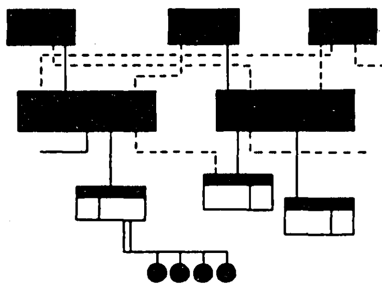
GENERAL ELECTRIC
COMPUTERS

GE-625/635 GMAP Implementation

SYSTEM
SUPPORT
INFORMATION



ABSTRACT

This manual describes the inner workings of GMAP, the General Macro Assembly Program for the GE-625/635. It presumes a familiarity with GMAP usage on the part of the reader.

GENERAL ELECTRIC

CPB-1078B

GE-625/635 GMAP IMPLEMENTATION

May 1965

Rev. August 1966

GENERAL ELECTRIC

INFORMATION SYSTEMS DIVISION

PREFACE

This publication has been prepared for the system programmers who will be maintaining and modifying GMAP. It presents a detailed breakdown of Pass 1 and Pass 2 control logic, a complete description of all pseudo-operations, and a bit-by-bit description of all GMAP control words and tables. It is assumed the programmer is familiar with the GE-625/635 Comprehensive Operating Supervisor (GECOS) Manual, CPB-1002.

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 Engineering Publications Standards, B-90, Computer Equipment Department, General Electric Company, 13430 North Black Canyon Highway, Phoenix, Arizona 85029.

CONTENTS

1. INTRODUCTION

The General Macro Assembly Program (GMAP) for the GE-625/635 produces an object program (and printouts) by processing the symbolic coding of the source program. The latter may be a GMAP symbolic program or may have been produced by COBOL or FORTRAN.

GMAP operates in two passes. Pass 1 processes the input file, forms the symbol table, and writes an intermediate file. Pass 2 processes the intermediate file and forms the output files, including the listing and binary information.

The following is a summary of Pass 1 and Pass 2 functions. Figure 1 shows core memory allocation for GMAP Pass 1 and 2.

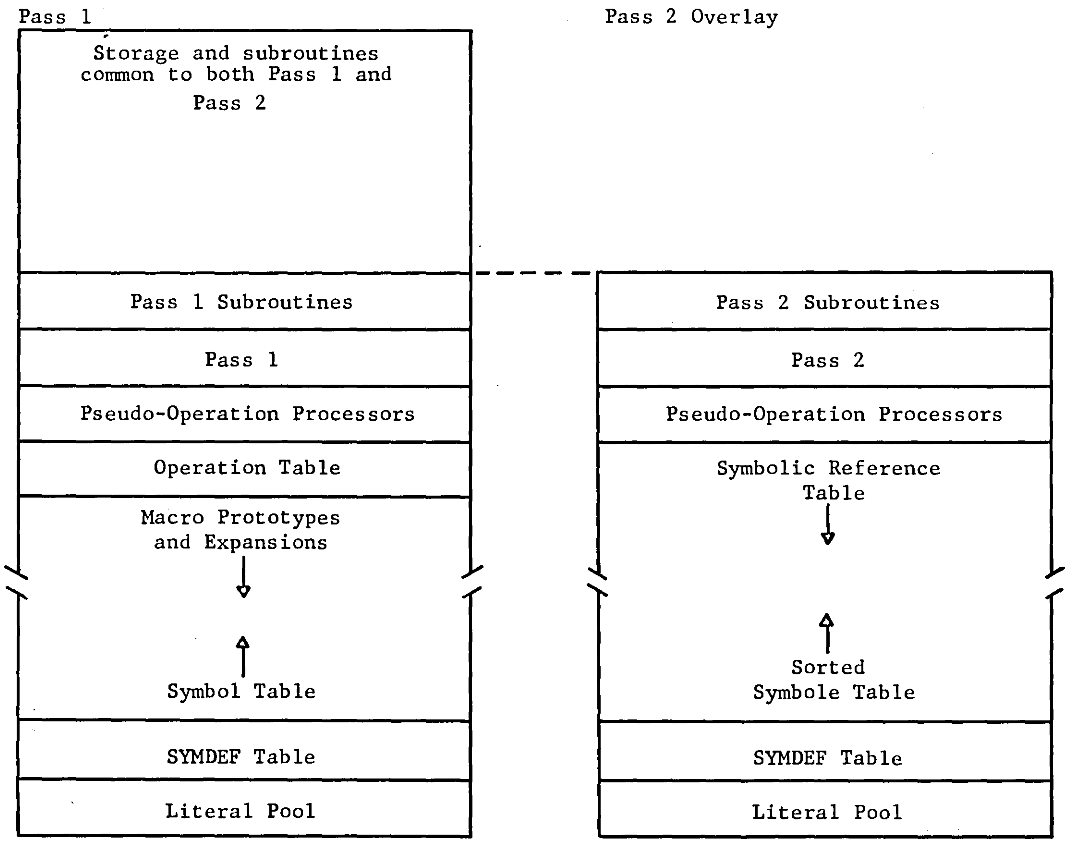


Figure 1. Core Memory Allocation for GMAP

PASS 1

1. Location symbols are placed in the symbol table along with their definitions.

2. The operation code is looked up in the operation table and the operation control word passed on to Pass 2. Pseudo-operations requiring Pass 1 processing are processed.

3. Literals that can be evaluated in Pass 1 are converted to binary and placed in the literal pool. Literals M, V, and nH, where $n > 12$, are not processed until Pass 2.

4. Macro definitions are entered in the macro prototype table.

5. Card images required by DUP and macro expansions are produced.

6. Tables are formed from information supplied by certain pseudo-operations (USE, BEGIN, SYMDEF, BLOCK, LIT).

Housekeeping at End of Pass 1:

7. The USE tables are processed in conjunction with the BEGIN information to determine the origin of each USE.

8. The location of the literal pool, the error linkage, and the program break are computed.

9. The symbol table is sorted; the symbols are given their true definitions based on the origin of their associated USE; and the table is checked for multi-defined symbols which are flagged.

10. The LIT table is processed to set the origins of each literal pool based on the origin of the USE under which the LIT occurred.

11. The values for the preface are computed.

12. Pass 2 is called.

PASS 2

1. The preface is punched and listed for relocatable programs.

2. The binary deck and listing are produced using information from the intermediate file and the tables built by Pass 1.

3. The symbolic reference table is formed.

PASS 1 CONTROL LOGIC

1. Initialize GMAP table locations and I/O routines. Read GMAP system macros.
2. Read a record; go to step 9. (G*)
3. If not in DUP mode, go to step 7.
4. If not the first time through the range of the DUP, go to step 6.
5. Save record for succeeding times through DUP range. Strip location symbol if not a SET and go to step 7.
6. Write intermediate file, retrieve next record from those previously saved, and go to step 10.
7. Write intermediate file. (*1)
8. If expanding a macro, get next record from macro processor and go to step 10.
9. Move record into working storage and read next record.
10. If any records are to be skipped, reduce count and go to step 9.
11. Set up controls for processing the record.
12. If processing a macro prototype, pack record in prototype storage and go to step 3.
13. Look up operation code and, if a pseudo-operation, go to appropriate processor.
14. Enter location symbol in the symbol table.
15. Increase location counter, process literal if it exists, and go to step 3.

PASS 2 LOGIC

1. Punch system macros, if required. (GECOS assembly only)
2. List and punch preface, if required. (P* & C*)
3. Read a record from the intermediate file. (*1)
4. If not a BCD card to be punched by the DCARD pseudo-operation, go to step 6.
5. Punch and list BCD card and go to step 3.
6. If pseudo-operation, go to appropriate processor.
7. Check location symbol for phase error.
8. If a literal is present, get address and go to step 12.

9. If I/O-type instruction, go to step 17.

10. Evaluate symbolic index, if required.

11. Evaluate address field, if present.

12. Assemble operation code.

13. Evaluate tag field, if present.

14. List and punch instruction; increase location counter.

15. If literal is <u>not</u> to be assembled at this point, go to step 3.

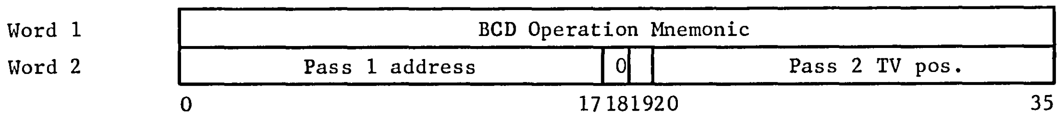
16. Assemble required literal and go to step 3.

17. Assemble I/O-type instruction word and go to step 14.

2. TABLE FORMATS

OPERATION TABLE

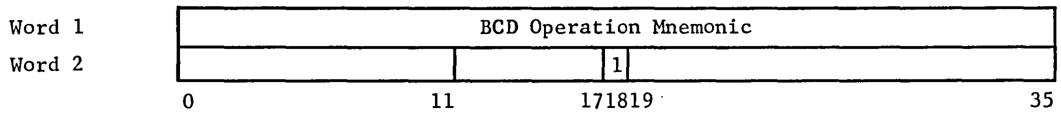
Pseudo-Operation



- <u>Word 1.</u> Left-adjusted and filled with blanks
- <u>Word 2.</u>

<u>Bits</u>	
0-17	Address of Pass 1 pseudo-operation processor
18	0 indicates this is a pseudo-operation
19	0, but is set to 1 if the operation is redefined
20-35	Position in the transfer vector giving the addresses of Pass 2 pseudo-operation processors

Machine Instruction



- <u>Word 1.</u> Left-adjusted and filled with blanks
- <u>Word 2.</u> Word 2 has two formats, depending on bit 19.

A) If bit 19 is 0 (machine instruction):

<u>Bits</u>	
0-11	Binary machine code
12-17	Unused

Bits (Cont'd)

18	1 indicates machine instruction
19	0 format indicator
20-25	0 indicates yes, 1 indicates no
20	Register modification allowed
21	Indirect addressing allowed
22	Indirect and tally allowed
23	DU allowed
24	DL allowed
25	SC and CI allowed
26-30	1 indicates yes, 0 indicates no
26	Address required
27	Address required even
28	Address required absolute
29	Symbolic index required
30	Octal tag field required
31-33	Not used
34-35	Listing format for binary word

```
00      XX   XXXX   XXXXXX
01      XXXXXXXXXXXX
10      XXXXXX   XXXXXX
11      XXXXXX   XXXX   XX
```

B) If bit 19 is 1 (I/O command):

Bits

0-17	Bits 18-35 of command
18	1 indicates machine instruction
19	1 format indicator
20-25	Bits 0-5 of command
26-28	Same as machine instruction

Bits (Cont'd)

29-30 I/O type

Type	Variable Field	Binary Word
00	DA,CA	XXDACAXXXXXX
01	NN,DA,CA	XXDACAXXXXNN
10	CC,DA,CA	XXDACAXXCCXX
11	A,C	AAAAAAXXCCCC

31-33 Not used

34-35 00 Listing type 0

SYMBOL TABLE

Word 1	BCD name (leading zeros)	
Word 2	relative address	control information
	0	1718 35

- **Word 1.** BCD symbol, right-adjusted with leading 0's
- **Word 2.**

Bits

0-17 Relative address

18 Pass 1 0

Pass 2 1 if a reference has been made (see symbolic reference table)

19 To be defined (see FEQU pseudo-operation)

20 Defined by SET pseudo-operation

21-23 Relocation type

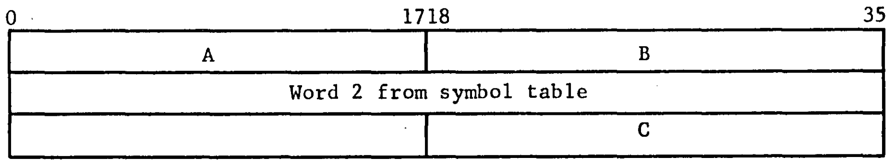
000	Absolute
001	Relocatable
010	Blank common
011	Block
100	SYMREF

Bits (Cont'd)

24	Multidefined	
25-26	Not used	
27	0	USE
	1	BLOCK
28-35	USE or BLOCK number	

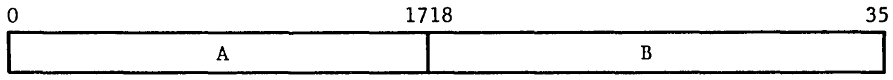
SYMBOLIC REFERENCE TABLE

First entry for symbol



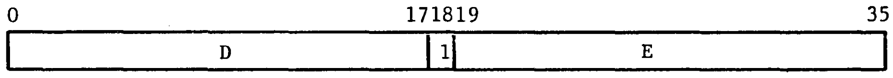
A	Address of next reference; or 0 if last reference
B	Alter number of reference
C	Alter number of definition

Second and succeeding entries



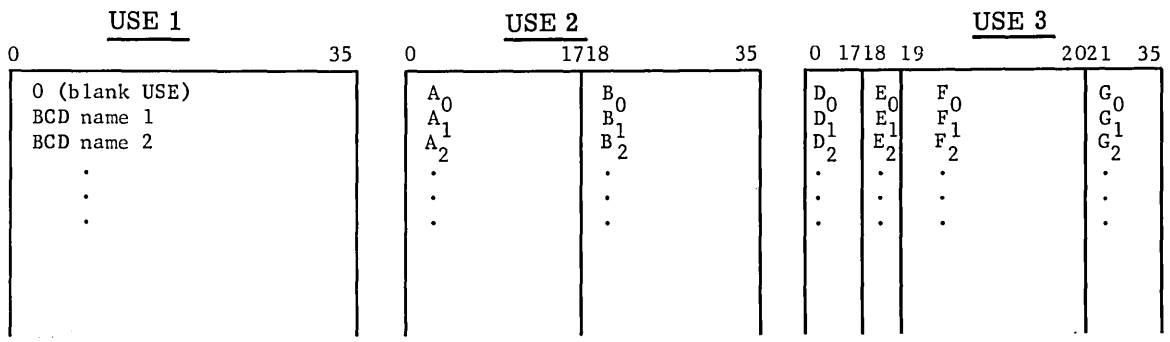
A,B Same as above

When the first reference to a symbol is made, word 2 of the symbol table is replaced by a word with the following format:



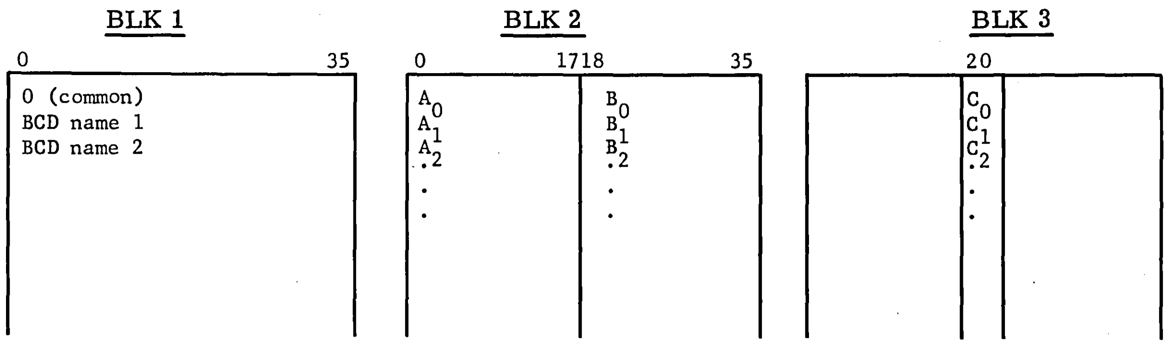
D	Address of first entry in reference table for this symbol
E	Address in the reference table of word 2 of the symbol table

USE TABLES



A_i	Next location available for this counter
B_i	Pass 1--largest location used by this counter
	Pass 2--origin of this counter
D_i	Relative address of BEGIN for this counter
E_i	0 no BEGIN for this counter
	1 BEGIN occurred for this counter
F_i	00 counter may begin in any location
	10 counter must begin in even location
	01 counter must begin at a multiple of eight
G_i	USE number under which relative address of BEGIN is defined

BLOCK TABLES

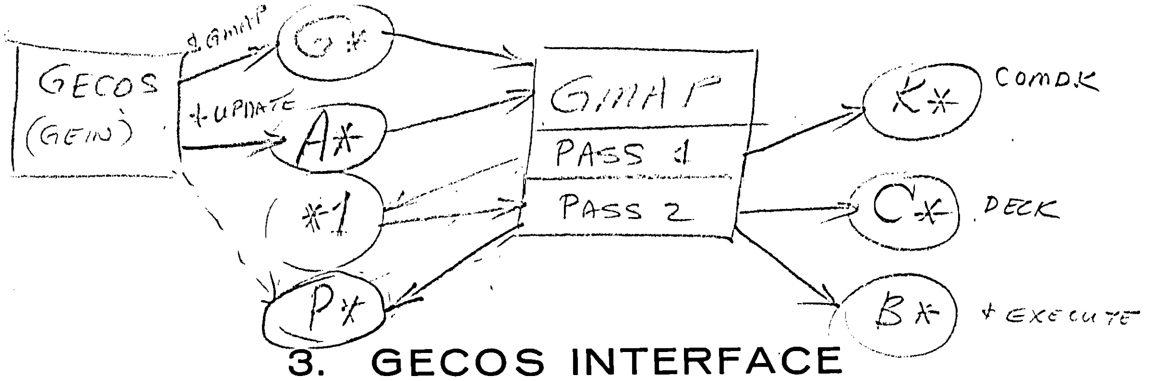


A_i	Next available location of this block
B_i	Total length of this block
C_i	0 counter may begin in any location
	1 counter must begin at a multiple of eight

INTERMEDIATE FILE

This file carries the original or generated source card image (84 columns) on to Pass 2 for further scanning and printing. The records are extended by four words which contain the information peculiar to the source statement learned in PASS 1. The contents of these four words, of course, varies depending on the type of operation code and type of variable field. In general, however, they contain the following types of information:

OP:	Operation definition from operation table
FLAGS:	All error flags, plus miscellaneous control flags (for example, generated datum, ETC follows, type of literal)
LVAL:	Literal value for DU, DL literals; count required by some pseudo-operations (BCI, BSS); result of expression evaluation for some pseudo-operations (BOOL, DUP); in general, unique usage depending on the pseudo-operation used.
P2ADD:	Address in literal pool for literal referenced in this operation. Occasionally used for second piece of data related to pseudo-operation.



3. GECOS INTERFACE

GMAP interfaces indirectly with GECOS in the allocation, reading, and writing of files, and directly in the requesting of elected options, loading overlays, and loading system macros.

To begin running, GMAP must have at least three files allocated to it: the symbolic input file (G*), the intermediate file (*1), and the output listing file (P*). Optional files are as follows: output binary deck for punching (C*) if a deck is requested, output binary deck for loading (B*) if execution is to follow, input alter file for symbolic updating (A*), and output file for new compressed deck (K*).

Determination of elected options is made via the program switch word and the Master Mode Entry, GESETS. The option and corresponding bit switch word setting is as follows:

Bit 5	Execution activity follows	1 = Yes
Bit 7	Produce binary deck for punching	1 = Yes
Bit 8	Produce output listing	1 = Yes
Bit 10	Print skipped cards following the conditional pseudo-operations (i.e., IFE, INE, IFL, and IFG).	1 = Yes
Bit 11	Load GMAP system macros	1 = Yes
Bit 13	GMAP interface	1 = Yes

Upon completion of the assembly process, the status of bit 13 of the switch word is examine. If it is zero (normal), the assembler will issue a MME GEFINI to terminate. If bit 13 is one (1), GMAP will load the Q-register with the file code *Z (000000005471_8), and transfer to location 26 (32_8) of the slave fault vector. this is all for use of GMAP with System Editor

The Alter and Comdeck options are determined as a function of the GEFRC input/output package.

All input/output is performed through GEFRC (and in turn through GECOS) via the standard CALL linkage.

Loading of overlays and system macros is normally made via the Master Mode Entry, GECALL. However, "handles" are there for using GERSTR instead, so that GMAP or new system macros may be run and debugged under control of the General Loader (GELOAD). In this case a special entry has been provided (GMAPD) which will alter all MME GECALL's related to loading GMAP to become MME GERSTR's. An exception is the MME GECALL for system macros. If this is to be made a MME GERSTR it must be patched to such. Thus, running under GELOAD, one may use the system macros from the system file or load a set with GMAP as a free-standing LINK, and load it from the Overlay file (H*) built by GELOAD.

4. PSEUDO-OPERATIONS

ABS

Pass 1

Set ABSFLG nonzero.

Pass 2

If in the FUL mode, punch what is there and terminate FUL mode.

ASCII

Pass 1

Define location symbol. Evaluate first field for number of words, same for Pass 2, and increase the location counter by that number.

Pass 2

Check location symbol for phase error. Transliterate each character of the second field into ASCII code. List and punch the number of words determined in Pass 1.

ASCIIC

Pass 1

Same as ASCII

Pass 2

Check location symbol for phase error. Evaluate second field as a 9-bit character and insert it as the first character of the generated information. Transliterate each character of the third field into ASCII code and insert into the next character position of the generated information. List and punch the number of words determined in Pass 1.

BCI

Pass 1

Define location symbol. Evaluate first field for number of words, save for Pass 2, and increase the location counter by that number.

Pass 2

Check location symbol for phase error, list, and punch the number of words determined in Pass 1.

BEGIN

Pass 1

The BEGIN flag and relative value are set in the USE tables for the USE determined by the first field.

Pass 2

List card with address.

BFS

Pass 1

Evaluate length of the table and save for Pass 2. Increase the location counter and define location symbol at next location.

Pass 2

Increase location counter by value from Pass 1. Check location symbol for phase error. Reset card punching controls, if value from Pass 1 is nonzero.

BSS

Pass 1

Define location symbol. Evaluate length of the table and save for Pass 2. Increase location counter.

Pass 2

Check location for phase error. Increase location counter by value from Pass 1. Reset card punching controls, if value is nonzero.

BLOCK

Pass 1

The BLOCK name is assembled from the variable field. If it is not already in the BLOCK tables, it is entered along with its initial value. The status of the current USE or BLOCK is stored in the appropriate table. The program counter is set to count in this BLOCK.

Pass 2

The status of the current USE or BLOCK is stored. The new values are set from the BLOCK tables. The card punching controls and program counter are reset.

BOOL

Pass 1

The variable field is evaluated as a Boolean expression,and the location symbol is defined as that value.

Pass 2

The location symbol is checked for phase error.

CALL

Pass 1

1. Define location symbol.
2. If absolute assembly, go to step 4.
3. Enter called program name in symbol table as SYMREF if not already in SYMDEF table.
4. Start at the beginning of the argument list.
5. Set word count to step 3.
6. Skip to next break character.
7. Determine which break character is present.
8. Transfer to associated routine (steps 9, 15, 16, 19, or 20).
9. [comma] Add one to word count and if the next character is not a blank, go to step 6.
10. If next card is ETC, return to Pass 1 control sequence; if not, go to step 19.

11. [Re-entry on ETC card] Skip next character if it is a left parenthesis (( ).

12. If next character is an apostrophe ('), go to step 20.

13. If next character is not an equal sign (=), go to step 6.

14. Evaluate literal. Get break character and go to step 7.

15. [Left parenthesis] Set error flag and go to step 6.

16. [Right parenthesis] If next character is an apostrophe, go to step 20.

17. If next character is not blank, go to step 9.

18. If next card is ETC, add one to word count and return to Pass 1 control sequence.

19. [Blank] Add one to word count.

20. [Apostrophe] Increase location counter by the word count and return to Pass 1 control sequence.

Pass 2

1. Check location symbol for phase error.

2. Evaluate subprogram address.

3. Evaluate modifier, if it exists, and publish TSX1.

4. Add 2 to the location counter and set the argument and error return counts to zero.

5. If break character is a blank, go to step 14.

6. If break character is a left parenthesis and is followed by a right parenthesis, skip over the right parenthesis.

7. If the end of the variable field has been reached, go to step 21.

8. If next character is an apostrophe, go to step 22.

9. Evaluate a subfield as an argument. (If it is a literal, get literal address.)

10. Increase argument count and location counter.

11. Store argument in save table for later use.

12. If break character is not a comma, go to step 16.

13. If next character is not a blank, go to step 17.

14. If next card is not ETC, go to step 7.

15. List card and return to Pass 2 control sequence.

16. If break character is not a right parenthesis, go to step 20.

17. Store argument count as beginning of error return count.

18. If next character is not a blank, go to step 20.

19. If next character is ETC, go to 15; otherwise, go to step 21.

20. If apostrophe, go to 22; otherwise, go to step 7.

21. Pick up alter number for E.I. and go to step 23.

22. Evaluate E.I.

23. Pick up current location for TRA address.

24. Reset location counter to the address of the TSX1 plus 1.

25. Publish TRA.

26. Assembly and publish error linkage word.

27. Retrieve and publish arguments in the order in which they were saved.

28. Retrieve and publish error returns in the reverse order.

29. Return to Pass 2 control sequence.

CRSM

Pass 1

The created symbol flag is set according to the variable field.

Pass 2

No processing required.

DCARD

Pass 1

The number of cards to be punched in BCD is obtained from the first field and passed on to Pass 2 along with that number of cards following the DCARD pseudo-operation.

Pass 2

The cards following the DCARD are punched in BCD. Column 1 of the punched cards is set to the character specified by the second field on the DCARD.

DEC

Pass 1

The location symbol is defined. The subfields are counted, and the location counter increased by that number.

Pass 2

The location symbol is checked for phase error. The subfields are converted, and the location counter increased by one for each subfield.

DELM

Pass 1

The location symbol is defined. The symbol is assembled from the variable field and looked up in operation table. It must be a macro name. Control word is disabled so that if macro is called again, an illegal operation code results. Prototype is located and deleted from memory. If any are above it, they are pushed down.

Pass 2

Location symbol is checked for phase error.

DETAIL

Pass 1

No processing required.

Pass 2

The DETAIL flag is set according to the variable field.

DUP

Pass 1

The location symbol is defined. The number of cards to be duplicated and the number of times they are to be duplicated are obtained, and these controls are set in the Pass 1 control sequence.

Pass 2

The location symbol is checked for phase error.

EDITP

Pass 1

No processing required.

Pass 2

Set Special Edit Mode Print flag according to the variable field.

EIGHT

Pass 1

If the location counter is not a multiple of 8, it is increased to such. The location symbol is defined. The current USE or BLOCK tables are set to reflect that the counter must also begin modulo 8.

Pass 2

If the counter is not a multiple of 8, a TRA *+n is generated. The location symbol is checked for phase error.

EJECT

Pass 1

No processing required.

Pass 2

The control is set to list the next line at the top of the next page.

END

Pass 1

Define location symbol. Store current USE or BLOCK data. Ignoring any USE that had a BEGIN, set the initial location of each to the sum of the lengths of all the USE's appearing before it in the USE table. Set the initial locations of the USE's having BEGIN's to their appropriate addresses. Retain the largest location used as the program break. If an ERLK pseudo-operation has not occurred, define the symbol .E.L.. at the program break and increase the program break by 2. Sort the symbol table. Sweep through the symbol table, redefining symbols (as required) from the data in the USE tables and flagging multidefined symbols. Compute word 3 for preface cards, if not in the ABS mode. Increase the program break by the number of remaining literals. Set the literal pool origins in the LIT tables for any LIT pseudo-operation that occurred. Close and rewind the input and the intermediate files. Call in Pass 2.

Pass 2

Assemble error linkage, if ERLK has not occurred. List and punch any remaining literals. If in the ABS mode, punch transfer card. Left-adjust the symbols in the symbol table and pad with blanks. Sort the symbol table again, list the symbolic reference table, list the undefined symbols, close out the I/O, and return to the monitor.

ENDM

Pass 1

See MACRO.

Pass 2

No processing required.

EQU

Pass 1

Evaluate variable field and define the location symbol as that value.

Pass 2

Scan variable field for symbolic references.

ERLK

Pass 1

Define .E.L.. at this location. Increase counter by 2.

Pass 2

Check location counter for phase error. Assemble error linkage cells at next two locations.

EVEN

Pass 1

If the location counter is odd, it is bumped by 1. The location symbol is defined. The "must begin even" bit is set in the USE table for the current use.

Pass 2

If the location counter is odd, an NOP is generated. The location symbol is checked for phase error.

FEQU

Pass 1

Assemble symbol from variable field. Look up in symbol table. If defined, treat this as normal EQU. If not, enter symbol in table with "as yet undefined" bit set. Value for table is pointer to next entry in table. Location symbol is entered in symbol table next. It has a value of 0 and the "as yet undefined" bit set. Counter of the number of undefined symbols in symbol table is increased by 2.

Pass 2

Check that location symbol is defined. Check for phase error. Check variable field for undefined symbols and references.

FUL

Pass 1

No processing required.

Pass 2

Punch any incomplete cards. Set FUL mode flag. Set card punching controls for FUL punching.

HEAD

Pass 1

Set up to 10 single-character subfields in the head table.

Pass 2

Set first character of the variable field as the current head character. Move head character(s) to subtitle image for display.

IDRP

Pass 1

If this is the first of the pair of IDRP's:

1. The current position in the prototype is saved,and the IDRP operative flag is set.

2. A tally word is set up to get the first subargument of the argument string corresponding to the argument number which controls the IDRP.

If this is the second IDRP of the pair:

1. If this was the last time through the range, the IDRP flag is reset and the macro expansion continues normally.

2. If this is not the last time, the position in the prototype is reset to the saved position and the subargument tally is set to the next subargument.

Pass 2

No processing required.

IFE, IFG, IFL, INE

Pass 1

The test switch is set for the appropriate conditional. The first two subfields are evaluated and compared. The number of cards following the conditional (specified by the third subfield) were skipped if the test fails. The ON5 option is tested. If it is present, enable skipped cards of main sequence to be printed by inserting an asterisk (*) in col. 84.

Pass 2

If the variable field contains no apostrophes, it is scanned for symbolic references. The alter number is adjusted if the following cards were not generated.

INHIB

Pass 1

No processing required.

Pass 2

Set INHIB flag according to the variable field.

LBL

Pass 1

If this is in the first card group, set label from the variable field.

Pass 2

Ignore if the first card group; otherwise, punch any incomplete card and set card punching controls to new card with label from the variable field.

LIST

Pass 1

No processing required.

Pass 2

If not in NLSTOU mode, set LIST flag according to the variable field.

LIT

Pass 1

If any double-precision literals have occurred, set the current USE to "must begin even." Set flag to write literals on the intermediate file following the LIT pseudo-operation. Set the current location, the current USE, and the literal counts in the LIT tables and reset controls to begin a new literal pool. Increase the location counter by the length of the pool.

Pass 2

Read in the literal pool. Punch and list it, increasing the location counter by 1 for each word assembled. Reset the literal controls from the next values in the LIT tables. Increase the location counter by the number of abnormal (=V, M, nH) literals which have been previously assembled.

LOC

Pass 1

Same as ORG.

Pass 2

Same as ORG except that the load address of the binary card is not changed.

LODM

Pass 1

Assemble symbol from variable field and call Load System Macros subroutine.

Pass 2

No processing required.

MACRO

Pass 1

1. A two-word entry is made in the operation table.

Word 1:		The location symbol, left-adjusted and filled with blanks.
Word 2:	0-17	Address of the macro call processor.
	18	0
	19-35	Relative location in the macro prototype table of the first word of the macro definition.

   If word 1 was already in the operation table, it is flagged as multidefined and the second word is stored over the original word 2.

2. The macro prototype switch is set in the Pass 1 control sequence to transfer to the routine which stores the prototype.

3. The cards following the MACRO pseudo-operation are packed into the prototype until an ENDM pseudo-operation is encountered. Control characters (6 bits) are included in the packing to control the expansion. They are:

00	End of record (00 00 indicates end of macro)
01	Operation field follows
02	Variable field follows
03	Argument number follows
04	String of characters follows
05	Next card is a generated ETC

   Types 01, 02, and 04 are followed by a character (6 bits) giving the number of characters that follow.

   Type 03 is followed by one character giving the argument number.

In general, packing ends with the first blank in the variable field.

Exceptions are:

a. REM--The entire card is packed.

b. TTL, TTLS, BCI, or the occurrence of a literal--The entire variable field is packed.

c. A left parenthesis is packed--Everything is packed until a closing right parenthesis is packed.

d. ENDM--A second 00 control character is packed, indicating the end of the macro prototype.

Example:

```
              L    LDA     B,#2     would be packed -

    P           040143010420        (note packing of column 7)

    P+1         432421020222

    P+2         73030200XX--
```

The next card would be packed, starting at XX. It should be noted that the operation field is considered to start in column 7 in order that even/odd indications may be carried.

4. When the ENDM pseudo-operation is encountered, the largest argument number detected is stored in the first word of the prototype. (The actual prototype that will be expanded begins in the second word.) The macro prototype switch in the Pass 1 control sequence is reset for normal processing and control is returned to it.

o <u>Macro Call</u>

1. When a macro call is made, the macro call processor sets up a table of arguments from the macro call card and any ETC cards that follow.

2. The arguments are stored as a sequence of characters, each argument beginning in a new word.

3. Commas normally delimit arguments. The exceptions are:

   a. When a left parenthesis is encountered, it is not stored, but every character up to (but not including) the closing right parenthesis is stored as one argument.

   b. If an argument is enclosed in brackets, everything is stored as one argument except the brackets and any enclosed blanks.

4. Created symbols are produced in two ways:

   a. If the number of arguments provided on the macro call is less than that required by the prototype, the remaining arguments each become a created symbol.

b. If the symbol # appears as an argument, a symbol is created for that argument.

5. A list of argument control words is appended to the end of the argument table. The control words are of the form:

0-17	Address of the argument's first word
18	0 normal argument
	1 created symbol
19-23	Not used
24-35	Number of characters in the argument sequence

Example:

Assume the macro ABC required three arguments and was called as follows:

```
L2    ABC    (ADLA   PUT,6),10
```

The argument table and control list would be:

L	212443212047
L+1	646373060000
L+2	010000000000
L+3	330000013300
L+4	(L)000012
L+5	(L+2)000002
L+6	(L+3)400005

6. The macro expansion switch in the Pass 1 control sequence is set to send control to the macro expansion routine instead of getting input records from the input file.

7. If the macro call had been generated by another macro, the latter's control words would have been saved in a table until completion of the expansion of the presently called macro.

● Macro Expansion

1. The card image is set to blanks.

2. Characters are set into the card image according to the controls from the prototype as follows:

00	(End of record.) The card is complete and control is returned to the Pass 1 control sequence to process the generated card.

01 (Operation field starts.) The tally word for storing characters is set to start storing in column 7.

02 (Variable field starts.) The tally word is set to column 16.

03 (Argument number follows.) The tally word for picking up characters is set to the correct argument. If this argument is under control of IDRP, the tally is set for the current subargument number.

04 (Characters follow.) The next n characters from the prototype are stored in sequence in the card image.

05 (Programmer ETC flag.) The ETC mode flag is set.

3. When the end of the prototype is reached, the level of the macro is checked to see if it is a nested macro call. If it is, the controls for the previous macro are restored and macro expansion continues. If not, the macro expansion switch in the Pass 1 control sequence is reset for normal processing and control returned thereto.

Pass 2

No processing required.

MAX, MIN

Pass 1

The location symbol is defined as the value of the appropriate subfield of the variable field.

Pass 2

The variable field is scanned for symbolic references.

MAXSZ

Pass 1

No processing required.

Pass 2

The variable field is evaluated, and the result saved for printing at the end of the listing.

NONOP

Pass 1

Defined as an undefined machine instruction.

Pass 2

No processing required.

NULL

Pass 1

The location symbol is defined as this location.

Pass 2

The location symbol is checked for phase error.

OCT

Pass 1

The location symbol is defined and the location counter increased by the number of subfields.

Pass 2

The location symbol is checked for phase error. Each subfield is assembled as one octal word.

ODD

Pass 1

If the location counter is even, it is bumped by 1. The location symbol is defined. The "must begin even" bit is set in the USE tables for the current use.

Pass 2

If the location counter is even, an NOP is generated.

OPD

Pass 1

The location symbol is entered into the operation table as a machine operation with its definition. The 36-bit definition is formed from the variable field (assumed to be in VFD format) with the machine instruction flag bit ORed into it. If the operation code is already in the table, only the new definition is entered.

Pass 2

The variable field is scanned for symbolic references.

OPSYN

Pass 1

Same as OPD except that the definition comes from looking up the symbol from the variable field in the operation table.

Pass 2

No processing required.

ORG

Pass 1

The variable field is evaluated. If the variable is defined under a different USE or BLOCK than the one currently in effect, the current USE or BLOCK data are stored and the data for the USE or BLOCK of the variable field are brought into effect (just as if a USE or BLOCK pseudo-operation had occurred). The location counter is set to the value of the variable field and the location symbol is defined. The value of the location counter and its respective USE or BLOCK numbers are saved for Pass 2.

Pass 2

The variable field is scanned for symbolic references. If the USE or BLOCK of the ORG is different from the current USE or BLOCK, the current USE or BLOCK data are stored and the new USE or BLOCK data are brought into effect. The location counter and card punching controls are reset to reflect the variable field value. The location symbol is checked for phase error.

ORGCSM

Pass 1

The numeric portion of the created symbols is reset to the value of the variable field.

Pass 2

The variable field is scanned for symbolic references.

PCC, PMC

Pass 1

No processing required.

Pass 2

The respective switch is set according to the variable field.

PUNCH

Pass 1

No processing required.

Pass 2

The new setting for the punch switch is determined and compared with its current state. The current status of the PUNCH flag is examined. If the two are the same, no further processing is required. If the new state is ON, the card punch controls are reinitialized; if OFF, the current card is wrapped up and moved to the output buffer; the PUNCH flag is reset to its new state.

PUNM

Pass 1

The length of the prototype area is determined and saved. The operation table is searched for all macros. Their names and definitions are saved in the next available prototype storage. The count of the number of macros is saved. A flag is set for the punching of these prototypes along with their definitions at the beginning of Pass 2.

Pass 2

No processing required.

REF

Pass 1

No processing required.

Pass 2

The reference list flag is set according to the variable field.

REM

Pass 1

No processing required.

Pass 2

The operation code is reset to blanks for listing.

RETURN

Pass 1

The location symbol is defined. The number of locations required for the RETURN is computed and added to the location counter.

Pass 2

The location is checked for phase error. The RETURN sequence is assembled, increasing the location counter one for each word assembled.

SAVE

Pass 1

The location symbol is defined and also entered in the SYMDEF table. The number of locations required for the SAVE is computed and added to the location counter.

Pass 2

The location symbol is checked for phase error. The SAVE sequence is assembled, increasing the location counter by 1 for each word assembled.

SET

Pass 1

The location symbol is looked up in the symbol table. If it is not there, it is defined as the value of the variable field. If it is there, it is redefined.

Pass 2

The value of the location symbol is reset to the value of the variable field.

SYMDEF

Pass 1

The symbols in the variable field are entered in the SYMDEF table with their primary or secondary flags.

Pass 2

The variable field is scanned for symbolic references.

SYMREF

Pass 1

If the symbols in the variable field have not been entered in the SYMDEF table and are nonnumeric, they are defined as SYMREF's.

Pass 2

The variable field is scanned for symbolic references.

TALLY, TALLYB, TALLYC, TALLYD

Pass 1

Processed the same as a machine operation, except that the address error flag is set if an abnormal literal is given as the first subfield.

Pass 2

For all four pseudo-operations, the first two subfields are evaluated in the same manner. The first subfield may be an expression or a simple literal and is assembled into bits 0-17 of the machine word. The second subfield may also be an expression. It is assembled into bits 18-29 of the machine word. The third subfield will be assembled into bits 30-35 and is examined differently for each of the pseudo-operations.

1. TALLYC--Third subfield is assembled as a normal modifier.
2. TALLYD--Third subfield may be any expression.
3. TALLY--Third subfield may be an expression with the value $0 \leq x \leq 5$.
4. TALLYB--Third subfield may be an expression with the value $0 \leq x \leq 3$. Bit 30 is set to 1.

TCD

Pass 1

No processing required.

Pass 2

Any incomplete binary card is punched. If in a relocatable assembly, nothing else is done; otherwise, the FUL mode flag is reset to normal absolute, and the variable field is evaluated and used as the address on the transfer card which is punched.

TTL

Pass 1

If it appears in the first card group and no other TTL's have occurred, it is retained as the initial page header. The next card is checked, and a flag passed on to Pass 2 if it is a TTLS pseudo-operation.

Pass 2

If it is in the first card group and the first TTL, no processing is required; otherwise, it becomes the new page header. The flag from Pass 1 is checked,and the list control is set to list the next line at the top of the page if the TTL is not followed by a TTLS.

TTLS

Pass 1

If it appears in the first card group and no other TTLS's have occurred, it is retained as the initial page subheader; otherwise, no processing is required.

Pass 2

If it is in the first card group and the first TTLS, no processing is required; otherwise, it becomes the new page subheader and the list control is set to list the next line at the top of the page.

USE

Pass 1

The symbol in the variable field is checked for PREVIOUS. If it is not PREVIOUS, it is looked up in the USE table. If it is not in the table, it is entered and its controls initialized (set to zero). The current USE or BLOCK data are stored. The new USE controls are brought into effect (if this was USE PREVIOUS, the previous USE controls are used).

Pass 2

The current USE or BLOCK data are stored. The new USE data are brought into effect. The card punching controls are reset. If this is the first occurrence of this USE, the location counter is odd, and if this USE must begin even, an NOP is assembled. Similarly, if the counter must begin modulo 8, a TRA *+n is generated and the counter set to that value.

VFD

Pass 1

The location symbol is defined. The total of the count fields on the VFD and any ETC's is computed, and the number of words required is added to the location counter.

Pass 2

The location symbol is checked for phase error. The subfields of the VFD are packed into succeeding words, the location counter being increased by 1 for each word assembled.

ZERO

Pass 1

Processed the same as a machine instruction, except that the address error flag is set if a literal is given as the first subfield.

Pass 2

The location symbol is checked for phase error. The two subfields are assembled as if they were 18-bit addresses.

5. LITERALS

LITERAL EVALUATED IN PASS 1 AND ENTERED IN THE LITERAL POOL

(Decimal, octal, and Hollerith of less than 13 characters.)

Pass 1

1. The literal is converted to binary, then:

   a. If it is followed by a DL or DU modifier, the converted literal is passed on to Pass 2.

   b. If it is not followed by a DL or DU modifier, it is entered in the literal pool and the relative address in the pool is passed on to Pass 2.

2. Control bits are passed on to Pass 2, giving the type of literal and, in the case of Hollerith literals, the character count also is passed along.

Pass 2

1. The control bits from Pass 1 are interpreted and:

   a. If the literal is followed by a DL or DU modifier, the literal itself is used as the address of the instruction.

   b. If the literal is not followed by a DL or DU modifier, the address is computed from the relative location in the pool and the pool's assembled address. The literal portion of the variable is skipped and the modifier evaluated.

Note: In each of the above cases a full 36-bit literal is formed. If DU or DL is indicated, however, only 18 bits are carried. For Decimal and Octal this will be the low-order 18 bits of the 36-bit literal, whereas for floating point and Hollerith it will be the high-order 18 bits. Hollerith literals will be blank filled to the right.

OTHER LITERALS

(=V, =M, and =nH where n > 12)

Pass 1

1. The control bits for type are set.
2. The number of words required for the literal are computed.
3. The relative address in a phantom pool is passed on to Pass 2.
4. In the case of the machine instruction literal, the operation code is looked up in Pass 1, and word 2 of the operation table entry is passed on.

Pass 2

1. The phantom pool is given an origin immediately after the normal literal pool.
2. The address of the literal,then, is relative to that origin.
3. The literal itself is assembled immediately after the instruction using it. It therefore appears directly in line in the listing, but with the correct address of where the literal would have been if it had been in the normal literal pool.

Note: There is no actual pooling of literals of this type. That is, a given literal (for example 13H...) will appear as many times in the phantom pool as it is referenced.

6. MAJOR SUBROUTINES

FUNCTION OF ROUTINES COMMON TO BOTH PASSES

Evaluate an Expression

- Calling Sequence is:

```
L       TSX1    SCAN
L+1             Null expression
L+2             Normal return
```

- Remarks. This function evaluates one expression of the variable field and sets error flags.

To evaluate the expression, SCAN:

1. Set the initial values of the term and expression to zero.
2. Return control, if the field is blank.
3. Set the previous operation to +.
4. Reset the first character and Boolean complement flags; pick up current head character and go to 6.
5. Pick up a new head character if a $ was encountered.
6. Store as head character to be used for this element.
7. Set the initial values of the element to zero and go to 9.
8. Set Boolean complement flag.
9. Assemble the next element and transfer to the routine which is designated by the operator following the element. If there is no operator, the "end of expression" flag is set and control is transferred to evaluate the element just assembled. The first two instructions of each routine for the operators are a TSX1 to evaluate the element and a TRA to the corresponding routine for Boolean evaluations. The routine that evaluates the element always returns to the previous operator and sets the operator just encountered as the previous operator. After processing, each operator routine transfers to 10.

The operator routines

a. (+ and -) The term is combined with the expression and the element (complement for -) is set as the initial value of the next term.

b. (* and /) The element and the term are combined if they form an allowable combination.

c. (+, -, *, and / in the Boolean) The Boolean counterparts of the operations are performed in like manner.

10. A test is made for the end of the expression and control is transferred to 4, if it is not the end.

11. The last term is combined with the expression.

12. The final number of relocatable units is checked for validity to determine if the expression was one of the allowable combinations of elements and terms.

13. Control is returned to the controlling routine with the value of the expression in the A-register. If there were any errors encountered in the SCAN,a flag (EXPRR) is set and a value of 0 is returned.

To evaluate an element:

1. If the element is symbolic, the head character is appended and the value of the symbol retrieved from the symbol table.

2. If the element is numeric, it is converted from decimal to binary or, if in the Boolean mode, from octal to binary.

3. If the element is *, it is assumed to be the current value of the location counter.

4. The control words (type, value, etc.) are set according to their evaluation.

Rules for determining the legality of the relocatability of an expression are given in the GMAP section of GE-625/635 Programming Reference Manual (CPB-1004).

Evaluate a VFD Expression

- Calling Sequence is:

```
L       TSX1    CVFD
L+1             Normal return
```

- Remarks. This function evaluates one VFD-type expression and leaves the result left-adjusted in the Q-register. Expressions are evaluated in the same manner for VFD as for normal expressions except that all operators are Boolean.

Decimal to Binary (Integer)

- Calling Sequence is:

```
L      TSX1    CTDEC
L+1            Normal return
```

- Remarks. This function converts the BCD element in SYMB2 and SYMB to binary and returns the result in the Q-register.

Octal to Binary

- Calling Sequence is:

```
L      TSX1    CTOCT
L+1            Normal return
```

- Remarks. This function converts the BCD element in SYMB2 and SYMB to binary and returns the result in the Q-register.

General Decimal to Binary

- Calling Sequence is:

```
L      TSX1    DTB
L+1            Normal return
```

- Remarks. This function converts one field of the variable field from decimal to binary according to the DEC pseudo-operation specifications. Result in the AQ-register.

General Octal to Binary

- Calling Sequence is:

```
L      TSX1    OCTCV
L+1            Normal return
```

- Remarks. This function converts one field of the variable field to binary according to the OCT pseudo-operation specification. Result is in the Q-register.

Assemble an Element

- Calling Sequence is:

```
L      TSX4   ASYM
L+1           Null field
L+2           Normal return
```

- Remarks. This function assembles an element (right-adjusted) in SYMB2 and SYMB until a break character is encountered. Leading characters are zeros.

Get Next Character

- Calling Sequence is:

```
L      TSX1   CO190
L+1           Normal return
```

- Remarks. This function returns the next character to be processed in the low 6 bits of the A-register.

Store USE Data

- Calling Sequence is:

```
L      TSX0   STUSE
L+1           Normal return
```

- Remarks. This function stores the current USE or BLOCK data in their respective tables.

Set USE Data

- Calling Sequence is:

```
L      TSX0   SETUS
L+1           Normal return
```

- Remarks. This function retrieves the new data for the USE or BLOCK being called from the USE or BLOCK tables.

Operation Table Search

o Calling Sequence is:

L	TSX4	OPCD
L+1		Undefined operation
L+2		Normal return

o Remarks. This function looks up in the operation table the symbol contained in the A-register and returns the second word of the entry in the Q-register.

Sort Symbol Table

o Calling Sequence is:

L	TSX1	SORT
L+1		Normal return

o Remarks. This function sorts the symbol table.

Card Initialization

o Calling Sequence is:

L	TSX1	INI
L+1		Asterisk in Column 1 or 84 (FORTRAN comment)
L+2		Normal return

o Remarks. This function initializes card SCAN parameters. It assembles the location symbol; sets CO190 so that the next character to be picked up is the first one of the variable field; forms the BCD operation code; and sets the odd or even flag as indicated by column 7.

Evaluate Switch Alteration for On/Off Type Pseudo-Operations

o Calling Sequence is:

L-1	LDA	SWITCH
L	TSX2	ON OFF
L+1		Normal return with updated switch status in A-register.

- Remarks. All of the following pseudo-operations may have their status saved and/or turned on, off, inverted or restored to the prior state. The current status may be pushed down to 35 levels deep. The function of this routine is to adjust the status of a switch based on the variable field of the pseudo-operation involved. Those which call this subroutine are:

Pass 1	CRSM
Pass 2	DETAIL, LIST, PCC, REF, PMC, INHIB, PUNCH, EDITP

FUNCTIONS OF PASS 1 ROUTINES

Pass 1 Symbol Table Search

- Calling Sequence is:

```
L      TSX1    SYMBT
L+1            Normal return
```

- Remarks. This function makes a linear search of the symbol for the value of the symbol in SYMB and returns value in the A-register. If the symbol is undefined, a value of zero is returned and the expression error flag (EXPRR) is set.

Define a Symbol

- Calling Sequence is:

```
L      TSX1    DSYM
L+1            Normal return
```

- Remarks. This function enters the symbol in LSYMB into the symbol table, along with its definition from ADDRS. If the symbol is already in the symbol table as an "as yet undefined" symbol, it is given its proper definition, and the symbol to which it points is given the same definition.

Define Operation

- Calling Sequence is:

```
L      TSX1    ENTOP
L+1            Normal return
```

- Remarks. This function enters the symbol in LSYMB into the operation table, along with its definition from LVAL. If the symbol is already in, the definition is changed to that of LVAL and the multiple-defined flag is set.

Load Macro Prototypes

- Calling Sequence is:

```
L      TSX1    LMAC
L+1            Normal return
```

- Remarks. This function issues a MME GECALL for the system macros named in LSYMB. It is called at the beginning of Pass 1 to load the GMAP systems macros, and it may be called by the LODM pseudo-operation to append the macro prototype tables with another set of macros. Using the control word loaded as the first word from the system file, it locates the macro definitions (also loaded) and enters them into the operation table. They are, of course, redefined so that the prototype pointers are adjusted relative to the load origin, and the address of the Macro Call processor is set.

FUNCTIONS OF PASS 2 ROUTINES

Symbol Table Search

- Calling Sequence is:

```
L      TSX1    SYMBT
L+1            Normal return
```

- Remarks. This function retrieves definition of the symbol in SYMB from the symbol table and makes appropriate entries in the symbolic reference table. It does a binary lookup in the now-sorted symbol table.

Pseudo-Variable Field Scan

- Calling Sequence is:

```
L      TSX1    DSCAN
L+1            Normal return
```

- Remarks. This function scans one field of the variable field of pseudo-operations whose variable field was processed in Pass 1 to make symbolic reference table entries.

Binary Card Entry

- Calling Sequence is:

```
L      TSX1    STAQ
L+1            Normal return
```

- Remarks. This function enters the A- and Q-register contents into the current binary card image. If the card becomes full, it starts a new card.

Punch Binary Card

- Calling Sequence is:

```
L      TSX1    BUNCH
L+1            Normal return
```

- Remarks. This function computes the checksum and punches the current binary card.

Initialize Binary Card

- Calling Sequence is:

```
L      TSX1    INCRD
L+1            Normal return
```

- Remarks. This function clears the card image and resets storing of binary words to the beginning of the card.

Set Binary Word Storing Controls

- Calling Sequence is:

```
L      TSX1    SETWC
L+1            Normal return
```

- Remarks. This function enters the address and word count for the previous binary words and sets up for the next words to be stored. Accomplishes multiple origining of binary instruction cards.

Enter Binary Word Into Card Image

o Calling Sequence is:

```
L      TSX1    BINRY
L+1            Normal return
```

o Remarks. This function enters BWORD into the binary card image; if NWORD is nonzero, it is also entered. If the card becomes full, appropriate routines are called to punch it and set controls for the next binary card.

Punch Macro Prototype Table

o Calling Sequence is:

```
L      TSX1    PMAC
L+1            Normal return
```

o Remarks. This function punches the macro prototype table on relocatable binary cards. The size of the prototype area to be punched and the number of macros defined there was determined in Pass 1 by the PUNM pseduo-operation. This deck will be given unique $ OBJECT and $ DKEND cards, and will require the addition of editing control cards only, for editing.

Print a Line

o Calling Sequence is:

```
L      TSX1    WROUT
L+1            Normal return
```

o Remarks. This function checks listing control flags and prints one line as required.

List Machine Instruction

o Calling Sequence is:

```
L      TSX1    MOUT
L+1            Normal return
```

o Remarks. This function sets up a line to be listed in the format for the machine operation specified by its list control flags in the operation table.

Evaluate Tag Field

- Calling Sequence is:

```
L      TSX1    TAG
L+1            Normal return
```

- Remarks. This function evaluates the tag field of an instruction and places it in the proper position in BWORD.

Evaluate Address Field

- Calling Sequence is:

```
L      TSX1    EVALA
L+1            Normal return
```

- Remarks. This function evaluates the address field of an instruction and places it in the proper position in BWORD. It determines the relocation and sets NWORD if a second word is required to handle the relocation.

Set Relocation Bits

- Calling Sequence is:

```
L      TSX1    STREL
L+1            Normal return
```

- Remarks. Based on the result of evaluating an expression, a relocation bit will be defined as absolute or relocatable relative to the program origin, blank COMMON, LABELED COMMON or some SYMREF. The function of this routine is to translate the established type of relocation, for each half-word of BWORD into the 5-bit code for the card and shift these in. Relocation bits are not carried for NWORD as it, by definition, contains absolute adders.

Checks for Phase Error

- Calling Sequence is:

```
L      TSX1    CHEK
L+1            Normal return
```

- Remarks. If a location symbol exists, its definition from the symbol table is checked against the current value of the location counter to ensure that they are the same.

APPENDIX
PSEUDO-OPERATIONS BY FUNCTIONAL CLASS

PSEUDO-OPERATION MNEMONIC	FUNCTION	PAGE NUMBER
CONTROL PSEUDO-OPERATIONS		
DETAIL	(Detail output listing)	18
EJECT	(Restore output listing)	19
LIST	(Control output listing)	23
REM	(Remarks)	31
LBL	(Label)	23
PCC	(Print control cards)	30
REF	(References)	30
PMC	(Print MACRO expansion)	30
TTL	(Title)	33
TTLS	(Subtitle)	33
INHIB	(Inhibit interrupts)	22
ABS	(Output absolute text)	13
FUL	(Output full binary text)	21
TCD	(Punch transfer card)	33
PUNCH	(Control card output)	30
DCARD	(Punch BCD card)	17
END	(End of Assembly)	19
HEAD	(Heading)	21
OPD	(Operation definition)	29
OPSYN	(Operation synonym)	29
LOCATION COUNTER PSEUDO-OPERATIONS		
USE	(Use multiple location counters)	33
BEGIN	(Origin of a location counter)	14
ORG	(Origin set by programmer)	29
LOC	(Location of output text)	23
SYMBOL DEFINING PSEUDO-OPERATIONS		
EQU	(Equal to)	20
FEQU	(Equal to symbol as yet undefined)	21
BOOL	(Boolean)	15
SET	(Symbol redefinition)	31

PSEUDO-OPERATION MNEMONIC	FUNCTION	PAGE NUMBER
MIN	(Minimum)	27
MAX	(Maximum)	27
SYMDEF	(Symbol definition)	32
SYMREF	(Symbol reference)	32
NULL	(Symbol EQU *)	28
EVEN	(Force Counter even before defining symbol)	21
ODD	(Force Counter odd before defining symbol)	28
EIGHT	(Force Counter to multiple of 8 before defining symbol)	19
DATA GENERATING PSEUDO-OPERATIONS		
OCT	(Octal)	28
DEC	(Decimal)	18
BCI	(Binary Coded Decimal Information)	14
VFD	(Variable field definition)	34
DUP	(Duplicate cards)	18
STORAGE ALLOCATION PSEUDO-OPERATIONS		
BSS	(Block started by symbol)	14
BFS	(Block followed by symbol)	14
BLOCK	(Block common)	15
LIT	(Literal Pool Origin)	23
CONDITIONAL PSEUDO-OPERATIONS		
INE	(If not equal)	22
IFE	(If equal)	22
IFL	(If less than)	22
IFG	(If greater than)	22
SPECIAL WORD FORMATS		
NONOP	(Undefined operation)	28
MAXSZ	(Maximum size of assembly)	27
ZERO	(Generate one word with two specified 18-bit fields)	34

PSEUDO-OPERATION MNEMONIC	FUNCTION	PAGE NUMBER
ADDRESS TALLY PSEUDO-OPERATIONS		
TALLY	(Tally--ID, DI, SC, and CI variations)	32
TALLYB	(Tally--SC and CI, 9 bit bytes)	32
TALLYD	(Tally and Delta)	32
TALLYC	(Tally and Continue)	32
MACRO PSEUDO-OPERATIONS		
MACRO	(Begin MACRO prototype)	24
ENDM	(End MACRO prototype)	20
CRSM	(Create symbols)	17
IDRP	(Indefinite repeat)	22
ORGCSM	(Origin Created Symbols)	29
PUNM	(Punch MACRO prototype)	30
LODM	(Load MACRO prototype)	24
DELM	(Delete a MACRO)	18
PROGRAM LINKAGE PSEUDO-OPERATIONS		
CALL	(Call external subroutine)	15
SAVE	(Enter subroutine)	31
RETURN	(Return to calling routine)	31
ERLK	(Error linkage)	20

DOCUMENT REVIEW SHEET

TITLE:	GE-625/635 GMAP Implementation
CPB #:	1078B

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.

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
COMPUTER EQUIPMENT DEPARTMENT
13430 NORTH BLACK CANYON HIGHWAY
PHOENIX, ARIZONA - 85029

ATTENTION: ENGINEERING PUBLICATIONS STANDARDS B-90

FOLD

Progress Is Our Most Important Product

GENERAL ELECTRIC

INFORMATION SYSTEMS DIVISION

LITHO U.S.A.