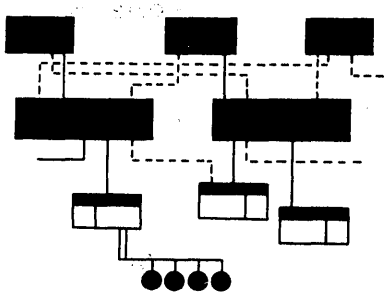
GE-625/635 General Loader

Information Systems

Information Systems Equipment



GENERAL ELECTRIC

CPB-1008E

GENERAL ELECTRIC INFORMATION SYSTEMS DIVISION COMPUTER EQUIPMENT DEPARTMENT	GE-600 SERIES TECHNICAL INFORMATION BULLETIN	DATE Oct. 1968 NO. 600-222
SUBJECT: Changes to GE-625/635 General Loader Reference Manual		REF. CPB-1008E

This TIB includes features implemented in GECOS-III System Development Letter 1.

Replace old pages in GE-625/635 General Loader reference manual, CPB-1008E, with attached new pages as follows:

Old	New
15, 16	14.1, 15, 16
17-20	17, 18, 18.1, 19, 20
23-27	23-27
35, 36	35, 36
39, 40	39, 40
53, 54	53, 54

Vertical bars in the margins of these new pages indicate changes or additions to the existing text. This new information and any changes it makes in the index will be included in the next edition of the manual.

Place this sheet in the front of your manual to show that the contents of this TIB have been incorporated.

This is at present the only TIB applying to CPB-1008E.

CPB-1008E

GE-625/635 General Loader

REFERENCE MANUAL

PROGRAM NUMBER

CD600B1.000

September 1964
Rev. May 1968

INFORMATION SYSTEMS

GENERAL ELECTRIC

Preface

The GE-625/635 General Loader (GELOAD) is an essential element of the integrated programming system. GELOAD's most extensive use is in the loading of relocatable subprograms from various sources and tying them together such that the subprograms execute as a whole. GELOAD's recognition of debug cards causes GELOAD to set up tables and additional linkages for printouts of intermediate results during execution in relation to the arguments on these cards. Other documents which will be of interest to programmers using GELOAD are:

GE-625/635 Programming Reference Manual, CPB-1004
GE-625/635 File and Record Control, CPB-1003
GE-625/635 Comprehensive Operating Supervisor, CPB-1195

This manual has been revised to the extent that the majority of pages have been affected. The manual is issued, therefore, as a new document, without change bars to indicate the changes.

This manual was produced using the General Electric Remote Access Editing System (RAES). RAES is a time-shared disc-resident storage and retrieval system with text-editing and manuscript formatting capabilities. The contents of the manual were entered into RAES from a remote terminal keyboard, edited using the system editing language, and formatted by RAES on reproduction masters.

The index was produced using a computer-assisted remote access indexing system. This system produces an index using source strings delimited at manuscript input time.

Suggestions and criticisms relative to form, content, purpose, or use of this manual are invited. Comments may be sent on the Document Review Sheet in the back of this manual or may be addressed directly to Documentation Standards and Publications, C-78, Processor Equipment Department, General Electric Company, 13430 North Black Canyon Highway, Phoenix, Arizona 85029.

(1.5M 5-68)

Contents

Page

Illustrations

1. Introduction

The GE-625/635 General Loader (GELOAD), one program of the GE-625/635 software system, is a general purpose loader whose primary function is to initiate an execution activity. Although GELOAD will load absolute as well as relocatable subprograms, its most extensive usage is in the loading of relocatable subprograms from various sources and tying them together in a rational fashion such that the subprograms execute as a whole.

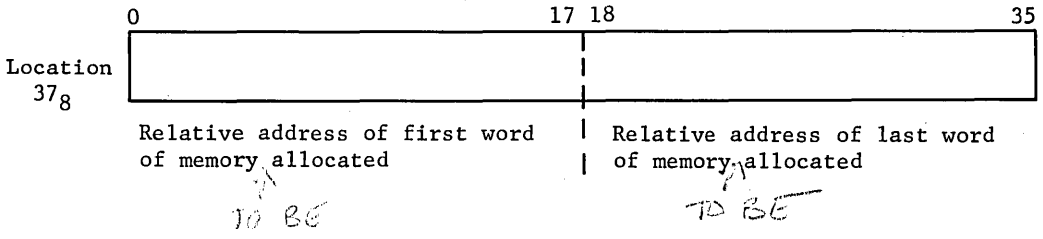
A relocatable subprogram is a binary object deck usually having a distinct but dependent function. Included within the subprogram are the necessary interfaces to link it to other subprograms which complement its function to the extent of achieving a desired goal. GELOAD completes these linkages, reserves storage for required data regions, calls in additional subprograms from established libraries, and, if desired, segments the program into reloadable overlays.

Additional features, included optionally within GELOAD, are the creation and linkage of file control blocks which may be required for execution of the user program. GELOAD, in conjunction with the FORTRAN I/O library, provides an extensive debugging aid for the user. The recognition by GELOAD of debug cards causes GELOAD to set up tables and additional linkages for printouts of intermediate results during execution in relation to the arguments on these cards.

The manuals listed under "References" contain supplementary information concerning the use of GELOAD and should be used in conjunction with this manual.

2. General Procedures for Loading GELOAD

The GE-625/635 General Loader (GELOAD) is called by the Comprehensive Operating Supervisor (GECOS) whenever a $ EXECUTE control card is encountered. GELOAD will load all object decks provided by the user or generated by the compilers and assemblers. Before loading is initiated, GECOS allocates slave memory requirements plus a required amount for GELOAD (unless slave memory or a portion of it may be shared with memory requirements for GELOAD). The relative address of the lower and upper limits of memory allocated to the object programs are placed in location 37_8 of the slave program prefix.



The addresses of the first and last words are not absolute, but are relative to the base address as set for GELOAD.

NORMAL LOADING

The normal loading procedure, from the high end of activity-allocated memory, is as follows:

1. Assign and reset contents of the amount of memory indicated by the program break on the object deck preface card.
2. Load the first subprogram in order of increasing addresses into allocated memory.
3. Assign below the first subprogram any Labeled Common regions requested by the first subprogram.
4. Assign and load the next subprogram below the Labeled Common regions (if present), followed by its Labeled Common regions (if present).
5. Continue this procedure until all of the subprograms have been loaded.

The largest amount of Blank Common that may have been requested by any of the subprograms loaded will be assigned to a region at the low end of the allocated memory immediately following the 64-word slave program prefix. Figure 1 illustrates the memory layout for high-loaded relocatable subprograms.

LOW LOADING

When a $ LOWLOAD control card is encountered by GELOAD, assignment and loading begins from the low end of activity-allocated memory following the 64-word slave program prefix. Succeeding subprograms are assigned to ascending memory locations. The $ LOWLOAD control card format is discussed in Chapter 3 of this manual. Figure 2 illustrates memory layout for low-loaded relocatable subprograms.

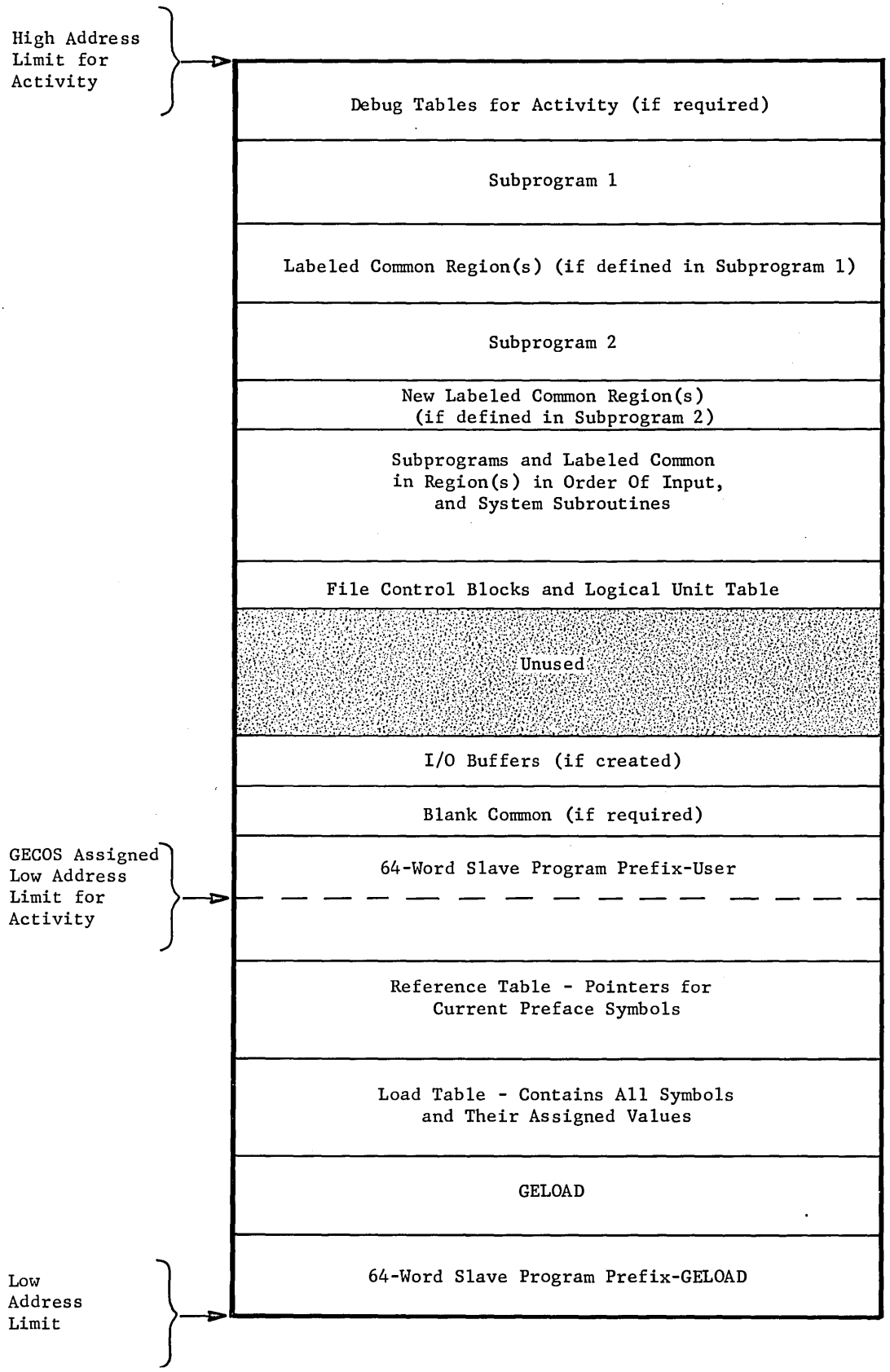


Figure 1. High-Loaded Memory Layout for Relocatable Subprograms

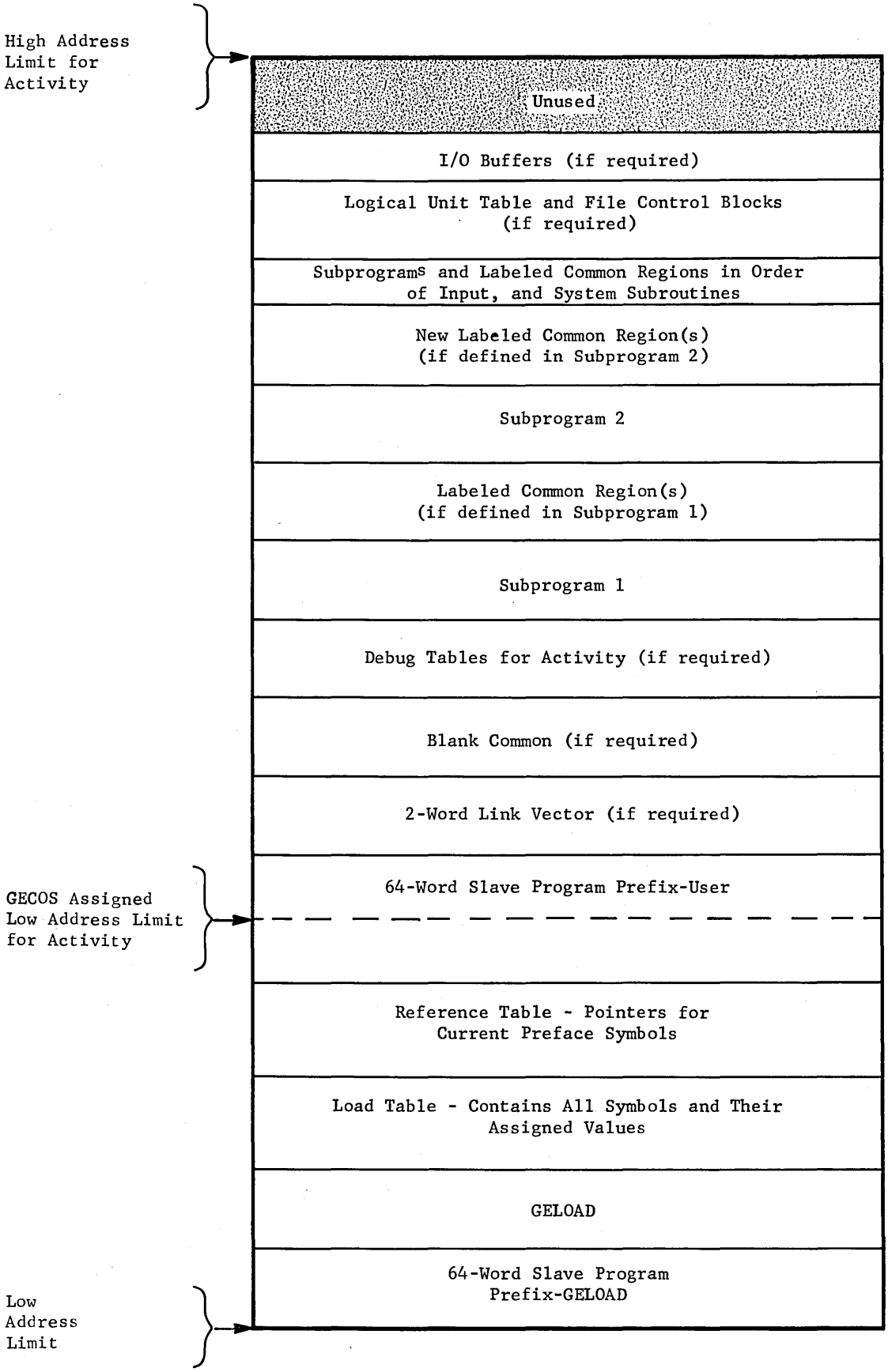


Figure 2. Low-Loaded Memory Layout for Relocatable Subprograms

3. Input to GELOAD

Input to GELOAD consists of control cards and object decks (in either relocatable or absolute form) such as those produced as output from the GE-625/635 GMAP Assembler. Each object deck is itself enclosed between two control cards, the $ OBJECT and the $ DKEND cards.

RELOCATABLE OBJECT DECK DESCRIPTION

Relocatable object decks may be further defined as being made up of two types of cards, those containing preface information and those containing text.

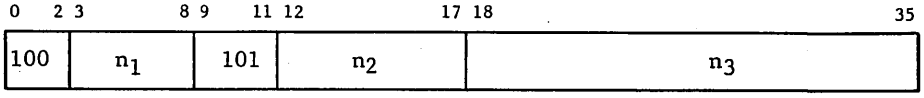
Preface Cards

The preface card(s) provides GELOAD with all pertinent size and linkage information, such as the following:

1. The length of the subprogram text region.
2. The length of Blank Common required, if any.
3. The total number of SYMDEF, SYMREF, and Labeled Common symbols, as well as the symbols themselves.
4. The relative entry value or the region length for each respective symbol.
5. In the special case of a preface card read from a random subroutine library containing a directory, the SYMREF entries contain the address of the record on the random device which contains the subprogram satisfying that particular SYMREF.

The specific content of each word on the preface card is as follows:

Word 1:



Bits 0-2 and 9-11 define the card as a binary preface card.

Field n_1 (V count bits), describes the number of bits required to express the total number of Labeled Common and SYMREF symbols referenced within the subprogram. A formula used to calculate this value is $n_1 = \log_2 (N + 1)$, where N is the count of the symbols.

Field n_2 is the count of words on the preface card beginning with word 3, which includes word 3 but not the checksum word.

Field n_3 is the length of the subprogram text.

Word 2: Checksum of columns 1-3, 7-72.

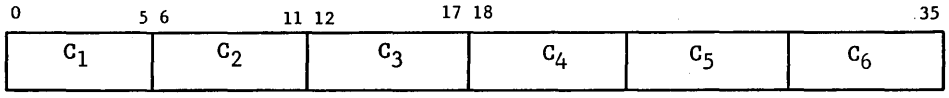
Word 3: Bits 0-17 define the length of Blank Common required by this subprogram.

Bit 18, if set, indicates loading of this subprogram should start at the next available address which is a multiple of eight.

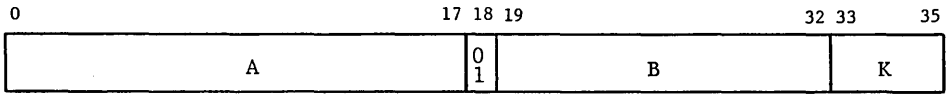
Bits 19-35 indicate twice the number of SYMDEF, SYMREF, and Labeled Common symbols contained on the preface card(s).

Word 4-23: Pairs of entries.

Word n



Word n+1



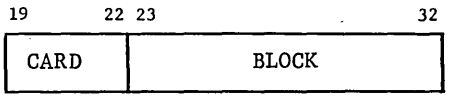
The first word of each pair is a symbol. The second word describes that symbol completely as to its usage by the subprogram being loaded. The value of K (bits 33-35) defines the type of symbol and thus the following implied meaning of the other fields involved:

K = 0 The symbol is a primary SYMDEF. Field A is a value equal to the position of the symbol relative to the beginning of the subprogram. Bit 18 and field B are not used.

K = 1 The symbol is a secondary SYMDEF. Usage of the remaining fields of word n+1 is the same as for a primary SYMDEF.

K = 3 The symbol is a constant which has special usage by subprograms generated by certain compilers. Field A contains the value of that constant. The remaining fields of word n+1 are not used.

K = 5 The symbol is a SYMREF. Field A must be zero and bit 18 is not used. When the preface card containing this pair has been read from a library file containing a directory on a random device, field B may contain an address pointing to the deck which satisfies this SYMREF. The use of field B is contingent on whether the satisfying deck appears anywhere on this particular library file. If used, the format of this address (bits 19-32) is:



CARD is the relative card number within the particular block.

BLOCK is the relative block address within the random file.

K = 6 The symbol is the name of a Labeled Common region. Field A contains the length of the region which may not be equal to zero and if bit 18 is on, indicates that the region is to be assigned, beginning at the next available address which is a multiple of eight.

K = 7 The symbol is .SYMT., a special form of Labeled Common. This region contains data tables such as those generated by the FORTRAN IV compiler (refer to GE-625/635 FORTRAN IV, CPB-1006) for use by the debug feature of GELOAD. Its data will be loaded at the next available area beyond that already designated for use by the text and Labeled Common regions associated with this subprogram. This data is temporary and will be overlayed by the next subprogram to be loaded by GELOAD. The fields in word n+1 conform to the rules of Labeled Common.

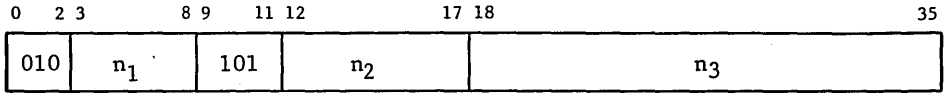
If the count in word 3 is greater than 20, then additional preface cards are required. On each additional preface card, word 3 will be repeated unchanged.

Relocatable Text Cards

The text cards carry information, assembled or compiled, required to execute the desired function. This information is formatted in such a way as to give GELOAD the necessary handles to form a useful block of data or executable instructions in memory.

The specific content of a relocatable text card is as follows:

Word 1:



Bits 0-2 and 9-11 define the card as a column binary relocatable text card.

Field n_1 indicates the symbol (obtained from the preface card) relative to which this text is to be loaded. If n_1 is zero, the text is loaded relative to the primary program region. When $1 \leq n_1 \leq$ number of Labeled Common symbols on the preface, n_1 then signifies the symbolic addresses relative to which the text is to be relocated.

Field n_2 is a count of the number of instructions associated with this control word. The count does not include the three words of relocation data and is not necessarily a count of the words on the card.

Field n_3 is the relative loading address under the load counter specified by field n_1.

Word 2: Checksum of columns 1-3, 7-72.

Word 3-5: Relocation data. Words 3 and 4 each hold seven 5-bit relocation identifiers, while word 5 holds five such identifiers. The five bits of each identifier carry relocation information for each instruction or data word in the text of the card. (Refer to "Relocation Scheme" below.)

Word 6-24: Instructions and data (maximum of 19 words per card). If the number of available instructions or data words on the card are not completely used up by the n specified by the control word (word 1) and at least two words are left vacant on the card, then a new control word (see format of word 1) may appear after the last utilized word, indicating a new word count n and new loading address n . The loading is then continued with the new address and with the relocation data continuously retrieved from words 3-5. The new control word does not have relocation bits associated with it. This process may be repeated as often as is necessary to fill the card.

Relocation Scheme

The five bits (A, BC, DE) of each relocation identifier (in words 3-5 of relocatable text cards) are interpreted by GELOAD as follows:

A = zero (reserved for future use by GELOAD)

BC = left half-word relocation identifier (bits 0-17)

DE = right half-word relocation identifier (bits 18-35)

For either half-word, four values apply (where XX stands for BC and DE)

XX = 00 Absolute - no relocation is applied

= 01 Relocatable - relative to the load address of the subprogram

= 10 Blank Common - relative to the beginning of the Blank Common region

= 11 Special relocation - relocate relative to the preface entry encoded in the half-word (SYMREF or Labeled Common)

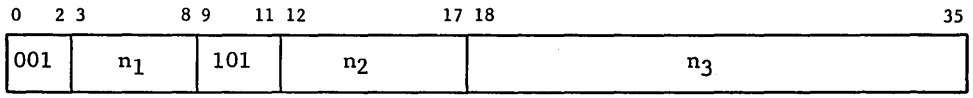
ABSOLUTE OBJECT DECK DESCRIPTION

An absolute object deck consists of one or more absolute text cards and a transfer card. Absolute text cards provide GELOAD with binary text and the absolute starting-location value for GELOAD to use in assigning core locations to the contents of the card.

Absolute Text Card

The absolute text card format is as follows:

Word 1:



Field n_1 = zero.

Field n_2 = word count.

Field n_3 = absolute address.

Word 2: Checksum of columns 1-3, 7-72.

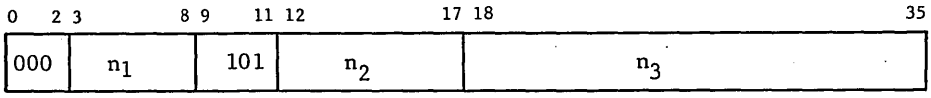
Word 3-24: Instructions and text (22 words per card, maximum). If the number of instructions or data words is not complete and at least two words are available, then word 1 may be repeated after the last word with a new word count n_2 and a new loading address n_3.

Transfer Card

The transfer card is generated automatically as the last card of an absolute subprogram assembly by the END pseudo-operation. The transfer card specifies to GELOAD the entry location for the program.

The transfer card can be used only with an absolute object deck. Its format is as follows:

Word 1:



Field n_1 = zero.

Field n_2 = zero.

Field n_3 = transfer address.

Word 2-24: Not used.

CONTROL CARD DESCRIPTIONS

The programmer, through the proper use of control cards, instructs GECOS and GELOAD in the execution of the individual activities of his job. The general rule for the placement of control cards is as follows: all control cards pertaining to a procedure must immediately precede that procedure, with the control card defining the procedure being the first card.

All control cards are identified by a $ symbol in column 1. Except for the $, columns 1-6 must be blank. The control card name begins in column 8 with delimiting blanks in columns 7 and 15. Variables, when present, begin in column 16 and are separated by commas. No embedded blanks are permitted. (A blank in the operand field terminates the field definition.) The operand field must not go beyond column 72.

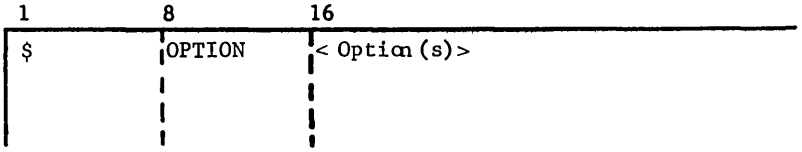
The control cards used to instruct GECOS and GELOAD are the following:

$ OPTION	$ LINK
$ DUMP	$ LIBRARY
$ USE	$ ENTRY
$ EQUATE	$ EXECUTE
$ OBJECT	$ RELCOM
$ DKEND	$ NOLIB
$ LOWLOAD	$ FFILE

$ SOURCE

$ OPTION

A $ OPTION control card is used to alter GELOAD options during loading. This card may contain, in any order, one or more fields as described below, separated by commas. The relative positions of these cards in the deck being loaded depend on the options being requested by the user. The format for the $ OPTION card is as follows:



If $ OPTION cards are not used, the options underlined below are considered standard and will be used.

The options are as follows:

1. Memory map:

   MAP — Produces a memory map.

   NOMAP — No memory map is produced.

2. Execute:

   CONGO — Executes the job regardless of any nonfatal errors detected during loading.

   GO — Executes the job only if no errors occurred during loading.

   NOGO — Loading proceeds to completion but no execution of loaded subprogram takes place. All execution pointers and indicators are set for wrapup. No dump is taken.

3. Set memory:

   SET/n/ — Sets unused allocated memory from 100_8 (Blank Common) to lowest relocatable address, to octal pattern specified by n. If not specified, these areas are set to zero. The number n can be any octal pattern up to 12 octal digits (n will be right-justified with leading zeros).

4. Set maximum error count:

   ERCNT/n/ — Sets a limit n on the number of fatal and nonfatal error messages which may be printed before loading is aborted. This count is normally set at 150. Refer to Chapter 10 for list of error messages.

5. Symbol references:

   SYMREF — SYMREF's used in each subprogram are printed with the memory map. This option may be set or reset at any time during loading.

   NOSREF — No SYMREF's are printed.

6. Low Common loading:

   LOCOMN — All Labeled Common is assigned below Blank Common. This option must be set before any Blank Common is assigned. The origin of Blank Common is readjusted upward.

7. Setup:

   NOSETU — Entry is made directly into the user's program, omitting the setup subroutine (.SETU.). Normal entry to the user program is through a setup routine which does certain initialization functions.

8. File Control Block (FCB) generation:

   FCB — Sets switches to generate FCB's for the activity as determined by control cards.

   NOFCB — Inhibits generation of FCB's.

9. Compiler generated program options:

   ALGOL — Required when loading programs generated by the ALGOL compiler. The request of this option sets all options required for the loading of an ALGOL program (i.e., FCB, LOWLOAD, and replace .SETU. with .ASETU, the ALGOL setup subroutine).

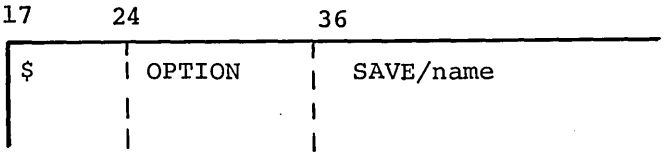
   COBOL — Required when loading programs generated by the COBOL compiler. The request of this option sets all options required for the loading of a COBOL program (i.e., NOFCB, LOWLOAD, and replace .SETU. with .CSETU, the COBOL setup subroutine).

   FORTRAN — Required when loading programs generated by the FORTRAN compiler. Sets FCB option, in addition to standard options.

   JOVIAL — Required when loading programs generated by JOVIAL compiler. Sets FCB option, in addition to standard options.

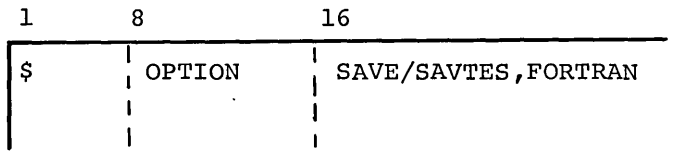
10. SAVE:

SAVE — Gives user ability to save a nonlink program on an H* file with a unique name such that it may be called and executed at a later date. The option is requested by:



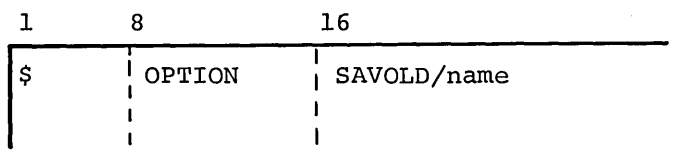
where "name" represents a unique 6-character name which is used in the MME GERSTR sequence for retrieval of the program.

If the FCB option is in effect when the program is saved, the generated FCB's may be retrieved by using the name saved with the program but with an asterisk (*) placed in front of the name. For example:



would save two text records on an H* file. The first would be called by SAVTES and the FCB's would be called by *SAVTE.

SAVOLD — Gives user ability to save additional nonlink programs on an already existing H* file. This option complements the SAVE option and is requested by:

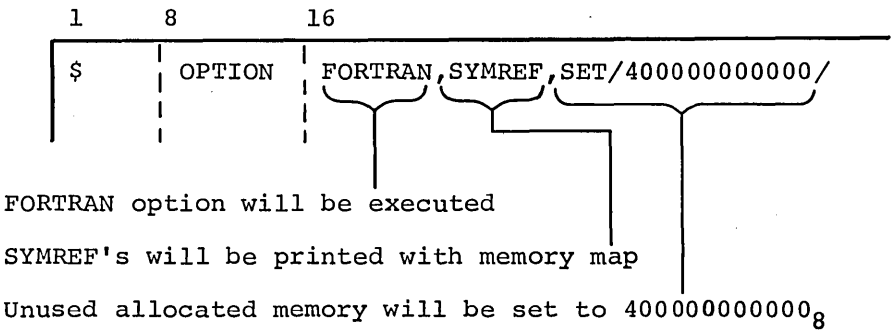


where "name" represents the unique 6-character (or less) identifier used in the MME GERSTR sequence for retrieving the program. The name is added to the catalog record of the H* file, and the program is appended to the current file.

11. NOMSUB:

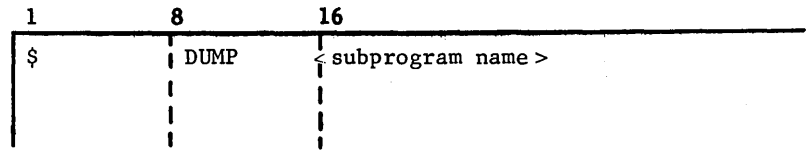
NOMSUB	Eliminates printout of message "NON FATAL ERROR MISSING ROUTINE XXXXXX". In large link jobs, the message might appear many times with each link, thus raising the error count and increasing the size of the load map. The message "NON FATAL ERROR MISSING ROUTINE MME GEBORT INSERTED AT REFERENCES" will still be printed as it would be under normal conditions.

An example of the use of a $ OPTION card is as follows:

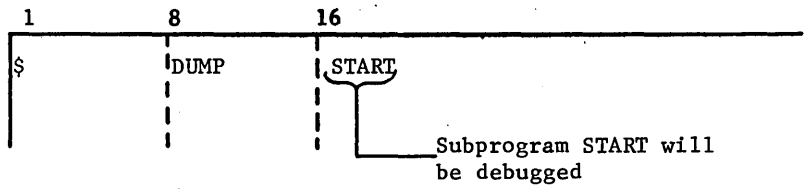


$ DUMP

The format for a $ DUMP card is as follows:

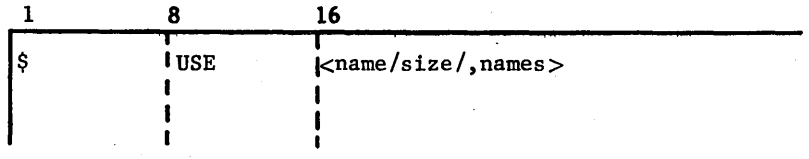


This control card must be the first card preceding a set of debug cards. The subprogram name to be debugged is equivalent to the first primary SYMDEF of that subprogram. If the $ DUMP card is used, it must precede all other cards of the program or link except the $ LOWLOAD card. At least one debug card must follow the $ DUMP card. (Refer to Chapter 8 for information pertaining to the use of debug cards.)



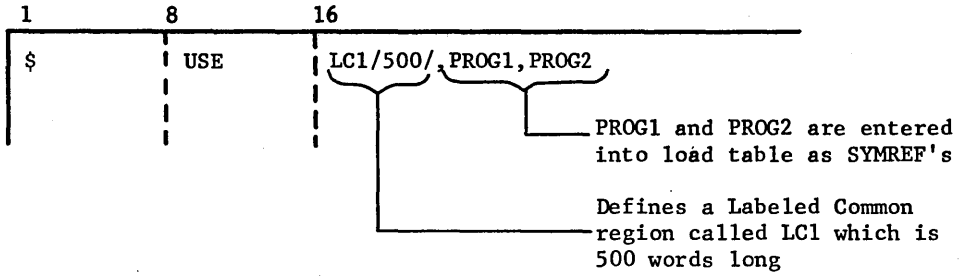
$ USE

The format for the $ USE card is as follows:

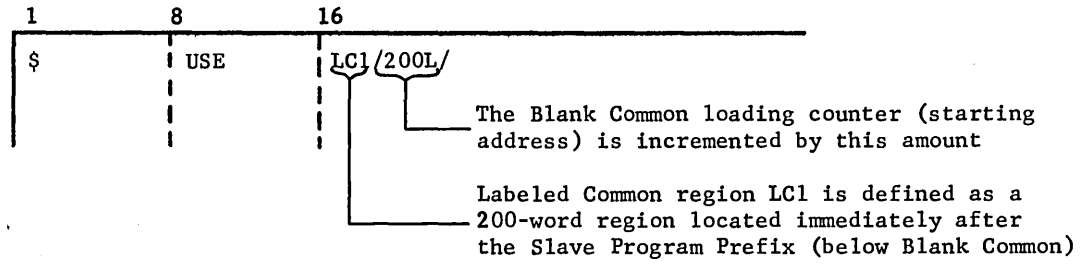


This control card permits the user to instruct GELOAD to enter a symbolic name into its symbol table to represent a Labeled Common region or SYMREF. A numeric size (enclosed in slants) following the name defines a Labeled Common region and designates the amount of storage to be set aside at that point of loading. If size is not given, the name is considered as a SYMREF. If size is terminated by the character L (i.e., /200L/), only that Labeled Common region name is handled as if under the LOCOMN option described under $ OPTION control card. All other Labeled Common regions are handled normally.

Example 1

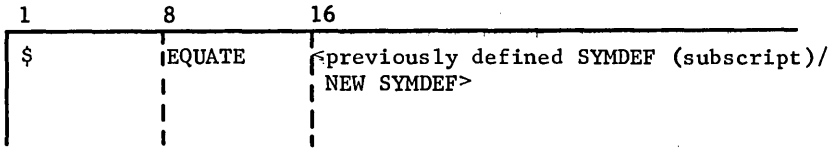


Example 2



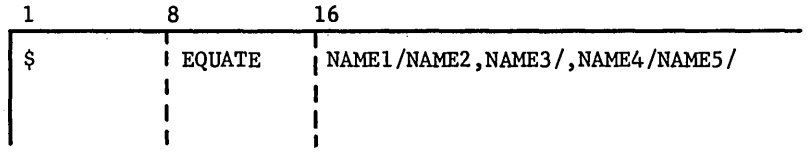
$ EQUATE

The format for the $ EQUATE card is as follows:



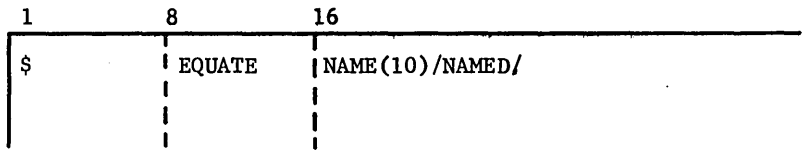
This control card permits defining of new SYMDEF's by SYMDEF's previously defined, defining new SYMDEF's relative to previously defined SYMDEF's, and equating Labeled Common regions relative to Blank Common.

Example 1



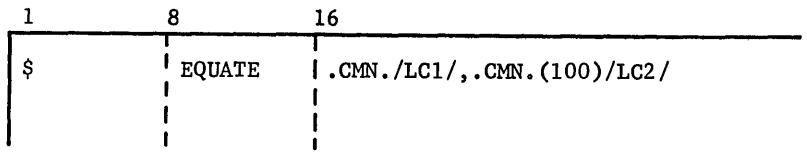
Defines NAME2 and NAME3 as SYMDEF's with the equivalent location of NAME1. A fatal error results if NAME1 is undefined. If NAME2 or NAME3 have been previously defined, they are redefined, and a nonfatal error message is printed. When more than one set of EQUATE's are to be included on a single card, they are separated by commas. NAME5 SYMDEF is defined as having a location equivalent to that of NAME4.

Example 2



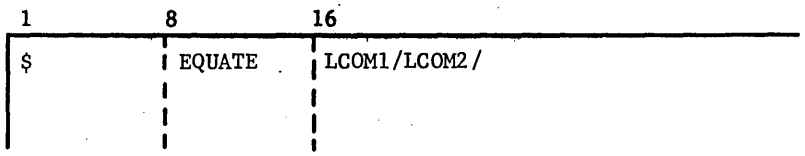
A new SYMDEF may be defined relative to a previously defined SYMDEF by enclosing the increment in parentheses. NAMED is defined as the location NAME+10.

Example 3



The .CMN. is a standard GELOAD symbol which is synonymous with the beginning of Blank Common. The Labeled Common regions (LC1 and LC2) are equated to the respective positions of Blank Common. Labeled Common region LC2 is assigned an address equal to the beginning of Blank Common plus 100. The length of Blank Common is adjusted accordingly.

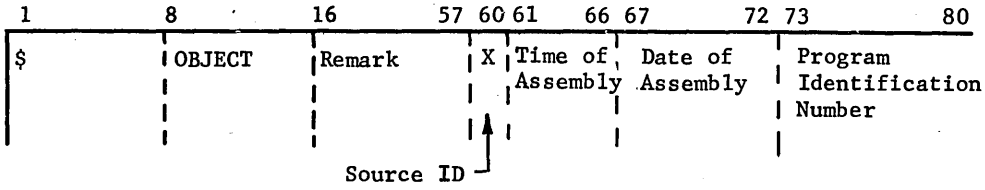
Example 4



A Labeled Common symbol may be equated to a previously defined Labeled Common region. The Labeled Common region LCOM2 assumes all of the properties (i.e., size, location) of the previously defined region LCOM1.

$ OBJECT

The format for the $ OBJECT card is as follows:



This control card must precede every absolute or relocatable subprogram object deck. The control card is produced by the assembler as the first card of all assembled subprograms.

Remark is an optional comment (a product of the second subfield of the LBL pseudo-operation) not exceeding 42 alphanumeric characters.

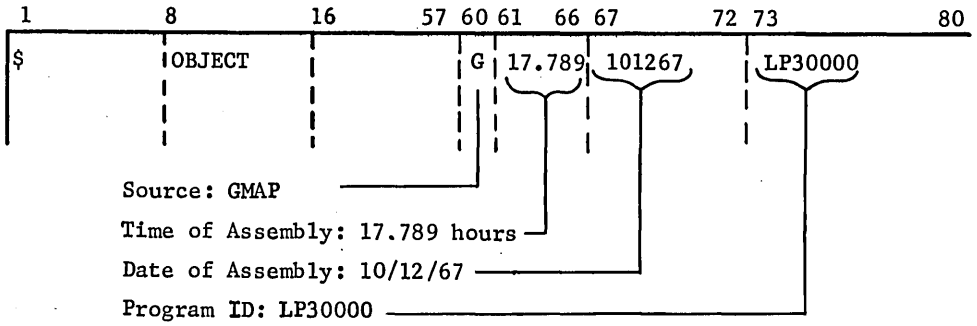
Source ID is the source of the object deck (C for COBOL, F for FORTRAN, G for GMAP, etc.).

Time of Assembly is the time of day of the current assembly. This information is supplied by the assembler.

Date of Assembly is the date of the current assembly. This information is supplied by the assembler.

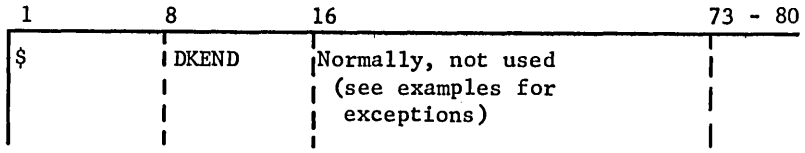
Program Identification Number is alphanumeric, designating the object program or subprogram. If LBL pseudo-op is not used, zeros will appear in this field.

Example



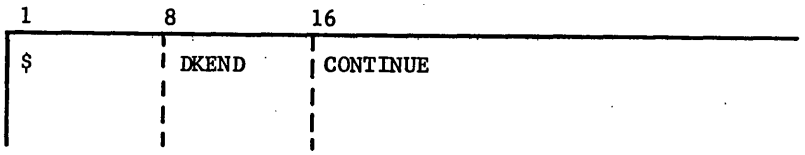
$ DKEND

The format for the $ DKEND card is as follows:

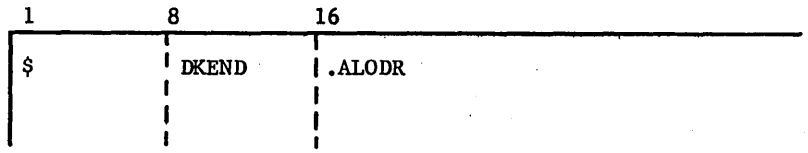


This control card must be at the end of every absolute or relocatable subprogram deck. This card is always produced by the assembler as the last card of each assembled subprogram. Columns 76-80 contain the natural sequence number of the card. Examples 1 and 2 show special cases of the use of the $ DKEND card. Example 1 would be generated in cases where the job involves batch compilation. Example 2 is generated only by the ALGOL compiler and instructs GELOAD to do certain maintenance functions during loading.

Example 1

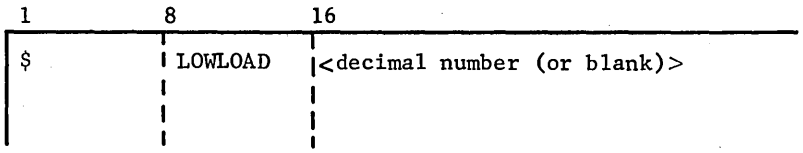


Example 2



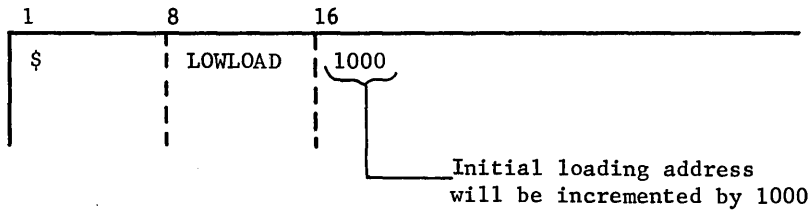
$ LOWLOAD

The format for the $ LOWLOAD card is as follows:



This control card causes the activity-allocated memory to be loaded from its low end, and must appear before any $ DUMP card and its associated debug statements, subprograms, $ OPTION control cards, or $ USE control cards are encountered. If any subprograms being loaded include Blank Common, a decimal number must be entered in the variable field of the $ LOWLOAD card. The decimal number represents (1) the largest amount of Blank Common appearing on any preface card of the object decks to be loaded plus (2) any Labeled Common assigned below or to Blank Common and extending above the top of Blank Common. If a number is present in the variable field, the loading address will be incremented by that number. If the variable field is left blank, loading will start immediately above the slave program prefix (and the 2-word link vector).

Example



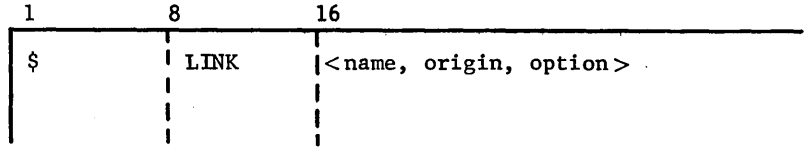
For a compile and execute, a $ LOWLOAD control card must appear immediately following the $ IDENT of the compile activity so that the binary deck(s) and system subroutines will be low-loaded.

Example

```
$ IDENT
$ LOWLOAD
$ GMAP
$ EXECUTE
```

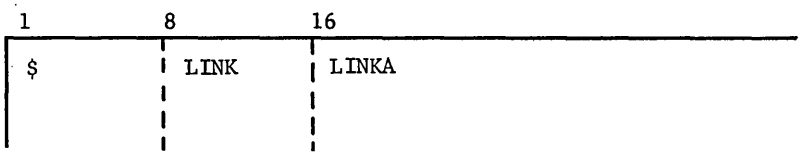
$ LINK

The format for the $ LINK card is as follows:



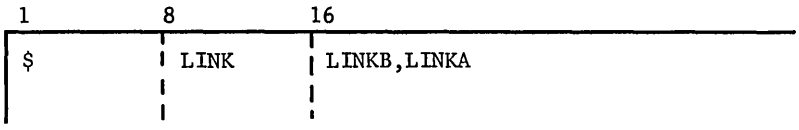
This control card specifies the beginning of a link section. The name (a maximum of 6 characters) is the unique identifier of the link to be loaded. The origin is the identifier of a previously created link at which the link being loaded is to be originated. The only option available is NOPAC, used only when references to SYMDEF's of the link(s) being overlayed are not to be purged from the load table. (Refer to Chapter 7 for details of link/overlay processing.)

Example 1



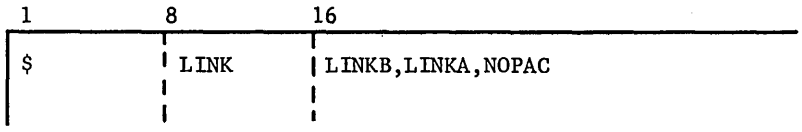
The subprograms following this card and preceding another $ LINK or $ EXECUTE card will be loaded and contained in an area defined as LINKA. The loading addresses for these subprograms would be the next available sequential addresses as in the loading of any set of subprograms.

Example 2



The subprograms following this card and preceding another $ LINK or a $ EXECUTE card will be loaded and contained in an area defined as LINKB. The loading addresses for these subprograms would begin relative to the address assigned to LINKA at the time of its loading. LINKA must have been defined on a previous $ LINK card as a name. The subprograms contained in LINKA are written in system loadable format onto an H* file. All SYMDEF's within these subprograms are purged from the GELOAD load table; no future references to these symbols are allowed.

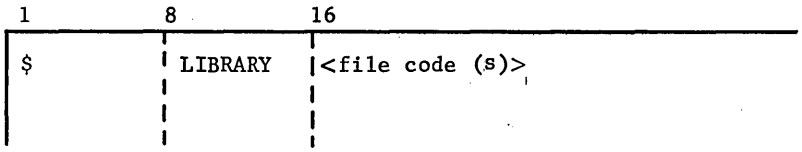
Example 3



The subprograms following this $ LINK card are handled exactly as in Example 2 except that the SYMDEF's used within LINKA are not purged from the GELOAD load table.

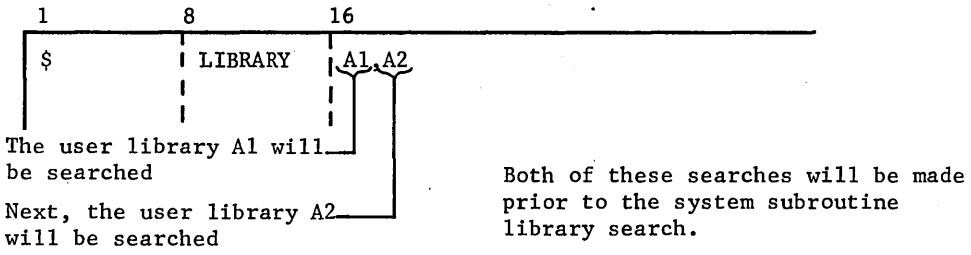
$ LIBRARY

The format for the $ LIBRARY card is as follows:



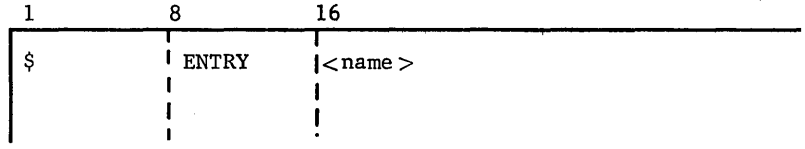
This control card indicates to GELOAD that user libraries are present. The variable field contains one or more 2-character field codes which must be further described on file control cards included in the activity deck. The libraries are searched in the order in which the file codes appear on this card and are followed by a system subroutine library search. When the file codes represent tape files, these files must be labeled. A maximum of 10 different file codes is allowed per job.

Example



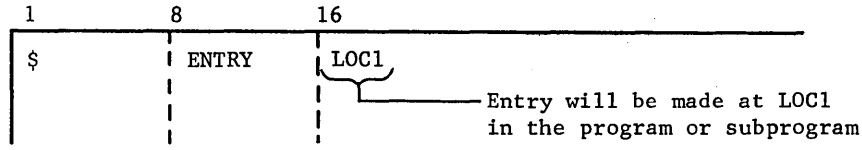
$ ENTRY

The format for the $ ENTRY card is as follows:



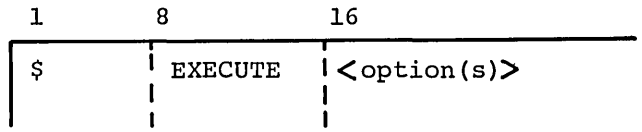
In this control card, the name is the desired primary or secondary SYMDEF entry point to the program. If this card is not present, the entry is made at symbolic location...(if ... has been defined in loading as a primary SYMDEF) or at the first defined primary SYMDEF of the first subprogram loaded. During linking operations, this card refers to an entry point of a subprogram in the current link.

Example



$ EXECUTE

The format for the $ EXECUTE card is as follows:



This control card is used to request the loading of object programs or subprograms. This card must appear after all subprograms to be executed, but before their data. Effectively, the EXECUTE control card requests GECOS to call GELOAD, which then loads all subprograms of the activity. The entries in the variable field are used by GECOS for activating sense switches 1 through 6 (which may be sensed during program execution by interrogating the Program Switch Word) and for electing the abort jump option. The options may appear in any order.

Options are as follows:

1. Sense switches

   ON1 - sense switch 1 on (Program Switch Word, bit 6 = 1)
   ON2 - sense switch 2 on (Program Switch Word, bit 7 = 1)
   ON3 - sense switch 3 on (Program Switch Word, bit 8 = 1)
   ON4 - sense switch 4 on (Program Switch Word, bit 9 = 1)
   ON5 - sense switch 5 on (Program Switch Word, bit 10 = 1)
   ON6 - sense switch 6 on (Program Switch Word, bit 11 = 1)

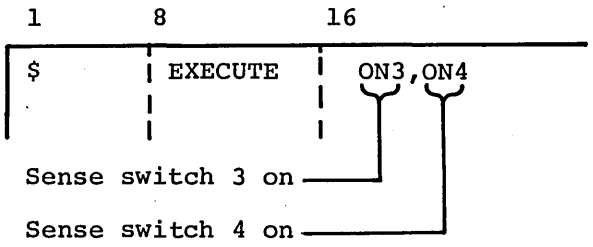
2. Dump option

   DUMP Slave core dump will be given if activity terminates abnormally (Program Switch Word bit 0 = 1).

   NDUMP Only program registers will be dumped if activity terminates abnormally.

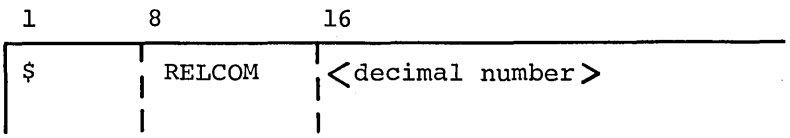
3. Blank Sense switches off and NDUMP implied (Program Switch Word bits 6 - 11 and bit 0 = 0).

Example



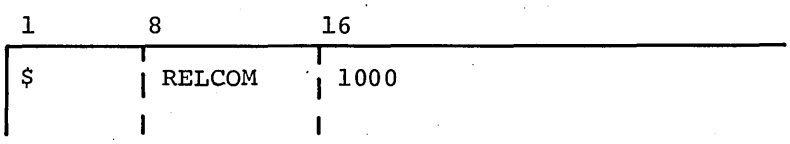
$ RELCOM

The format for the $ RELCOM card is as follows:



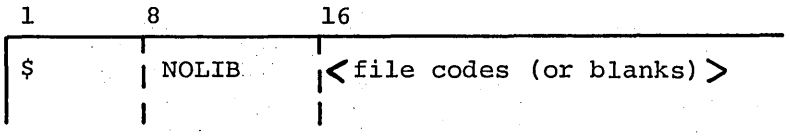
This control card causes the Blank Common loading counter to be incremented by the amount specified in the variable field. The Blank Common loading counter and its reference symbol .CMN. are normally set to the address just above the slave program prefix (cell 100_8). This control card must precede any object deck(s) and compiler/assembler activities.

Example



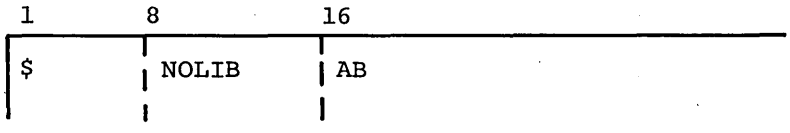
$ NOLIB

The format for the $ NOLIB card is as follows:



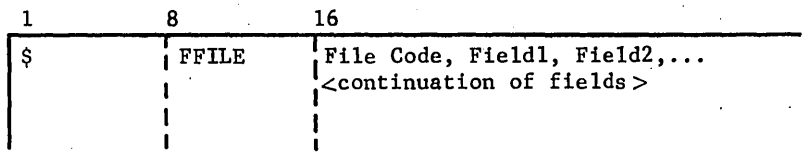
This control card prevents a library search from being made for the current activity or link being loaded. The control is only in effect during the loading of the link in which it appears. If the variable field is blank, no search is made. If the variable field contains a file of L*, all file codes specified on the $ NOLIB card must have been previously specified on a $ LIBRARY card.

Example



$ FFILE

The format for the $ FFILE card is as follows:



This control card is used to describe nonstandard file control blocks to be generated as a result of the FCB option and must be located after the $ EXECUTE card. One file control block will be created for each $ FFILE card found in the deck. The first field (beginning in column 16) must be the 2-character file code as specified in a GECOS file control card. The comma-separated fields (described below) following the file code are used to produce the nonstandard entries in the file control block to be generated. The fields need not appear in any specific order.

Field	Word and Bit Positions Altered in FCB		Function
STDLBL	Word -5	Bit 24	Create or verify standard labels. Unless NSTDLB is specified, the STDLBL will be assumed.
NSTDLB	Word -5	Bit 24	Do not create or verify standard labels.
NBUFFS/n	Word -5	Bit 24	Assign "n" (0,1, or 2) buffers.
BUFSIZ/nnn	Word +4	Bits 0-17	Set buffer size = nnn, where nnn is a decimal number.
LGU/(II,JJ...)	Not FCB		Logical files II, JJ,... (as many as desired) are on the same device as logical file FC, where II, JJ, etc. are 2-digit numerics ≤ 43.
RETPER/nnn	Word -8	Bits 0-17	Set retention period = nnn, where nnn is a decimal number ≤ 999.
MLTFIL	Word -5	Bit 25	This is a multifile device.
MODBCD	Word 0	Bits 20,21	Set recording mode to BCD.
FIXLING/nn	Word 0 Word +1	Bits 18,19 Bits 0-17	The records are a fixed length of nn, where nn is a decimal number.
NOSRLS	Word -5	Bit 23	Do not use block serial numbers.
LODENS	Word 0	Bit 22	Set device to low density (magnetic tapes only).

Field	Word and Bit Positions Altered in FCB	Function
IGNORE	Not FCB	The user wishes to specify his own FCB. GELOAD will not create an FCB for this file code.
PREHEAD/SYMDEF	Word -14 Bits 0-17 Word -8 Bit 18	SYMDEF is the entry to a preheader label checking routine supplied by user.
POSHED/SYMDEF	Word -13 Bits 0-17 Word -8 Bit 18	SYMDEF is the entry to a postheader label checking routine supplied by user.
PRETRL/SYMDEF	Word -12 Bits 0-17 Word -8 Bit 18	SYMDEF is the entry to a pretrailer label checking routine supplied by user.
POSTRL/SYMDEF	Word -11 Bits 0-17 Word -8 Bit 18	SYMDEF is the entry to a post-trailer label checking routine supplied by user.
ERRXIT/SYMDEF	Word -5 Bits 0-17	SYMDEF is the entry to a user-supplied error handling routine.
MIXLNG/SYMDEF	Word 0 Bits 18,19 Word +5 Bits 0-17	SYMDEF is the entry to a user-supplied routine to determine length of records being read. Bits in word 0 are set to indicate mixed length records.
NOSLEW	Word -6 Bit 23	When set, indicates no automatic slewing is to be done by the FORTRAN I/O library subroutines. Slew characters coded as part of the format are passed with the line.
PTMODS	Word 0 Bits 20,21	Set recording mode for single-character paper tape.
PTMODD	Word 0 Bits 20,21	Set recording mode for double-character paper tape.
PTMODE	Word 0 Bits 20,21	Set recording mode for paper tape edit.
MODMIX	Word 0 Bits 20,21	Set recording mode for mixed binary and BCD card files.

Field	Word and Bit Positions Altered in FCB	Function
ASA9	Word 0 Bits 20,21	Set recording mode for 9-channel tape commands.
DSTCOD/(Printer, MTAPE,...)	Not FCB	Final destination codes for each of the logical units specified in LUG/(...) field. These are used only with programs generated by the ALGOL compiler.

Fields recognized with DSTCOD:

Device	Mnemonic
Printer	PRNTR
Card Reader (Binary)	BCRDR
Card Reader (BCD)	DCRDR
Card Reader-Mixed	MODMIX
Card Punch (Binary)	BCPNCH
Card Punch (BCD)	DCPNCH
Magnetic Tape (standard or ASA 7-track)	MTAPE
Magnetic Tape (ASA 9-track)	ASA9
Disc	DISC
Drum	DRUM
Paper Tape-Single Character	PTMODS
Paper Tape-Double Character	PTMODD
Paper Tape-Edit	PTMODE

The following example shows the use of the $ FFILE card:

```
1       8          16
|       |          |
$       | EXECUTE  |
$       | TAPE     | 17,X1R,,,,0001,MYFILE
$       | FFILE    | 17,NSTDLB,NOSRLS
```

4. Loading Relocatable Object Decks

To load a relocatable object deck, the cards comprising the subprogram deck must be arranged in the following order:

$ OBJECT control card
Preface card(s)
Relocatable text card(s)
$ DKEND control card

The loading of the relocatable subprogram deck begins with the recognition of a $ OBJECT control card. If the card contains the date on which the deck was originally produced, the date information will be saved so that it may be printed subsequently as part of the memory map.

INTERPRETING THE PREFACE CARD

The card following the $ OBJECT card must be a preface card, recognizable by a 12-7-9 punch pattern in column 1. The only legal alternative would be an absolute text card. (Refer to Chapter 6 for details concerning absolute text cards.)

The various fields of the preface card supply all size information and such linkage details as the relative entry points to the subprogram (SYMDEF's) and which subprograms and data regions (SYMREF's and Labeled Common) are required for execution of the relocatable subprogram. There are three considerations made by GELOAD to determine if enough memory is available to load the subprogram -- the length of the subprogram being loaded, the length of Blank Common region required, and the number of preface symbols contained on the preface card(s). This latter consideration is used in determining if the increase in the size of the load table (calculated as 5 times the number of preface symbols) caused by the addition of these symbols to the table will result in an overlap with the subprograms previously loaded into memory. GELOAD then proceeds to pick up and process each of the respective preface entries. All SYMDEF symbols appear first, followed by all Labeled Common and SYMREF symbols. Because of field size, up to 63 Labeled Common regions is the limit that may be referenced by one subprogram.

SYMDEF

When a SYMDEF (primary or secondary) is encountered, GELOAD searches the load table to determine if the symbol has been previously defined. If such is the case, the symbol is ignored; otherwise, the symbol, with its

definitions calculated as the sum of the beginning address of the subprogram and the relative entry value taken from the preface, is entered into the load table. (The symbol may appear in the load table undefined as yet. In this case, only the defining address is added to the table.) If all SYMDEF entries on the preface card(s) have been previously defined, the scanning of the preface card is terminated and the subprogram is bypassed. A nonfatal message will be printed on the memory map when this occurs while reading from the B* and R* files.

Labeled Common

When a Labeled Common symbol is encountered, GELOAD searches the load table for its definition. If not previously defined, it is assigned storage at the next available memory location, according to its requirements as stated on the preface card. An entry is then made in the load table stating the defining address and also the size when defined. When it is found that a Labeled Common symbol has been previously defined, the size from the preface card is compared to that of the definition. If the size in the definition is greater than or equal to that on the new preface card, loading continues; otherwise, a nonfatal type error message is printed.

When the symbol has satisfied the requirements of the load table, a pointer to this location in the load table is placed in the temporary reference table.

SYMREF

When a SYMREF is encountered, GELOAD searches the load table to determine if it has been previously defined. If such is the case, a pointer to this definition in the load table is entered in the temporary reference table. If the SYMREF has not appeared previously, or is yet undefined, it is entered in the load table as undefined. Again, a pointer to this entry is entered in the temporary reference table.

When the subprogram being loaded has been read from a library file containing a directory, bits 19-32 of the control word may contain the address of the block in the file containing the defining subprogram for this SYMREF. In cases where the defining subprogram does not appear on the same library file, this will not be true. When this address is present, it is moved to a table of similar addresses of subprograms required from this library for the current execution activity.

LOADING RELOCATABLE TEXT CARDS

Each text word is picked, in turn, from its position in the relocation text.card image. Relocation is applied to the respective half-words (18-bit fields) as designated by the encoded relocation scheme in words 3-5 of the card. The following is a description of what takes place as a result of each type of half-word relocation before the word is stored into memory. The digits in parentheses are the binary bit pattern signifying this type relocation.

Absolute (00)

The 18-bit field is an absolute value and must not be modified.

Subprogram Relocatable (01)

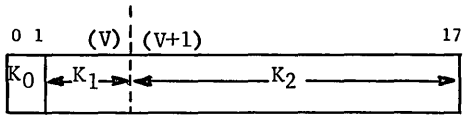
The 18-bit field is relocated by the value assigned to the beginning location of the subprogram.

Blank Common (10)

The 18-bit field is relocated by the value assigned to the beginning location of Blank Common.

Special Relocation (11)

Special relocation implies that the 18-bit field is a reference to a word in an external subprogram or data region. Although the following example of special relocation uses bits 0-17, it should be noted that special relocation is available in bits 18-35 when required.



where:

K_0 is the sign of the addend; 0 implies plus and 1 implies minus.

K_1 is a variable-length field (V count bits) dependent on the number of bits required to express the total number of Labeled Common and SYMREF symbols appearing on the preface card. The value K_1 indicates which of the preface symbols is being referenced; for example, K = 1 implies the first Labeled Common appearing on the preface card(s). V is variable terminal bit position of field.

K_2 is the addend to be added to (K_0=0) or subtracted from (K_0=1), the address associated with the referenced symbol. If this field, because of the size of K_1, is not large enough to express the entire addend, then all bits of the field K_2 are set to 1's. This implies that the next word in the card image contains the complete addend in the same relative half-word position. When a second word is used in this manner, no bits of the relocation scheme (words 3-5) are used to describe it.

When a special relocation reference is made to a symbol yet undefined, an address chain is used to link all references until the deck containing the symbol is loaded. Later in the wrap-up phase of loading,

all address chains are filled in with their definitions or with an abort procedure such that during execution, when these cells are referenced, an abort will occur. It should be noted that if, by means of octal patching (see Chapter 12), one of these address chains is broken, numerous undetectable errors could occur. Thus, any form of patching is illegal when applied to words containing SYMREF's.

Figure 3 shows an example of printed output from the GMAP Assembler utilizing the various types of relocation. The information may be read from left to right as: location (octal), octal equivalent of instructions or data to be loaded, relocation scheme, alter number of card, and the card image. A further description of the output format may be found in GE-625/635 GMAP Implementation, CPB-1078.

```
04660 1   06-02-66


     000000  010005  2350 00  030     1 EX1    LDA      A2
     000001  000001  0750 07  000     2        ADA      1,DL
     000002  000000  7550 00  020     3        STA      COMMON
                                      4 *
                                      5 *
                                      6 *
                                      7 *
                                      8        SYMDEF   EX1
     000003  000000  000000   001     9 EX2    ZERO     0,EX1
                       000000        10        BLOCK    DATA
                       000000        11 A1     BSS      5
                       000005        12 A2     BSS      5
                       000000        13        BLOCK
                       000000        14 COMMON BSS      100

     ERROR LINKAGE

      000004  000000000000     000
      000005  256701202020     000
                                      15        END
     6  IS THE NEXT AVAILABLE LOCATION,
  THERE WERE    NO  WARNING FLAGS IN THE ABOVE ASSEMBLY, AID 042766
```

Figure 3. Types of Relocation

All four types of relocation are illustrated in Figure 3. The numerics used to express the relocation scheme (i.e., the field 030 associated with alter number 1) are not octal characters but are made up of the five bits described earlier under "Relocation Scheme" in Chapter 3. For example, taking the field 030, the first character (0) may be ignored, the second character (3) implies special relocation to the left-most 18 bits and the third character (0) implies absolute or no relocation to the right-most 18 bits of the word being loaded. A discussion of Figure 3 by alter number follows:

Alter 1 — The left-most 18 bits of this word require special relocation while the remaining 18 bits are an absolute field. Although symbol A2 is within the Labeled Common region named DATA (assembled with this subprogram), the contents of this region are maintained under a separate location counter and look to GELOAD as data

assembled into a separate subprogram; therefore, special relocation must be used to reference it. The left-most 18 bits, when analyzed in the light of the above description of special relocation, result in:

bit 0 (K_0) = 0 (sign of addend is plus)

bits 1-5 (K_1) = 1 (assuming V count of 5, use first non-SYMDEF entry in preface)

bits 6-17 (K_2) = 5 (apply addend of 5 to address resulting from lookup of symbol determined by K_1)

Alter 2 The relocation scheme indicates that both 18-bit fields are absolute.

Alter 3 The relocation scheme associated with the left-most 18 bits indicate this field is an address relative to the origin of Blank Common.

Alter 9 The 18 right-most bits of this word contain an address relative to the origin of the subprogram.

Alter 10-12 This is a string of coding which does not produce any loadable text but does define the size and the symbolic names which make up the Labeled Common block named DATA. GELOAD obtains this information via the preface card.

Alter 13-14 This is a string of coding which does not produce any loadable text but does define the size of Blank Common used by this subprogram. GELOAD obtains this information via the preface card.

Figure 4 illustrates the layout of a 5120-word segment of memory after the coding shown in Figure 3 is loaded. Previous to loading this coding, the last cell used was 10300_8. The load address for the subprogram is determined by subtracting the size of the subprogram (6), taken from the preface card, from the last address used by the previous subprogram. Thus the address assigned to symbol EX1 is 10272_8. The SYMDEF EX1 is placed in the load table with the defined value of 10272_8. The origin of the Labeled Common region DATA is calculated in the same manner as above; thus its origin is $10272_8 - 12_8 = 10260_8$. This value is also entered in the load table with the respective Labeled Common description. The Blank Common region (100 words) is assigned directly above the slave program prefix (cells $0-77_8$) and extends to 243_8. The yet unused portion of memory (cells $244_8 - 10257_8$) is available for additional loading of subprograms and data or for working storage during execution. This information is stored in slave program prefix at location 37_8.

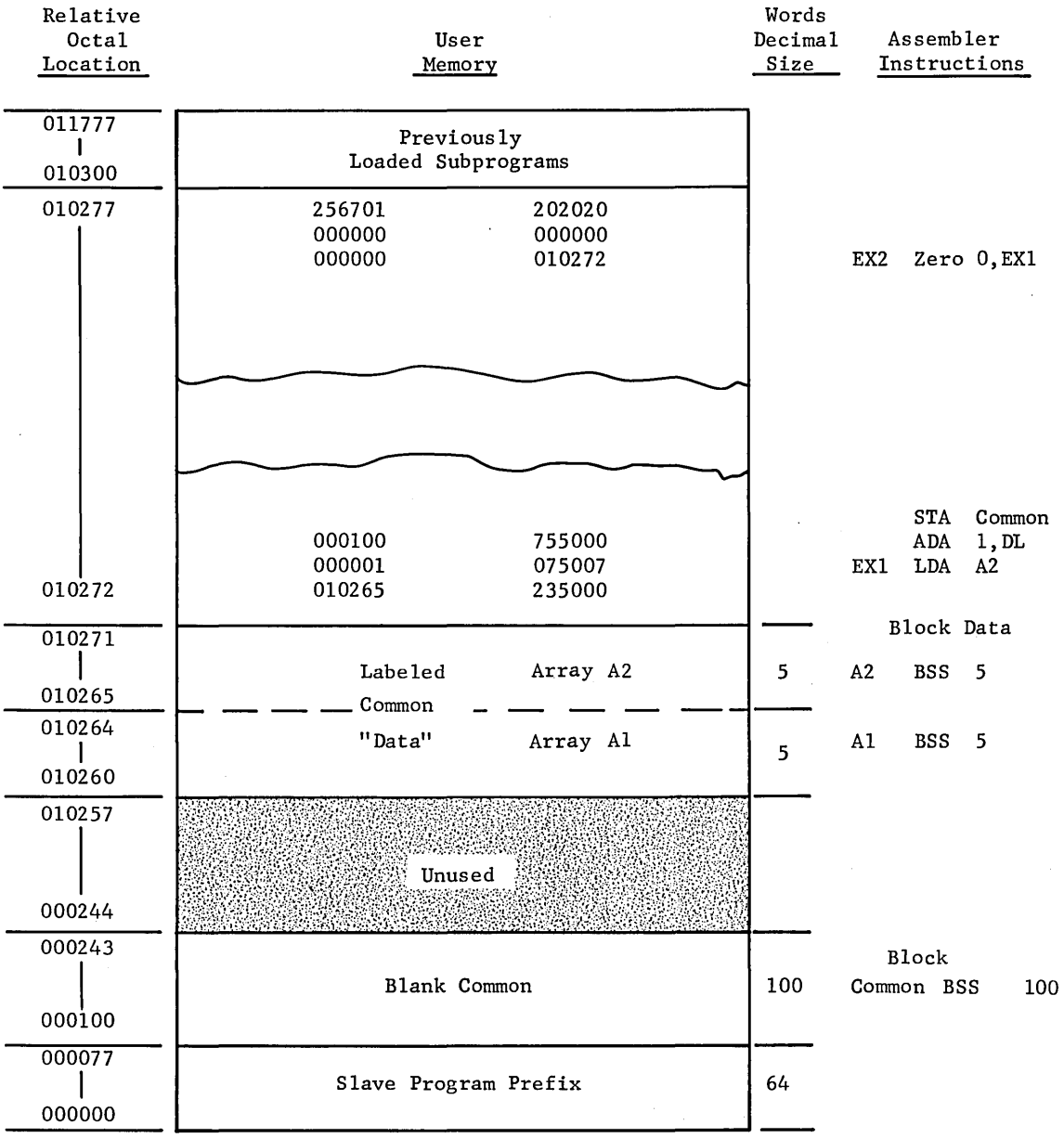


Figure 4. Layout of 5120-Word Segment of Memory

5. Use of Libraries

LIBRARY FILES

During the loading process, GELOAD builds a load table containing all SYMDEF, SYMREF, and Labeled Common symbols with related information used in their definitions. It is very likely that not all symbols encountered as SYMREF's will have been defined when the end of the input stream ($ EXECUTE on system file R*) to GELOAD is reached. This situation triggers GELOAD to search library files available to it for subprograms that have SYMDEF symbols which satisfy the undefined SYMREF's in the load table.

The system subroutine library file (file code L*) is always available to GELOAD. It should contain all high-use subprograms required for executing major software packages. Because of the space requirements of these and other subprograms added at certain user installations, a secondary system subroutine library (file code *L) may be used. When this secondary library is used, it is scanned before the primary file (L*). To allow more efficient use of high-speed storage, it is generally assumed that the secondary library contains the less frequently used subprograms and resides on a slower device.

Besides the system library files described above, the user may create his own library files using the Object File Editor (refer to GE-625/635 System Editor, CPB-1138) containing additional subprograms conforming specifically to his needs. Such a user library file is recognized by GELOAD when its file code is encountered on a $ LIBRARY control card (see Chapter 3) located in the input stream. If the card is present, user library files are searched in the order in which they are encountered and before the system subroutine library file L*.

PRIMARY AND SECONDARY SYMDEF SYMBOLS

A primary SYMDEF is any symbol appearing on a preface card and used as an entry point to the subprogram. A secondary SYMDEF follows all the rules associated with primary SYMDEF symbols, with the additional qualification that the segment of coding for which it is an entry point appears elsewhere on the library file as a freestanding subprogram (primary SYMDEF). The following example may clarify the usefulness of secondary SYMDEF symbols.

Assume that subroutine A on the library file contains within itself another subroutine B. The advantages of this are in linkage efficiency and conservation of storage, assuming logic of A and B use the same temporary working storage. This deck then contains a primary SYMDEF A and a secondary SYMDEF B. Further along on the library file the subroutine B exists as a freestanding subprogram for which B is a primary SYMDEF.

In searching a library file, only primary SYMDEF symbols are scanned when looking for subprograms to satisfy undefined (SYMREF) symbols in the load table. Assume a SYMREF for subprogram A had been entered in the load table. In reading A from the library file, B is brought in as part of it. Had a SYMREF for subprogram B been in the load table instead of A, the subprogram having the primary SYMDEF B would have been loaded instead. Thus, only the matching of a SYMREF symbol and a primary SYMDEF symbol causes a particular subprogram to be loaded from a library file. During the loading sequence, all SYMDEF symbols, whether primary or secondary, are defined and are used in turn to define the value of undefined SYMREF symbols.

6. Loading Absolute Object Decks

Although GELOAD is primarily designed to load relocatable subprograms, it can load absolute subprograms as well. Absolute binary cards appear between the $ OBJECT and $ DKEND control cards.

PROCESSING THE ABSOLUTE TEXT CARD

After the $ OBJECT control card is read, GELOAD will encounter an absolute text card. An absolute text card is recognized when a binary card is read having a bit pattern of 001 in bit positions 0-2 of word 1. (Refer to Chapter 3 for description of absolute text card format.)

GELOAD obtains the word count and the absolute address from word 1. The data word in word 3 of the card is loaded into the initial address, and each word thereafter is loaded upward in consecutive memory addresses until the count is exhausted. If the word count is less than 20, it is possible that the word directly after the last word loaded is another control word like that in word 1. If so, loading continues into the new load address, starting with the word directly after the control word. The checksum of all the words on the card, excluding the checksum word, is found in word 2.

PROCESSING THE TRANSFER CARD

If GELOAD encounters a binary card after the $ OBJECT control card but prior to the $ DKEND control card and the card has a bit pattern of 000 in positions 0-2 of the first word, it interprets it as an absolute transfer card. (Refer to Chapter 3 for description of absolute transfer card format.)

GELOAD obtains the absolute transfer address and saves it to be used at the entry point when the end of the input file is reached.

Pseudo-operations SYMDEF and SYMREF cannot be used in subprograms which will be loaded absolute.

7. Link/Overlay Processing

Because of size and complexity, it may be beneficial to segment a program in order to make more efficient use of memory and available storage media. Each of these segments may be referred to as a "link." When using links, the programmer must organize his program in such a way as to retain the more commonly used subprograms in the links which will reside in memory and the lesser used subprograms in links which will be used as temporary overlays. All of the subprograms loaded which precede the first $ LINK control card and all subprograms loaded as a result of the first library search are commonly referred to as being in the "main link." As this first link has no identifier by means of a $ LINK control card, it is assigned the standard "main link" identification of ////// when it is written on an H* file.

USE OF THE $ LINK CONTROL CARD

The $ LINK control card is used to specify the positions in the input stream at which segmentation is to take place. When this control card is encountered, all requested library files are searched to satisfy any undefined references (SYMREF's) in the link being terminated. The $ LINK control card specifies, in its first variable field, a unique identifier for the new link. When variable field 2 is present on the card, it indicates that the new link is to overlay a previously loaded link(s) whose identifier appears in field 2. In this situation, the new link assumes the origin of the link specified in field 2. All links which are to be overlayed by the new link are written in system loadable format (refer to GE-625/635 System Editor, CPB-1138) onto the file having the file code H*.

When the $ EXECUTE control card is encountered, much of the same procedure as described for $ LINK control cards is followed in order to wrap up the linking process. If file control blocks are to be generated by GELOAD (see Chapter 11), an additional utility link is created and written onto the H* file having identifier /////1.

REFERENCING BETWEEN LINKS

The amount of cross-reference between the various subprograms which comprise the link program dictate the desired segmentation. The main link, as defined above should contain all high-usage subprograms because of its permanent status in memory throughout the execution. Subprograms contained in any other link may always reference subprograms in the main link.

As all cross-references between subprograms are established during loading by GELOAD, only those subprograms contained in links which reside in memory at the same time may reference each other. (The usage of the word "subprograms" is meant here to also include data areas.) This is to say, if link B is loaded as an overlay of link A (i.e., $ LINK B,A) then subprograms of link B cannot reference subprograms of link A. The optional third variable field (NOPAC) of the $ LINK control card may be used discreetly to override the above rule of references between links. Consider an example having three links, A, B, and C, which are loaded using the following control cards:

```
$ LINK A
$ LINK B
$ LINK C,A
```

Under normal conditions of loading, no references may be made between link C and link B, for when link C overlays link A it may possibly overlap into link B and thus destroy some subprograms of link B. However, if the size of link C is smaller than or equal to the size of link A, the user may increase his referencing capabilities by adding NOPAC to the card defining link C. The example now becomes:

```
$ LINK A
$ LINK B
$ LINK C,A,NOPAC
```

This now allows references between subprograms of link A and link B while link A is in memory, and between link C and link B when link C is in memory.

LINK MANIPULATION AT EXECUTION TIME

As described above, GELOAD builds file H* containing the links defined by the input stream. During execution of the program, it is necessary to reload these links in the order required to achieve the function of the program. To do this, a subprogram is supplied on the system subroutine library (L*) which has two entries, LINK and LLINK. The entry LINK may be called to load a particular link and transfer control to a predesignated SYMDEF within that link. This SYMDEF must have been designated to GELOAD by means of a $ ENTRY control card when the link was originally created on the H* file. The subprogram entry LLINK may be called to load a particular link and return control to the point in the program at which LLINK has been called. Thus, the two respective calling sequences are as follows:

1. To load a link and transfer control to it:
   ```
   CALL LINK (ARG)
   ```
2. To load a link and return to the calling:
   ```
   CALL LLINK (ARG)
   ```

In both sequences, ARG could be defined in a GMAP program as:

```
ARG   BCI   1,   Link Identifier
```

Two additional SYMDEF's are included in the LINK subroutine: LENTRY and IDLINK. The LENTRY enables a user to find his link entry point when issuing an LLINK call.

In the Linear Programming language (to cite one example in the use of LENTRY), there is a provision of patch agenda at execution time. Therefore, when using this provision, the link to be patched must be loaded and then modified before entering. Hence, the entry point must be known; therefore, the LENTRY SYMDEF is used.

The other SYMDEF, IDLINK, points to the last previously loaded link and is useful in debugging a core dump. A detailed discussion of these routines may be found in GE-625/635 FORTRAN IV I/O Library, CPB-1137.

EXAMPLE OF A LINKED PROGRAM

Figure 5 illustrates a sample deck setup utilizing the $ LINK control card to create a linked program. Figure 6 illustrates the memory layout that results from loading the linked program.

This example consists of four links, a main link, and Blank Common available as storage, accessible to all of the links.

GELOAD loads the subprograms comprising the main link until the first $ LINK control card is recognized. At this point, GELOAD has already read the $ ENTRY control card and established LOC1 as the entry location of the main link. The $ USE control card defines a 100-word Labeled Common region to be created and called VCL1. If there are any undefined SYMREF symbols, a search of the system subroutine library will now be made in an attempt to define these symbols.

Since the $ LINK control card and defining link A have no origin (none may appear on the first $ LINK control card), GELOAD will load decks and assign storage for link A directly below the main link. When the next $ LINK control card is encountered, another system subroutine library search will be made if again there are any undefined symbols. The next $ LINK control card indicates that link B will have its origin at link A. As no $ ENTRY control card was read during the processing of link A it is assumed that a call to the LLINK subprogram will be used to restore this segment during execution. Link A is then written onto file H* and all symbols associated with link A are purged from the GELOAD symbol table. The decks which comprise link B are loaded and entry location is set according to the definition of the symbol appearing on the $ ENTRY control card. The decks making up link C are loaded below link B. The next $ LINK control card indicates link D will overlay link C and initiates the same procedure as that initiated for the loading of link B.

The $ EXECUTE control card indicates to GELOAD that it has reached the end of the loadable input stream. (If the FCB option has been set, the utility link (/////1) would be created at this time.) The remaining links are now written onto the H* file in the order in which they appear in the GELOAD symbol table (i.e., the main link, link B, and finally link D). Control is then transferred to the specified entry point of the main link (LOC1) for execution. The main link (as well as link B and link D) does not need to be restored into memory since it still resides there from loadings.

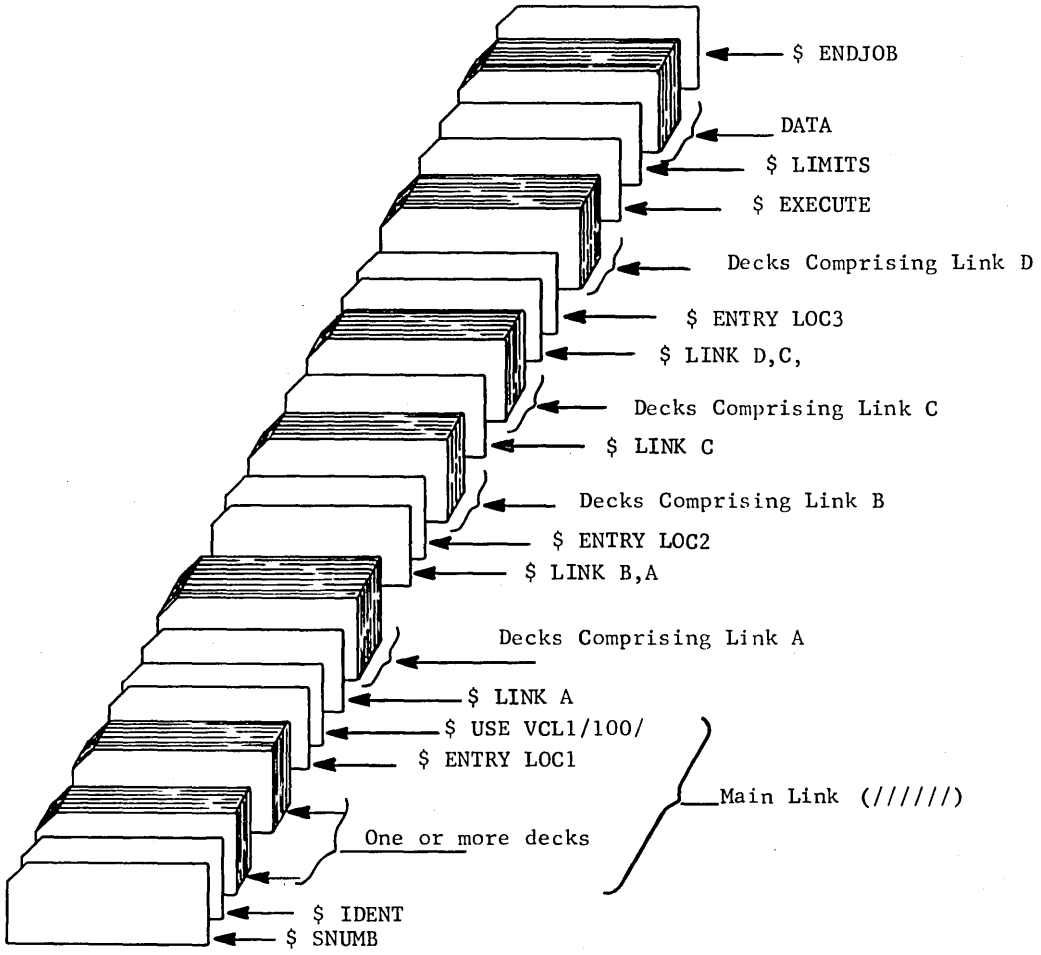


Figure 5. Deck Setup For A Linked Program

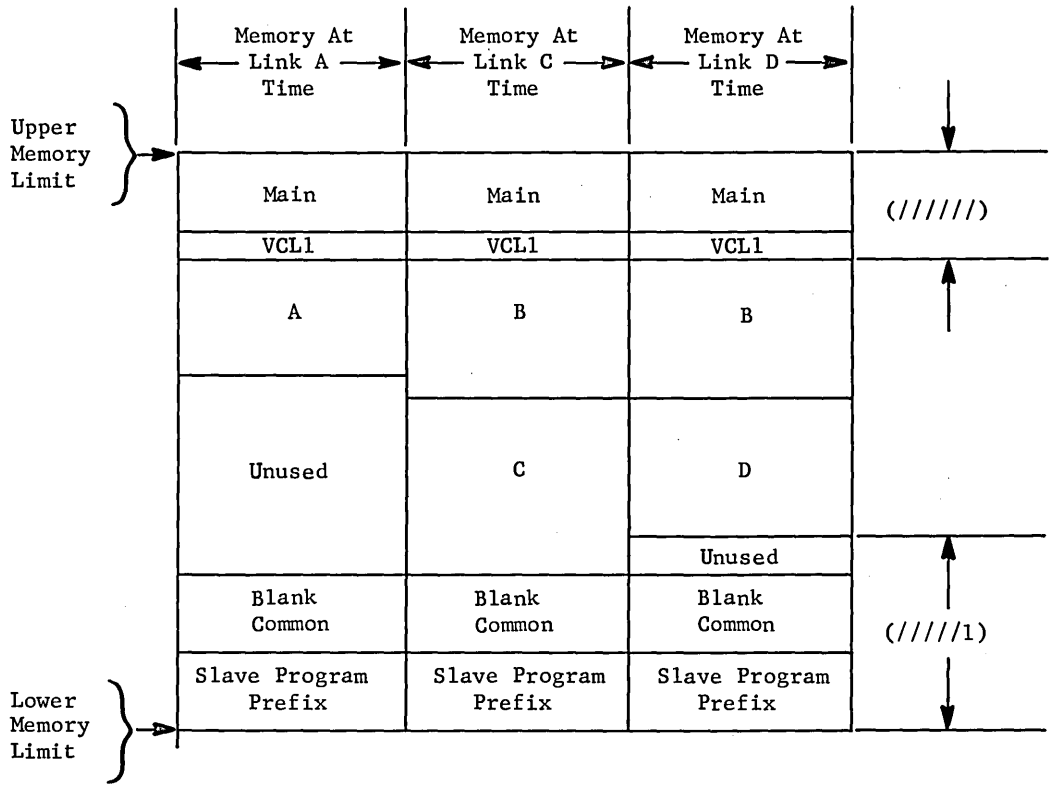


Figure 6. Memory Layout Resulting From Loading Linked Program Of Figure 5.

8. Using The Debug Feature At Load Time

DEBUG FEATURE

The debug feature included in GELOAD is similar to debug as provided in the FORTRAN compiler. Dumping of specified memory locations in selected formats during execution of a program is accomplished by cards presented to GELOAD at load time. Thus, the debug feature may either be used or not used, depending on the presence of these cards at load time.

Usage of the GELOAD debug feature requires different procedures by the programmer when running with subprograms compiled by FORTRAN than with subprograms generated by other means (GMAP and other compilers). In FORTRAN, the programmer specifies in the $ FORTRAN control card, by means of the STAB option, that the compiler generate a Debug Symbol Table containing all symbols and pertinent information describing them. The programmer then need only concern himself with the contents of his GELOAD debug cards.

When the debug feature is used with subprograms generated by software other than FORTRAN, the symbols used to indicate the positions in the subprogram at which dumps are to be taken and the areas which are to be dumped may be any SYMDEF or Labeled Common symbol defined in the load table. The programmer may gain additional debug capabilities by generating a Debug Symbol Table similar to that produced by FORTRAN. This table takes the form of a special Labeled Common region (.SYMT.).

DEBUG SYMBOL TABLE .SMYT.

A Debug Symbol Table (e.g., such as that produced by FORTRAN compilations) may be created by using macros and pseudo-operations supplied by the GMAP Assembler. The name .SYMT., when used with the BLOCK pseudo-op, is recognized by GMAP as being a special Labeled Common region and indicated as such on the preface card of the subprogram. When GELOAD encounters this preface entry, switches are set to assign space for the loading of this special Labeled Common such that it will be overlayed by the next subprogram loaded. This table contains, when loaded, relocated values for the symbols used within the subprogram. Thus, once the information is used to fill in unknown addresses specified on the debug control cards, this table is no longer needed and may be overlayed.

When the STAB option is taken, FORTRAN generates a symbol table by use of VTAB and LTAB macros. A VTAB macro is generated for each variable named. An LTAB macro is generated for each external formula number--the number which is the FORTRAN statement number.

The system macro VTAB, supplied by GMAP, is used to generate the Debug Symbol Table within the region .SYMT.. The structure of the table may be best defined by examining the prototype of the VTAB macro, as follows:

```
VTAB MACRO
     IDRP      #2
     BCI       1,#2
     VFD       18/#2,12/0,06/#1
     IDRP
     ENDM      VTAB
```

The first argument to the macro is an octal value indicating the type code for the symbol list which appears as argument two. The legal octal values for type designation are listed below:

Type	Internal Format	Output Format
20	Binary	Octal
21	Binary Integer	Single Precision Integer
22	Real (floating point)	Single Precision Exponential (E)
23	Double Precision Real	Double Precision Exponential (D)
24	Complex	Two Exponential Fields (E)
26	Logical	Characters T or F
77	Instruction; this code is used to indicate where debug is to take place.	Octal

Example of use of VTAB macro:

```
     BLOCK     .SYMT.
     VTAB      22, (A,B,C,D)
     VTAB      77, (X,Y,Z)
     VTAB      21, E
```

Variables A,B,C, and D are entered in the Debug Symbol Table (.SYMT.) as real variables and variable E is entered as a binary integer.

Symbols X,Y, and Z indicate locations of instructions used to indicate where debug is to take place (relative addressing cannot be used with these symbols).

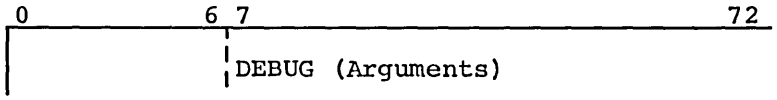
The following is the prototype of the system macro LTAB, supplied by GMAP:

```
LTAB MACRO
     BCI       1,#1
     VFD       18/#2,12/0,06/77
     ENDM      LTAB
```

#1 is the external formula number and #2 is the internal (GMAP) formula number.

DEBUG STATEMENT

The information which comprises a debug statement may be placed into three catagories. The first category is a symbolic location, necessary to determine where the debug is to take place. If debugging is to be conditional on data at execution time, the conditions are listed as the second category. The list of variables to be dumped when conditions for the debug are met is the third category. The format of the debug statement is as follows:



The DEBUG statement does not have a dollar sign ($) punch in column 1. The word DEBUG may begin in column 7 and the arguments may start in any column following the word DEBUG (blank columns are ignored). If the arguments continue to a second card, a continuation punch (any non-blank character) must be placed in column 6 of the subsequent card. All symbols punched in the DEBUG statement must be defined either by means of the VTAB or LTAB macro in the subprogram being debugged or by the existence of SYMDEF or Labeled Common symbols defined in the load table.

DEBUG statements may take any of the following five syntactical forms:

1. DEBUG m/(List)
2. DEBUG m/FOR n_1, n_2, n_3/(List)
3. DEBUG m/IF $(a_1 \pm a_2)$ K_1, K_2, K_3/(List)
4. DEBUG m/FOR n_1, n_2, n_3/IF $(a_1 \pm a_2)$ K_1, K_2, K_3/ (List)
5. DEBUG m/IF $(a_1 \pm a_2)$ K_1, K_2, K_3/FOR n_1, n_2, n_3/(List)

Explanation of the symbols used in the formats are as follows:

m — Specifies the symbolic location at which DEBUG is to take place. This would be the statement number when debugging a subprogram compiled by FORTRAN or a symbolic reference ($\pm$ addend) in a non-FORTRAN subprogram.

n_1, n_2, n_3 — Specifies limits and increment to be used with the FOR clause. The values of n_1, n_2, n_3 are integers used to control debugging of iterations of control through the FOR clause of the DEBUG statement. Taken literally, the variables indicate that debugging should occur, beginning on the n_1 iteration and every n_3 iteration thereafter until n_2 iterations have been made. If n_3 is null, it is assumed to be 1. If n_2 and n_3 are null, the debug is done only on the n_1 iteration.

($a_1 \pm a_2$) Specifies a relationship on which the IF clause depends. The symbols used are restricted to nonsubscripted variables or subscripted variables where the subscript is a constant. The variable a_2 may be a constant but a_1 may not. Some examples of this relationship are: (A-3.4), (B+C), (TIME-TIMED), (TEMP (5)-4.1). The mode of the relationship is determined by the mode of the argument a_1.

K_1, K_2, K_3 Each may assume one of four literals (NO, YES, EXIT, or DUMP) to specify the disposition of the IF statement as shown as a result of the relationship ($a_1 \pm a_2$). The action taken results from the definition of each literal. NO means do not debug here; YES means condition is satisfied so debugging may proceed; EXIT means transfer control to the EXIT subroutine which is to be loaded from the system subroutine library (L*); and DUMP means control is transferred to the DUMP subroutine which is to be loaded from the system subroutine library. In cases where the literals EXIT or DUMP are specified, GELOAD enters these symbols in its load table as SYMREF's.

The value of the relationship ($a_1 \pm a_2$) determines if the procedure associated with the literal used for K_1, K_2, or K_3 is to be involved. For the values of negative, zero, or positive the debug will respond, respectively, to the request of K_1, K_2 or K_3. If any of these literals are explicitly null, they are assumed to be NO.

(List) Specifies those variables which the programmer wishes to dump when all conditions of the statement have been satisfied. Each variable, array or directive is separated by a comma and the entire list must be enclosed in parentheses. If a Debug Symbol Table (.SYMT.) was created for the subprogram being debugged, the variable type associated with each symbol dictates the format in which the data will be dumped. If the symbol can not be found in .SYMT., or if one does not exist, it is looked up in the load table (SYMDEF or Labeled Common) and dumped in octal format.

If the symbol can not be found in either of these two tables, the list statement is in error. The list statement may include any combination of the following:

- Single-celled variable--for example, A, G, I, FAD (5). Any nonsubscripted variable or any single-dimensioned constant subscripted element of an array.

- Arrays (subscripting starts at 1)--for example, D (I_1-I_2, I_3) where I_1, I_2, and I_3 are integers and implies array D is to be dumped from elements D (I_1) to D (I_2) in increments of I_3. If written as D (I_1-I_2) the increment is assumed to be 1.

- Special--for example, OCTAL DUMP (I_1 - I_2). This requests an octal dump of memory from location I_1 to location I_2, where I_1 and I_2 may be any single-celled variable, or any symbol appearing in either the .SYMT. table or the load table (SYMDEF's and Labeled Common regions).

The following are examples of the various types of DEBUG statements with respective explanations. They are not intended to be used together as listed but are examples of the individual types.

```
$ DUMP PROG
DEBUG LOC1/(A,B,C)
DEBUG 10/FOR 1,10,2/(A,B,C)
DEBUG EN1/IF (A-10.2) NO DUMP,YES/(A)
DEBUG L1/FOR 1,50/IF(VALUE)DUMP,NO,YES/(VALUE)
DEBUG L5+2/IF(D)YES,,YES/FOR 1,10/(OCTAL DUMP (ALPHA-BETA))
```

$ DUMP PROG

The $ DUMP control card must precede the DEBUG statement cards for each subprogram being debugged. In this particular example, PROG is assumed to be the first primary SYMDEF of a subprogram.

DEBUG LOC1/(A,B,C)

This example shows the simplest form of a DEBUG statement. It calls for an unconditional print of the current values of variables A, B, and C in the format specified in the VTAB macro each time symbolic location LOC1 is executed.

DEBUG 10/FOR 1,10,2/ (A,B,C)

This is an example of a conditional debug called for at FORTRAN statement 10. The conditions indicate that on the first, third, fifth, seventh, and ninth iterations through statement 10, the values of A, B, and C will be printed. The FORTRAN compiler indicates via the .SYMT. table that the variables are type real. The STAB option must have been specified on the $ FORTRAN control card when compiling the activity.

DEBUG EN1/IF (A-10.2) NO,DUMP,YES/(A)

Another example of a conditional debug, this time using an IF clause. On each iteration through symbolic location EN1 the current value of variable A is used to evaluate the simple expression (A-10.2). If the result of the evaluation of the expression is negative, processing of the subprogram continues immediately. When the value of A is 10.2, all slave memory is dumped in octal format and the activity terminates normally. When the result of the evaluation of the expression is positive, the value of A is printed in the format specified in the VTAB macro and the execution continues.

```
DEBUG L1/FOR 1, 50/IF(VALUE)DUMP,NO,YES/(VALUE)
```

This example shows the use of compound conditions. It may be noted that an IF and FOR clause may be used together in a statement in either order but neither two IF's nor two FOR's may be used together. The effect of this statement is that for the first 50 iterations through symbolic location L1, the current value of variable VALUE is tested. If negative, a dump of slave memory terminates the run. All positive values of VALUE are printed in the format specified in the VTAB macro; zero is not printed.

```
DEBUG L5+2/IF(D)YES,,YES/FOR 1,10/(OCTAL DUMP (ALPHA-BETA))
```

This example shows how it is possible to use the debug feature at a location not flagged with a symbol. Symbol L5 must have been either specified in a VTAB macro or have been a defined SYMDEF, but the location at which debug will take place is two cells beyond L5. The statement will result in an octal dump of the memory inclusively between symbols ALPHA and BETA on each of the first 10 iterations when the value of variable D is nonzero. (Refer to "Processing of DEBUG Cards" below for instructions which cannot be used at debug location symbols.)

PROCESSING OF DEBUG CARDS

When requesting debug at load time, all DEBUG cards must be placed in front of the first object deck of the program or link. The first card of a set of DEBUG cards to be used for a given subprogram is a $ DUMP control card which specifies the deck name (first primary SYMDEF) of the subprogram being debugged. As many DEBUG statement cards follow each $ DUMP control card as are needed to define the conditions for dumping.

As each DEBUG statement card is encountered, it is scanned and reduced into tabular form, still retaining all symbolic information. These tables are generated in the high-address end of memory. If the LOWLOAD option is in effect, the tables are moved to a respective position in the low-address end of the user's memory, just above Blank Common and the slave prefix. The size of the generated tables and the additional conversion subprograms requested from the system subroutine library may cause the user's program to exceed the limits he has requested on a $ LIMITS control card for running the program without debug. A general way of estimating the size of debug tables follows:

Transfer vector (one per program or link)		3 words
FOR clause		3 words
IF clause	one argument (a1)	3 words
	two arguments (a1+a2)	5 words
Debug location (one per statement)		5 words
List entries:		
	Single cell variable	2 words
	Single cell variable without subscript	3 words
	Array	3 words
	Octal dump	3 words

As each subprogram to be debugged (name on $ DUMP cards) is loaded by GELOAD, the symbolic information in the debug table is replaced by the corresponding addresses found from either the .SYMT. table loaded with the subprogram or the load table if the symbols are either SYMDEF or Labeled Common. When a location symbol is encountered in GELOAD-generated debug tables, this implies that, during execution, the subprogram must be interrupted at this point in order to accomplish the debugging. To do this, the instruction corresponding to the location symbol is picked up by GELOAD and the operation code (bits 18-26) is saved in the debug table. The instruction is restored to memory with the operation code replaced by a Derail Instruction (DRL).

When executed, the DRL causes a fault to occur which will transfer control to the system library subroutine DEBUG. The DEBUG subprogram then tests to determine if all conditions of the statement have been satisfied and, if so, accomplishes the debug. There are certain instructions which cannot be used as debug location symbols because, at execution time, following the debug, the replaced operation codes are executed by the DEBUG subprogram in an interpretive mode. These instructions are:

RPD	STC2	DIS
RPT	XEC	Any instruction with IC modification
RPL	XED	
STC1	MME	

GELOAD tests for any of these instructions before inserting the DRL operation and, if found, prints the following message on the memory map:

AT DEBUG XXXXXX INSTRUCTION NOT LEGAL
where: XXXXXX is the location symbol.

All programs should terminate via a CALL EXIT or a CALL DUMP in order to have the debug file print out onto P* as part of the system output during wrapup. If a MME GEBORT/GEFINI termination is taken, wrapup must be indicated or else the dump file will be lost.

What is the format of the output for each DEBUG card?
What do you get from a DEBUG card?

EXAMPLE DECK SETUP

An example of a 2-subprogram input deck follows. The $ DUMP control card contains the first SYMDEF of each subprogram.

```
$  SNUMB
$  IDENT

Note: If the subprograms are to be low-loaded, insert a  $  LOWLOAD
control card here. This is the only control card  permitted  before
the $ DUMP control card.

$  DUMP    NAME1
   DEBUG   .....
   DEBUG   .....
$  DUMP    NAME2
   DEBUG   .....
   DEBUG   .....
$  OPTION FCB or FORTRAN
$  OBJECT
      ⎧
      ⎪
      ⎪
      ⎨   NAME1
      ⎪
      ⎪
      ⎩

$  DKEND

$  OBJECT
      ⎧
      ⎪
      ⎪
      ⎨   NAME2
      ⎪
      ⎪
      ⎩

$  DKEND

$  EXECUTE
$  LIMITS
      ⎧
      ⎪
      ⎩

$  ENDJOB
```

Example of debug output ??

(PS - pls pass me my example listing)

9. Memory Map Printout

The GELOAD option to print a memory map of the subprograms being loaded is considered the normal mode of operation. If no memory map printout is required, the NOMAP option on the $ OPTION control card must be specified.

The memory map printout (Figure 7) includes the following:

- Origin and name of the subprograms
- Origin and name of all primary SYMDEF's
- Origin and name of all Labeled Common regions
- All $ control cards used by GELOAD
- Error messages from GELOAD
- List of all subprograms obtained from the user's library
- List of all subroutines obtained from the system subroutine library (L*)
- Optional list of all SYMREF's by routine
- Estimate of optimum amount of memory required to run the job.

```
88861 02  09-26-68   09.061                                                    PAGE    1
(1) ORIGIN (2)061468     ENTRY LOCATION  ENTRY LOCATION  ENTRY LOCATION  ENTRY LOCATION  ENTRY LOCATION


                         SUBPROGRAMS INCLUDED IN DECK,

                         (3)$     OPTION  SYMREF,FCB
(4) 037574 (5)092668     (6)START (7)037574
   (8) BLOCK COMMON      (9)LC1  (10)037474     LC2     037434
      (11) SYMREFS       (12)XYZ


                         SUBPROGRAMS OBTAINED FROM SYSTEM LIBRARY,

(4) 037360 (5)082068     (6)'SETU (7)037565
      (11) SYMREFS       (12)'FLTPR
   037346    SDL#12      .FLTPR  037346

                                          RANGE                    SIZE
                         (13) ALLOCATED CORE     000000 THRU 037777     040000
                              OBJECT PROGRAM
                              RELOCATABLE        037344 THRU 037777     000434
(14) *** NON FATAL ERROR - MISSING ROUTINE  XYZ

                         (15) FCB AND BUFFER SPACE

                              AVAILABLE          000101 THRU 037343     037243
                              FILE CTRL BLKS     037214 THRU 037344     000131
                              MAXIMUM BUFFER SPACE REQUIRED             001200

                         (16) 2K, IS THE MINIMUM MEMORY REQUIRED FOR THIS JOB WITH ALL FILES OPEN

                         (17) EXECUTION   PROGRAM ENTERED AT     037574

                         (18) THERE WERE  000001 WARNING FLAGS IN THE ABOVE LOAD
```

1 Heading line

2 Assembly date of this version of GELOAD

3 Control card printout

4 Subprogram load origin

5 Assembly date of subprogram being loaded

6 Name of subprogram being loaded (SYMDEF)

7 Address of SYMDEF within subprogram

8 Indicates assignment of Labeled Common regions assigned for this subprogram

9 Labeled Common region name

10 Labeled Common region assigned address

11 Indicates loaded subprogram references another subprogram (printout due to use of SYMREF option)

12 Name of subprogram referenced

13 Summary of core memory used in loading subprograms

14 Error diagnostic; indicates routine XYZ not found

15 Summary of core memory used for file control blocks and buffers

16 Indicates optimum core memory limits for this activity. (If no additional memory required for running, this number may be placed on $ LIMITS card for optimum run efficiency.)

17 Address where activity execution begins

18 Summary count of diagnostic messages issued for this load

Figure 7. Memory Map Printout

10. Error Messages

Errors within GELOAD are of two types--fatal (F) and nonfatal (N). Execution is inhibited by the occurrence of any fatal error. A nonfatal error does not halt execution under normal conditions. However, if the programmer wishes to execute his program only if no errors occur, he must specify the GO option on the $ OPTION control card. Figure 8 lists all the errors recognized by GELOAD, the actions taken to correct them, and the messages printed out on the user's execution report, following the memory map with which they are associated.

NOTES:

1. If an error is caused by an illegal binary-coded decimal (BCD) card, the contents of the card are printed with the message.

2. If an error is caused by a binary card, only the contents of card columns 73-80 are added to the message printout.

TYPE	MESSAGE	CAUSE	ACTION
N	AT DEBUG XXXXXX IF NOT IN DICTIONARY	IF variable undefined	Statement ignored
N	ILLEGAL BINARY CARD	Illegal card type encountered	Deck ignored
N	CONTINUATION NOT EXPECTED	New debug card expected --col. 6 not blank	Card ignored
F	$ ENTRY NAME NOT LOADED	Name on $ ENTRY card undefined	Immediate return to monitor
F	ARGUMENT 1 NOT DEFINED	$ EQUATE Name1 undefined	Remainder of card ignored
N	ARGUMENT 2 DEFINED PREVIOUSLY	$ EQUATE Name2 undefined	Remainder of card ignored
F	ORIGIN ILLEGAL ON FIRST $ LINK CARD	First $ LINK card has a second field	Second field ignored
F	FIELD DEFINED PREVIOUSLY	Field of $ USE card previously defined	Field ignored - scan continues
N	INCONSISTENT FIELD-PREFACE CARD	Labeled Common without size or SYMREF with size specified	Deck ignored
F	ILLEGAL LOAD ADDRESS	Attempting to load above program	Deck ignored

Figure 8. List of Error Messages

TYPE	MESSAGE	CAUSE	ACTION
F	ORIGIN ILLEGAL	Origin on LINK card currently undefined	No origin assumed
N	ILLEGAL CHECKSUM	Checksum incorrect	Card loaded
N	MISSING ROUTINE XXXXXX-MME INSERTED AT REFERENCES	Undefined name	*Name undefined for this segment of load
F	END-OF-FILE READING CONTROL CARD	Premature EOF on input file	Immediate return to monitor
F	END-OF-FILE READING BINARY	Premature EOF on input file	Immediate return to monitor
F	LOAD TABLE AND PROGRAM OVERLAP	Insufficient Memory available	Immediate return to monitor
F	COMMON AND PROGRAM OVERLAP	Insufficient Memory available	Loading continues when loading from a library file
N	LOADER SKIPPING TO NEXT BCD CARD	Binary card out of place	Read and ignore cards until a BCD card is encountered
N	NONLOADER CONTROL CARD IGNORED	BCD card without $ in column 1	Card ignored -- processing continues
N	PREVIOUS DEBUG CARD TERMINATED INCORRECTLY	Continuation card expected	Previous DEBUG list deleted
N	ILLEGAL IF CLAUSE	Illegal character in clause	Card ignored
N	ILLEGAL FOR CLAUSE	Illegal character in clause	Card ignored
N	ILLEGAL LIST CLAUSE	Illegal character in clause	Card ignored
N	ILLEGAL CLAUSE	Illegal delimiter encountered	Remainder of card ignored
N	AT DEBUG XXXXXX SYMBOLIC REFERENCE NOT IN DICTIONARY	Location of debug request undefined	Statement ignored
N	AT DEBUG XXXXXX INSTRUCTION	The instruction at XXXXXX cannot be simulated by the DEBUG subroutine	Request ignored
N	LABELED COMMON XXXXXX SIZE INCONSISTENT	Labeled Common defined previously with smaller size	Initial size retained
F	PROGRAM HAS EXCEEDED TOP OF MEMORY	Insufficient memory available under LOWLOAD	Return to monitor

* A TSX7 X is placed at all references. IR7 in the panel dump points to the instruction having the undefined SYMREF. The two instructions at S are LDQ = 3HOL1,DL and MME GEBORT.

Figure 8. List of Error Messages (cont.)

TYPE	MESSAGE	CAUSE	ACTION
N	ILLEGAL CONTROL CARD	Non-Loader control card appearing before $ EXECUTE	Read next card
N	MAXIMUM NUMBER OF LIBRARIES	More than 10 Library File Codes on $ LIBRARY cards	Terminate table and continue
N	ENTRY NAME NOT LOADED	Link or program ENTRY cannot be found in Load Table	Continue using no entry with link
N	COMMON ALREADY ASSIGNED	Use of LOCOMN option after assignment of BLANK COMMON	Assign as Labeled Common
N	XXXXXX ILLEGAL IN ALGOL TABLE	Name xxxxxx is not defined to ALGOL	Continue table scan
N	ETC ONLY LEGAL FOLLOWING FFILE	ETC card out of sequence	Ignore card and continue
F	TOO MANY FILE CODES	Number of file codes exceeds 30	Return to monitor
N	FILE CODE USED MORE THAN ONCE	Same file code used on more than one FFILE card	Ignore first reference to file code
F	FFILE TABLE OVERFLOW	Table used for encoding of $ FFILE cards too small	Return to monitor
N	FILE CODE XX NOT DEFINED	File code XX appeared on an $ FFILE card followed by IGNORE but did not appear on a FILE card previously	Continue processing file cards
N	TWO LGU FIELDS	LGU field redefined	Ignore 2nd field
N	RECORD LENGTH REDEFINED MIXED	Record length field previously defined	Redefine as mixed length records
N	RECORD LENGTH REDEFINED FIXED	Record length field previously defined	Redefine as fixed length records
N	ILLEGAL FFILE FIELD	Illegal clause on $ FFILE card	Ignore field
F	LGU TABLE OVERFLOW	Too many logical units have been specified	Return to monitor

Figure 8. List of Error Messages (cont.)

11. File Control Block Generation

GELOAD may be requested to generate file control blocks (FCB) for the user by listing the FCB option on the $ OPTION control card. (This option is automatically set by listing either the FORTRAN or the ALGOL options on the $ OPTION control card).

When this option is set, normal loading takes place until the $ EXECUTE control card is encountered on R* (GELOAD system input file). At this time, the file control block generator overlay of GELOAD is called and executed. First, any GECOS file control cards or $ FFILE control cards following the $ EXECUTE control card are scanned and the information tabulated. File control blocks will only be generated for GECOS file control cards containing numeric file codes with values less that 44 for all $ FFILE specified file codes, and for files I* (Input) and P* (Output). Only one file control block will be generated when the same file code appears on a GECOS file control card and a $ FFILE control card. When all control cards following the $ EXECUTE have been scanned, file control block generation begins. Beginning at the next available load address, an area of memory (currently set at 22 words but variable on reassembly of GELOAD) is set aside as the Logical Unit Table (LGU). This area is used during execution by FORTRAN, ALGOL, and JOVIAL I/O library subprograms and contains pointers to all generated file control blocks. Each file control block is generated following this table from the encoded data taken from the file cards. Twenty-two words are used to form each block; 20 words comprise a standard file control block and 2 words are used as control by the I/O library subprograms. A standard file control block is generated (see description of standard file control block in GE-625/635 File and Record Control, CPB-1003) unless the files are described by $ FFILE control card. For descriptions of the pointer (location 25) to the LGU table, the structure of the LGU table, and the relationship of numeric file codes to system files, refer to GE-625/635 FORTRAN IV I/O Library, CPB-1137 and GE-625/635 FORTRAN IV, CPB-1006.

12. Octal Correction Cards

GELOAD has the capability of processing octal corrections to a relocatable object subprogram deck when supplied according to the conditions and formats given in the following paragraphs. Corrections are made in memory by the use of one or more octal correction cards. These cards are inserted in the object deck immediately preceding the $ DKEND control card.

The use of octal corrections is restricted in that they may not be used to overlay words in the object subprogram which contain undefined SYMREF's. Until an undefined reference (SYMREF) is defined by an object subprogram containing its respective SYMDEF, an address chain is maintained through all memory locations which reference the given symbol. The contents of a correction card over one of these chained addresses causes a break in the chain at that point. Thus, when the symbol is defined and the definition is being stored at each of the referencing locations, all those references appearing after the point of the corrective patch will not be filled in with the correct address.

The octal correction card has the following format:

Columns	Contents
1-6	Address in octal - this address is relative to the beginning of the subprogram and may be punched anywhere in the field. Non-numeric characters in this field are ignored.
7	Blank
8-12	The word OCTAL
13-15	Blank
16-72	The corrections in the form of one or more subfields are as follows: If only one subfield is specified, it will replace the contents of the word whose address is specified in columns 1-6. Multiple subfields, which must be separated by commas, replace successive words starting at the address specified.

A subfield may contain up to twelve octal digits. If less than twelve are specified, they will be right-justified, with leading zeros inserted. A null field (,,) constitutes one word containing zeros.

Each subfield may be prefixed and/or suffixed by the letter R. Prefix R will cause the left-most 18 bits of the word to be modified relative to the program load point. Suffix R results in the equivalent modification to the right-most 18 bits. Absence of the letter R will imply no modification to the respective 18-bit field.

GELOAD terminates the scan of subfields of octal correction cards when either a blank column or column 72 is encountered. Following the terminal blank column, comments may be added.

References

GE-625/635 GMAP Implementation, System Support Information, CPB-1078

GE-625/635 Comprehensive Operating Supervisor, Reference Manual, CPB-1195

GE-625/635 File and Record Control, Reference Manual, CPB-1003

GE-625/635 FORTRAN IV I/O Library, System Support Information, CPB-1137

GE-625/635 FORTRAN IV, Reference Manual, CPB-1006

GE-625/635 System Editor, Reference Manual, CPB-1138

GE-625/635 General Loader, System Support Information, CPB-1127

Index

DOCUMENT REVIEW SHEET

TITLE: GE-625/635 General Loader

CPB #: 1008E

From:

Name: _____

Position: _____

Address: _____

_____

Comments concerning this publication are solicited for use in improving future editions. Please provide any recommended additions, deletions, corrections, or other information you deem necessary for improving this manual. The following space is provided for your comments.

COMMENTS: _____

Please cut along this line

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

Fold on two lines shown on reverse side, staple, and mail.

STAPLE

STAPLE

FOLD

FIRST CLASS
PERMIT, No. 4332
PHOENIX, ARIZONA

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

GENERAL ELECTRIC COMPANY
PROCESSOR EQUIPMENT DEPARTMENT
13430 NORTH BLACK CANYON HIGHWAY
PHOENIX, ARIZONA 85029

ATTENTION: Program Documentation C-78
Systems and Processors Operation

FOLD

INFORMATION SYSTEMS

GENERAL ELECTRIC

LITHO U.S.A.