29

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:

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

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)	<u>ALL CAPITAL LETTERS, UNDERLINED</u>
2	<u>Initial Capital Letters, Underlined</u>
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

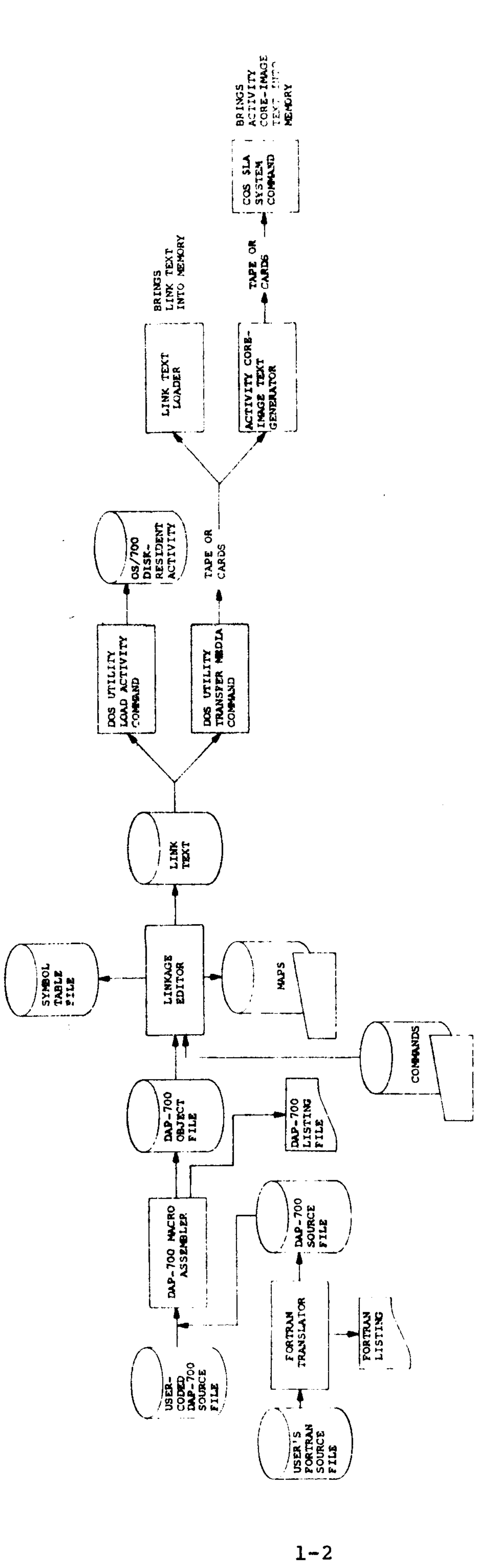
SECTION I
LINKAGE EDITOR FUNCTIONS

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.



USER-CODED DAP-700 SOURCE FILE
USER'S FORTRAN SOURCE FILE
FORTRAN TRANSLATOR
FORTRAN LISTING
DAP-700 SOURCE FILE
DAP-700 MACRO ASSEMBLER
DAP-700 OBJECT FILE
DAP-700 LISTING FILE
COMMANDS
LINKAGE EDITOR
SYMBOL TABLE FILE
MAPS
LINK TEXT
DOS UTILITY LOAD ACTIVITY COMMAND
DOS UTILITY TRANSFER MEDIA COMMAND
OS/700 DISK-RESIDENT ACTIVITY
TAPE OR CARDS
LINK TEXT LOADER
BRINGS LINK TEXT INTO MEMORY
ACTIVITY CORE-IMAGE TEXT GENERATOR
TAPE OR CARDS
COS $LA SYSTEM COMMAND
BRINGS ACTIVITY CORE-IMAGE TEXT INTO MEMORY

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 <u>OS/700 DAP-700 Macro Assembler</u> 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:

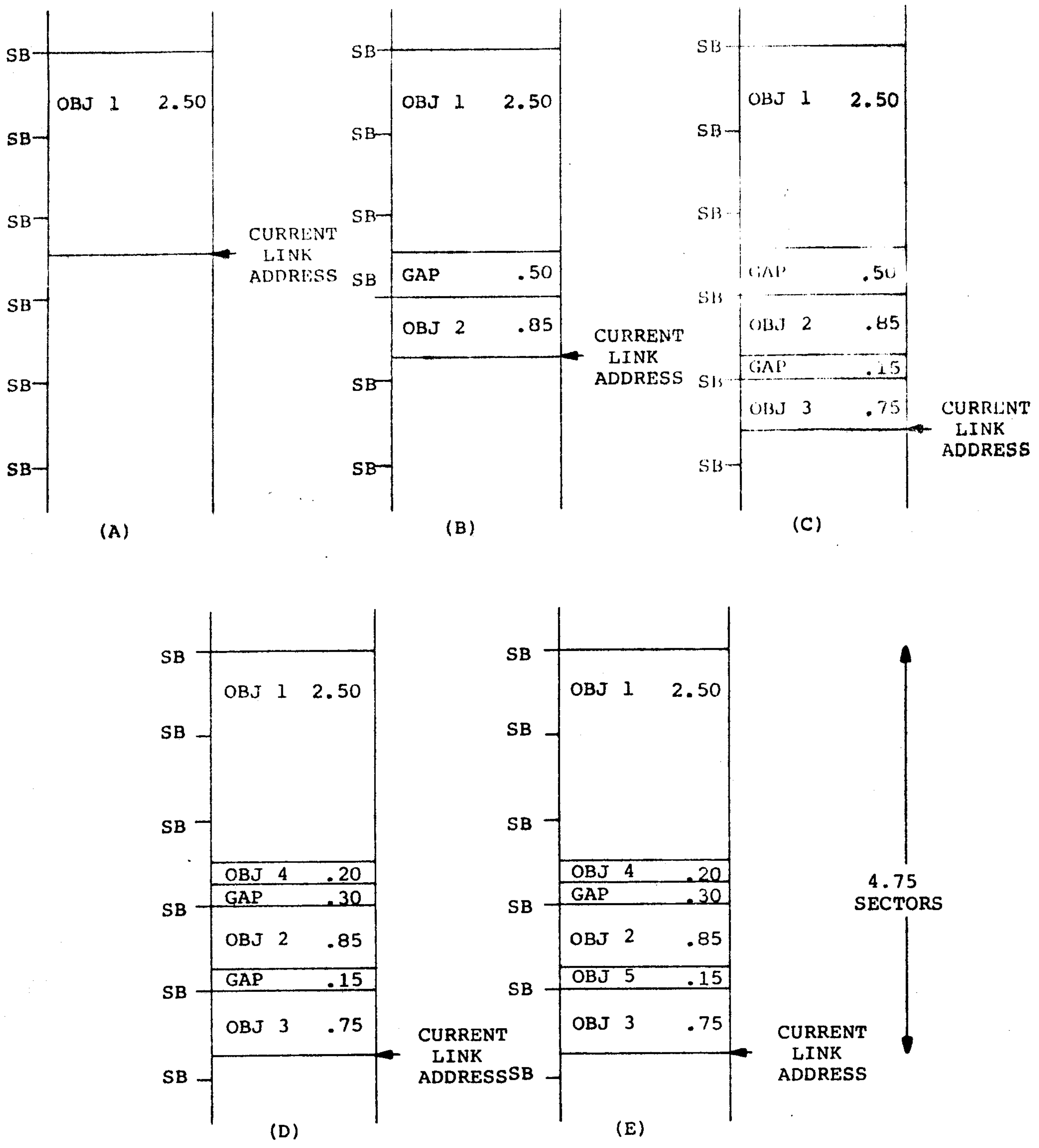
- If a gap resulting from a previous SECT mode operation or MOKG command is large enough to contain the module, the module is linked in that gap.[1] The link address is not altered.
- If the module does not fit into a gap, it is linked at the link address, provided the remainder of the current sector 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

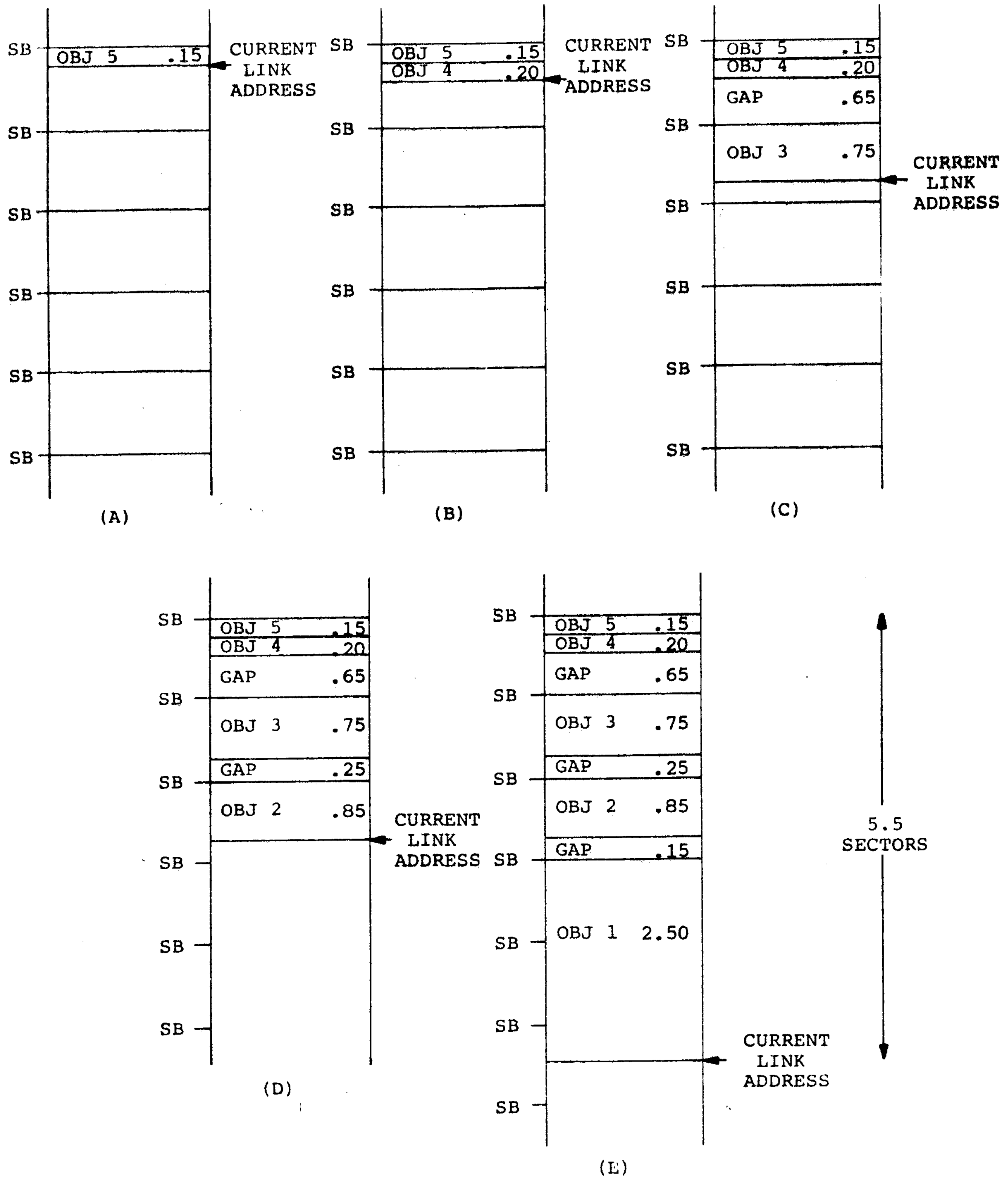
Figure 1-3 shows the same modules linked in ascending order by size, resulting in less efficient memory usage.

[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.

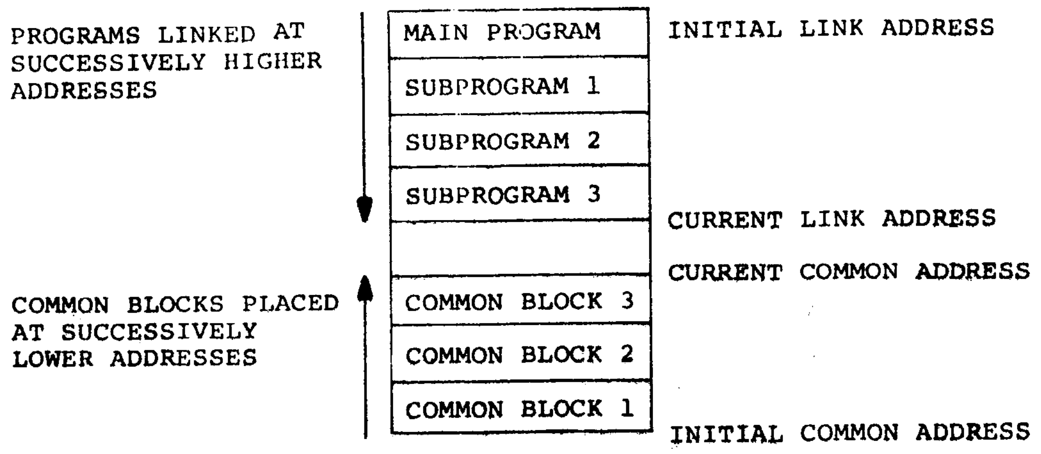


Figure 1-4. Allocation of the FORTRAN COMMON

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
```

The DEF command allows a symbol to be defined explicitly during a linking process. A symbol can also be defined by an entry point in an object module. DEF is useful in:

- Defining hardware device addresses for programs with external symbols in the device address field of I/O instructions
- Defining symbolic parameters for linkage editor commands before the command file in which they appear is assigned to the command input stream

NCULL and CULL govern the criteria by which the linkage editor enters symbol definitions into its memory-resident symbol table. When NCULL is in effect, any symbol defined by DEF or by an object module entry point is entered into the symbol table, unless the symbol was defined previously in the current transient code group. When CULL is in effect, the symbol must satisfy a previously unresolved reference from the current transient code group or from transient code group 0.

NCULL and CULL are complementary commands; each remains in effect until cancelled by the other. Typically, CULL mode is used when linking a symbol table file from a previously-linked program, so that symbols defined in the first program are retained only if they are referenced in the program currently being linked.

TRANSIENT CODE GROUP COMMAND

```
                              TCG
```

The TCG command directs the linkage editor to assign code in subsequently linked object text to a specified transient code grcup. Symbols subsequently defined by DEF, or by an object module entry point with no explicit transient code group assignment, are assigned to that transient code group. The TCG command is equivalent to the TCG pseudo-operation.

CONDITIONAL COMMAND EXECUTION COMMANDS

```
                              IFZ
                              IFN
                              ELSE
                              ENDC
```

IFZ, IFN, ELSE, and ENDC commands allow other linkage editor commands read from the command input stream to be conditionally executed. These commands have the same functions as similarly-named DAP-700 pseudo-operations. With conditional command execution, a linkage editor command file can be written that suits a variety of situations.

LINKING PROCESS TERMINATION COMMANDS

```
INIT
FIN
QUIT
```

INIT, FIN, and QUIT commands terminate the current linking process; link text generation is completed and the BO stream is closed. INIT reinitializes the linkage editor to its original state, except that the common address assigned by the most recent COMM command is preserved. FIN differs from INIT in that all symbol and COMMON block definitions are preserved for a subsequent linking process. QUIT terminates the linkage editor activity.

SECTION III
LINKAGE EDITOR COMMANDS AND ERROR MESSAGES

The linkage editor sequentially reads and processes command lines. Each command line consists of one or more linkage editor commands. Some commands require numeric parameters, which are denoted in the command syntax by num. Num can be one term or a sequence of terms separated by a plus sign (+), denoting addition, or a minus sign (-), denoting subtraction. The results of arithmetic operations are truncated to 15 bits. Valid terms are:

Decimal constants -	One to five decimal digits (maximum value is 32767)
Octal constants -	Apostrophe (') followed by one to six octal digits (maximum value is '177777)
Hexadecimal constants -	Dollar sign ($) followed by one to four hexadecimal digits (maximum value is $FFFF)
External symbol name -	Must be defined in the current transient code group[1] or in the root program. If defined differently in both places, the value in the root program is used.

LINKAGE EDITOR INITIALIZATION

The linkage editor is initialized by entering the schedule activity ($SA) system command through the console:

```
(P)$SA ZLE(CR)
```
[2]

(P) - Control-P (nonprinting character)

ZLE - Linkage editor activity name

[1] The current transient code group is the last transient code group specified by a TCG pseudo-operation or by the linkage editor TCG command.

[2] (CR) designates carriage return.

In response, there is a version identification typeout in the form:

```
    ZLE    LE-700  REV. n  yy/mm/dd
```

Command input is initially requested by the typeout:

```
    nn  ZLE     !
```

nn - Message number for associating operator response with message printed

ZLE - Linkage editor activity name

Each command line entered through the console must be in the form:

```
    (P)nn<command string>(CR)
```

(P) - Control-P (nonprinting character)

nn - Message number; identical to the one printed in the command request

<command string> - Series of linkage editor commands, separated by commas and terminated by carriage return

Additional commands can be entered through the console, or a command input (CI) stream can be assigned to a file containing linkage editor commands. Commands issued from a CI file do not require the (P) and message number.

After each command line is processed, the linkage editor issues another command request so that more commands can be entered; when the linkage editor receives a QUIT command, the linkage editor activity terminates.

Linkage Editor Initial Conditions

The linkage editor is initially in the state defined by the following commands: LIB, NORM, FORCE, EXD, GBASE=0, ADDR='1000, TCG=0, and NCULL. (See "Linkage Editor Command Set" below.) It is returned to this state after an INIT command is processed.

The default value of the COMMON address is the linkage editor activity's ending address plus 1, which is configuration dependent. (See "COMMON Address Command (COMM)" later in this section.) This value can be ascertained with a MAPS command.

The linkage editor establishes a primary desectorization area in locations '163 through '777 if not user defined before it is required by the linkage editor. (See "BASE Command" later in this section.) Sector 0 is initially considered the base sector.

LINKAGE EDITOR COMMAND SET

Linkage editor commands are described below, alphabetically. Some examples are provided to illustrate command usage. See "MAP Command" later in this section for a description of MAP, MAPF, and MAPS, which are often designated in examples.

Address Command (ADDR)

Format:	ADDR=num
Function:	Sets the linkage editor link address to the specified address. If the address is odd, it is advanced by one. If the next module to be linked is relocatable, it is linked starting at the specified address, unless it is being linked in SECT mode. (See Section I, "SECT Mode Linking.")
Default:	ADDR='1000

Example:

```
00  ZLE      ! 00ADDR='2000,MAPS

ZLE
ZLE     STATE
ZLE
ZLE          ADDR   02000
ZLE          COMM   40300
00  ZLE      !
```

BASE

BASE Command

Formats: BASE=num1<num2
BASE=num1

Function: Establishes a primary desectorization area. The area starts at the address specified by num1 and extends to the address specified by num2. If num2 is not in the same sector as num1, or if num2 is omitted, the area extends from num1 to the end of the sector containing num1. When desectorizing subsequent instructions, the sector containing the primary desectorization area is considered the base sector. The base sector relocation register should be set to the same sector when the instructions are executed.

NOTE: If a program with a relocated base sector (J-base) is to be linked with the linkage editor, the base area should begin at the sector boundary plus 6 to bypass hardware register addresses 0 to 5 in the relocated base sector; i.e., base would be xxx06 for a program starting on sector boundary xxx00. No error message is produced if this is not done, but the linked program does not execute properly.

Default: None. If a base area becomes necessary for desectorization, BASE='163 becomes effective automatically.

Example 1:

BASE command with one parameter:

```
    00   ZLE      !  00BASE='1400,MAPS

    ZLE
    ZLE      STATE
    ZLE
    ZLE           ADDR    01000
    ZLE           BASE    01400-01400-01777
    ZLE           COMM    40300
    00   ZLE      !
```

Example 2:

BASE command with two parameters:

```
    00   ZLE      !  00BASE='400<'767,MAPS

    ZLE
    ZLE      STATE
    ZLE
    ZLE           ADDR    01000
    ZLE           BASE    00400-00400-00767
    ZLE           COMM    40300
    00   ZLE      !
```

Block Storage for Desectorization Command (BSD)

Format: BSD=num

Function: Generates a secondary desectorization (BSD) area of the specified size at the current location. If the area does not completely fit into the current sector, the remainder goes into the next sector. The link address is advanced to the next even-numbered address beyond the area.

Default: None

Example:

```
00  ZLE     ! 00MAPF

ZLE
ZLE     STATE
ZLE
ZLE         ADDR  01000
ZLE         COMM  40300

00  ZLE     ! 00BSD='20,MAPF

ZLE
ZLE     STATE
ZLE
ZLE         ADDR  01020
ZLE         COMM  40300
ZLE
ZLE     BASE AREAS
ZLE
ZLE         01000-01000-01017
00  ZLE     !
```

Identification With Copyright Command (CIDNT)

Format: CIDNT:character string

character string - May contain as many characters as can be typed on one line; all printing characters, including embedded blanks, are allowed.

Function: Places the characters following the colon, through the rightmost nonblank character, into a text identification block at the start of the link text. A second text identification block containing a Honeywell copyright notice is also generated. If used, this command must precede the first LINK command. If multiple CIDNT commands are given, all but the first are ignored.

Default: None

COMMON Address Command (COMM)

Format: COMM=num

Function: Defines the top (beginning) of the COMMON storage area as num; num replaces the current value and the initial value. The highest location used for the COMMON storage area is num minus 1. (The COMMON is allocated from num minus 1 down.) This area is used for the FORTRAN COMMON and for the COMMON defined by the COMM pseudo-operation in DAP-700 source programs.

Default: Linkage editor activity's ending address plus 1.

Example:

```
00  ZLE     ! 00MAPS

ZLE
ZLE      STATE
ZLE           ADDR    01000
ZLE           COMM    36750

00  ZLE     ! 00COMM='4000,MAPS

ZLE
ZLE      STATE
ZLE           ADDR    01000
ZLE           COMM    04000
00  ZLE     !
```

CULL Mode Command

Format: CULL

Function: Puts the linkage editor into CULL mode. In this mode, a symbol defined by a module or by a DEF command is recorded in the symbol table only if it was previously referenced within another module but not previously defined.

Default: NCULL

Definition of Symbol Command (DEF)

Format: DEF:sym1=num1[;sym2=num2]...

Function: Defines the value of each symbol specified. Each symbol is defined in the current transient code group; if the symbol is already defined there, this definition has no effect. If the linkage editor is in CULL mode, only symbols previously referenced but not yet defined are recorded in the symbol table.

Default: None

Example:

```
00  ZLE     ! 00MAPF

ZLE
ZLE     STATE
ZLE
ZLE         ADDR   01000
ZLE         COMM   40300

00  ZLE     ! 00DEF:XLNK='1001,MAPF

ZLE
ZLE     STATE
ZLE
ZLE         ADDR   01000
ZLE         COMM   40300
ZLE
ZLE     DEFINED SYMBOLS
ZLE
ZLE         XLNK   01001
00  ZLE     !
```

Enter Extended Desectorization Mode Command (EXD)

Format: EXD

Function: Puts the linkage editor into extended desectorization mode (15-bit addresses), which is used when the program being linked will be executed using extended addressing mode. EXD remains in effect until terminated by an LXD command or by an LXD pseudo-operation in a module being linked.

Default: EXD

sh Command (FIN)

Format: FIN

Function: Directs the linkage editor to complete link text generation and to reinitialize for another linkage editor job. See "Linkage Editor Initial Conditions" earlier in this section. Records of undefined symbols, desectorization areas, and caps are deleted from the symbol table. COMMON block defintions, the current COMMON address, and symbol definitions are preserved.

Default: None

Example:

```
00  ZLE     ! 00MAPF

ZLE
ZLE     STATE
ZLE
ZLE         START  01000
ZLE         LOW    01000
ZLE         HIGH   02672
ZLE         ADDR   02674
ZLE         BASE   00163-00270-00777
ZLE         COMM   40277
ZLE
ZLE     UNDEFINED SYMBOLS
ZLE
ZLE         LINKED
ZLE         ICOMM
ZLE
ZLE     DEFINED SYMBOLS
ZLE
ZLE         GENIO  01000
ZLE         ERROR  02323
ZLE         F$C1   02540
ZLE         F$C2   02543
ZLE         F$C3   02550
ZLE         F$C4   02555
ZLE         F$C5   02561
ZLE         F$C6   02565
ZLE         F$RB   02571
ZLE         F$CG   02615
ZLE         F$AT   02623
ZLE
ZLE     BASE AREAS
ZLE
ZLE        *00163-00270-00777
ZLE
ZLE     COMMON BLOCKS
ZLE
ZLE         SYMTOP 40277

00   ZLE        ! 00FIN,MAPF

ZLE
ZLE     STATE
ZLE
ZLE         ADDR   01000
ZLE         COMM   40277
ZLE
ZLE     DEFINED SYMBOLS
ZLE
ZLE         GENIO  01000
ZLE         ERROR  02323
ZLE         F$C1   02540
ZLE         F$C2   02543
ZLE         F$C3   02550
ZLE         F$C4   02555
ZLE         F$C5   02561
ZLE         F$C6   02565
ZLE         F$RB   02571
ZLE         F$CG   02615
ZLE         F$AT   02623
ZLE
ZLE     COMMON BLOCKS
ZLE
ZLE         SYMTOP 40277
00   ZLE        !
```

FORCE Mode Command

Format:	FORCE
Function:	Causes linking of the next module read after a LINK command, regardless of whether the module contains referenced external symbols. FORCE mode remains in effect only until the next module is linked or skipped.
Default:	FORCE

Gap Table Generation Command (GAPT)

Formats:	GAPT GAPT=num
Function:	Causes the linkage editor to write into the binary output stream a table of memory gaps and unused areas in desectorization areas residing in transient code group 0. The table is generated when the GAPT command is processed. The table's starting address is the value of num, or the address HIGH plus 1 (see Figure 3-1) if num is omitted. The gap table can be examined by any program that wants to utilize the gaps as data areas. Appendix A describes gap table format.
Default:	None

Gap Base Command (GBASE)

Formats: GBASE
GBASE=num

Function: When modules are linked in SECT mode, memory areas may be left unused. The linkage editor records the location and size of these gaps for linking future object text modules in SECT mode. The GBASE command instructs the linkage editor to ignore gaps below the current link address. GBASE=num instructs the linkage editor to ignore gaps below the value num. Modules are not linked into an ignored gap, but the gap is in the gap table and in maps. If a subsequent GBASE command lowers the gap base, gaps between the old and new gap base are not ignored.

Default: GBASE=0

Identification Command (IDNT)

Format: IDNT:character string

character string - May contain as many characters as can be typed on one line; all printing characters, including embedded blanks, are allowed.

Function: Places the characters following the colon, through the rightmost nonblank character, into a text identification block at the start of the link text. If used, this command must precede the first LINK command. If multiple IDNT commands are given, all but the first are ignored.

Default: None

Conditional Command Execution Command (IFN, IFZ)

Formats: $\left\{\begin{matrix}\text{IFN}\\\text{IFZ}\end{matrix}\right\}$:num...[ELSE]...ENDC

Function: Allows other linkage editor commands to be executed conditionally, based on the value of num. Num can be any expression, including previously defined symbols. The IFZ command directs the linkage editor to execute subsequent commands only if num is 0; otherwise command execution is inhibited. An ENDC command indicates the end of this conditional command execution. The IFN command functions like the IFZ command, except that subsequent commands are executed only if num is not 0.

The ELSE command can be placed between an IFZ or IFN command and an ENDC command. It reverses the effect of the IFZ or IFN command on subsequent commands; if command execution was enabled, it is inhibited until ENDC is reached.

A sequence of the above commands can be included within another such sequence. If command execution is inhibited because of a previous IFZ, IFN, or ELSE command, the sequence is ignored; otherwise, the sequence is executed normally. An ELSE or ENDC command that is not matched by a prior IFZ or IFN command constitutes a command error. (See "Error Messages" later in this section.)

If an INIT or FIN command is encountered when command execution is enabled, the linkage editor is reinitialized and the effect of previous IFZ, IFN, and ELSE commands is cancelled. If command execution is inhibited, all commands (including IFZ, IFN, and ELSE) are ignored.

The effect of previous IFZ, IFN, and ELSE commands is cancelled whenever the linkage editor encounters errors.

Default: Command execution enabled $\left\{\begin{matrix}\text{IFZ:0}\\\text{IFN:1}\end{matrix}\right\}$

Example:

```
00  ZLE      ! OODEF: SWITCH=1
                    :
               
ZLE          BI=QLIB
ZLE          IFZ:SWITCH
ZLE          SKIP,LINK   These commands are not executed.
ZLE          ELSE
ZLE          LINK,SKIP  These commands are executed.
ZLE          ENDC
ZLE          LINK
00  ZLE      !
```

Initialize Command (INIT)

Format: INIT

Function: Directs the linkage editor to complete the link text generation and to reinitialize for another linkage editor job. (See "Linkage Editor Initial Conditions" earlier in this section.) The INIT command is executed when the user has produced one link text file and wants to build another one without restarting the linkage editor. All records of defined symbols, undefined symbols, COMMON blocks, gaps, and desectorization areas are deleted from the symbol table.

Default: None

Library Mode Command (LIB)

Format: LIB

Function: Puts the linkage editor into LIB mode, in which each module read is linked only if:

1. The module defines at least one referenced, previously undefined, external symbol; or
2. The module contains a FRCE pseudo-operation; or
3. A FORCE command is in effect.

The TOTAL command terminates LIB mode.

Default: LIB

LINK Command

Format: LINK

Function: Initiates the reading and processing of object text. The linkage editor stops reading object text and resumes command processing at the end of the first linked module if in STEP mode, or when an end of file is detected if in NORM mode. See Section I, "Module Selection and Placement," to determine whether a module is linked or ignored.

Default: None

Leave Extended Desectorization Mode Command (LXD)

Format: LXD

Function: Puts the linkage editor into normal desectorization mode (14-bit addresses), which is used when the program being linked will be executed using normal addressing mode. LXD remains in effect until terminated by an EXD command or by an EXD pseudo-operation in a module being linked.

Default: EXD

MAP Command

Formats: MAP
MAPS
MAPF

Function: Directs the linkage editor to produce a link map on the MO output stream. The MAP command creates a map that contains the linkage editor "state" information and a list of undefined symbols; the MAPS command creates a map containing only the linkage editor "state" information; and the MAPF command creates a full map.

The link map comprises the following sections:

- Link Test Identification - Copy of ASCII character string provided by CIDNT or IDNT command; printed only if MAPF is specified.
- State - START - Execution starting address.

START	Execution starting address.
LOW	Lowest memory location occupied by programs, except base areas and COMMON blocks.
HIGH	Highest memory location occupied by programs, except COMMON blocks.
ADDR	Current link address.
GAPH	Highest memory location occupied by the gap table.
BASE	Primary desectorization area's first location, next available location, and last available location. If the next available location is beyond the last available location, the area is full.
COMM	Current COMMON address.
CLOW	Lowest COMMON location initialized.
CHIGH	Highest COMMON location initialized.
CTCG	Current transient code group number.
TCDL	Low address provided by a TCD pseudo-operation.
TCDH	High address provided by a TCD pseudo-operation.

- Undefined Symbols - Lists all referenced external symbols that have not been defined; it is not printed if the MAPS option is selected or if there are no undefined external symbols. If an undefined symbol is referenced by a transient code group other than 0, the transient code group number is printed in brackets to the right of the area's ending address.

NOTE: The map information listed below is printed only if MAPF is specified.

- Defined Symbols - Lists all defined, external symbols and their corresponding values. If the symbol was defined in a transient code group other than 0, the transient code group number is printed in brackets to the right of the symbol name or symbol value.
- Base Areas - Consists of one line per desectorization area, listing the desectorization area's starting address, next available location, and ending address. Primary desectorization areas are flagged with an asterisk. If a secondary desectorization area is in a transient code group other than 0, the transient code group number is printed in brackets to the right of the area's ending address.
- Gaps - Lists the starting and ending addresses of all gaps. If the gap is in a transient code group other than 0, the transient code group number is printed in brackets to the right of the area's ending address. If the starting address is higher than the ending address, the gap is full.
- COMMON Blocks - Lists all COMMON block names and their respective low addresses.

Default: None

Figure 3-1 illustrates a complete sample link map.

```
00  ZLE      ! 00MAPF
ZLE      LINK TEXT IDENTIFICATION
ZLE
ZLE    STATE
ZLE
ZLE        START   01024
ZLE        LOW     01024
ZLE        HIGH    02527
ZLE        ADDR    03000
ZLE        GAPH    04003
ZLE        BASE    01006-01010-01023            } MAPS
ZLE        COMM    36604
ZLE        CLOW    36604                                    } MAP
ZLE        CHIGH   36750
ZLE        CTCG    00000
ZLE        TCDL    01305
ZLE        TCDH    02514
ZLE
ZLE    UNDEFINED SYMBOLS
ZLE
ZLE        TCM
ZLE        TCMI                                                     } MAPF
ZLE
ZLE    DEFINED SYMBOLS
ZLE
ZLE        PROGRM 01024
ZLE        OVRLAY 01305
ZLE
ZLE    BASE AREAS
ZLE
ZLE        02511-02511-02514 [00002]
ZLE        02214-02214-02216 [00001]
ZLE       *01006-01010-01023
ZLE
ZLE    GAPS
ZLE
ZLE        02530-02777
ZLE
ZLE    COMMON BLOCKS
ZLE
ZLE        ARRAY 36604
00  ZLE    !
```

Figure 3-1. Sample Link Map

Modular Origin Command (MORG)

Format: MORG=num

Function: Causes the linkage editor to advance the link address to the next location divisible by num, which must be from 2^1 through 2^9; i.e., 2 through 512. Skipped locations are recorded as a gap.

Default: None

Example:

```
00  ZLE      ! 00MAPS

ZLE
ZLE     STATE
ZLE
ZLE          ADDR   01000
ZLE          COMM   40300

00  ZLE      ! 00MORG='400,MAPS

ZLE
ZLE     STATE
ZLE
ZLE          ADDR   01000
ZLE          COMM   40300
00  ZLE      ! 00ADDR='1001,MORG=4,MAPS

ZLE
ZLE     STATE
ZLE
ZLE          ADDR   01004
ZLE          COMM   40300
00  ZLE      ! 00MORG='400,MAPS

ZLE
ZLE     STATE
ZLE
ZLE          ADDR   01400
ZLE          COMM   40300
00  ZLE      ! 00MORG='1000,MAPF

ZLE
ZLE     STATE
ZLE
ZLE          ADDR   02000
ZLE          COMM   40300
ZLE
ZLE     GAPS
ZLE
ZLE          01400-01777
ZLE          01004-01377
ZLE          01002-01003
00  ZLE      !
```

NCULL Mode Command

Format:	NCULL
Function:	Puts the linkage editor into NCULL mode. In this mode, a symbol defined by a module or by a DEF command is recorded in the symbol table if it was not previously defined in the current transient code group. The CULL command terminates NCULL mode.
Default:	NCULL

Normal Mode Command (NORM)

Format:	NORM
Function:	Puts the linkage editor into NORM mode. In this mode, a LINK command causes modules to be read from the binary input stream until an end of file is encountered. (See Section I, "Module Selection and Placement," to determine if each module is linked or ignored.) The linkage editor then reads the command input stream for further commands. The STEP command terminates NORM mode.
Default:	NORM

QUIT Command

Format:	QUIT
Function:	Terminates the linkage editor activity; used when linkage editor processing is complete.
Default:	None

SKIP
STEP

SKIP Command

Formats: SKIP
SKIP=num

Function: When the SKIP command is given, one module is read and ignored from the BI stream. When SKIP=num is given, the designated number of modules (num) are read and ignored. If an end of file is encountered before the designated number of files are read, the process terminates. If num is 0, no operation is performed. Unless num is 0, the SKIP command terminates FORCE mode.

Default: None

STEP Mode Command

Format: STEP

Function: Puts the linkage editor into STEP mode. In this mode, a LINK command causes modules to be read from the binary input stream until a module is linked or end of file is reached. (See "Module Selection and Placement" in Section I to determine whether each module is linked or ignored.) The linkage editor then reads the command input stream for further commands. Used in conjunction with the SKIP command, STEP mode allows modules in an object library to be linked individually. The NORM command terminates STEP mode.

Default: NORM

Stream Assignment Commands (BI,BO,CI,MO,OO)

Formats:
$$\left\{\begin{matrix}\text{BI}\\\text{BO}\\\text{CI}\\\text{MO}\\\text{OO}\end{matrix}\right\}\text{=file name}$$

$$\left\{\begin{matrix}\text{CI}\\\text{MO}\end{matrix}\right\}$$

Function: Assigns I/O stream (designated by two-character mnemonic) to an OS/700 file. File names are from one to six alphanumeric characters; the first character must be alphabetic. If the CI stream is not assigned to a disk file, commands must be entered through the console. If the MO stream is not assigned to a disk file, memory maps are printed at the console.

NOTES:

1. Each input file must be in the OS/700 default library for files (FIO).
2. Each output file is created in FIO; there must not already be a file in FIO with the same name.

BI - Object text input stream (binary). Object text can be either the result of a DAP-700 assembly or a symbol table file previously generated by the linkage editor using the SYMT command. (See "Symbol Table File Generation Command (SYMT)" below.) An object text file produced by the DAP-700 Macro Assembler consists of one or more object modules. An object text file containing more than one object module is called an object text library. The object text input stream must be reassigned every time modules in a new object file are to linked.

BO - Link text output stream (binary). The BO stream is assigned if a link text image of the program being linked is desired. The stream must be assigned before any modules are linked, and it cannot be reassigned during the linking process.

CI - Command input stream (ASCII). The linkage editor usually receives command input from the console, but command input may also be read from an OS/700 file assigned to the CI stream. When an end of file is reached, or if CI is specified without a file name, the CI stream is reassigned to the console. The stream can be reassigned at any time during the linking process.

MO - Map output stream (ASCII). A map provides information regarding the layout of a linked program. The linkage editor usually prints maps on the console, but maps may be written into an OS/700 file using the MO stream assignment command. If MO is specified without a file name, maps are printed on the console. If a file name is specified, maps are written into an OS/700 file.

The MO stream can be reassigned at any time during the linking process. Any number of maps, representing different stages of the linking process, can be written into a single file using the MAPS, MAP, and MAPF commands without reassigning the stream. Individual files can be obtained by reassigning the stream before each MAPS, MAP or MAPF command.

OO - Object text output stream (binary). By assigning an OO stream and using the SYMT command, an object text file is produced which preserves the values of symbols defined during a linking process. These symbols can be used in subsequent linking processes by reading the object text file from the binary input stream.

The OO stream can be assigned at any time during a linking process. A symbol table file is not produced until the SYMT command is executed. If another symbol table file is desired, the stream must be reassigned.

Default:	BI - None
	BO - None
	CI - Console
	MO - Console
	OO - None

Symbol Table File Generation Command (SYMT)

Format:	SYMT

Function: Directs the linkage editor to write, in the object output stream (OO), a module consisting of the following object text blocks:

- Identification block
- External symbol definition blocks for all currently defined symbols
- END block

If a second symbol table file is desired, the OO stream must be reassigned. (See the OS/700 DAP-700 Macro Assembler manual, Appendix D.) This command saves information about a linked program so other programs can reference symbols it defines. When linking a program that requires this information, link the symbol table file.

Default: None

TOTAL Mode Command

Format: TOTAL

Function: Puts the linkage editor into TOTAL mode, in which all modules read by a LINK command are linked regardless of whether they define previously referenced, undefined, external symbols. TOTAL mode remains in effect until terminated by a LIB command.

Default: LIB

Establish Transient Code Group Command (TCG)

Format: TCG=num

Function: Establishes the next linked module as a transient code group. Num can be from 0 to 126. This command does not affect the link address.

Default: TCG=0

ERROR MESSAGES

When the linkage editor detects an error, it issues a message in the following format, and then requests command input from the console.

$$\text{ERROR xx}\begin{Bmatrix}\text{FILE=ffffff [CODE=cccccc]}\\ \text{[LINE=111111]}\end{Bmatrix}$$

xx -	Two-letter error mnemonic.
ffffff -	File name; printed for file-related errors (if xx is BL, DF, IO, RS, TD, TI, UC, or UF).
cccccc -	OS/700 executive macro call error return code (decimal); printed if xx is IO, denoting I/O error. (See the OS/700 Operators Guide.)
111111 -	Line number (decimal) of the command input file being processed when the error occurred; printed only if command input file is being used, and xx is BO, CE, IA, MO, NT, TO, or US.

Table 3-1 lists and explains error messages, and suggests remedies.

Table 3-1. Linkage Editor Error Messages

Error Mnemonic	Type of Error	Meaning	Operator Action
BL	Block error	Object text has illegal format.	Do not continue linking. Verify that file is an object file (i.e., file was produced by assembler), or regenerate object file.
BO	Base sector overflow	Primary desectorization area has been fully utilized. The linkage editor cannot desectorize an instruction because there is no desectorization area for the indirect address word generated.	Do not continue linking. Provide additional primary or secondary desectorization area, or rearrange order of linking to minimize cross-sector references.
CE	Command error	Syntax error or unrecognized command.	Retype command on console. If in CI stream, correct contents of file.
DF	Duplicate file	File assigned to the MO, BO, or OO stream already exists.	Delete old file or assign a different file to the stream.
IA	Illegal assignment	File is assigned to the BO stream after a LINK command.	Restart linking process and assign the BO stream before entering a LINK command.
IO	Input/output	Error detected by OS/700 during file handling.	Do not continue linking. Consult <u>OS/700 Operators Guide</u> for I/O error codes and remedies.

Table 3-1 (cont). Linkage Editor Error Messages

Error Mnemonic	Type of Error	Meaning	Operator Action
MO	Memory overflow	Program code overwrites COMMON storage; i.e., HIGH exceeds COMM. (See Figure 3-1.)	Do not continue linking. Reorganize linking process so any unused areas are used, or decrease memory requirements of one area.
NT	No transient code definition (TCD)	TCD assembly pseudo-operation was not processed before the first transient code group (TCG) command or assembly pseudo-operation (where TCG≠O) was encountered.	Do not continue linking. Be sure module with TCD pseudo-operation is linked before TCG command or pseudo-operation is encountered.
RS	Record sequence	Object text records are improperly sequenced.	Do not continue linking. Verify that file is an object file, or regenerate object file.
TD	TCD error	TCD assembly pseudo-operation occurred in object text after code-generating object text.	Do not continue linking. Verify that module containing TCD pseudo-operation is linked first.
TI	Illegal object text type	Object text contains illegal data.	Do not continue linking. Verify that file is an object file, or regenerate object file.

Table 3-1 (cont). Linkage Editor Error Messages

Error Mnemonic	Type of Error	Meaning	Operator Action
TO	Symbol table overflow	Linkage editor's symbol table is fully utilized.	Do not continue linking. Use fewer symbols; or Rearrange linking process to minimize the number of unresolved references to external symbols; or Generate a linkage editor activity that has a larger symbol table.
UC	Undefined COMMON block	Reference to an undefined COMMON block name. Data read through the BI stream does not conform to the definition of object text (see the OS/700 DAP-700 Macro Assembler manual).	Do not continue linking. Verify that file is an object file, or regenerate object file before referencing it in a linkage editor command.
UF	Undefined file	Undefined input file is assigned to the BI or CI stream. Attempt to read nonexistent file.	Be sure to generate a file before using it. Verify file name entered through the console or CI stream.
US	Undefined symbol	Command parameter contains an undefined symbol.	Reorganize linking procedure, defining symbols before referencing them.

APPENDIX A
LINK-700 LINK TEXT

Link text is the memory-image representation of a linked program, with provisions for text identification. It is the interface between the linkage editor, System 700 link text loaders, the OS/700 online utility Load Activity (LA) command, and other System 700 programs.

INTERNAL DATA FORMAT

Link text comprises five types of binary records (blocks); the first word of each contains the block type and block length. Maximum block length is 54 words. The blocks occur in the following order, except that data blocks and transient code group blocks can be intermixed.

1. Identification block (optional)
2. Transient code definition block (optional)
3. Data blocks
4. Transient code group blocks (optional)
5. End block

Identification Block - Block Type 2

The first word of an identification block contains a 2 in the high-order byte as the block identifier, and the word count (n) in the low-order byte. The next n-1 words contain text identification, packed two characters per word. (See Figure A-1.)

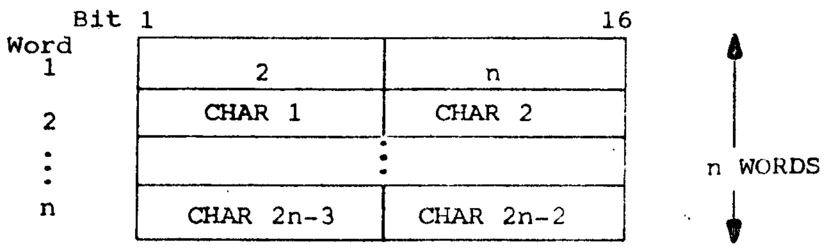
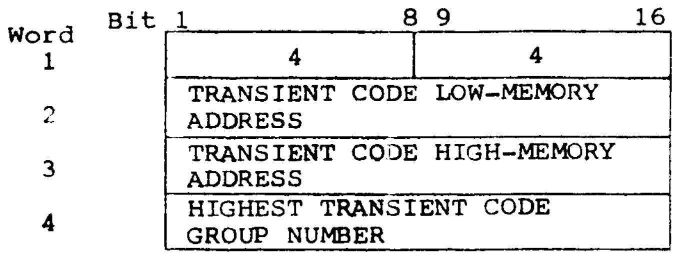


Figure A-1. Identification Block

Transient Code Definition Block - Block Type 4

The first word of a transient code definition block contains a 4 in the high-order byte as a block identifier, and the word count (always 4) in the lower byte. The second word contains the low-memory address of transient code groups in the link text. The third word contains the high-memory address of the transient code groups. The fourth word contains the highest transient code group number. (See Figure A-2.)



Word
Bit 1
8 9
16
1
4
4
2
TRANSIENT CODE LOW-MEMORY ADDRESS
3
TRANSIENT CODE HIGH-MEMORY ADDRESS
4
HIGHEST TRANSIENT CODE GROUP NUMBER

Figure A-2. Transient Code Definition Block

<u>Data Block - Block Type 0</u>

The first word of a data block contains a 0 in the high-order byte as the block identifier, and the word count (n) in the lower-order byte. One or more data groups follow, formatted as shown below:

Word 1 - Data group origin

Word 2 - Bit 1 - Set if repeated constant
Bits 2-16 - Number of data words to be stored (j)

If bit 1 of word 2 is 1, the next word is stored in j consecutive locations, starting at the group origin. If it is 0, the next j words in the group are stored consecutively starting at the group origin. (See Figure A-3.)

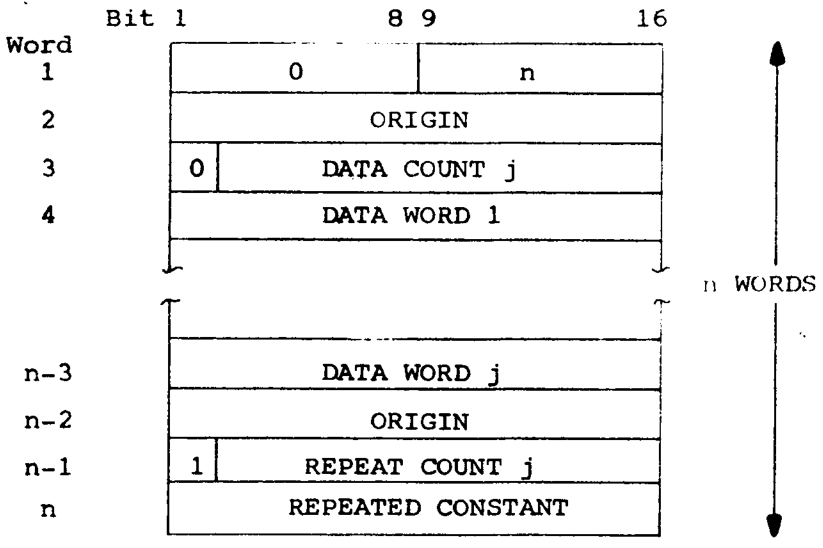


Figure A-3. Data Block

Memory gap tables are included in data blocks, along with data derived from object text, and are indistinguishable from such data. When loaded into memory, a gap table is 2n consecutive words, where the first n-1 pairs of words represent n-1 free-memory areas. The first word of each pair is the low-memory address of the area; the second word is the high-memory address. The last two words of the gap table contain the value '177777 and mark the end of the table.

Transient Code Group Block - Block Type 5

The first word of a transient code group block contains a 5 in the high-order byte as a block identifier, and the word count (always 2) in the low-order byte. The second word contains the number representing the transient code group to which subsequent data belongs. Data preceding the first transient code group block belongs to transient code group 0. (See Figure A-4.)

Word	Bit 1 … 8	9 … 16
1	5	2
2	TRANSIENT CODE GROUP NUMBER	

Figure A-4. Transient Code Group Block

End Block - Block Type 1

The first word of an end block contains a 1 in the high-order byte as the block identifier, and the word count (always 2) in the low-order byte. The second word contains the starting address of the program. (See Figure A-5.)

Word	Bit 1 … 8	9 … 16
1	1	2
2	START ADDRESS	

Figure A-5. End Block

APPENDIX B
SAMPLE PROGRAM AND MAP

Figures B-1 and B-2 illustrate, respectively, a sample program and the resulting map.

```
                              * SAMPLE PROGRAM TO ILLUSTRATE MAP

0001                          * SAMPLE PROGRAM TO ILLUSTRATE MAP
0002                          *
0003                          *
0004                                 REL                     RELOCATABLE CODE
0005                                 EXD                     EXTENDED MODE DESECTORIZATION
0006                                 TCD     OVRLAY,OVEND,10 DEFINE TRANSIENT CODE GROUPS
0007                                 ENT     PROGRM,INIT     PROGRAM STARTING ADDRESS
0008                                 ENT     OVRLAY          START OF OVERLAY AREA
0009                          *
0010                                 SETB    *+6,14          PRIMARY DESECTORIZATION AREA
0011                                 ORG     *+20            SKIP AROUND BASE AREA
0012                          *
0013 00024     0 15 01515     INIT   STX     PTCB            SAVE ADDRESS OF TCB
0014 00025     0 10 00000E           CALL    TCMI            CALL TRANSIENT CODE MANAGER INITIALIZER
0015 00026     0 001515              DAC     ìPTCB           ADDRESS OF WORD CONTAINING ADDRESS OF TCB
0016 00027     0 001516              DAC     TCGTBL          ADDRESS OF TABLE FOR TRANSIENT CODE GROUPS
0017 00030     0 000305              DAC     OVRLAY          START ADDRESS OF OVERLAY AREA
0018 00031     0 001514              DAC     OVEND-1         END ADDRESS OF OVERLAY AREA
0019 00032     000012                DEC     10              NUMBER OF TRANSIENT CODE GROUPS
0020 00033     0 01 00206            JMP     TCMIER          TCMI ERROR OCCURRED
0021                          *
0022 00034     0 02 01530            LDA     =1              LOAD TRANSIENT CODE GROUP 1
0023 00035     0 10 00000E           CALL    TCM
0024 00036     0 01 00223            JMP     TCMERR          TRANSIENT CODE MANAGER ERROR
0025 00037     0 10 00305            JST     SUBRT1          CALL SUBROUTINE IN TRANSIENT CODE GROUP 1

                                     .
                                     .
                                     .

0034                          *
0035                          *      TRANSIENT CODE GROUP 1
0036                          *
0037           000305         OVRLAY EQU     *
0038                                 TCG     1
0039 00305     0 000000  A    SUBRT1 DAC     **
                                     .
                                     .
                                     .

0044                          *
0045 01213    *0 01 00305            JMP*    SUBRT1          RETURN TO CALLER
0046 01214                           BSD     3               DESECTORIZATION AREA
0047                          *
0048           001217         OVEND  SET     *
0049                          *
0050                          *      TRANSIENT CODE GROUP 2
0051                          *
0052                                 ORG     OVRLAY
0053                                 TCG     2
0054 00305     0 000000  A    SUBRT2 DAC     **
                                     .
                                     .
                                     .

0059                          *



                              * SAMPLE PROGRAM TO ILLUSTRATE MAP

0060 01510    *0 01 00305            JMP*    SUBRT2          RETURN TO CALLER
0061 01511                           BSD     4               DESECTORIZATION AREA
0062                                 IFP     *-OVEND         IF FURTHER THAN PREVIOUS END OF OVERLAY AREA
0063           001515         OVEND  SET     *               UPDATE END OF OVERLAY AREA
0064                                 ENDC
0065                          *
0066                          *      RETURN TO ROOT
0067                          *
0068                                 ORG     OVEND
0069                                 TCG     0
0070                          *
0071                          *      DATA AREA
0072                          *
0073 01515     0 000000  A    PTCB   DAC     **              ADDRESS OF TCB
0074 01516     000000         TCGTBL BSZ     10              TCG TABLE
0075                          ARRAY  COMM    100             COMMON ARRAY INITIALIZED TO ZERO
0076                                 ORG     ARRAY
0077 00000C    000000                BSZ     100
0078                                 MAIN
0079 01530     000001                END     INIT

INIT   000024    OVEND  001515    OVRLAY 000305    PTCB   001515
SUBRT1 000305    SUBRT2 000305    TCGTBL 001516    TCMERR 000223
TCMIER 000206

COMMON

ARRAY  000144

0000 WARNING OR ERROR FLAGS
DAP-700   REV. F    74-05-09
```

Figure B-1. Sample Program

```
$SA ZLE

ZLE      LE-700  REV. D  74/10/10
00  ZLE      ! 00IDNT:LINKED 700 DEMONSTRATION    Provides link text identi-
                                                  fication.

00  ZLE      ! 00BO=TDEMO                         Assigns file to binary out-
                                                  put stream.

00  ZLE      ! 00COMM='4000                       Sets COMMON address.

00  ZLE      ! 00BI=QDEMO,LINK                    Links main program.

00  ZLE      ! 00BI=QTCM,LINK                     Links subroutine (Transient
                                                  Code Manager).

00  ZLE      ! 00ADDR=OVRLAY                      Sets link address to begin-
                                                  ning of program's overlay
                                                  area.

00  ZLE      ! 00BI=QTCMI,LINK                    Links subroutine (Transient
                                                  Code Manager Initializer).

00  ZLE      ! 00MAPF,QUIT                        Obtains map; then terminates.
ZLE      LINKED 700 DEMONSTRATION
ZLE
ZLE      STATE
ZLE
ZLE           START  01024
ZLE           LOW    01024
ZLE           HIGH   02657
ZLE           ADDR   01502
ZLE           BASE   01006-01011-01023
ZLE           COMM   03634
ZLE           CLOW   03634
ZLE           CHIGH  03777
ZLE           CTCG   00000
ZLE           TCDL   01305
ZLE           TCDH   02514
ZLE
ZLE      DEFINED SYMBOLS
ZLE
ZLE           PROGRM 01024
ZLE           OVRLAY 01305
ZLE           TCMI   01306
ZLE           TCM    02532
ZLE           TCMGL  02641
ZLE           TCMGH  02642
ZLE           TCGBUF 02645
ZLE           TCMSEG 02646
ZLE           ZALINK 02647
ZLE
ZLE      BASE AREAS
ZLE
ZLE           02511-02511-02514  [00002]
ZLE           02214-02214-02216  [00001]
ZLE          *01006-01011-01023
ZLE
ZLE      COMMON BLOCKS
ZLE
ZLE           ARRAY  03634
ZLE      END OF JOB
```

Figure B-2. Map Resulting From Sample Program

APPENDIX C
SUMMARY OF LINKAGE EDITOR COMMANDS

Command	Syntax	Description
Address	ADDR=num	Sets the linkage editor link address.
BASE	BASE=num1<num2 or BASE=num1	Establishes a primary desectorization area.
Assign object text binary input stream	BI=file name	Assigns object text binary input stream to OS/700 file.
Assign link text binary output stream	BO=file name	Assigns link text binary output stream to OS/700 file.
BSD	BSD=num	Generates a BSD block at current location.
Assign ASCII command input stream	CI or CI=file name	Assigns ASCII command input stream to console or to OS/700 file.
Identification with copyright	CIDNT	Places character string and Honeywell copyright in text identification blocks.
COMMON address	COMM=num	Defines top of the FORTRAN COMMON.
CULL mode	CULL	Enters CULL mode, in which a symbol defined by a module or by a DEF command is recorded in the symbol table if it was previously referenced but not yet defined.
Definition of symbol	DEF:symbol=num	Defines the numeric value (num) of symbol.
Reverse condition of command execution	ELSE	Reverses effect of preceding IFZ or IFN command; e.g., if command execution was enabled, it becomes inhibited.
End conditional command execution	ENDC	Removes condition imposed upon command execution by preceding IFZ or IFN command.
Enter extended desectorization mode	EXD	Enters extended desectorization mode, which is used when the program being linked will execute using extended addressing mode.

Command	Syntax	Description
Finish	FIN	Completes link text generation but does not reinitialize.
FORCE	FORCE	Forces linking of the next module read.
Gap table	GAPT or GAPT=num	Produces a link text table of memory gaps in the binary output stream.
Gap base	GBASE or GBASE=num	Ignores gaps below current link address or num when linking SECT mode object text.
Identification	IDNT:character string	Places character string in text identification block.
Conditional command execution	$\left\{\begin{matrix}\text{IFN}\\\text{IFZ}\end{matrix}\right\}$:num	Enables execution of subsequent commands if num is not 0 (IFN) or is 0 (IFZ); if condition is not met, subsequent commands are ignored.
Initialize	INIT	Completes link text generation and reinitializes.
Library mode	LIB	Enters LIB mode, which links a module if certain conditions are met. See "Library Mode Command (LIB)" in Section III.
LINK	LINK	Initiates the reading and processing of object text.
Leave extended desectorization mode	LXD	Enters normal desectorization mode, which is used when the program being linked will execute using normal addressing mode.
MAP	MAP or MAPS or MAPF	Produces a link map.
Assign ASCII map output stream	MO or MO=file name	Assigns ASCII map output stream to console or to OS/700 file.
Modular origin	MORG=num	Advances link address to next location divisible by num.
NCULL mode	NCULL	Enters NCULL mode, in which a symbol defined by a module or by a DEF command is recorded in the symbol table if it was not previously defined in the current transient code group.
Normal mode	NORM	Enters NORM mode, in which modules are read from the binary input stream until an end of file is encountered.

Command	Syntax	Description
Assign binary object text output stream	OO=file name	Assigns binary object text output stream to OS/700 file.
QUIT	QUIT	Terminates linkage editor activity.
SKIP	SKIP or SKIP=num	Reads and ignores one or num modules.
STEP mode	STEP	Enters STEP mode,in which a LINK command causes object text to be read until a module is linked or an end of file is encountered.
Symbol table	SYMT	Transfers the symbol table as a module on the object text output stream.
Establish transient code group	TCG=num	Establishes the next modules as a transient code group.
TOTAL mode	TOTAL	Enters TOTAL mode, which links all modules read.

INDEX

INDEX

INDEX

Honeywell Bull

Technical Publications Remarks Form* *(please print)*

Title:

SYSTEM 700 OS/700
LINK-700 LINKAGE EDITOR

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

Dated: DECEMBER 1974

Errors in publication:

Suggestions for improvement to publication:

from : **Name**

Company

Title

Address

* Your comments will be promptly investigated by appropriate technical personnel, action will be taken as required, and you will receive a written reply. If you do not require a written reply, please check here : ☐

CUT ALONG LINE

Please hand this technical publication remark form to your Honeywell Bull representative,

or mail to :

Honeywell Bull

Marketing Communications
Documentation/Publications

94, avenue Gambetta
75960 PARIS CEDEX 20 - FRANCE

Honeywell Bull

HONEYWELL INFORMATION SYSTEMS