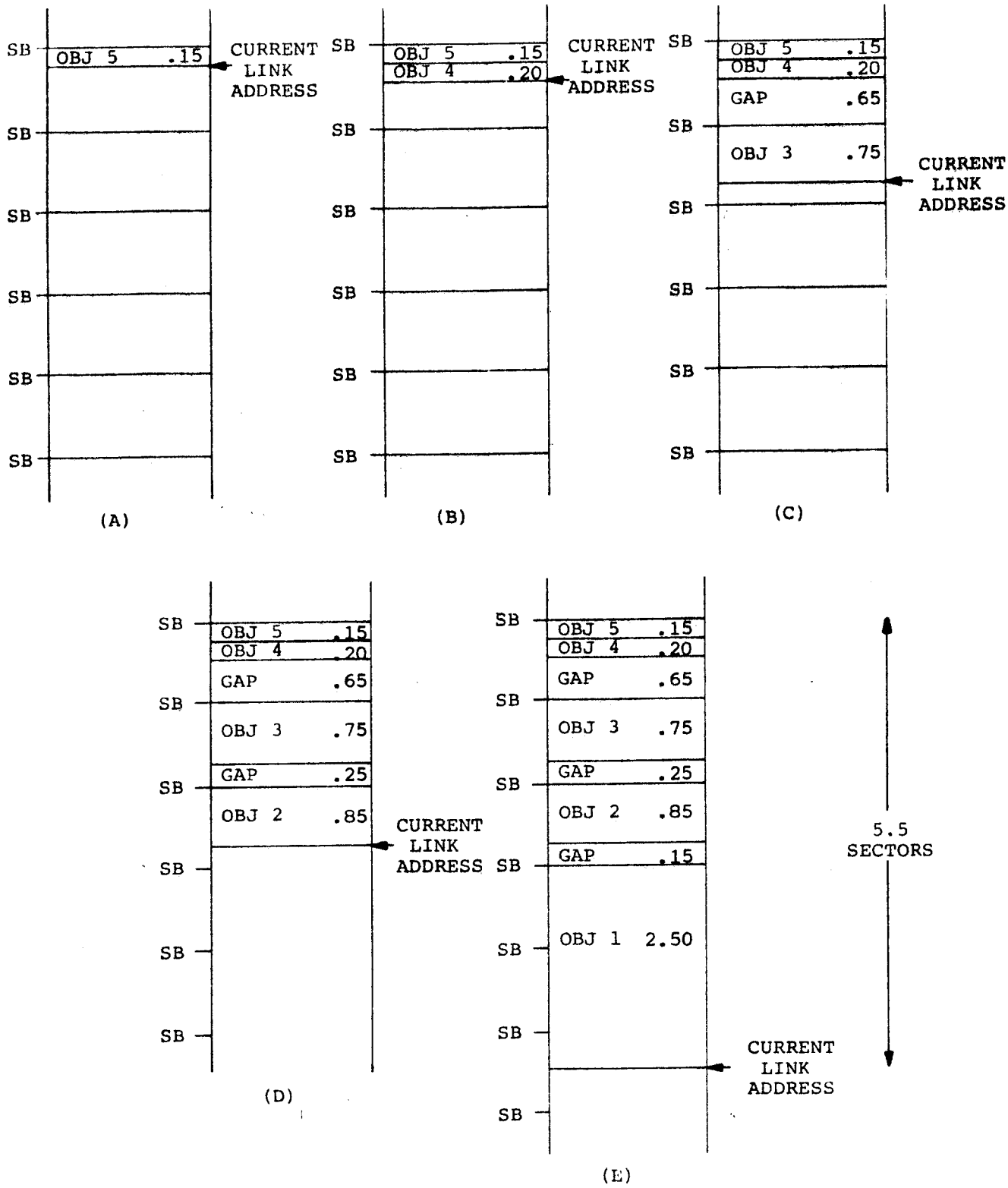
Figure 1-3 shows the same modules linked in ascending order by size, resulting in less efficient memory usage.

[1]Gaps resulting from unused portions of BSD (block storage for desectorization) areas cannot be used for linking modules.



SB - Sector boundary

Figure 1-2. SECT Mode Linking of Modules in Descending Order by Size



SB - Sector boundary

Figure 1-3. SECT Mode Linking of Modules in Ascending Order by Size

DESECTORIZATION UNDER SECT MODE LINKING

SECT mode linking allows the user to determine the maximum number of indirect address words which can be generated when a module is linked. If a module is smaller than a sector, indirect address words result only from references to locations outside the module. If a module is larger than a sector, the module is linked starting at a sector boundary, and the user can (1) determine which instructions cause indirect address words to be generated, (2) organize the module to minimize the number of indirect address words generated, and (3) provide BSD areas where needed.

Relocation of Modules

The assembler assigns addresses comprising a number and a relocation flag to program code. The address can be absolute or relocatable. (See Section II of the OS/700 DAP-700 Macro Assembler manual.) The linkage editor performs a mapping function between the assembler-assigned addresses and actual memory addresses according to the following rules:

- If the address is absolute, the numeric value is the actual memory address.
- If the address is relocatable, the current link address is added to the numeric address to determine the actual memory address. The link address is initially set to '1000 and can be modified by the linkage editor ADDR command. After each module is linked, the link address is set to the next even-numbered address beyond the module.

Resolution of External References

When the linkage editor encounters an external reference while linking a module, it determines whether the referenced external symbol was previously defined; if it was, the instruction's address can be specified immediately. Otherwise, the linkage editor records in the symbol table the external symbol name, the instruction that made the reference, and the instruction's address, so that the instruction can be generated when the symbol is defined. Linkage editor memory map commands produce lists of defined and undefined symbols.

Allocation of the FORTRAN COMMON

The COMMON is a data area that can be shared by numerous FORTRAN programs and subprograms. DAP-700 programs can also access this area. A COMMON block is defined in a FORTRAN program by the COMMON statement, and in DAP-700 by the COMM pseudo-operation.

The COMMON area is allocated starting at the base address, which the user can designate (see Section III, "Linkage Editor Initial Conditions"), and extends to successively lower addresses (towards program storage). (See Figure 1-4.) If the COMMON area overlaps program storage, memory overflow occurs and the program must be restructured.

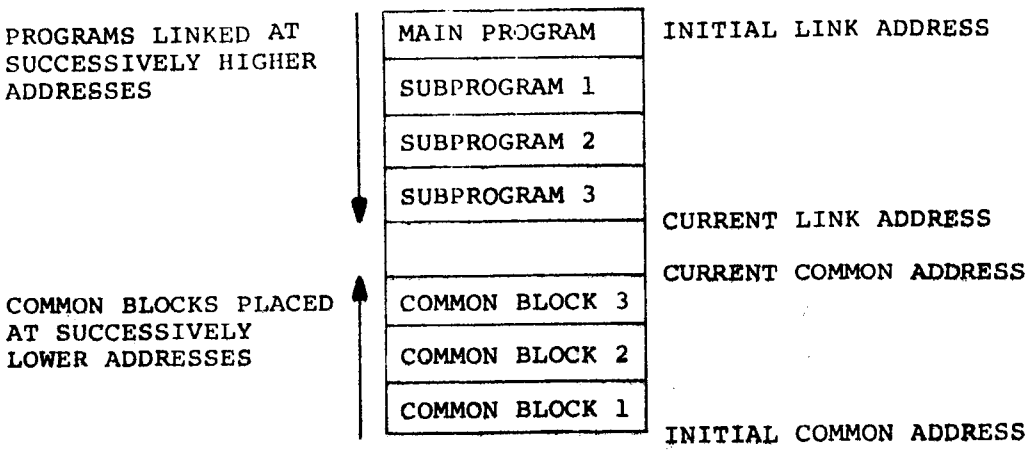


Figure 1-4. Allocation of the FORTRAN COMMON

Establishment of Overlays

Users can establish overlays in a program by using transient code groups and the DAP-700 Macro Assembler. (See the example in Appendix B.) Transient code groups are memory images into which the linkage editor and the assembler can store code. Any overlay can be loaded and executed, when required, into the program's overlay area. The Transient Code Manager Initializer and Transient Code Manager are subroutines used by OS/700 activities for managing overlays. (See the OS/700 DAP-700 Macro Assembler manual.)

The procedure for establishing overlays in a program is:

1. Use the TCD pseudo-operation to specify the beginning and ending addresses of the overlay area, and the number of overlays associated with the program. Overlays are assigned transient code group numbers from 1 to 126. The root program (the permanently memory-resident portion of the program) is transient code group 0. The TCD pseudo-operation must be in the first module linked in the program, and must be before any executable code in that module.
2. Prior to linking a module that will be part of an overlay, use the linkage editor TCG command or the TCG pseudo-operation in the source code to specify the transient code group number. Link the overlays in the defined overlay area using the linkage editor ADDR command or the ORG pseudo-operation.

EFFECT OF TRANSIENT CODE GROUPS ON DESECTORIZATION

Indirect address words formed by the linkage editor are stored in desectorization areas. A secondary desectorization area can be designated in a transient code group by using the linkage editor BSD command or the BSD pseudo-operation. The desectorization area is used to desectorize instructions in that transient code group only. An unused BSD area that is part of a transient code group other than 0 is not included in a gap table produced by the GAPT command. (See Section III, "Gap Table Generation Command (GAPT).")

EFFECT OF TRANSIENT CODE GROUPS ON SECT MODE LINKING

The linkage editor records, in its symbol table, the number of gaps in each transient code group. Only gaps in transient code group 0 are included in a gap table. (See Section III, "Gap Table Generation Command (GAPT).")

EFFECT OF TRANSIENT CODE GROUPS ON EXTERNAL LINKING

A symbol can have the same name, but different values, in several transient code groups.

An external reference by a module in transient code group 0 (the root program) can be resolved by a symbol defined in any transient code group; the most recent definition of the symbol is used. If the symbol is not yet defined, the first subsequent definition of that symbol is used.

An external reference by a module in transient code group 1 through 126 can be resolved only by a symbol defined in that transient code group or in the root program. If defined differently in both places, the definition in the root program is used.

When a symbol is defined in the root program, all previously unresolved references to that symbol are resolved.

When a symbol is defined in transient code group 1 through 126, all previously unresolved references to that symbol in the root program and in that transient code group are resolved. References in other transient code groups are not resolved.

SECTION II
FUNCTIONAL GROUPS OF LINKAGE EDITOR COMMANDS

Linkage editor commands have the following functions:

- Assign streams to OS/700 files
- Control object text input
- Control module selection
- Control module placement
- Control desectorization
- Produce special link text
- Produce link maps
- Produce a symbol table file
- Control symbol definition
- Specify a transient code group
- Control command execution
- Terminate the linking process

Command functions are described below. See Section III for command usage and formats.

STREAM ASSIGNMENT COMMANDS

BI
CI
BO
MO
OO

Stream assignment commands are used to assign I/O streams to OS/700 files. The linkage editor reads object text through the binary input (BI) stream, and commands through the command input (CI) stream. It writes link text into the binary output (BO) stream, maps into the map output (MO) stream, and symbol table files into the object output (OO) stream. Commands can also be entered from the console, and maps can be printed on the console.

OBJECT TEXT INPUT COMMANDS

LINK
SKIP
NORM
STEP

LINK and SKIP commands cause object text to be read from the binary input stream. LINK causes object modules that are read to be linked or ignored according to the rules of module selection. (See "Module Selection and Placement" in Section I.) If a binary output stream is assigned when a module is linked, link text is produced as object text is read. SKIP causes object modules that are read to be ignored. This command is used to bypass object modules within an object library.

NORM and STEP determine how object modules are read from the binary input stream following a LINK command. If a NORM command was executed, LINK causes object modules to be read from the binary input stream until an end of file is encountered. If a STEP command was executed, a LINK command causes object modules to be read until one is linked (not ignored).

NORM and STEP commands are significant only when reading object libraries. NORM is usually used with the LIB command to scan a library of subroutines and link those subroutines referenced by a main program. STEP is used when modules in an object library must be linked individually.

NORM and STEP are complementary commands; each remains in effect until cancelled by the other.

MODULE SELECTION COMMANDS

FORCE
LIB
TOTAL

When a LINK command is used to read a module, FORCE, LIB, and TOTAL commands can be used to determine whether the module will be linked.

A FORCE command causes the next module read using a LINK command to be linked unconditionally. The FORCE command affects that module only. If read using the SKIP command, the module is ignored.

Following the execution of a LIB command, each object module read using a LINK command is linked if one of the following conditions apply: a FORCE command is in effect, the module contains a FRCE pseudo-operation, or the module contains at least one entry point that satisfies a previously unresolved reference from the same transient code group or from transient code group 0. Otherwise, the module is ignored.

Following the execution of a TOTAL command, all object modules read using a LINK command are linked unconditionally.

LIB and TOTAL are complementary commands; each remains in effect until cancelled by the other, although a FORCE command overrides a LIB command for the first module read following a FORCE command.

MODULE PLACEMENT COMMANDS

ADDR
MORG
GBASE
COMM

ADDR and MORG commands change the link address. ADDR changes the link address to a designated value; MORG advances the link address to the next address that is a multiple of a designated value, and produces a gap corresponding to the area between the previous and current link addresses.

ADDR is used to set the initial link address, or to change the link address when the next module read will not be linked immediately following the previous module.

MORG is used when a module must be linked at a particular boundary; e.g., a sector boundary.

GBASE causes gaps below the link address, or below a designated address, to be ignored during SECT mode linking. GBASE allows SECT mode linking to be performed while controlling module placement.

COMM selects the beginning address for allocation of COMMON blocks. COMMON blocks are assigned successively lower addresses.

DESECTORIZATION COMMANDS

BASE
BSD
EXD
LXD

BASE and BSD commands allocate desectorization areas, which are used by the linkage editor for storing indirect address words. BASE specifies a primary desectorization area, and designates the sector containing this area as the base sector for subsequent modules. BSD specifies a secondary desectorization area. BASE and BSD commands are equivalent to SETB and BSD pseudo-operations, respectively.

EXD and LXD commands cause the linkage editor to desectorize subsequent modules for execution in extended or normal addressing mode respectively. EXD and LXD are complementary commands that are equivalent to EXD and LXD pseudo-operations. Each command remains in effect until cancelled by the other.

SPECIAL LINK TEXT COMMANDS

IDNT
CIDNT
GAPT

IDNT and CIDNT commands are used to write link text identification blocks into the binary output stream. IDNT produces an identification block which contains a specified string. CIDNT produces two blocks; one contains a specified string and one contains a Honeywell copyright notice. These identification blocks are printed on the console by Honeywell programs that read link text.

GAPT causes link text data blocks to be written into the binary output stream. Link text data blocks are a memory image of a table of all gaps and unused desectorization areas in transient code group 0. (See Appendix A "LINK-700 Link Text.") By examining the gap table, a program can locate unused memory areas that can be used to store data.

LINK MAP COMMANDS

MAPS
MAP
MAPF

MAPS, MAP, and MAPF commands cause link maps to be written into the map output stream. These commands can be specified at any time during linking to write any number of maps. MAPS, MAP, and MAPF generate short, intermediate, and full-size maps, respectively.

SYMBOL TABLE FILE COMMAND

SYMT

The SYMT command causes the linkage editor to write into the object output stream an object module that contains an entry point for every currently defined symbol in the linking process. These symbols can be used in future linking processes.

For example, if program B references external symbols defined in program A, the symbol table is produced when linking program A and linked as part of program B. The same effect can be achieved using the FIN command if both programs are linked during the same execution of the linkage editor.

SYMBOL DEFINITION COMMANDS

DEF
NCULL
CULL