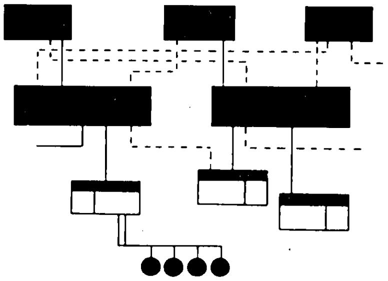
THE COMPATIBLES/600

GE-600 Series GELOAD General Loader

SYSTEM
SUPPORT
INFORMATION



ABSTRACT

This document describes the Compatibles/600 General Loader. Divided into three phases, GELOAD reads and generates a table from debug control cards, provides logic for linking subprograms, and generates file control blocks required by the user.

GENERAL ELECTRIC

GELOAD

GENERAL LOADER

May 1965

GENERAL ELECTRIC

COMPUTER DEPARTMENT

CONTENTS

INTRODUCTION

GELOAD is a general purpose loader with the ability to load absolute or relocatable programs which may be considered in three phases or overlays. Phase 1, which is optional, reads and generates a table from debug control cards supplied with the user's deck. This information will be used at object time, to give snapshot printouts of specified locations within the program during execution. Phase 2, the loader itself, provides the logic for linking subprograms together, reserving storage for data regions, calling in subroutines from established libraries, and segmenting the relocatable program into loadable overlays. Phase 3, also optional, generates the file control blocks required by the user as specified on file control cards.

PHASE 1

1. Entered by reading loader control card: $ DUMP.

2. Read debug statement cards.

3. Encode information from cards and generate debug table.

4. Enter names of library subroutines used by debug in load table.

PHASE 2

1. Read loader control cards and set conditions accordingly.

2. $ OBJECT card indicates a subprogram is to be loaded from the R* input file.

a. Generate load table from preface cards.

b. Assign entry location, if not given.

c. Pick up and assign proper relocation to instructions and data of object subprograms until $ DKEND card is encountered.

d. Fill in debug tables, if applicable, with any references within subprogram just loaded using special table loaded as LABELED COMMON.

e. The debug reference table above will be overlayed by the next subprogram loaded.

f. Return to read next control card.

3. $ SOURCE card indicates a subprogram is to be loaded from the B* file. (Starts with $ OBJECT and reads until $ DKEND encountered then searches the $ DKEND parameters to determine if reading goes back to R* or C from B*.

4. $ LINK card causes subprograms already loaded to be saved on a file for later chain overlay processing.

5. $ EXECUTE card signals GELOAD that no additional object programs have been supplied by the user. If Phase 3 is required, it will be called when this card is encountered.

6. Available libraries are searched for undefined subprograms.

7. All remaining undefined subprogram references are filled in with MME GEBORT.

8. All GELOAD files are closed.

9. Unused allocated core is reset and released to GECOS.

10. Transfer to user's entry point.

PHASE 3

1. Read GECOS $ FILE cards and $ FFILE control cards. from R*.

2. Build table of file codes and file control block parameters from the cards.

3. Generate a file control block for:

   a. Each file code mentioned in a FFILE card.

   b. Each numeric file code from a GECOS file card which is less than 44.

   c. I* and P* files.

4. No file control blocks will be generated only if:

   a. There are no FFILE control cards and all file codes on GECOS file cards are nonnumeric.

   b. The option NOFCB is requested in a $ OPTION control card.

5. All file control blocks will be generated in the unused memory (as specified by cell 31 of the user's fault vector) at the completion of loading.

TABLE FORMAT - LOAD TABLE

- Location: Immediately follows GELOAD

- Type of Entries:

SYMDEFs:

```
L       BCI     1,NAME
L+1     ZERO    X,Chain
```

where:

```
X       =0 when NAME is undefined (SYMREF)
        =L(NAME) when NAME has been defined
Chain   =L(First reference to NAME).  When all
         references are filled in, this field is zero.
```

SYMREFs with addends: (NAME+4)

```
L       VFD     6/77,12/A,18/L(NAME  in
                Load Table)
L+1     ZERO    Addend L(Reference to NAME)
```

where:

```
A       =1: Addend applied to lower 18 bits of word.
        =0: Addend applied to upper 18 bits of word.
```

LABELED COMMON:

```
L       BCI     1,NAME
L+1     ZERO    L(NAME)
L+2     VFD     H6/$,12/0,18/(Size of LABELED
                COMMON)
L+3     ZERO    L(NAME)
```

TABLE FORMAT - DEBUG TABLE

- Location: Loaded in reverse order at current loading address
  when $ DUMP card is encountered.

- Transfer vector

  Depending on the control cards used, the subroutine DEBUG and two optional subroutines DUMP and EXIT may be called from the library file. A three-word transfer vector at the beginning of the table provides linkage with the user's program.

- Standard entries

  Five standard entries are generated for each set of DEBUG conditions.

```
L   ZERØ     T1,L(DRL)
L-1 Replaced Instruction (Operation code only)
L-2 BCI      1,NAME (Routine)
L-3 BCI      1,SYMBØL (Location for test of conditions)
L-4 ZERØ     0, addend (Applies to SYMBOL)
```

where:

T1 points to beginning of next encoded set of debug conditions.

L(DRL) is location in user's program where return is to be made after debug is complete and the replaced instruction has been executed.

- IF clauses (3-word minimum; 5-word maximum entry)

  IF clauses are encoded into the Debug Table in a variable format depending upon the arrangement of variables, subscripts, constants and null fields within the conditional description (set of parentheses) of the clause. A control word (location L of the table entry) contains bits further describing the type and content of table entries L-1 through L-4:

  L VFD Ø6/77,1/0,1/A,1/B,1/C,2/D,2/K_1,2/K_2,2/K_3,18/0

  where the above codes imply the following memory contents:

Code	Implies
A = 0	Mode is fixed
A = 1	Mode is floating
C = 0	Variable operator is add
C = 1	Variable operator is subtract
K_1,K_2,K_3 = 0	No
= 1	Yes
= 2	EXIT
= 3	DUMP

Code	Location	Before Program Loaded	After Program Loaded
B = 0	L - 1	BCI 1,VAR1	ZERØ L(VAR1),0
B = 1	L - 1	ZERØ 0,subscript	ZERØ 0,subscript
	L - 2	BCI 1,VAR1	ZERØ L(VAR1),0

(In the following description, the location symbol "n" will be used to illustrate the variable nature of this table entry, such that for B = 0, n = L - 2 and for B = 1, n = L - 3)

D = 0	n	BCI 1,VAR2	ZERØ L(VAR2),0
D = 1	n	ZERØ 0,subscript	ZERØ 0,subscript
	n - 1	BCI 1,VAR2	ZERØ L(VAR2),0
D = 2	n	DEC constant	DEC constant
D = 3		second variable is null	

- FOR clauses

L	VFD	06/76/30/0
L-1	ZERØ	n_2, n_1
L-2	ZERØ	0 , n_3

Where n_1, n_2, n_3 imply debug for count from n_1 to n_2 in increment of n_3.

- Variable list

1. Single Cell (No subscript) (VAR

	Before program loaded	After program loaded
L	ZERØ	VFD 18/LØC,12/-1,6/Type
L-1	BCI 1,VAR	BCI 1,VAR

2. Single Cell (subscripted) (VAR(5)) i_1 is the computed location relative to the array orig

L	ZERØ i_1, i_1	ZERØ i_1, i_1
L-1	VFD 18/0,12/1,6/0	VFD 18/LØC,12/1,6/Type
L-2	BCI 1,VAR	BCI 1,VAR

3. Array (VAR(5,20,[illegible])) i_1 = [illegible] i_2 = [illegible] increment

L	ZERØ i_2, i_1	ZERØ i_2, i_1
L-1	VFD 18/0 12/i_3,6/0	VFD 18/LØC,12/i_3,6/Type
L-2	BCI 1,VAR	BCI 1,VAR

4. Octal Dump (OCTAL DUMP (NAME, NAME(2))) SUB1 = 0 SUB2 = 2

L	ZERØ SUB1,SUB2	ZERØ maximum loc. minimum loc.
L-1	BCI 1,$NAME_2$	VFD 18/0,12/1,6/16
L-2	BCI 1,$NAME_1$	DEC -1

Where:

i_1,i_2 and i_3 are integers specifying the variables from i_1 to i_2 in increments of i_3.

SUB1 and SUB2 are possible subscripts of symbols NAME1 and NAME2 respectively.

TABLE FORMAT - FFILE CARDS

```
L     VFD          18/L(Next Segment),6/0,H12/File Code
L+1   VFD          18/Retention Period,18/Bits to be set in LØWER
                   of LØCSYM
L+2   VFD          18/Buff. Size,18/Bits to be set in LØWER of
                   LØCSYM-5
L+3   VFD          18/PREHED,18/0
L+4   VFD          18/PØSHED,18/0
L+5   VFD          18/PRETRL,18/0
L+6   VFD          18/PØSTRL,18/0
L+7   VFD          18/ERRXIT,18/0
L+8   VFD          18/MIXLNG,18/FIXLNG
L+9   VFD          6/LU1,6/LU2, . . . 6/LUn
```

Notes:

1. The lower 18 bits of words L+1 and L+2 are set only in the event of corresponding fields on the FFILE cards. For example:

   Bit 25 of LØCSYM-5 (Bit 25 of L+2) is set only if the field MLTFIL is present on the FFILE card.

2. In word L+8, either the upper or the lower condition will be set but not both. The upper 18 bits of the word is the location of SYMDEF supplied as the name of the subroutine to handle mixed-length records. The lower 18 bits of the word is the length, in number of words, of each record.
3. Word L+9 may be repeated as required to include all logical unit numbers specified on the FFILE card. The LGU's are packed up to six per word, with zeros filling in unfinished words.
4. Table is terminated by a word of zeros with the pointer in L indicating it to be the next segment.

PHASE 1 CONTROL - DBG

1. Phase 1 is entered on reading $ DUMP control card.

2. Macro EDBGT is used to make entries into debug table. Location of entry is in index 7 which is decremented and compared with the load table address to check for overlap. Argument #1 is the entry.

3. Test LOWLOAD. If not requested, set current loading address in index 7 and go to step 5.

4. If LOWLOAD, set highest allocated core address (MXCR) in index 7 and lowest available address (MLDADD) in LDTCL to be used to test for debug-table/program overlap.

5. Set flag indicating presence of debug statements.

6. Save beginning address of debug table for memory map printout.

3-word table linkage

7. First cell of table is used as transfer vector for DEBUG subroutine from the library file. Two additional cells are also reserved for possible extension of vector to include linkage to subroutines DUMP and EXIT.

8. Set transfer to beginning of debug transfer vector in symbolic location TWO, for later storage in cell 13_{10} of the slave ~~user's fault~~ pref ~~vector~~ (the DRL fault vector ~~cell~~).

9. Get name of program being debugged from $ DUMP card.

ignore $ card

10. Read next card. If not DEBUG card, then error.

11. Skip one cell to be used as a pointer

    (a) Address of next segment of table (upper)

    (b) Address plus one of instruction when DRL is inserted (lower)

12. Save this table location as beginning of current segment.

13. Skip one cell in table for storage of operation code of instruction where DRL is inserted.

14. Enter symbolic name of program.

15. Get symbol or statement number where debug is to take place and enter it, with possible addend, in table.

16. Test characters for possible IF clause. If not go to step 24.

17. Test if IF clause is legal. If not, print message and reset table entry address to overlap any entries pertaining to illegal clause. Then go to step 36.

18. Set flag for IF clause.

19. Enter control word (770000000000). See table format for breakdown of bit codes.

20. Enter variable names and subscripts involved in IF clause.

21. Set appropriate bits in control word for conditionals, YES, NO, DUMP, or EXIT.

22. If DUMP or EXIT is used, enter symbols in transfer vector. (See step 7 above.) Later during loading these will be defined by subroutines from the system library.

23. If statement contains another clause go to step 16, otherwise go to step 29.

24. Test characters for possible FOR clause. If not, go to step 29.

25. Test if FOR clause is legal. If not, print error message and reset entry address to overlay entries pertaining to this statement. Then go to step 36.

26. Set flag for FOR clause.

27. Enter control word (760000000000).

28. Enter n_2, n_1 and n_3 in the next two cells respectively, then go to step 16.

29. Test for left parenthesis as beginning of list. If not, this is an error. Print message and skip to next debug statement causing the deletion of the entire statement containing the error.

30. Set flag for list.

31. Beginning of list entries is signaled by a word of zeros in table.

32. List is scanned and entries are made according to the table format description defined earlier.

33. Final right parenthesis ends list.

34. One complete set of conditions has been encoded into the table.

35. Set pointer (upper 18 bits of word at step 11) to next unused cell below table.

36. Read next control card.

37. Test card:

    a. If $ DUMP card, go to step 9.

    b. If not $ DUMP card, go to step 11.

    c. If other control cards, close table with word of zero.

38. If HIGHLOAD go to step 40.

39. For LOWLOAD, debug tables are generated at high end of allocated core just as they would be for HIGHLOAD. They are then moved, with their relative positions unchanged, to the low end of allocated core. Where, in relation to blank common?

40. Table is still stored in reverse order; that is, transfer vector at high end and word of zeros signaling end of table at low end.

41. Enter the symbolic names, DEBUG, DUMP, and EXIT (the latter two only if required in IF clause) in load table as SYMREFs with reference pointers to the relative positions in the transfer vector.

42. TRA instructions are inserted in place of the symbolic names previously in the vector.

43. Load Phase 2.

PHASE 2 CONTROL

1. Save first unused core location to be used in print of storage map.

2. Read loader control card (subroutine RMCD).

3. Isolate type field in card and compare with list of possible types.

4. Transfer to subroutine associated with control card type.

   Note: Controls for card types other than $ OBJECT are described beginning at step 53.

5. $ OBJECT indicates one subprogram is to be loaded from the R* file using subroutine LOAD.

6. On entry to LOAD all necessary buffer references are set to the file being loaded.

7. Read preface card using subroutine PCRD.

8. On return from PCRD, symbolic location CURF contains the address of card image and LOADT4 is the tally on which individual words are picked up.

9. Size of the load table is increased by four times the number of entries on the preface card to handle the maximum possible entries. The temporary reference table is located immediately following load table for this subprogram.

10. An entry is taken from the preface card and tested for being one of the following types. The table below indicates where the control for each type is described.

Type	Step
SYMDEF	11
SYMREF	16
LABELED CØMMØN	19
DEBUG SYMBØL TABLE CØMMØN	25

11. Enter SYMDEF symbol in load table with entry location in upper 18 bits of cell following symbol.

12. Enter pointer, to cell containing entry location, in temporary reference table.

13. If this is first SYMDEF entry to load table, save as possible entry point to program.

14. Increment counters for load table, reference table and preface card entries.

15. If more entries on preface card go to step 10. Otherwise go to step 27.

16. Enter SYMREF symbol in load table with a word of zeros in the cell following symbol.

17. Enter pointer to cell containing zeros in temporary reference table.

18. Increment preface card entry counter. If more entries, go to step 10. Otherwise go to step 27.

19. Enter LABELED COMMON symbol in load table.

20. Enter beginning address of LABELED COMMON in upper 18 bits of cell following symbol.

21. Two additional entries flag this as a LABELED COMMON entry.

22. Enter size of LABELED COMMON region in lower 18 bits and $ as first character of next sequential cell of load table.

23. Fourth cell in this group of entries is same as second cell (see step 20 above).

24. Enter pointer to cell following symbol in temporary reference table. Then go to step 18.

25. No entries are made in load table for the debug symbol table.

26. For debug symbol table, enter pointer in temporary reference table to cell containing location of the first unused cell beyond the program and LABELED COMMON. This will ensure that the symbol table is always loaded outside the area reserved for instructions. The next subprogram to be loaded may then overlay the debug symbol table. Go to step 18.

27. Get V-bit count from preface card and store in appropriate shift command to be used later in special relocation.

28. Print storage map , if requested by user.

29. Move SYMREF and LABELED COMMON entries of the temporary reference table to working area. For LOWLOAD, this area is the top of available memory and for normal loading it is just below the last allocated subprogram or LABELED COMMON region. This is done to allow additional entries such as SYMREF's with addends to be made in the load table during loading of the subprogram.

30. Read binary program card using subroutine BCRD.

31. Set counters and storage constants from control word (word 1 of binary card):

    a. Relative loading address in X3

    b. Tally word LOADT2 used to pull words from card image.

    c. X6 and X7 to pick up relocation bits.

32. Load instruction or data word into Q-register.

33. Isolate relocation bits for this word.

34. Go to respective subroutine for relocation according to bit pattern for upper and lower 18 bits of word being stored.

    Note: The following subroutines are called for relocation of the appropriate 18-bit segment of the word. The main flow of description is continued with step 45.

35. Absolute entries (Type 00) are loaded directly into memory after the address specified has been tested as being within the limits of allocated core. Go to step 32.

36. Normal relocation (Type 01) adds the relocation constant to the respective 18 bits and stores the word in its proper core location with respect to the beginning of the subprogram being loaded. Go to step 32.

37. BLANK COMMON relocation (Type 10) adds the relative beginning address of the BLANK COMMON region to the respective 18 bits and stores the word in core with respect to the beginning of the subprogram or data region. Go to step 32.

38. Special relocation (Type 11) requires the use of the V-bit shift which was set up in step 28.

39. Bits 2 through V+1 of the 18-bit segment of the word being relocated are isolated to determine which entry in the reference table will be used for this relocation.

40. Using pointer in reference table, determine if symbol referred to has been defined. If so apply addend to entry location associated with symbol and go to step 43.

41. The symbol is not defined and is being referred to with addend or in the lower 18-bits of the word making reference. This condition causes the following entry in load table.

```
        VFD     6/77,12/LOW,18/L(Symbol in load table)
        ZERO    Addend, L(cell making reference)
where:  LOW=1   Reference is in lower 18 bits
        LOW=0   Reference is in upper 18 bits
```

42. The symbol is not defined and referred to directly in the upper 18 bits of the word making reference. This condition alters the original SYMREF entry in the load table (see step 16). The contents of the lower bits of the cell following the symbol (in the load table) replaces the upper bits of the word being loaded. The location of the word being loaded is set in the lower 18 bits of the load table entry. If several such entries occur before the symbol is defined, a chain is formed which is terminated by zeros in the upper of the <u>first</u> reference to the symbol.

43. When both 18-bit card segments have been relocated, the word is stored in memory relative to its subprogram origin.

44. Return to step 32 until all instructions or data words have been loaded from the card image.

45. A $ DKEND control card signals the end of a subprogram deck. Under certain conditions, a $ DKEND control card may contain additional control information in its variable field.

<u>Variable</u>	<u>Control</u>
CONTINUE	Signifies a batch compile or more than one subprogram to be loaded from the B* file before returning control to R*. The current file is tested and if this control card was not read from B*, the CONTINUE is ignored.
LODER:	This is an ALGOL control which allows for further processing of the load table before additional subprograms are loaded.

46. When the entire subprogram and its LABELED COMMON regions have been loaded, the presence of debug statement cards with this job is tested. If none, return to step 2 to read next monitor control card.

47. In Phase 1, a table was generated from the debug statement, symbolic identifiers were stored in the table indicating specifically where in the source program snapshot dumps were to occur and what was to be dumped.

48. The debug statement table is searched for statements pertaining to the subprogram just loaded.

49. The debug symbol table (if one was loaded with the subprogram is searched).

50. If no symbol table exists, the load table of SYMDEF and LABELED COMMON entries is searched.

51. As defined, each of the symbols in the debug statement table is replaced with an address. Operation codes of the instructions at the locations where debug is to take place, are replaced with the DRL instruction after testing the operation codes against a table of codes which cannot be simulated by the DEBUG program.

52. Variable and locations which are still undefined at the end of the search are flagged with error messages. If only one variable in a list is undefined, the DRL is removed from the instruction and the operation code is replaced.

Control Cards

Type	Step Number
$ ENTRY	53
$ EQUATE	55
$ EXECUTE	56
$ LIBRARY	57
$ LINK	58
$ LØWLØAD	59
$ ØPTIØN	62
$ RELCØM	63
$ SØURCE	64
$ USE	66
$ NØLIB	70

At the conclusion of processing of each control card, control returns to step 2.

53. Get NAME of entry from ENTRY card.

54. If processing a link overlay, store symbol as a link entry; otherwise store it as an entry to main program. Then return to step 2.

55. a. Variable field of $ EQUATE card is scanned for first symbol.

    b. If break character is not a /, control goes back to step 4.

    c. The load table is searched for this symbol. If it is yet undefined, a fatal error message is printed and control goes back to step 2.

d. Test possible increment to symbol. An addend may be applied to a symbol which has been defined.

e. Resume field scan for symbols to be defined.

f. Test each symbol for previous definition in the load table. If previously defined, print nonfatal message and redefine.

g. If symbol being used as definition is that of BLANK COMMON (.CMN.), the symbol being defined will be entered in the load table as a LABELED COMMON region and given a defining address within BLANK COMMON. The size of this region will be filled in later.

h. Each set of equations is enclosed in slashes (/) and separated by commas. The last set is terminated by a blank column. Equation fields may not be continued on additional cards but additional $ EQUATE cards may be used.

56. a. $ EXECUTE, a GECOS control card, signals the end of program loading. All final housekeeping necessary for closing Phase 2 is begun.

b. Available libraries are searched for any routine yet undefined.

c. The storage map is printed.

d. The load table is searched for chained references and undefined references. Chained references are completed in the CHN subroutine.

e. Remaining references to undefined SYMREFs are filled in with:

```
        TSX7    .MSNG.
```

where:

```
       .MSNG. LDQ  =3HØL1,DL       ABØRT CØDE
              MME  GEBØRT          ABØRT
```

The .MSNG. coding is placed in the next two available locations following loading of the user's program.

(The user may include his own subroutine .MSNG. on his library if a different procedure is preferred.)

f. If any fatal error messages have been printed, loading and execution are deleted at this point.

g. The entry to the user's program is determined and given to the setup subroutine (.SETU.). (The setup subroutine is optional and may differ between user's of FORTRAN, COBOL, ALGOL, etc.) The priority of entry is determined as:

(a) $ ENTRY card

(b) FORTRAN Standard Entry (......)

(c) First SYMDEF placed in load table from preface card.

h. If job was loaded under LOWLOAD control, the unused memory above the program is cleared.

i. Place address of bounds of the unused memory in cell 31 of the user fault vector.

j. Correct addresses in the user's debug table (if one exists) relative to his own base zero.

k. Close B* and L* files.

l. Go to Phase 3 unless NOFCB option was requested, in which case the R* and P* files are closed and entry is made into the user's program via the setup subroutine.

57. a. Use GFLD subroutine to obtain library file codes from $ LIBRARY card.

b. Enter file codes in table in order in which they appear on card.

c. L*, the system library, is a permanent entry in the table.

58. a. The $ LINK control card signals the end of the current link and the beginning of a new one.

b. A link is concluded much the same as a separate program in that libraries are searched and a storage map is printed.

c. $ LINK cards without origins (that is, first link and links which are not overlays) are handled as follows:

(a) Enter link ID in load table with defining address.

(b) Print $ LINK card.

(c) Read next control card. If a $ DUMP card, load Phase 1; if not, go to step 1, Phase 2.

d. $ LINK cards with origin fields are handled as follows:

(a) Enter link ID in load table with defining address.

(b) Get origin ID and search load table for definition. An undefined origin is an error.

(c) Test NOPAC option. If set as the third field on the card, a flag is turned on so that the load table will not be purged of references to SYMDEF's located within the link being overlayed.

(d) Fill in chained references and undefined references within the link being overlayed.

(e) Write links being overlayed on H* file using MME GESAVE.

(f) If NOPAC option is not set, purge all SYMDEF's and references to SYMDEF's of the overlayed link from the load table.

(g) The new load address (the new origin) is set to continue loading.

(h) Print $ LINK card.

(i) Read next control card. If $ DUMP card, load Phase 1; otherwise go to step 1, Phase 2.

59. $ LOWLOAD card is used to initialize the loading address at the lower end of allocated core.

60. When a decimal address is punched in the card, the initial loading address is incremented by this amount. This is to allow for BLANK COMMON.

61. All constants are initialized with respect to the lower memory loading address. Then return to step 2.

62. The following table lists cells affected by $ OPTION card. Underlined options are assumed if no $ OPTION card is included in the deck.

Option	Cell in Memory	Contents	Effect on Loading
MAP	MAP	Zero	Produce memory map
NØMAP	MAP	Nonzero	No memory map printed
CØNGØ	ERXEQ	Zero	Execute regardless of nonfatal loading errors.
GØ	ERXEQ	Nonzero	Execute only if no loading errors
NØGØ	XEQ	Nonzero	No execution
SET/n/	SETC	=n	Core will be preset to value n which may be up to 12 octal characters. (Normal contents of SETC; n=0)
ERCNT/n/	MWRCT+3	=n	Maximum number of error messages allowed. (Normal contents = 150)
SYMREF	PREFF	Nonzero	Print SYMREF's on loader map.
NØSREF	PREFF	Zero	Do not print SYMREF's.
LØCØMN	LLCMF	Nonzero	Assign all LABELED COMMON below BLANK COMMON
NØSETU	SETF	Nonzero	Do not load the setup subroutine ,SE from the system library. Enter the user program directly.
NØFCB	NØFCBF	Nonzero	Do not call Phase 3 of GELOAD. No file control blocks will be generated by GELOAD.
ALGØL	ALGF	Nonzero	Set LOWLOAD option, prepare for the LODER: field on the $ DKEND card, ,ALODR and set FCB options.

What happens for less than 12 — left or right just, zeros, blanks? what?

Option	Cell in Memory	Contents	Effect on Loading
FCB	NØFCB	Zero	Call PHASE 3 of GELOAD and generate file control blocks according to the control cards supplied.
FØRTRAN	NØFCBF	Zero	Sets all standard loader options for the FORTRAN user. These include highload, FCB, and the use of the standard setup routine (.SETU.).
CØBØL	NØFCBF	Nonzero	Sets all standard loader options for the COBOL user. These include LOWLOAD, NOFCB, and the use of the COBOL setup routine (.CSETU).

63. Convert decimal number in $ RELCOM card and increment beginning address of BLANK COMMON by this amount.

64. The $ SOURCE card will cause one subprogram to be loaded from the B* file by the procedure in steps 6-52.

65. Deviation from the above procedure occurs only when loading object subprograms generated by the TASMIN or G compilers. In these cases it is possible to encounter a $ DKEND control card with the word CONTINUE in the variable field. This would signal GELOAD to load another subprogram from the B* file. The CONTINUE option is present only when more than one source subprogram is batch compiled and a single $ SOURCE card appears on the R* file.

66. $ USE causes symbolic names to be entered in the load table as undefined SYMREF's.

67. When a symbol in the variable field is terminated by either a blank or a comma (break characters) it is entered in the next available location of the load table followed by a cell of zeros signifying it is an undefined SYMREF.

68. When a symbol is terminated by the slash (/) break character, the symbol is a LABELED COMMON entry and the number following the slash is converted as the size of the COMMON region.

69. A LABELED COMMON entry of the form:

```
          $ USE   NAME/nL/
```

implies that the LABELED COMMON NAME of size n, is to be assigned in an area below BLANK COMMON or just above the user's fault vector. If any BLANK COMMON has already been assigned, a nonfatal error message is printed, the field is ignored and the next field is considered.

70. $ NOLIB causes no library search to be made during the loading. In link jobs, a $ NOLIB card must be included with each link for which a search is not to be made.

PHASE 3 CONTROL - GENERATE FILE CONTROL BLOCKS

1. Determine limits of user's memory still available after loading program (cell 31).

2. Read file control card. (GECOS type, FFILE
Where from? from R* !!!

3. If card is not a file card (LIMITS, ENDJOB, etc.) print it and go to step 2.

4. Test type of file card.

   a. GECOS--Only file code is saved in a table.

   b. FFILE--All information on this card is pertinent to the construction of the FCB.

   c. ETC--Only applicable to FFILE and is considered an error if it appears elsewhere.

5. The file codes are carried from the GECOS cards in BCD. Each new code is compared with all other entries for duplication. A nonfatal error message is printed when duplication exists.

Note:
Steps 6 thru 9
deal with FFILE
Card handling

6. Each of the possible fields on the FFILE card is encoded into a respective position in a 10-word segment of the table pertaining to this FFILE card. The format of each segment of the table is as described earlier under Table Formats.

7. Each field is picked up from the card image by the GFLD subroutine. The card is scanned until it is terminated by a blank column or the end of the card (column 72).

8. When a card is terminated, the next card is considered. If it is an ETC card, the procedure in step 7 is continued; otherwise, control goes to step 4.

9. The respective fields are handled as follows:

Field	Action
STDLBL	Set bit 24 in word L+2 of table segment.
NBUFFS	Test format correct. Convert number using CVT subroutine. Shift bits into proper position. OR bits into word L+2 of table segment.
BUFSIZ	Test format correct. Convert size using CVT subroutine. OR into upper of word L+2.
LGU	Test if field appeared previously (error). Test format correct. Initialize tally for storing logical unit numbers. Convert numeric file code. Convert unit numbers Store in table segment. Repeat until list is complete.
PREHED POSHED PRETRL POSTRL ERRXIT	Test format correct. Get SYMDEF using GFLD subroutine. Search load table for SYMDEF. Store definition in respective location of table.

Field	Action
MIXLNG	Test format correct.
	Test FIXLNG already set. If so, reset.
	Get SYMDEF using GFLD subroutine
	Search load table
	Store definition in table.
RETPER	Test format correct
	Convert number
	OR into upper 18 bits of word L+1
MLTFIL	Set bit 25 in word L+2 of table
MODBCD	Set bit 21 in word L+1 of table
FIXLNG	Test format correct
	Test MIXLNG already set. If so, reset.
	Convert record length using CVT subroutine.
	Store length in lower of 18 bits of word L+8 of table.
NOSRLS	Set bit 23 of word L+2 of table.
LODENS	Set bit 22 of word L+1 of table.
IGNORE	Reinitialize beginning of table.
	Search table of GECOS file codes for match.
	If match exists, delete code in table.
	Delete is accomplished by replacing code with 7777
	If match is not found, print error message.

10. When all cards have been scanned (signaled by EOF on R* or encountering $ ENDLD card), control is transferred to the generator part of Phase 3.

Generated by GEIN when the end of activity on jrt is determined

present on Object Library file input to GELOAD under System File Editor control.

11. Up to this point, all temporary tables have been set up outside of the user's memory.

12. The logical unit table and all FCB's will be generated in the user's unused memory and will be loaded in the same manner as additional subroutines.

13. Area (22 words) is assigned for the logical unit table.

14. Test present tables (GECOS file codes and FFILE) to determine which, if any, FCB's will be generated.

15. FCB's for FFILE file codes are generated first. (steps 16 - 20 are FCB for FFILE card.)

16. The standard FCB is assigned 22 words--20 words for the FCB itself and 2 words working areafor system library usage.

17. The table of GECOS file codes is searched for the current code and if present, the code is deleted from the table.

18. Begin generating FCB, filling in options as stated on FFILE card and encoded into table segment.

19. All options not expressed on FFILE card are assumed to be standard options.

20. Logical units are transferred into the logical unit table using a tally modification.

    The format of the logical unit table is:

    ```
    VFD  18/FCB, 6/LU1, 6/LU2, 6/LU3
    ```

    using as many cells as are required to assign all logical unit numbers. FCB is the address of the LØCSYM.

21. When FCB's have been generated for all FFILE cards, the table of GECOS file codes is checked for presence of numeric file codes which are numerically less than 44.

22. Standard FCB's are generated for all numeric file codes less than 44 which appeared on GECOS file cards.

23. The file code is converted to binary and entered in the logical unit table for this FCB.

24. Steps 22 and 23 are repeated until FCB's are generated for all numeric file codes.

25. If to this point, FCB's have not been generated for P* and I* files, they are generated now.

26. When no GECOS File Control cards and no FFILE cards are present with the job, two standard FCB's for P* and I* will be generated. If only GECOS File Control Cards are present, no FCB's are generated. Control goes to step 29.

27. All necessary FCB's have been generated.

28. Calculate new limits of the unused memory and store them in cell 31 of the user's fault vector.

29. Return to GELOAD COMMON to close R* and P* files.

30. Test if memory is to be released back to GECOS.

31. Transfer control (to .SETU routine or) to user's program.

presets parameters in .SETU then releases memory (GEMRELS) + returns control to .SETU which clears out only the hole + blank common + then sets up the slave prefix before turning control over to entry symbol.

MAJOR SUBROUTINES COMMON TO ALL PHASES

FUNCTION	CALLING SEQUENCE	REMARKS
Read Monitor Card	L TSX1 RMCD L+1 TRA (EØF) L+2 Normal	Reads BCD card and checks for $ in column 1. Uses RBCD subroutine.
Read BCD Card	L TSX1 RBCD L+1 TRA (EØF) L+2 Normal	Reads Card and checks for BCD. If not, error message is printed and read continues until BCD card is found. Blank cards are ignored.
Print BCD Card	L TSX1 PBCD L+1 Normal	Writes one card image from the R* file onto the P* file. Uses MWR subroutine.
Message Writer	L TSX1 MWR-CØDE*2 L+1 } See MACRØ expansion L+n } of the calling sequence	Writes message on P*. Return and type code are specified as part of the macro.

FUNCTION	CALLING SEQUENCE	REMARKS
Get Field	L-1 LDA A,DU L TSX1 GFLD L+1 Normal	A is a composite mask of legal break characters. Characters are picked up using GCHR and packed in FLD until a break character is found. The field is in FLD and the break character number is in X2 upon return.
Convert Fixed-Point Numbers From Cards	L-1 LDA A, DU L TSX1 CVT L+1 Return with number in CVTNM	A is a composite mask of legal break characters. Numerics are converted until a non-numeric is encountered. An illegal break character will cause an error message.
Enter Symbol In Load Table	L-1 LDA (Symbol) L TSX1 ETBL L+1 Return - In and Defined L+2 Return	Uses SLTB to search load table. If symbol is already in table and defined, return is at L+1. If symbol is already in table but not defined, then return is at L+2 with

FUNCTION	CALLING SEQUENCE	REMARKS
		zero indicator on. If symbol is not in table, it is entered as undefined and the return is at L+2 with zero indicator off.
MAJOR SUBROUTINES OF PHASE 1		
Get Character From Card Image (Pass 1)	L TSX1 GCHR L+1 Return	Loads one column from a BCD card image and isolatesit in the rightmost character position of the A-register. Ignores blanks.
Read Debug Card	L TSX1 RDCD L+1 Continuation Card Return L+2 $ Card Return L+3 Normal Return	Reads card using RBCD. Dollar sign card may be an error. Continuation card has punch in column 6. Tally is set for scan of debug statement.

MAJOR SUBROUTINES OF PHASE 2

FUNCTION	CALLING SEQUENCE	REMARKS
Get Character From Card Image (Pass 2)	L TSX1 GCHR L+1 Return	Loads one column from a BCD card and right adjusts it in the A-register. Blanks are included.
New Limits	L TSX1 NLMT L+1 Return	Calculates limits of storage allocated to this program or link but unused by it. On return, the A-register has lowest unused cell address in the upper 18 bits and highest unused cell address in the lower 18 bits.
Load User's Libraries	L TSX1 ELNK L+1 Return	Picks user library file codes from list and searches them in turn. If user has not specified any libraries, only the system library, L*, is searched.

FUNCTION	CALLING SEQUENCE	REMARKS
Print Storage Summary	L TSX1 PMAP L+1 Return	Prints amount and bounds of storage used for object program, COMMON and debug tables.
Define SYMREF	L TSX1 CHN L+1 Return	Searches load table for undefined SYMREF's. Fills in chain addresses if any exist. Calls MRT to handle missing routines.
Missing Routine	L TSX1 MRT L+1 Return	Fills in address of MME GEBORT subroutine at all references to missing routines.
Convert Octal For Printout	L-1 LDQ (NUMBER) L TSX1 CVOCT L+1 Return	Converts 6-digit octal address for printing. Converted number is returned in the A-register.
Print Storage Map	L TSX1 SMAP L+1 Return	Prints names and locations of entries to the subroutine being loaded. Also prints names and locations of LABELED COMMON Regions and SYMREF's (optional).

FUNCTION	CALLING SEQUENCE	REMARKS
Search Load Table	L-1 LDA NAME L TSX1 SLDT L+1 Return (In) L+2 Return (Not In)	Searches load table in 256-word blocks. If return is to L+1, index register 2 points to location in load table.
Make Entry In Load Table	L-1 LDAQ (ENTRY) L TSX1 ENTB L+1 Return	Makes entry in next two available locations of load table. A fatal error message is printed when table overlaps with user's program.
Set Reference Table Location	L TSX1 SREFT L+1 Return	Gets size of reference table from preface card and calculates a storage location relative to load table.
Search Symbol Table	L-1 LDA NAME L TSX1 SYTB L+1 Return (Not In) L+2 Return (In)	Uses logic of SLDT (above) to search debug symbol table. Q-register contains location of symbol when return is to L+2.

FUNCTION	CALLING SEQUENCE	REMARKS
Load Absolute Card	L TRA ABS (Return to read next card)	Loads one absolute binary card. Checksum is calculated.
Read Preface Card	L TSX1 PCRD L+1 Return	Card type is checked. Appropriate messages are printed if card is not preface card. ABS subroutine is entered if card is absolute. Tally words are initialized for preface cards.
Read Binary Card	L TSX1 BCRD	Picks a card image from buffer and checks type. Illegal cards cause message to be printed. BCD cards are checked for $ DKEND, signaling end of binary deck.
Compute Checksum	L TSX1 CKSUM L+1 Return	Zero checksums are ignored. Message is printed when checksum on card does not compare with that calculated. Loading of

FUNCTION	CALLING SEQUENCE	REMARKS
		card with illegal checksum continues unchanged.
Get Load Address	L-1 LDA (Length) L TSX1 GLD L+1 Return	Calculates new loading address by applying the length (A-register) of the subprogram or data region to the current load address. Tests are made for the possibility of overlap with load table and top of memory. Counters concerning the highest and the lowest cells used are maintained by GLD.
Enter SYMREF In Load Table	L-1 LDAQ (Entry) L TSX1 SRF L+1 Return	1. Makes normal entry in load table: BCI 1, NAME ZERO 0, Pointer 2. If entry is a LABELED COMMON region instead of SYMREF, two additional words are entered:

FUNCTION	CALLING SEQUENCE	REMARKS
		VFD6#/$,12/0,18/(Length) ZERO Loc, 0 3. Entry is made in reference table pointing to the second cell of the load table entry.
Enter Primary or Secondary SYMDEF In Load Table	L-1 LDAQ (Entry) L TSX1 SETY L+1 Return	1. Make standard entry in load table: BCI 1, NAME ZERO Loc, 0 2. Make entry in reference table pointing to the second cell of the load table entry.

EFFECT OF LOWLOAD OPTION

Cell or Control Affected	HIGHLOAD	LOWLOAD
LDADD	Contains loading address of current subroutine being loaded. Calculated by subtracting length of subprogram from previous load address.	Same as HIGHLOAD except address is calculated by adding length of previous subprogram to previous load address.
MLDADD	Contains address of first cell above user's allocated memory.	Contains address of first unused cell above subprogram being loaded.
LLDADD	Contains address of first unused cell below subprogram being loaded.	Contains address of first unused cell above subprogram being loaded (same as MLDADD).
BLDADD Base Load address	Contains address of lowest cell used by user's program. (Not BLANK COMMON)	Same as HIGHLOAD.
LWLD	Contains zero.	Contains base load address for user's program. This cell is used as a flag signaling LOWLOAD option.

Cell or Control Affected	HIGHLOAD	LOWLOAD
BASE	Base address of user's allocated memory.	Same as HIGHLOAD.
MXCR	Upper limit of user's allocated memory.	Same as HIGHLOAD.
Location of Reference Table (Relative to current sub-program)	Stored downward beginning at address contained in LLDADD.	Stored downward beginning at top of user's memory.
Location of Debug Tables	Generated downward from highest available location in user's allocated memory.	Generated as in HIGHLOAD and moved to a position relative to user's lowest loading address. (Above BLANK COMMON region.)

DEFINITIONS OF TERMS FREQUENTLY USED

Primary SYMDEF - Symbolic location in the user's subprogram which denotes an entry point to the subprogram.

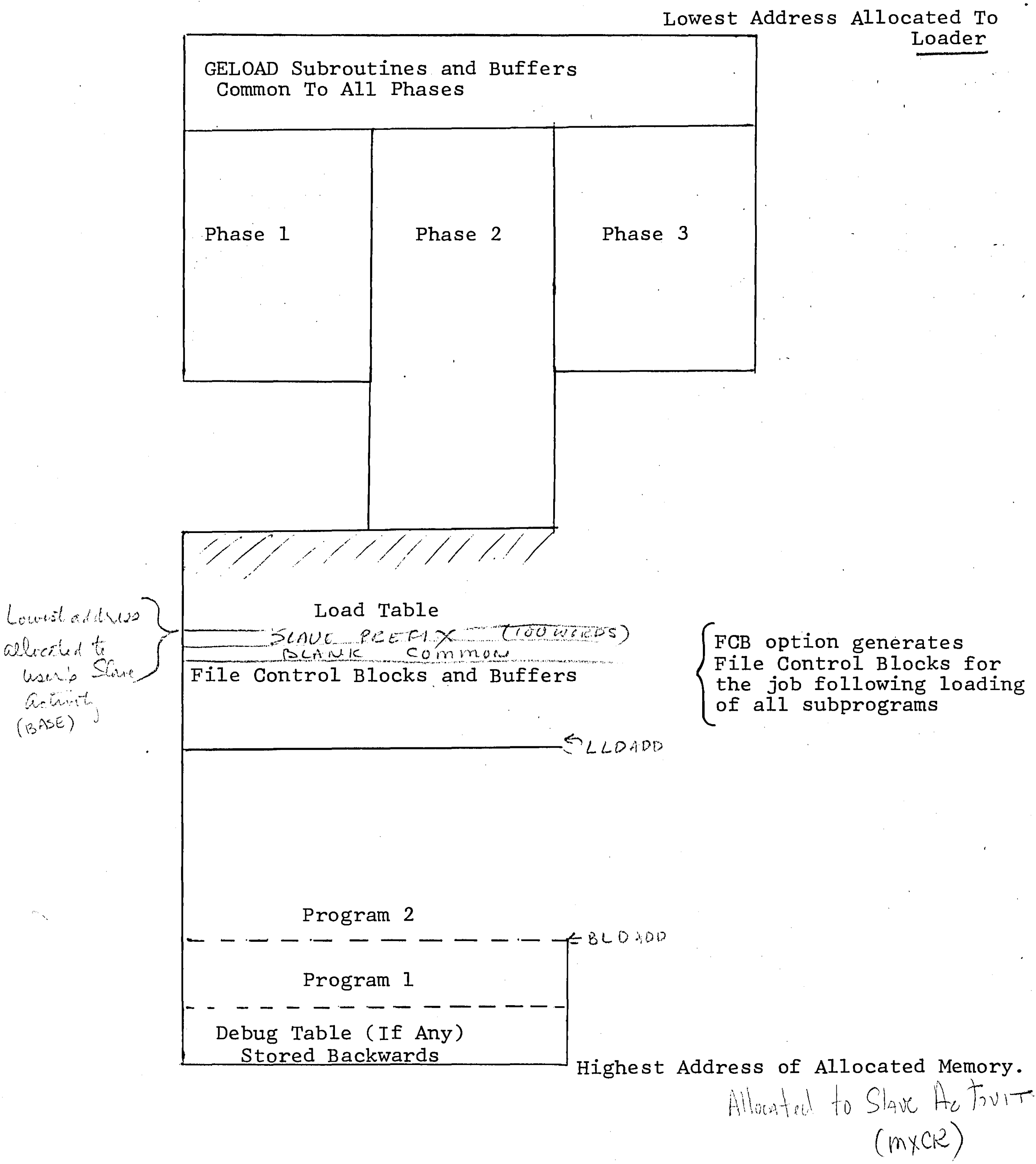
Secondary SYMDEF - Symbolic location in the user's subprogram to be referred to from outside the subprogram but which is not necessarily an entry point.

SYMREF - Symbols referred to within the subprogram which are defined externally as SYMDEFs in other subprograms.

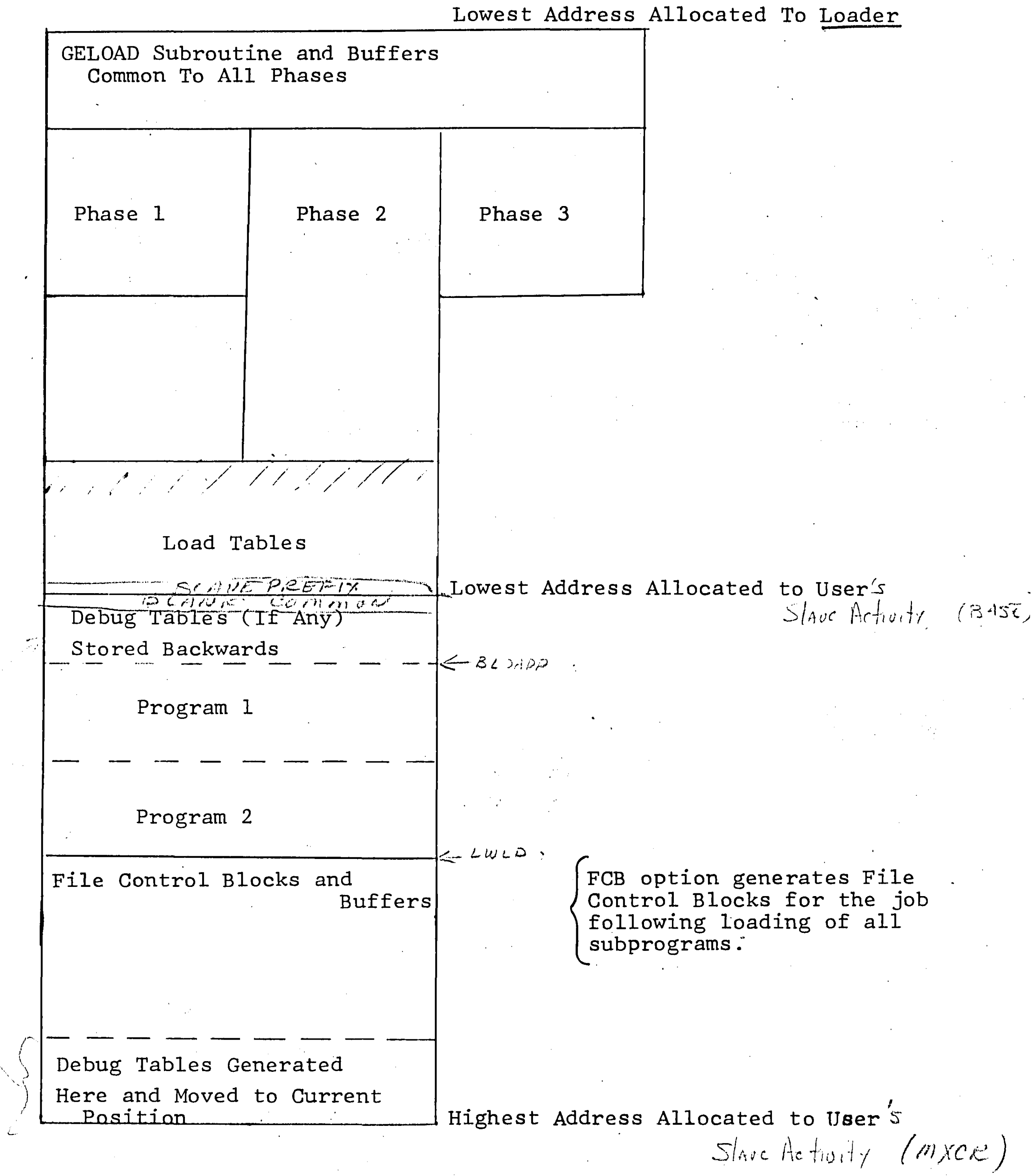
LABELED COMMON - Common regions which are loaded with the subroutines and referred to by name. Entries are made in the load table to enable referencing by any subprogram.

BLANK COMMON - The COMMON region located just above the user's fault vector at the low end of his allocated memory. BLANK COMMON is referenced by absolute addressing relative to the user's base address register.

MEMORY DIAGRAM FOR HIGHLOAD


Lowest Address Allocated To Loader
GELOAD Subroutines and Buffers Common To All Phases
Phase 1
Phase 2
Phase 3
Lowest address allocated to user's Slave Activity (BASE)
Load Table
SLAVE PREFIX (100 WORDS)
BLANK COMMON
File Control Blocks and Buffers
FCB option generates File Control Blocks for the job following loading of all subprograms
LLDADD
Program 2
BLDADD
Program 1
Debug Table (If Any) Stored Backwards
Highest Address of Allocated Memory.
Allocated to Slave Activity (MXCR)

MEMORY DIAGRAM FOR LOWLOAD


Lowest Address Allocated To Loader
GELOAD Subroutine and Buffers Common To All Phases
Phase 1
Phase 2
Phase 3
Load Tables
SLAVE PREFIX
BLANK COMMON
Lowest Address Allocated to User's Slave Activity (BASE)
Debug Tables (If Any) Stored Backwards
← BLDADR
Program 1
Program 2
← LWLD
File Control Blocks and Buffers
Debug Tables Generated Here and Moved to Current Position
Highest Address Allocated to User's Slave Activity (MXCE)

FCB option generates File Control Blocks for the job following loading of all subprograms.

Progress Is Our Most Important Product

GENERAL ELECTRIC

COMPUTER DEPARTMENT • PHOENIX, ARIZONA

LITHO IN U.S.A.