GE-225
GECOM-II
OPERATIONS
MANUAL

GENERAL ELECTRIC

COMPUTER DEPARTMENT

OBJECT-PROGRAM CHARACTERISTICS

The object program is the "raison d'etre" of GECOM. The need for its creation is what leads the programmer to use GECOM in the first place. It is important, therefore, to understand the principal characteristics of an object program in order that the programmer can build into it certain properties that are not automatically attended to by the compiler, and at the same time avoid duplication or counter action in those areas where the compiler has been designed to take away automatically much of the detail burden ordinarily placed on the programmer.

We can best approach this matter from the standpoint of how data is handled by the object program. We will relate this to the programmer's job, so that he can see what he must do, and should not do. At the same time, we can clear up an important distinction, the difference between external data and internal data in GECOM object programs.

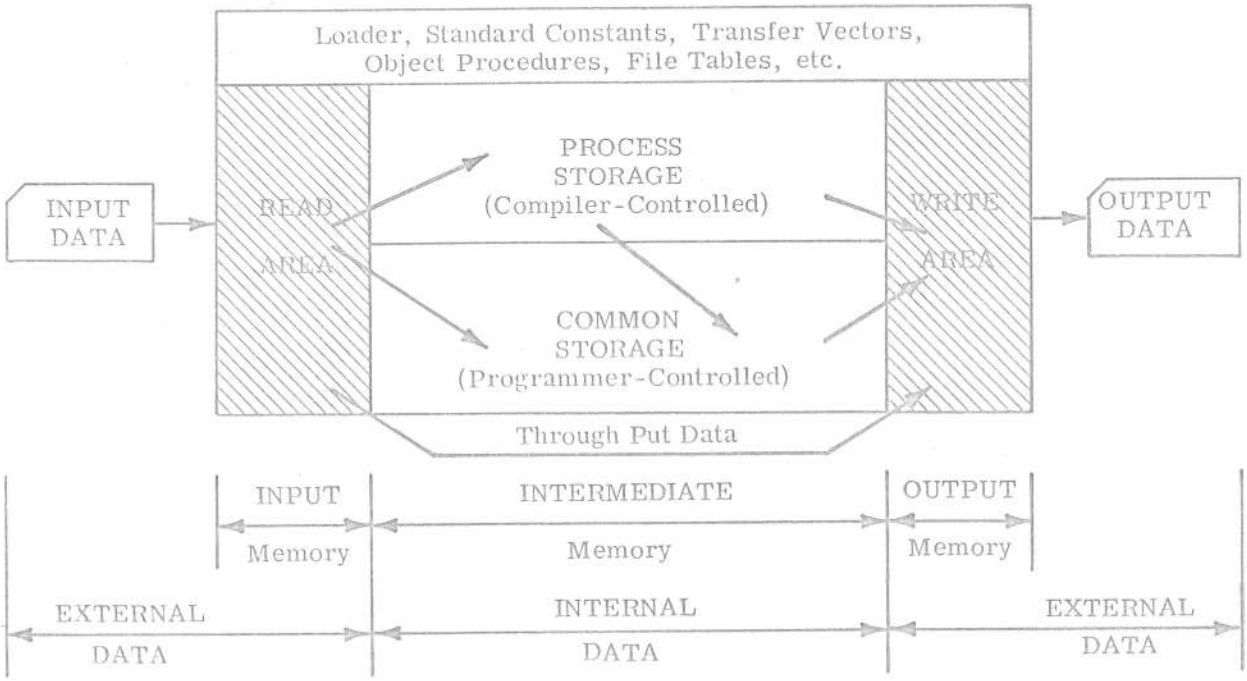
In the part of this manual called "Sections (data)," we discuss the various blocks of memory allocated for different purposes. In quick review, the programmer describes input/output data under the headings of FILE SECTION, OUTPUT FILES, and INPUT FILES. Here he shows data as it appears on an input or output medium, which is a duplicate facsimile of the data as it appears in the input and output buffers, called Read Area, and Write Area. This is external data.

Next, the programmer describes internal data in the intermediate or working areas of memory, using the headings WORKING-STORAGE SECTION, CONSTANT SECTION, COMMON-STORAGE SECTION, and *COMMON-STORAGE SECTION. We have recommended that Common Storage be used in place of a combination of Working Storage, Constants, and Common Storage, as a simplification. Our discussion follows this recommendation, but is, however, completely valid if all these sections are used instead of just the one.

We have also mentioned the presence of a compiler-controlled area of memory called Process Storage, that is not under the control of the programmer.

Now, we come to an explanation of the relationships between these various memory areas in order to show why the programmer does not necessarily describe all his internal data in the Common Storage Section, but always describes all his input and output data in complete detail. Consider the following diagram . Arrows show possible avenues of data flow.

MEMORY SCHEMATIC



Memory is shown as having two "gray areas," the Read Area (input buffers), and the Write Area (output buffers.) Although certainly a part of memory, these areas are external, from the standpoint of data. Each contains a duplicate facsimile of the data appearing on an external medium, say a punched card. Effectively, data does not "enter memory" until it passes the border between the input area and intermediate memory. It does not "leave memory" until it passes the border between the input area and intermediate memory. It does not "leave memory" without passing the border between intermediate memory and the output area.

A GECOM object program processes all data from intermediate memory, never from the input/output areas. Hence, we conclude that all input data needed for arithmetic or logical comparisons or other processing must be transferred to intermediate memory. Ordinarily, the transferred data will require a change in format or mode to prepare it for processing. This is done on the way, by the object program.

If there is a risk that data will be overlayed and destroyed in the input area by execution of a following READ instruction, the programmer may decide to move it into intermediate memory to save it, even though the data may not be processed further except to be again transferred at the appropriate time to the Write Area for output.

The point is that no data needs to be transferred into intermediate memory, unless the transfer serves some useful purpose. It is not considered sufficient purpose to move data into intermediate memory when the data will be used only as output and does not risk being destroyed. This would introduce an extraneous transfer because data can be moved directly from the input area to the output area, as shown. Such data is referred to as throughput data.

The GECOM Compiler is designed to analyze a programmer's procedure statements and draw certain conclusions about transferring input data to intermediate memory. In many cases, the compiler can deduce that transfers are useful and necessary. If it can deduce this, the compiler automatically provides object program coding to transfer given data to Process Storage. Here are the four categories of data for which the compiler knows positively that a transfer must be made:

1. Any input data whose data name is referenced, that is, appears in any ADD, SUBTRACT, MULTIPLY, DIVIDE, EQUAL, IF, VARY, GO TO DEPENDING, or ENTER GAP sentence.

2. Any input array whose data name appears in any MOVE or EXCHANGE sentence.

3. Any input data that is a repeated group. (See Reference Manual.)

4. Any input data from punched cards using comma-separated input format.

Data in these categories is always unpacked, as the transfer process is called.

For input data in any other category, the compiler cannot be certain that a transfer is necessary. Thus, it takes no action. Other data must be moved by the programmer to Common Storage, by using a MOVE or EXCHANGE sentence in his procedure statements. He provides memory space in the Common Storage by listing data names in the Common Storage Section of the Data Division on his coding sheets, and moves the input data to the new data name. If the programmer specifically wants an input quantity in Common Storage even though it is automatically assigned to Process Storage, he can move it in the same manner.

Thus, we have automatic input data transfers to Process Storage, made by the compiler without directive from the programmer, and programmed transfers to Common Storage, made by the compiler based on the programmer's procedure entries. All other input data has no memory assignment in intermediate memory, and is termed throughput. (The question of conversions of input formats and modes during these transfers is covered in the section called Modes.) Automatic transfers to Process Storage are made during input time, as a part of the function of the READ sentence.

In calculations, new data names may arise as arithmetic expressions are evaluated, as in TEMP = A*B; also, there may be a number of variables that appear in a program only as subscripts or counters. Such quantities are called intermediate fields, if they are not input quantities. They always must be assigned memory space in intermediate memory in one of the following manners.

Memory assignments of intermediate floating-point quantities, such as TEMP above and intermediate integers, such as subscripts, are made automatically by the compiler in Process Storage in the most usual case, such as when the computation-mode is floating-point. In such a case, all intermediate data names are assumed to be in floating-point mode unless otherwise specified. Thus, intermediate fields such as TEMP are automatically assigned to Process Storage when the compiler finds their data names written in procedure sentences. No entries are needed in the Data Division. Integers, if listed in the Integer Section, are also automatically assigned to Process Storage with no additional entries needed in the Data Division. (This is explained in further detail in the section, Modes.)

On the other hand, if the programmer specifically wants his intermediate quantities in Common Storage, in order to be available for use by other programs, or if the computation-mode is fixed-point, the programmer must request memory assignments for all intermediate quantities required in Common Storage, by listing the data names he wants under the heading COMMON-STORAGE SECTION. Either automatic or requested memory assignments must be made so as to include all intermediate fields.

The most important concept to grasp with respect to output data is that output is merely selected intermediate and input data. Since output cannot contain any data that is neither input nor an intermediate result, there never can be any new data name in the description of output.

The GECOM compiler, in preparing the coding which will gather data for output according to the names given in the output description, makes a careful analysis of the output format or mode requested for each field. It then decides on the best source for the output data, which is to say, it decides whether it is most efficient to gather certain data from intermediate memory, or from the Read Area, if the data exists in both places. The programmer must take caution on this point. The compiler assumes that no additional reading has been performed which would destroy the desired data in the Read Area. If more reading is to be done, the programmer should first move out of the Read Area the data he wishes to save. There is less problem when comma-separated field input cards are used since all data is sent to Process Storage, as shown in category 4 above.

There is an interesting distinction to be made concerning input data assigned to Process Storage. Some of this data may be changed in value during the execution of a program. Other data may not be changed. For example, take two fields called X and Y that enter as input, and are automatically transferred to Process Storage, according to the rules given previously. Say that the value of X is subsequently changed by any of these GECOM sentences, for example:

```
X = A+B-C.
MOVE Z TO X.
EXCHANGE X AND P.
```

X is then called a receiving field. It receives new data. On the other hand, say that X and Y also enter into calculations, but the value of Y is never changed by these operations. For example:

```
C = X/Y.
MOVE Y TO J.
```

Y then is called a nonreceiving field. It never receives new data.

In both X and Y are requested for output, the compiler will always take X from Process Storage, since, as a receiving field, it has a new value. There is no problem here. However, after analysis of the desired output format or mode given for Y, the compiler may decide to take Y from the Read Area, knowing that Y is a nonreceiving field and thus has experienced no change in value. Here again the programmer must caution himself. If more reading has been done, the correct value of Y in the Read Area is destroyed, and erroneous data is sent to output, even though the correct value of Y may still rest in Process Storage. The programmer should save nonreceiving fields by moving them from Process Storage to Common Storage if he feels this condition will occur.

At this point, the programmer may wonder how he accomplishes a move from Process Storage to Common Storage, and how this differs from a move from the Read Area to Process Storage, or from the Read Area to Common Storage. Finally, he may wonder about moves into the Output Area. We can summarize our discussions by reviewing these points.

Still remembering that we have simplified our diagram by assuming that Working Storage, Constant Section, Common Storage, and *Common Storage are all rolled into one storage area (returning storage), and remembering that Process Storage is not under programmer control, we note that no new names can appear under Output Files. So all data names are found under Input Files or Common Storage, or are intermediate fields automatically assigned in Process Storage as explained.

If we say R is a data name in input, W is a data name in Common Storage and Z an intermediate data name in Process Storage, we can make the following observations about transfers:

- The transfer of R from the Read Area to Process Storage, if performed, is automatic, as already stated. The programmer has no direct control over this.

- The sentence, MOVE R TO W. moves the input R to Common-Storage W regardless of whether or not R has been previously transferred to Process Storage. The compiler will decide from which place to make the move to W for best efficiency and effectiveness. The warnings given previously about destroying throughput fields and nonreceiving fields apply here if R is either of these types. On the other hand, if R is receiving field, it will always be moved from Process Storage by the sentence above.

- The sentence, MOVE Z TO W. moves the intermediate quantity Z from Process Storage to Common Storage. (This will seldom be necessary except to change modes.)

- Moves to output are not directly controlled by the programmer. He only writes the sentence, WRITE so-and-so RECORD, and the compiler generates all coding necessary to gather, convert, edit, and record on the output medium all the data items listed in the data description for so-and-so Record. Specifically, the programmer never writes MOVE sentences to handle transfers to output. He indicates the sources of his output by the data names he lists in his output description.

- The format or mode of an output quantity is the same as its format or mode at its source unless the programmer specifies otherwise by a data image in his output description.

- There is a special kind of move to output wherein all throughput data in an input record is transferred from the Read Area to the Write Area. The programmer accomplishes this by writing the name of the input record in his output description, giving the name of the input file as a qualifier. Together, these are the names of the output record. If any field in the input record has been a receiving field, the new value is sent to output (from Process Storage.)

GE-225

GECOM-II

COBOL COMPATIBLE

OPERATIONS MANUAL

CD225H1.005

NOVEMBER 1962

(Revised April 1963)

(Revised December 1963)

(Revised April 1964)

GENERAL ELECTRIC

COMPUTER DEPARTMENT

CONTENTS

I. INTRODUCTION

Page No.

II. GE-225 GENERAL COMPILER COMPILATION PROCESS

III. COMPILER OPERATING INSTRUCTIONS

IV. SOURCE PROGRAM ERROR MESSAGES

Page No.

V. EDITED LIST

VI. OBJECT PROGRAM OPERATING INSTRUCTIONS

VII. OBJECT PROGRAM MESSAGES

VIII. OBJECT PROGRAM SUBROUTINES

ILLUSTRATIONS

[+]

Material in paragraphs so marked is not available at the present time. Addenda will be forwarded to you as the material becomes available.

General Electric GE-225 Information Processing System

I. INTRODUCTION

The General Compiler (GECOM) is an automatic coding system which translates a program described in user oriented language (source program) into a machine language program (object program). The compiler is used to obtain the object program. The object program is used to obtain the results the user desires from the computer. The user must provide whatever input data is necessary for the object program to arrive at the desired output results. Thus, there are three basic steps to be followed in proceeding from problem definition to end results:

1. Describe the program and data in source language;

2. Compile the object program from the source program;

3. Execute the object program upon the input data to obtain the output results.

The rules for writing GECOM source programs are stated in the GE-225 GECOM-II REFERENCE MANUAL. It is the purpose of this GECOM-II OPERATIONS MANUAL to provide the information necessary for the operation of the compiler and the object programs on the computer.

II. GE-225 GENERAL COMPILER COMPILATION PROCESS

This chapter provides a brief, general description of the process followed by the compiler in the development of an object program.

The compiler is similar to a business data processing application in the following respects:

1. It operates upon input: the source program;
2. It consists primarily of a series of sequential runs (overlays) on a master instruction tape (the compiler system tape);
3. Compiler runs produce intermediate files of internal data;
4. It produces an output: the object program;
5. It produces a report: the Edited List.

Figure 1 shows the compilation process including the major input and output options. To perform a compilation, the start card and SCC card are placed in the card reader. These cards control locating GECOM, then reading of the first part of the compiler into the machine memory and initiate execution.

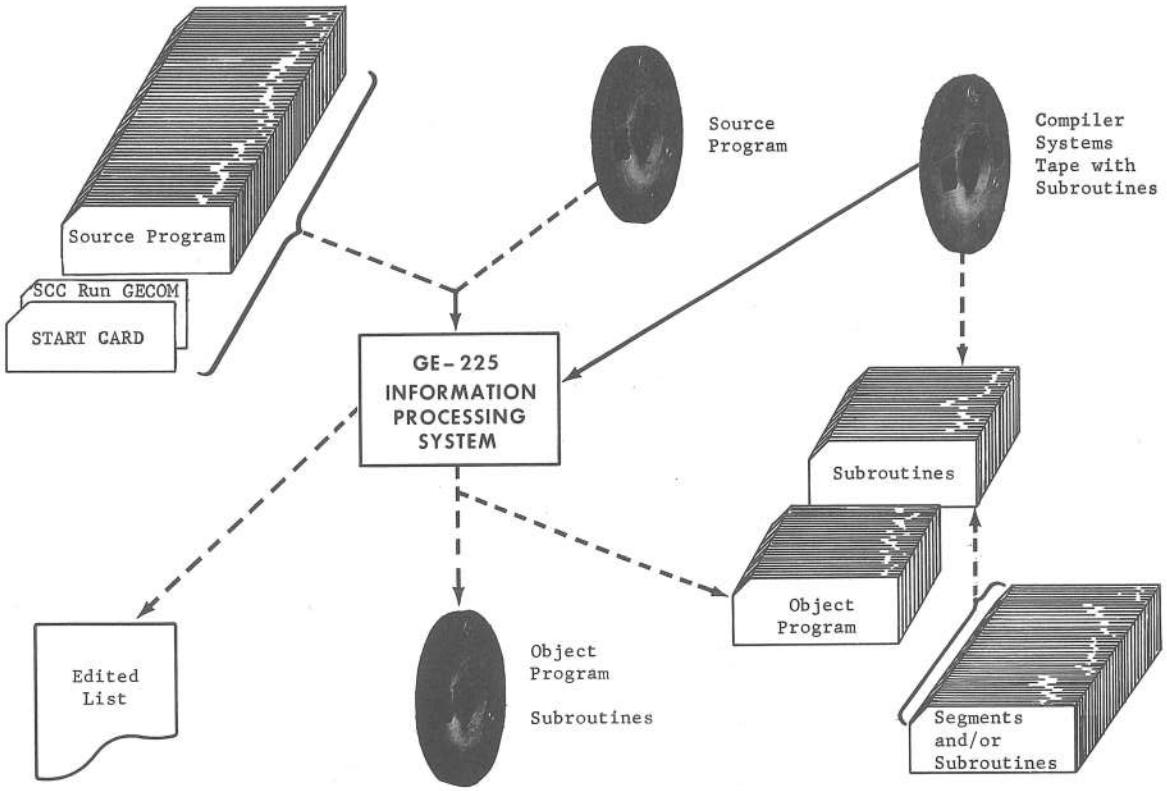


Figure 1. Compilation Process Chart

As indicated previously, the compiler consists of a series of sequential runs (overlays) on a system tape. Figure 2 shows the major overlays in the system. Note that a library of object program subroutines is filed at the end of the compiler.

#OVT010EN	Environment Division
#OVT020RW	Report Writer Translator
#OVT020RW	Report Writer Analyzer
#OVT030DD	Data Division
#OVT031DD	
#OVT032DD	
#OVT040PD	Procedure Division
#OVT050TA	TABSOL
#OVT051TA	
#OVT060PN	Procedure Name Analyzer
#OVT061PN	
#OVT070LT	Internal Language Translator
#OVT071LT	
#OVT0800A	Output Analyzer
#OVT0810A	
#OVT090CC	
#OVT090AA	Arithmetic Analyzer
#OVT100SI	Simple - IF Analyzer
#OVT110ME	Move-Exchange Analyzer
#OVT120PN	Constant Pooling
#OVR010	Reformer
#OVEXGEXG	Expression Generator
.	Simple and IF Generators
.	Move and Exchange Generators
.	Input/Output Generators
.	Pseudo Instruction Generator
.	Assembler
#OVEDT11	Editor
.	Object Program Subroutines
.	
.	

Figure 2. Compiler System Tape Layout

There are four phases in the compiler:

1. Transformer
2. Reformer
3. Assembler
4. Editor

Each phase consists of one or more runs.

TRANSFORMER PHASE

The Transformer phase transforms the source program into an internal language more suitable for processing by the generators. The source program is broken down into short records containing codes, indicators, and symbols (all called generator parameters) which describe for the rest of the compiler what action is to be performed on what data. The parameters are dispersed into files by category of function.

The ENVIRONMENT DIVISION run prints the Identification Division on the printer and copies the Identification and Environment Divisions onto the listing tape. It checks the Environment Division for validity and consistency, and prints messages whenever any errors or apparent errors are detected. It sets indicators in memory specifying the object computer configuration and other environment options. It starts building the procedure name table in memory by assigning short symbols for section names encountered. It also prepares a table of input/output file parameters (codes, indicators, symbols, etc.) for each data file showing hardware assignment, options such as rerun, etc. These tables are written out in a tape file (called IOFP).

The REPORT WRITER runs consist of a translator run followed by an analyzer run. The translator processes the Report Section and copies it onto the listing tape. It checks all entries for validity and consistency, and prints messages whenever any errors or apparent errors are detected. It scans report line layout information in conjunction with line definition information to create generator parameters for output data descriptions (GPOD) and generator parameters for literals and constants (GPPS). These are the same type of internal parameters as those created later by the Data Division run (see below) for files described in the File Section. In addition to these functions, the translator recognizes Report Writer procedure parameters in source form and creates packed report parameter lists (RPL) on tape for use in the Report Writer Analyzer run. The analyzer run accepts the RPL parameters from tape, performs further analysis, error detection, and consolidation of these parameters in order to generate Report Writer procedures. Pseudo procedure statements containing preassigned symbolic data names are written on tape in blocked form for efficient handling by the Procedure Division run. Process storage requirements of the Report Writer generated procedure are communicated to the Data Division by means of data report parameter lists (DRPL) written on tape during the analyzer run.

The DATA DIVISION run copies the Data Division onto the listing tape. As it does so, it checks all entries for validity and consistency, and prints messages whenever any errors or apparent errors are detected. It forms a data name table in memory including coded data descriptions for each entry. Symbols are assigned for file names, record names, literals, and constants. The file tables started in the Environment Division run are read in and information is added to the tables from the file descriptions contained in the Data Division, (e.g., block size, control keys, etc). The generator parameters for output data descriptions are written out in a tape file (called GPOD). Generator parameters for literals and constants are written out in a tape file (called GPPS or Generator Parameters for Pseudo Instructions) for use by the Pseudo Instruction Generator (PIGN). The data name table and file tables are written on magnetic tape.

The PROCEDURE DIVISION run copies the Procedure Division onto the listing tape. As it does so, it checks all sentences for validity and consistency and prints messages whenever any errors or apparent errors are detected. Noise words are bypassed and dropped from further processing. Verbs and other key words are converted into codes and indicators (parameters) designating the functions and options specified by the sentences. This run creates generator parameters from each sentence. These parameters are records containing verb (or generator) numbers, sentence numbers, symbolic sentence names, and other codes and indicators designating the various functions and options specified by the sentences. Data names used as operands are also carried among the parameters. At this point, they have not been assigned symbolic names. The generator parameters derived are selected and written out on one tape file.

The TABSOL run processes all decision tables. As it does so, it checks all table entries for validity and consistency and prints messages whenever errors are detected. It creates generator parameters from each secondary and primary block combination. The parameters are identical to those generated by the Procedure Division and are merged onto the same tape with the Procedure Division parameters.

PROCEDURE NAME ANALYZER processes all section, segment, and Procedure statement names from the Procedure Division. It builds these names into an internal search table, replacing each source name within the parameter lists with the appropriate 3-character symbolic name. The MOVE, EXCHANGE, and WRITE on typewriter generator parameters are merged onto one magnetic tape and the other generator parameters on a separate magnetic tape.

The INTERNAL LANGUAGE TRANSLATOR run integrates information from the Data Division with information from the Procedure Division and translates the resulting parameters into the internal language format. Simultaneously, this run detects additional errors, and prints messages accordingly. It reads in the parameters created by the Procedure Division run. It assigns unique symbolic names to those data names referenced in the Procedure Division thereby completing the data name table. The source program data names among the parameters are replaced with the assigned symbols. "IF" generator parameters are created for conditional names. The expanded generator parameters for MOVE and EXCHANGE and other type sentences are written out in tape files (GPME and GPOT). Additional information is inserted into the input file tables, and they are written out in a tape file (INFP). Input data "unpacking" parameters are developed for input fields that are used in arithmetic or IF sentences. These parameters are written out in a tape file (called GPID).

The OUTPUT ANALYZER run reads in the parameters for output data descriptions to build output data packing parameters. The parameters for output descriptions are then written out again. The output analyzer adds information to the output file tables and writes them out in a tape file (called OTFP). The analyzer adds parameters to the generator parameters for pseudo instructions to designate storage reservations for working storage, process storage, buffer areas, etc. These are placed in the GPPS file. Finally, the Output Analyzer run writes the data name table on the listing tape.

The ARITHMETIC, SIMPLE-IF, and MOVE-EXCHANGE ANALYZERS perform a very detailed analysis upon the parameters produced by the preceding runs. The parameters which are input to the analyzers are general in nature and correspond roughly to the broad verb-like actions specified in procedure sentences. GECOM generators are not excessively general in nature. (There is not a one-for-one correspondence between verbs and generators). One sentence can result in calls for many generators. GECOM generators are tailored to specific functions in given sets of circumstances. Therefore, the Analyzers refine input parameters into more detailed parameters for specific generators, dependent upon functions and options specified in sentences, format of data, etc. Because of the detailed analysis performed, the Analyzers are able to detect subtle errors which have not been caught in preceding runs. They print appropriate error messages. Thus, they read in the parameters derived from sentences, analyze them, and formulate more detailed parameters. These are separated and written out in three tape files. One file (GPEX) contains the parameters for the Expression Generator (EXGN). A second file contains the parameters for SIMPLE and IF Generators (SIGN). The third file (GPME) contains the parameters for the Move and Exchange Generators (MEGN). Parameters for constants used in the Procedure Division are added to the GPPS file for use by the Pseudo Instruction Generator (PIGN). The analyzers leave two lists of generator names in memory. One tells the Reformer which SIMPLE and IF generators are required. The other tells the Reformer which MOVE and EXCHANGE generators are required.

REFORMER PHASE

The GECOM Reformer is essentially an executive routine. It calls in only those generators which are required. It relocates them in memory and allows them to operate upon the parameters. It includes input, output, and other service subroutines which are used in common by the various generators. Generator loads are brought into memory in the following sequence:

1. Expression Generator if required.
2. Required Simple and If Generators.
3. Required Move and Exchange Generators.
4. Input File Table Generator.
5. Input Coding Generator.
6. Input Unpacking Generator.
7. Output File Table Generator.
8. Output Coding Generator.
9. Output Packing Generator.
10. Input/Output Service Generators - three loads.
11. Pseudo Instruction Generator.

As the generators operate, they produce General Assembly Program (GAP) symbolic coding to perform the functions specified by the parameters upon the data described by the parameters. It should be noted at this time that there are many constants and subroutines which are common to many (if not all) object programs. It would be wasteful of computer time for the compiler to generate such constants and subroutines during every compilation. Therefore, the generators only produce references to the constants and/or calling sequences to the subroutines, both of which are assumed to be placed in the object program by other methods mentioned later.

If any coding is produced by the Expression Generator, it is written out on tape. The parameters derived from the source program Procedure Division were dispersed to the various parameter files. The Reformer must merge the symbolic coding back into one consecutive file in source program sentence sequence. As the SIMPLE and IF Generators produce coding, it is merged into one tape file with any coding produced by the Expression Generator. When the MOVE and EXCHANGE Generators produce coding, it is merged into one tape file with the coding produced by the EXPRESSION, SIMPLE, and IF Generators. From this point on, all coding produced in subsequent generator loads is simply added onto the end-of-tape file.

ASSEMBLER PHASE

The Assembler phase is a modification of the General Assembly Program (GAP). The first pass of GAP is omitted because it would perform unnecessary work. GAP reads the tape file containing the generated symbolic coding, assembles the coding into machine language, and punches out the object program. At a later date, GAP will optionally write the object program on magnetic tape. Instead of printing the usual assembly listing, GAP writes it on the end-of-listing tape.

EDITOR PHASE

The Editor phase is described in detail in Chapter V of this manual. Essentially, it uses the listing tape to print the documentation of the program. This documentation, the Edited List, includes the entire source program as written by the user, a merged list showing the generated symbolic coding side by side with the assembled machine coding produced for each sentence and several cross reference tables. The Editor optionally punches out the required object program subroutines if the object program is on cards or writes the subroutines on tape if the object program is on tape.

At the end of compilation, the user has an Edited List (if he has not inhibited it through console switch settings) and an object program on tape or cards. If the object program is on tape, a loader will have been written at the beginning of the tape. If the object program is on cards, the user will have to place a copy of the MCML II Loader and a type 1 in front of the object program deck and the required object program subroutines at the back of the deck (if the subroutines have not been punched out by the Editor). If no error messages have been printed during compilation, the user is then in a position to test his object program. He should consult the operating instructions for object programs. (See Chapter VI.)

III. COMPILER OPERATING INSTRUCTIONS

GECOM - Model II - compiles with four or five magnetic tape handlers. The source program may be input from the card reader or magnetic tape. (See Figure 3.)

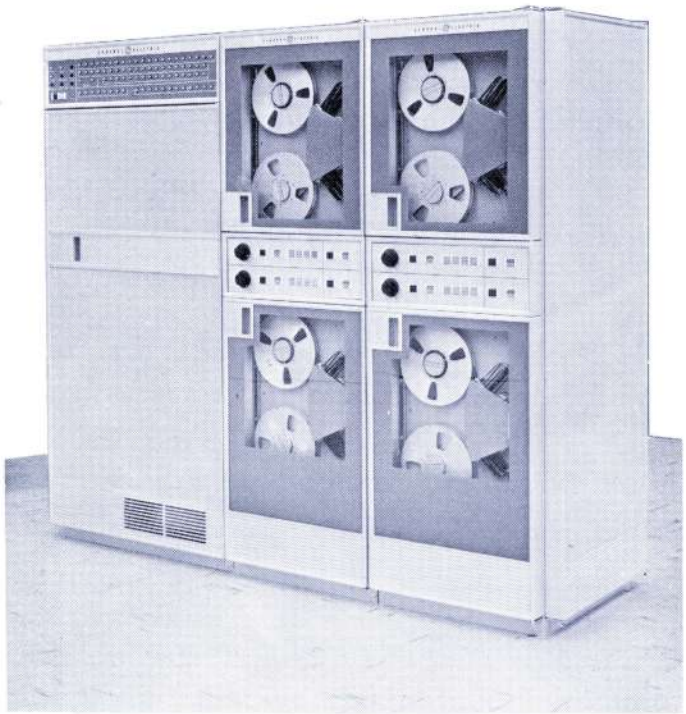


Figure 3. GE-225 Magnetic Tape Handlers and Controllers

SOURCE COMPUTER CONFIGURATION

The minimum source computer (compiling) configuration for GECOM is:

GE-225 Central Processor with 8192 words
Typewriter
Card Reader
Card Punch
High Speed Printer
Magnetic Tape Controller
4 Magnetic Tape Handlers

The maximum configuration may include:

5 Magnetic Tape Controllers
5 Magnetic Tape Handlers

TAPE HANDLER SETUP

Tape Handler	1	System Tape containing GECOM
Tape Handler	2	Blank
Tape Handler	3	Blank
Tape Handler	4	Blank
Tape Handler	5	Optional. May contain the source program.

Four Magnetic Tape Option:

The source program must be input to the compiler from the card reader.

Five Magnetic Tape Option:

1. If the source program is input to the compiler from the card reader:

   a. The source program will be written on tape 5 in the BCD mode, single card images. Tape 5 will not be in BRIDGE II format.

   b. GECOM will not rewind tape 5 before or after compilation.

2. If the source program is input to the compiler from tape 5:

   a. Tape 5 may be in BRIDGE II format, or

   b. Tape 5 may be in BCD mode, single card images. (See paragraph 1, a above.)

   c. GECOM will not position tape 5 to start compilation. (See Option 2 under "Source Deck Setup for Compilation".)

   d. GECOM will not rewind tape 5 before or after compilation.

   e. For placement of source programs on tape 5 in BRIDGE II compatible format, see BRIDGE II (CD225J1.001), Symbolic Tape Maintenance.

SOURCE DECK SETUP FOR COMPILATION

1. START Card to rewind system tape and call in locator. For explanation and preparation instructions see BRIDGE II, CD225J1.001.

2. SCC Card (optional) to locate source program on tape 5 and leave tape 5 positioned for input to GECOM. For explanation and preparation instructions, see BRIDGE II, CD225J1.001.

3. SCC Card to locate GECOM run label and start execution. For explanation and preparation instructions see BRIDGE II, CD225J1.001.

4. Plug Parameter Card (optional). The GECOM system contains a built in Plug Parameter Card. The system card is designed for four tape compilations on a standard configuration of tapes 1, 2, 3, and 4 on plug 1, and printer on plug 6. The customer may modify the compiler system card to conform to his own installation's normal plug assignments if different than the standard configuration. (See "Modification of GECOM for Nonstandard Plug Assignments".)

   Furthermore, the user may override the system card by placing a Plug Parameter Card in the source deck and setting console switch 13 down. This is a Hollerith punched card in the following format:

Column	
1	Plug number of tape handler 1.
2	Plug number of tape handler 2.
3	Plug number of tape handler 3.
4	Plug number of tape handler 4.
5	Plug number of tape handler 5.
6	Not used.
7	Plug number of printer.
8	4 punch if four tape option is required. 5 punch if five tape option is required.

5. Source Program if on cards. See Appendix B of GECOM Reference Manual for sequence of source program deck.

6. Blank Card if source program is on cards.

7. EBS Card. For explanation, preparation instructions, and information on optional run completion and linkage, see BRIDGE II, CD225J1.001.

Option 1: 4 or 5 Tape Compilation:

Source program is on cards. Source deck sequence is as follows:

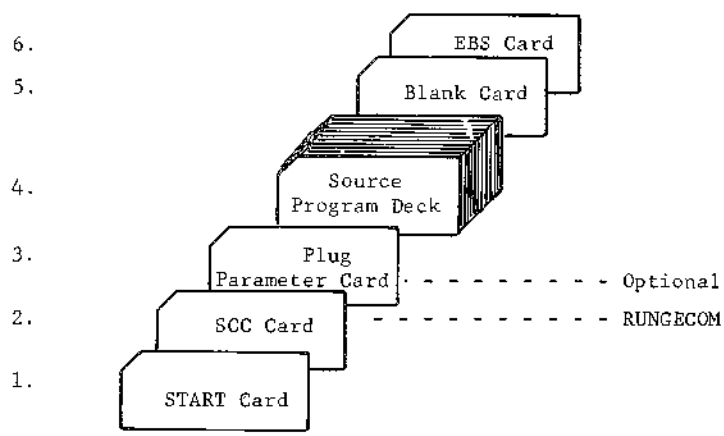
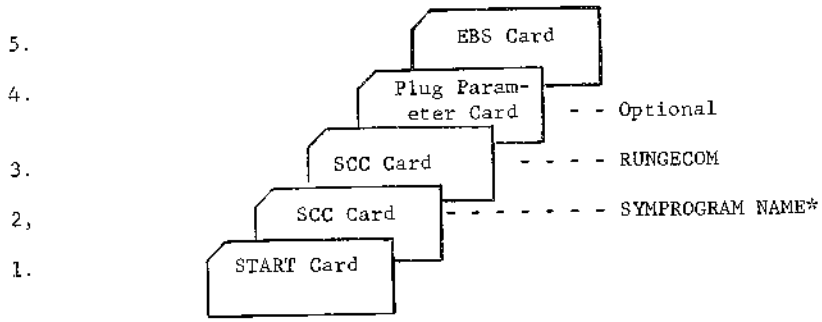


Figure 4. Option 1 - Source Deck Sequence

Option 2: 5 Tape Compilation:

Source program is on tape 5 in BRIDGE II compatible format. Source deck sequence is as follows:



*SCC SYM (Source Program Name) will locate the named source program on tape 5 and leave it positioned for input to GECOM.

Figure 4a. Option 2 - Source Deck Sequence

Option 3: 5 Tape Compilation:

Source program is on tape 5 but not in BRIDGE II compatible format. Source deck sequence is as follows:

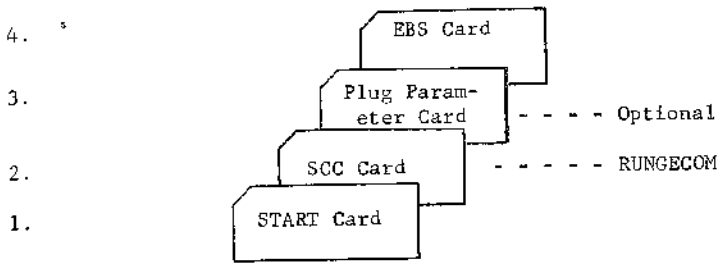


Figure 4b. Option 3 - Source Deck Sequence

STARTING COMPILATION

Place source deck in card reader.

Read the START card into memory.

Reset "P"; start.

The plug, tape, and label of the system tape are typed.

The SCC card is read.

"LOCATING RUNGECOM" is typed.

A console switch loop is entered.

Set the console switches for this compilation. (See options below.)

Toggle switch zero to cause GECOM to be located and compilation to start.

CONSOLE SWITCHES AND OPTIONS (Control Console shown in Figure 5.)

Console Switch 1: Down - If the object-program is assigned to the card reader, the required subroutines will be punched after the main program.

Switch 1 should be down whenever the object program requires the linkage subroutine AP2 or MI2 since these routines are generated by the compiler and are variable. A linkage subroutine is required whenever an API OPEN entrance is given in the Environment Division.

If the object program is written on magnetic tape by the compiler, the required subroutines are always supplied after the main program regardless of the setting of switch 1.

Console Switch 2: Down - The input/output, REPORT WRITER, and GECOM common coding is listed.

Console Switch 3: Down - The Editor may be rerun at end of compilation.

Up - The standard run completion routine is entered at the end of compilation; see BRIDGE II, CD225J1.001.

Console Switch 4: Down - The complete Edited List is suppressed.

Console Switch 5: Down - The complete object program (punched cards or magnetic tape) is suppressed.

Console Switch 13: Down - A plug parameter card is supplied after the SCC RUNGECOM card.

Console Switch 16: Down - The source program is on tape handler 5 in the BCD mode, single card images.

Console Switch 17: Down - The compiler loops ($1553-1560_8$) at the end of the Transformer phase to permit examination of error messages. Compilation may be continued by raising switch 17.

Console Switch 18: Down - The source program is sequence checked.

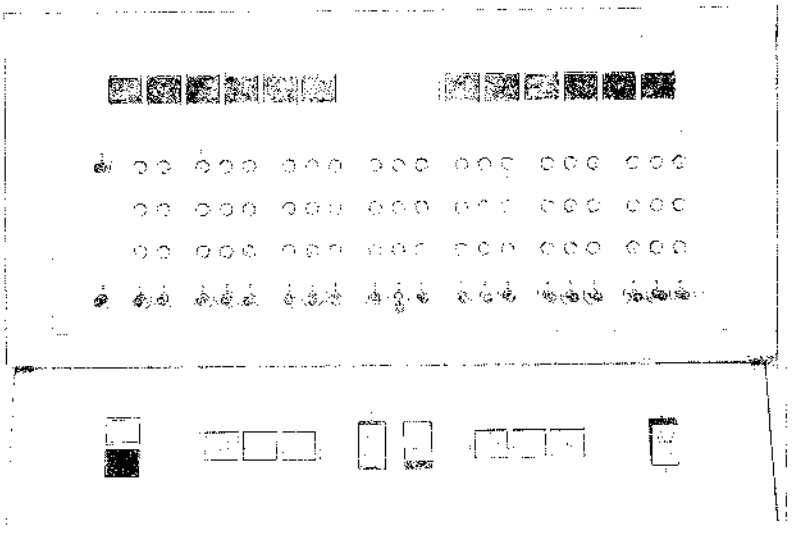


Figure 5. GE-225 Control Console

OUTPUT

High Speed Printer

The compilation log (run identifications and source program error messages followed by the Edited List as controlled by the console switch settings.

Card Punch

The binary object program if assigned to cards optionally followed by the required subroutines.

Tape Handlers:

1. System tape containing GECOM
2. Blank or object program. (See typeout
3. "OBJECT PROGRAM ON.")
4.
5. Optional. May contain source program.

Typewriter

Logged messages for this compilation follow.

LOGGED TYPEWRITER MESSAGES

STOP	TYPEWRITER MESSAGE	EXPLANATION	OPERATOR ACTION
	"Bad Tag in PIGN"	Compiler error. Incorrect input to the Pseudo Instruction Generator has caused many undefined pseudo instructions.	Report this to the G. E. Computer Department with source deck included.
	"CARD RD"	The last card was read in error.	Put the computer in manual. (Depress manual.) The card that was read in error is the last card in the hopper. This card must be reread. Empty the Read station. Place the card that was read in error on the read station followed by the remainder of the unread cards; put the computer in Automatic; start; toggle switch zero.
	"COL 8 ERR"	Column 8 of plug parameter card does not contain "4" or "5" to indicate number of tapes for compilation.	Correct plug parameter card, place in card reader, and toggle zero to accept corrected card.
	"END GECOM (Program~ID)"	End of compilation.	If switch three is down, the Editor may be rerun to obtain additional copies of the Edited List and/or the object program. To initiate Editor rerun, reset console switches to indicate new listing or object program preparation instructions; then toggle switch zero.
	"xxxxx EQU"	An "EQU" has been improperly used at location xxxxx. The operand symbol has not been previously defined.	None.
	"ERR T Y "	Error on tape Y that cannot be corrected.	Mount new tape or change handlers and recompile.
	"E7 T Y "	Control error. Possible causes: 1. Tape handler Y in local or nonexistent. 2. Two or more tape handlers assigned to Y.	Check tape assignments. Reset alert halt on controller. Toggle switch zero. Compiler attempts to rectify error.

	"GECOM XT MODAYR (PROGRAM~ID) (NEXT~PROGRAM)"	Start of compilation. "X" is "4" or "5" indicating number of tapes to be used in compilation. MODAYR represents date of last GECOM change letter.	None.
	"LOCATING RUNGECOM"	Sequential run locator is ready to locate compiler and begin compilation.	Set appropriate console switches, and toggle zero to start.
	"OBJECT PROGRAM ON" [PXTY] or ["CARDS"]	The location of the resulting object program.	
	"PLUG ERR"	Plug parameter card: 1. has tape and printer assigned to same plug or 2. plug number is not 0-7.	Correct plug parameter card; place in card reader; toggle zero to accept corrected card.
	"PRINTER"	Printer is out of paper or does not come ready.	Supply paper or make it ready and toggle 0.
	"PUNCH"	Card punch does not come ready.	Fix; then toggle switch zero.
	"PX TY (BTL)"	Label of system tape on plug X, tape handler Y.	None.
	"PX TY E4"	Parity error from tape handler Y plug X.	Mount a new tape or change handlers and start from the beginning.
	"PX TY E5"	Mod 3 or 4 error.	Mount a new tape or change handler and start from the beginning.
	"PX TY E6"	I-O buffer error.	Mount a new tape or change handlers and start from beginning.
	"PX TY E7"	Control Error Possible causes: 1. Tape Handler Y in local or non-existent. 2. Two or more tape handlers assigned to Y.	Put computer in manual. Recheck tape assignments, correct error. Reset alert halt on tape controller. Put computer in automatic. Start, toggle switch 0. (The program will attempt to rectify the error.

	"SWITCHES"	Inconsistent switch settings have been sensed.	Correct the settings and toggle 0. If correction cannot be supplied, put switch 19 down and then toggle 0 (in this case, program assumes lowest of inconsistent switches and ignores others). Example: both 1 and 5: 1 assumed both 2 and 4: 2 assumed.
	(S) "SYMBOL LOST"	The symbol (S) has been lost from the symbol table. Normally this is a result of symbol table overflow.	Toggle switch 19 to ignore and continue compilation. The symbol will be typed out, being undefined in the GECOM listing. Typeout and halt will be repeated for each lost symbol unless switch 19 is depressed, in which case, typeout will be repeated but halt will not. If SYMBOL TABLE OVERFLOW has not occurred, mount a new tape or change handlers and start from the beginning.
	"SYMBOL TABLE OVERFLOW"	More than 1185 symbols have been referenced	This will result in one or more undefined symbols on the GECOM listing and a non-executable Object Program. Stop compilation, return all output to the author.
	"TAPE X"	"X" represents controller.	Device does not come ready. Fix; then toggle switch zero.
	"WTS"	Errors have been detected in Run T030DD which do not permit compilation to continue.	Correct the errors indicated in messages printed on the listing and recompile.
	"WTV"	The source program Data Division has exceeded the internal table in Run T030DD	See "Overflow Condition" in Chapter V, GECOM II REFERENCE MANUAL.

COMPILER STOPS

1553_8 loop 1560_8	- - -	This loop occurs at the end of the Transformer if switch 17 is down.	Examine listing on printer. If there are no error messages, or if it is desired to continue compilation in spite of error messages, toggle switch 17 to continue
17777_8	- - -	Compiler has halted in the TABSOL run.	Check the last error message on the listing in the printer to determine reason why compilation cannot continue.

COMPILATION ERROR DOCUMENTATION

The following procedure is necessary to provide documentation for an error occurring during a GECOM compilation. The indicated documentation should accompany the completed GE-225 SOFTWARE ERROR REPORT form.

Standard Procedure

1. Printer output of compilation error messages.
2. Manual dump of the first 300 memory locations on the printer.
3. Memory dump.
4. Logged typewriter messages.
5. A copy of the source program cards.
6. A source program listing.

The following information should be included depending on the phase of the compiler in which the error occurred. The particular phase in which the error occurred is the last run identification printed following the IDENTIFICATION DIVISION. All tape dumps must be octal, unless otherwise specified below.

Run Identification

T010EN	IDENTIFICATION, ENVIRONMENT DIVISION Standard procedure and dump of tapes 2, 3, and 4.
T020RW	REPORT WRITER TRANSLATOR Standard procedure and dump of all files on tapes 2 and 4.
T021RW	REPORT WRITER ANALYZER Standard procedure and dump of all files on tapes 3 and 4.
T030DD	DATA DIVISION Standard procedure and dump of all files on tapes 2, 3, and 4.
T040PD	PROCEDURE DIVISION Standard procedure and dump of tape 4.
T050TA	TABSOL Standard procedure and dump of all files on tapes 3 and 4.
T060PN	PROCEDURE NAME ANALYZER Standard procedure and dump of all files on tapes 3 and 4.
T070LT	INTERNAL LANGUAGE TRANSLATOR Standard procedure and dump of tapes 2, 3, and 4, approximately 8 files on tape 2, and 5 files on tapes 3 and 4.
T080OA	OUTPUT ANALYZER Same as T070LT.
T090AA	ARITHMETIC ANALYZER Standard procedure dump of 1 file each on tapes 2, 3, and 4.

T100SI SIMPLE-IF ANALYZER
Standard procedure and dump of 4 files on tapes 2, 3, and 4.

T110ME MOVE-EXCHANGE ANALYZER
Standard procedure and dump of all files on tapes 2, 3, and 4.

REFORMER STANDARD Procedure and DECK of source program is always necessary.

MODIFICATION OF COMPILER FOR NONSTANDARD PLUG ASSIGNMENTS

The GECOM system tape is created and maintained by the BRIDGE II Service System, CD225J1.001. To modify the compiler, knowledge of BRIDGE II and the Octal to Binary BRIDGE Correction Protection routine, CD225B6.004R, is required.

System Plug Parameter Card

The compiler system contains a built in plug parameter card. The customer may modify the compiler system cards to conform to his own installation normal plug assignments if different than the standard configuration. Alternatively, a plug parameter card may be placed in the source deck for each compilation. (See "Plug Parameter Card" page III-3 and "Console Switches and Options" page III-5.) If the user does change the system plug parameter card, the option of using one in the source deck still remains.

The system plug parameter card is designed for the 4-tape option with all tapes on plug 1 and the printer on plug 6.

Format of the system plug parameter card is as follows:

Col.	1	=	1	(Plug number of tape 1)
Col.	2	=	1	(Plug number of tape 2)
Col.	3	=	1	(Plug number of tape 3)
Col.	4	=	1	(Plug number of tape 4)
Col.	7	=	6	(Plug number of printer)
Col.	8	=	4	(Number of tapes to use for compilation)

Modification of System Plug Parameter Card

The following BRIDGE control and change cards must be prepared by the user:

1. START Card
2. SCC Card
3. DOP Card
4. THA Card
5. COP Card
6. COR Card
   Correction Card
7. CPT Card
8. Blank Card
9. EBS Card

The following example of card preparation assumes:

1. BRIDGE II system tape is mounted on plug 1 tape 7.
2. GECOM system tape is mounted on plug 1 tape 6.
3. Blank tape for updated GECOM system to be written is on plug 1 tape 1.
4. Printer is on plug 6.

(Above plug and handler numbers are for purposes of example only. User may assign accord- to his own environment.)

Control Deck Example

1. START Card for plug 1 tape 7
   (See BRIDGE II, CD225.J1.001.)

2. SCC Card (to locate BRIDGE II) format:

   Col. 1-3 SCC
   Col. 22-33 RUNBRIDGEΔII
   Col. 53 1 (Plug number)
   Col. 54 7 (Tape number of BRIDGE II system)
   All other columns blank

3. DOP Card format:

   Col. 1-3 DOP
   Col. 7-12 MODAYR (Current date)
   Col. 22-23 NO (No library functions)
   Col. 38 1 (Plug number of output tape)
   Col. 39 1 (Tape number of output)
   Col. 61-62 NO (Do not produce SCC card)
   All other columns are blank.

4. THA Card format:

   Col. 1-3 THA
   Col. 7 1 (Plug number of input)
   Col. 8 6 (Tape number of input)
   All other columns are blank.

5. COP Card (Copy function) format:

   Col. 1-3 COP
   Col. 7-18 BTLGECOMΔΔΔΔ (Label of tape containing GECOM)
   All other columns are blank.

6. COR Card format:

   Col. 1-3 COR
   Col. 7-18 RUN GECOMΔΔΔΔ (GECOM run label)
   Col. 22-23 #OVEXECUT (None of overlay to be changed)
   Col. 58-60 CHK (Request for correction protection)
   All other columns are blank.

6.a Octal Correction Card:

Format:

Col. 5	0	(Relocatable Bit indicator, always zero)
Col. 7-10	1400	(Octal location, always 1400)
Col. 12-18	0AABBCC:	

AA represents: tape 1 plug number. If tape 1 is to be on plug 2, the content would be 02.

BB represents: tape 2 plug number. If tape 2 is to be on plug 2, the content would be 02.

CC represents: tape 3 plug number. If tape 3 is to be on plug 2, the content would be 02.

Col. 20-26 0DDEEFF:

DD represents: tape 4 plug number. If tape 4 is to be on plug 2, the content would be 02.

EE represents: tape 5 plug number. If tape 5 is to be on plug 2, the content would be 02.

FF: blank, content is 60.

Col. 28-34 0GGHHII:

GG represents: printer plug number. If printer is to be on plug 5, the content is 05.

HH represents: number of compilation tapes. If 5 tapes are to be used for compilation, the content is 05.

II: blank, content is 60.

Col. 67-75 #0VEXECUT (Name of overlay to which the changes are to be applied)

Thus, an octal correction card for a change to 5 tape option, all tapes on plug 2, printer on plug 5, would be punched as follows:

Col. 5	7	12	20	28	67
0	1400	0020202	0020260	0050560	#0VEXECUT

An octal correction card for a change to 5 tape option, tape 1 on plug 1, tape 2 on plug 2, tape 3 on plug 1, tape 4 on plug 2, tape 5 on plug 1, printer on plug 6 would be punched as follows:

Col. 5	7	12	20	28	67
0	1400	0010201	0020160	0060560	#0VEXECUT

6.b Binary Correction Card:

The user must process the octal correction card through the Octal to Binary BRIDGE Correction Protection routine, (CD225B6.004R), to obtain the binary correction card for insertion in the BRIDGE control deck. The sequence of the example deck for input to the Correction Protection routine is as follows:

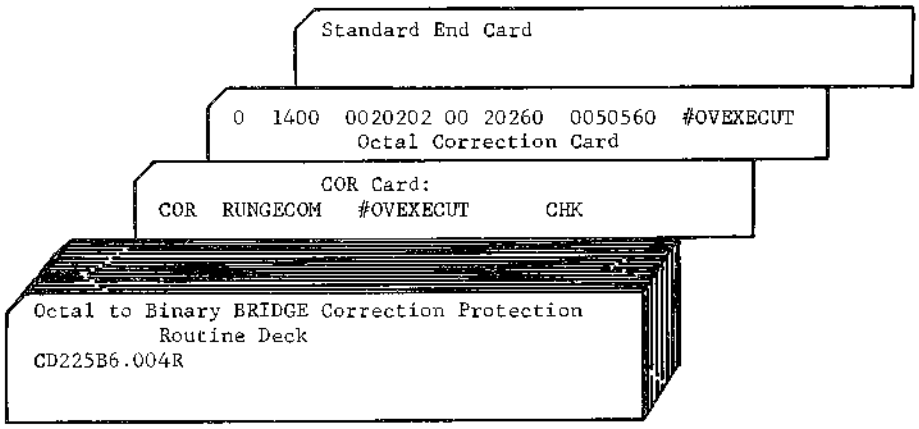


Figure 5a. Sequence of Example Deck for Input to Correction Protection Routine

The Correction Protection routine punches out the following cards:

COR Card
Binary Correction Card

These two cards should be placed in the BRIDGE control deck between the COP and the CPT cards.

7. CPT Card format:

Col. 1-3 CPT

8. Blank card

9. EBS Card format:

Col. 1 0-7-8 punch
Col. 2 0-7-8 punch
Col. 3 0-7-8 punch
Col. 4-6 EBS

Operating Instructions

After mounting tapes and placing control deck in the card reader, follow operating instructions for BRIDGE II, (CD225J1.001). The sequence of the BRIDGE control deck is as follows:

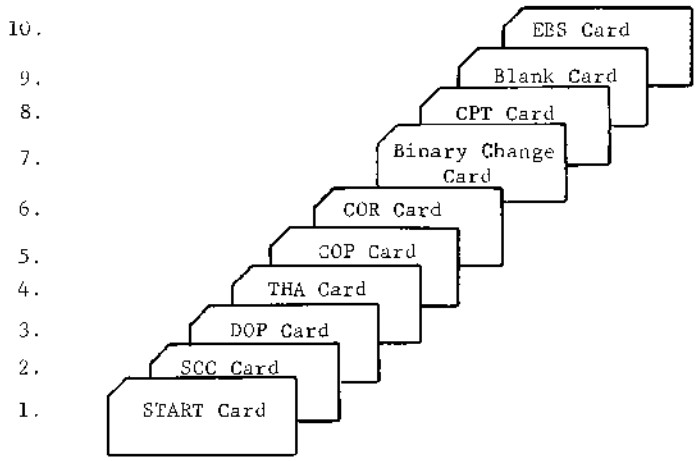


Figure 5b. Sequence of BRIDGE II Schedule Control Deck

The output from the previous example will be an updated GECOM system tape on plug 1 handler 1.

OPERATING INSTRUCTIONS FOR APPLYING CHANGES TO COMPILER

Preparation for operation of the computer for correcting the compiler with BRIDGE II requires the BRIDGE II schedule control deck to be arranged as follows:

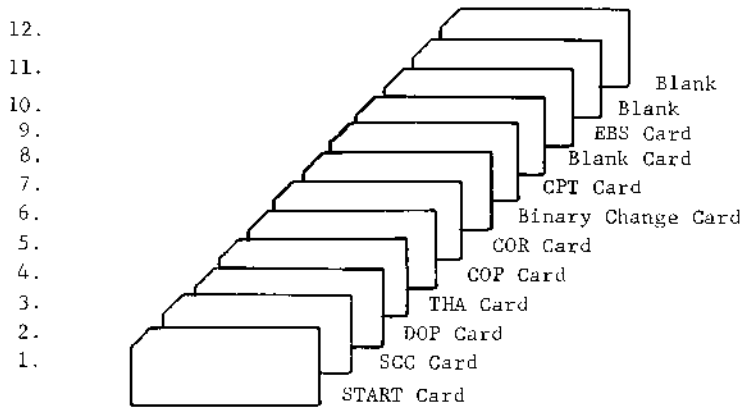


Figure 5c. Sequence of BRIDGE II Schedule Control Deck for Correcting the Compiler.

The START card must have the plug number and tape handler number of the BRIDGE II system tape punched in columns 79 and 80 respectively. These are the same plug and tape numbers that are punched in the SCC card.

Operator setup is as follows:

1. The BRIDGE II system tape to be mounted on the tape handler specified in the START and SCC cards.
2. The GECOM system tape must be mounted on the tape handler specified on the THA card.
3. A blank tape must be mounted on the tape handler specified in the DOP card as the output for the updated GECOM System tape.
4. The BRIDGE II schedule control deck, shown in Figure 5e, must be placed in the card reader.
5. Then normal operator action must be performed to:
   a. set the computer for manual operation
   b. reset the accumulator, alarms and depress the A→I switch
   c. reset P
   d. load card
   e. reset alarms
   f. reset P
   g. load card
   h. set the computer for automatic operation
   i. depress the START button

STOP	TYPEWRITER MESSAGE	EXPLANATION	OPERATOR ACTION
	PXTY BTL (8 words)	Typed when the START card is used to begin operation.	
	LOCATING (Run-Name)	Typed when entering the search routine. May be accompanied by a halt to verify the run name.	If a halt occurs, toggle switch 0 to search or toggle switch 19 to read an SCC card.
	(Run Name) COMPLETED	Typed when the Run Completion subroutine gives control to the Sequential Run Locator.	
	LOADING (Run-Name)	Typed when a program is loaded from the card reader.	
	END OF SCHEDULE	Typed when the EBS card is detected. Program enters a read console switch loop.	Place SCC card in the reader and toggle switch 0 to continue schedule.
	CHECKSUM ERROR SW0 TO TRY AGAIN. SW 19 TO ACCEPT.		Toggle switch 0 to try again. Toggle switch 19 to accept.
13335_8	(Switch 19)	If this switch is toggled any time during searching, the Run Locator is set to control mode and a halt occurs at location 13335_8.	Position the proper SCC card in the reader. Toggle switch 0 to read the new SCC card.
	PROGRAM NOT ON THIS TAPE. PLACE SCC CARD IN READER AND TOGGLE ZERO.	Self explanatory	
	NO BOOTSTRAP, PLACE SCC CARD IN READER AND TOGGLE ZERO.	Typed when a program has been specified by name and is not preceded by a bootstrap label. (no loader).	

Note: Card/Tape Read Error Messages are described in the write-up of Symbolic Input System/Card Reader CD225E1.007 and Symbolic Input/Output System CD225E2.001.

IV. SOURCE PROGRAM ERROR MESSAGES

A detailed error analysis is performed on the Source Program Language during compilation in the Transformer Phase. For each type of error found, a message is logged on the printer following the IDENTIFICATION DIVISION listing. The individual error messages from each run are headed by a run identification.

The run divisions and their corresponding run identification are shown in Table 2.

Table 2. Run Identification

PRINTER	GECOM MASTER TAPE
ENVIRONMENT DIVISION	T010EN
REPORT WRITER TRANSLATOR	T020RW
REPORT WRITER ANALYZER	T021RW
DATA DIVISION	T030DD
PROCEDURE DIVISION	T040PD
TABSOL	T050TA
PROCEDURE NAME ANALYZER	T060PN
INTERNAL LANGUAGE TRANSLATOR	T070LT
OUTPUT ANALYZER	T080OA
ARITHMETIC ANALYZER	T090AA
SIMPLE-IF ANALYZER	T100SI
MOVE-EXCHANGE ANALYZER	T110ME

When an error is found, the message is logged on the printer and the necessary recovery actions are taken to continue compilation. Corrective actions are taken only when it expedites the recovery. It must be remembered that an error in any run may cause subsequent errors to be noted in the following runs.

The lists starting on page IV-5 contain error code messages with an explanation of the probable causes and the corrective action (if any) taken by the compiler. In addition, the recommended programmer action is given for each of these errors. The lists give error messages for the Environment, Data, and Procedure Divisions, in that order.

Preceding the lists, the formats for the logged error messages are given in Figure 6.

ERROR MESSAGES IN THE TRANSFORMER PHASE

ENVIRONMENT DIVISION

Error Code, right justified with blank fill	ΔΔΔ	SENTENCE~NAME	Δ	+	Δ	Relative Card Position	

REPORT WRITER TRANSLATOR

Error Code	ΔΔ	Report Name	Δ	Δ	Major Heading	Δ	Δ	Minor Heading	Δ	+	Δ	Rel. Pos.	Δ	Invalid wd. or char.

REPORT WRITER ANALYZER

Error Code	Δ Δ	Report Name	

DATA DIVISION

Error Code, right justified with blank fill	Δ	Δ	Δ	+	Δ	Relative Position	Major Heading	Minor Heading

PROCEDURE DIVISION AND PROCEDURE NAME ANALYZER

Error Code, right justified with blank fill	ΔΔΔ	SENTENCE~NAME	Δ	+	Δ	Relative Position	

TABSOL RUN

Error Code	ΔΔ	Table Name	Δ	Row Number	ΔΔΔ	Column Number	ΔΔΔΔ	Memory Location where error was detected.

INTERNAL LANGUAGE TRANSLATOR

Error Code, right justified with blank fill	ΔΔΔ	SENTENCE~NAME	Δ	+	Δ	Relative Position	ΔΔΔ	Major Heading	Minor Heading

OUTPUT ANALYZER

Error Code, right justified, with blank fill	ΔΔΔ	OUTPUT	Δ ——————→	Δ	Major Heading	Minor Heading

ARITHMETIC, SIMPLE-IF AND MOVE-EXCHANGE ANALYZERS

Error Code, right justified with blank fill	ΔΔΔ	SENTENCE~NAME	Δ	+	Δ	Relative Position	

Figure 6. Format For Error Messages

When running a sequence check on the source program with console switch 18 down the format shown in Figure 7 is used.

ENVIRONMENT, DATA AND PROCEDURE DIVISIONS

```
| △ S E Q . △ LAST SEQUENCE  △  △  △   CURRENT   CURRENT |
|             NUMBER                   SEQUENCE  SENTENCE |
|                                      NUMBER             |
```

Figure 7. Format With Console Switch 18 Down

Glossary

The Glossary below contains definitions of various terms used in the error message lists.

1. Destination Field: The field to which the information is being moved.
2. Dummy Symbolic Name: A constant symbolic name that is used when a field was not described in the source program.
3. Generator Class: A compiler term used for a particular generator in the compiler.
4. Receiving Field: Same as the destination field.
5. Source Field: The field from which the information is being moved.
6. Tag: Compiler term for a descriptive parameter.
7. Undefined Symbol Flag: A General Assembly term for a flag to denote a symbolic name that was not defined in the symbol field of GAP coding.

References

All references given in the following error message listings are to the GECOM-II Reference Manual, unless otherwise indicated.

ENVIRONMENT DIVISION

Error Code, right justified with blank fill	ΔΔΔ	SENTENCE~NAME	Δ +	Δ	Relative Card Position

Sentence Name + Relative Card Position

If the error occurred in a card containing the Sentence Name, the card is identified by the Sentence Name with a relative card position of 000. If the error occurred in a card not having a Sentence Name, the card is identified by its position relative to the last named sentence encountered. For example, the first unnamed sentence after a sentence named "COMPUTE~1" is identified as "COMPUTE~1 + 001".

DE and FL Prefixes

The numeric error codes in the ENVIRONMENT DIVISION are prefixed by either a DE or FL. All error codes prefixed by a DE are listed first in numeric order followed by those prefixed by an FL. DE and FL, the two categories of errors, are explained below:

DE Delete: The compiler has found an error with respect to the GECOM language specifications. The statement or phrase was deleted. The resulting object program is not complete with respect to the problem definition. The error must be corrected and the problem recompiled.

FL Flag: The compiler has found an apparent error with respect to the GECOM language specifications. Action was taken which was the best possible action to be attempted at this time. This action, however, may have caused an erroneous object program with respect to the problem definition. The author must review all FL errors carefully. In most cases the condition causing the error must be corrected and the problem recompiled.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔDE006	No Identification Division was found.	Compilation continued with "PROGRAM~ID" and "NEXT~PROGRAM" as blanks.	Check the order of the source program (see Appendix B). Make the necessary correction to the source program and recompile.
ΔDE015	Duplicate I~O~CONTROL sentence names have been found.	The result is an incorrect object program.	Check the order of the source program (see Appendix B). Make the necessary corrections to the source program and recompile.
ΔDE017	An illegal Rerun statement nas been found.	The statement was deleted and compilation continued with the next statement. The object program will have no rerun coding.	Check convention 2 for the Rerun statement under I~O~CONTROL, in Page VII-4. Make the necessary corrections to the source program and recompile.
ΔDE020	A "MULTIPLE FILE" statement was found in error with respect to positions.	All positions must be given or no positions given. The positioning is probably incorrect.	Check the third convention under the I~O~CONTROL sentence, Page VII-5. Make the necessary corrections to the source program and recompile.
ΔDE021	An error in a file name or the position of a file was found in the "MULTIPLE FILE" statement.	The statement was deleted. The object program is wrong with respect to the positioning of the files.	Check the third convention under the I~O~CONTROL sentence, Page VII-5. Make the necessary corrections to the source program and recompile.
ΔDE022	More than one section name was used in a "USE" statement. A key or noise word may have been misspelled.	The last section name given was used.	Check the fourth convention under the I~O~CONTROL sentence, Page VII-5. Make the necessary corrections to the source program and recompile.
ΔDE028	Sufficient information was not given in the "USE" statement.	The statement was deleted.	Check the options and choices and the conventions for the USE statement under I~O~CONTROL, Page VII-4. Make the necessary corrections in the source program and recompile.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔDE030	More than one FILE~CONTROL sentence~name was found.	An incorrect object program will be the result.	Check the order of the source program. Make the necessary corrections in the source program and recompile.
ΔDE034	The word printed with this message is not a key word or valid noise word. It has been encountered in the FILE~CONTROL sentence.	Processing continued with the next "SELECT" or next sentence, whichever was first.	Check the options under the FILE~CONTROL sentence. Make the necessary corrections to the source program and recompile.
ΔDE045	More than four qualifiers have been used on a data name in the DSU~CONTROL sentence.	Compilation continues with possibly incorrect coding produced.	Reduce qualification to at most four levels and recompile.
ΔFL001	An illegal character or a character which is not normally expected in the Environment Division has been found. This may have been due to a misspelling.	The character was ignored by the compiler.	See "CHARACTERS", in GECOM Reference Manual then check the indicated sentence for presence of an illegal character.
ΔFL002	A character other than a hyphen has been found in column 7.	Compilation was continued by accepting the character.	Check the general conventions for using the compiler forms in Reference Manual.
ΔFL003	An illegal use of a hyphen was found.	The hyphen was ignored.	Check the general conventions for using the hyphen.
ΔFL004	A period was found at the beginning of new sentence.	It was ignored.	Check the general conventions for the use of a period.
ΔFL005	More than one Identification Division heading was found.	If a "PROGRAM~ID" or NEXT~PROGRAM was found under the second heading, the compiler used the latest information.	Check the order of the source program.
ΔFL008	No ENVIRONMENT DIVISION heading was found.	The omission of the heading was ignored.	Before the next compilation insert heading to delete the error note. Check the order of the source program.

ΔFL010	The memory size given was greater than 4 modules.	Two modules are assumed, (i.e., $8192)_{10}$ words.	Check conventions for memory size under OBJECT~COMPUTER, Page VII-2.
ΔFL012	An illegal object program assignment.	Tape zero, plug one, has been substituted.	Check the conventions for Object Program assignment under OBJECT~COMPUTER, Page VII-2.
ΔFL013	An illegal number of tapes has been specified for the object computer.	Six tapes were assumed.	Check the conventions under OBJECT~COMPUTER, Page VII-2.
ΔFL016	No I~O~CONTROL sentence name has been found.	The sentence name was assumed and compilation continued.	Check the order of the source program in Appendix B.
ΔFL018	A hardware assignment must be magnetic tape in a RERUN statement.	Tape zero, plug one, was substituted as the hardware name.	Check the second convention for the RERUN statement under I~O~CONTROL, Page VII-4.
ΔFL019	Illegal tape or plug number used in the indicated sentence.	Tape zero or plug one was substituted.	Check the general conventions for assigning tapes and plugs under OBJECT~COMPUTER, Page VII-3.
ΔFL023	Two file names have been found in one "USE" statement. A key or noise word may have been misspelled.	The last file name given was used.	If more than one file name is actually necessary, a separate USE statement should be written for each file name.
ΔFL024	The key word "INPUT" and a file name are present in a "USE" statement. A key or noise word may have been misspelled.	The last one given was assumed.	Check the options, choices, and conventions for the USE statement under I~O~CONTROL, Page VII-4.
ΔFL025	A file name and/or "INPUT" is present with "OUTPUT" in a "USE" statement.	"OUTPUT" was assumed in the statement.	Check the options, choices, and conventions for the USE statement under I~O~CONTROL, Page VII-4.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔFL026	"FILE" and "REEL" are present in a "USE" statement.	The last one given was assumed.	Check the options, choices, and conventions for the USE statement under I-O-CONTROL, Page VII-4.
ΔFL027	A section name was not given in a "USE statement.	The dummy symbolic name "8" was used as the section name. This symbol will be undefined in the listing.	Check the options, choices, and conventions for the USE statement under I-O-CONTROL, Page VII-4.
ΔFL029	The words "BEGINNING" and "ENDING" were both present in a "USE" statement.	Only one, the last word given, was used.	Check the options, choices, and conventions for the USE statement under I-O-CONTROL, Page VII-4.
ΔFL031	No hardware name or two or more hardware names are assigned for the same file name.	If two or more hardware names, the last assigned was used.	Check the general convention for assigning hardware names to files under FILE-CONTROL, Page VII-7.
ΔFL032	More than two tape controllers were used.		
ΔFL033	Two file names were found under one select. The first one taken as a file name may have been a misspelled vocabulary word.	The last file name mentioned took precedence.	Check the options under FILE-CONTROL, Page VII-7.
ΔFL035	Hardware floating point has been specified in the Computation Mode sentance but is not listed in the Object Computer sentence.	Compilation continues assuming floating point hardware.	
ΔFL036	Object Computer sentence was not given.	Compilation continues.	

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔFL037	No Journal Tape has been SELECTed, but a DSU file has been ASSIGNed to the Journal Tape.	The assignment to Journal Tape is ignored.	Check DSU~CONTROL sentence.
ΔFL039	There was more than one SELECT JOURNAL~TAPE, or more than one tape ASSIGNment made for a Journal Tape.	The last tape and plug number ASSIGNed to the JOURNAL TAPE is used.	Check DSU~CONTROL sentence.
ΔFL040	RESERVE ALTERNATE AREA clause is suspected to be out of order.	Two buffer areas are assigned to the last name file. Incorrect coding may be produced.	Check DSU~CONTROL sentence.
ΔFL041	No PLACE clause in OBJECT~COMPUTER sentence.	Compilation continues assuming "PLACE MAIN IN LOWER."	Check OBJECT~COMPUTER sentence and conventions.
ΔFL042	An OVERLAY~SEGMENTED object program was ASSIGNed to the CARD READER.	Compilation continues. Incorrect coding may be produced.	Check OBJECT~COMPUTER sentence and conventions.
ΔFL045	API CLOSE was specified but no API OPEN.	Compilation continues.	
ΔFL046	MIO entrance was specified but no API OPEN or CLOSE.	Compilation continues.	
ΔFL047	API is missing when DSU is used.	Compilation continues.	

REPORT WRITER TRANSLATOR

Error Code	ΔΔ	Report Name	Δ	Δ	Major Heading	Δ	Δ	Minor Heading	Δ	+	Δ	Rel. Pos.	Δ	Invalid wd. or char.

Major Heading

The first word of the last named report description header. During report layout processing this is the word LAYOUT. During the report definition processing this is the first word of one of the last named section headers (LINE CONTROL, LINE DEFINITIONS, LINE SECTIONS, PAGE CONTROL).

Minor Heading

During layout processing, the line code from the Line Image Entry. During definition processing, the second word of the last named section header.

Relative Position

The position of the card being processed relative to the header named under major and minor headings. The error is normally detected during processing of the card containing the error, but in some instances processing of the next card has begun before detection is possible. Therefore, the programmer should examine the card referenced and if no error is found then examine the immediately preceding card.

Invalid Word or Character

On certain errors (i.e. RE001, RE002, RE008) the illegal word or character is printed. If a character is in error its octal equivalent is printed. An invalid word is printed in decimal mode.

REPORT WRITER ANALYZER

Error Code	ΔΔ	Report Name

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔRE001	An illegal character has been found in either a data name or a data image. See Report Writer section, Figure R-13 for allowable Report Writer data image symbols and Chapters III and V for allowable characters in data names and file descriptions. Other examples of this error are: 1. A slash (/) other than in LINES/PAGE or a line definition entry. 2. A right parenthesis immediately preceded by a space. 3. A "G" symbol in other than the leading position of a data image. 4. Parentheses used in a data image. 5. A number instead of an alphabetic code as a Line Number code. 6. A numeric entry contains other than numeric characters.	The octal equivalent of the illegal character is printed and ignored.	
ΔRE002	An unidentifiable or extraneous word had been encountered (i.e. words after "LAYOUT," "OF" followed by "OF").	The word is printed and ignored.	
ΔRE003	An improper or incomplete entry has been detected, (i.e. "LINE SECTIONS" with no following Line Section Entries, "CONTROL BREAKS" with no data names, "LAYOUT" with no line images).		
ΔRE004	A Line Number Entry is missing a numeric value, or the alphabetic symbol is more than one character in length.	A value of 99 is assumed.	

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔRE005	More image names are listed in the Line Definition Entry than there are separated images in the matching Line Image Entry. The extra names may be data names or "L".	The data names are added to the print line with the data description given in Input or Working Storage. Any L's are ignored.	
ΔRE006	An incorrect numeric value has been detected in an entry. See the BLOCK clause on the RFD Entry and Page Control Entries.	A missing or 0 block size is given a value of 341. A block size of greater than 341 (unacceptable to the off-line printer or PIP*)is accepted but flagged by this error message.	
ΔRE007	An illegal card continuation has been detected. See Report Description Form Conventions on page R-30.	The card is ignored.	
ΔRE008	A vocabulary word has been used illegally. It is either used as a data name or it is out of correct context.	The illegal word is ignored and displayed in the printer message.	
ΔRE009	An unidentifiable card has been detected or a Line Image card exceeds the limit of 30 lines pèr report.	The card is ignored.	
ΔRE010	Compiler error.	Compilation has been stopped and cannot continue.	Try recompilation from the the beginning. If error occurs again, report to GE Computer Department sending copies of the source program deck, typewriter log; printer listing and HSM dump.

* Peripheral Package ,CD225E1.009

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔRE011	The identified section under Report Definition was used twice within the same report.	The invalid section header card is ignored. Following cards will be processed as though part of the last legal section.	
ΔRE012	Although a "REPORT SECTION" header was found, no reports were found or only unidentifiable cards were found.		
ΔRE013	The adjacency symbol (slash) was used with a literal identifier (L).	The slash is ignored.	
ΔRE014	A Report Layout Header after a Reports File Definition (RFD) Entry does not contain a report format code.	A value of 7 will be used.	See Report Writer, page R-8.
ΔRE015	Under "PAGEΔCONTROL", LINES/PAGE or LAST~DETAIL are used twice or together.	The second entry is ignored.	
ΔRE016	Report name or File Name (in off-line case) is missing.	"RDX" is inserted for missing Report name and "RFDX" is inserted for missing file name.	
ΔRE017	An "RD" header is not explicitly identified as "LAYOUT" or "DEFINITION".	The header is assumed to be a Report Definition Header.	
ΔRE018	A Line Definition Entry has been detected although the Line Image Entries have been exhausted.	The Line Definition Entry is ignored.	

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔRE019	A data image has been detected in the Line Image Entry after the image names in the Line Definition Entry have been exhausted.	The data image is treated as a literal, as though an "L" was given in the Line Definition Entry.	
ΔRE020	An "OF" clause has been used illegally.	The "OF" clause is ignored.	
ΔRE021	A "FOR" clause has been used illegally.	The "FOR" clause is ignored	
ΔRE022	The line after a "LINEΔ CONTROL" header does not have a blank "Line Code" column. This indicates that the Control Breaks Entry is missing or misplaced.	The line is ignored.	
ΔRE023	Within the Line Definition Entry an ACC or ACCUMULATION field has no immediate "OF" clause.	If it is immediately followed by a "FOR" clause, the "FOR" clause will be accepted as an "OF" clause. Otherwise the data name "ACC" is assigned.	
ΔRE024	Within the Line Definition Entry a COUNT field has no immediate "FOR" clause.	If it is immediately followed by an "OF" clause, the "OF" clause will be accepted as a "FOR" clause. Otherwise the data-name "COUNT" is assigned.	
ΔRE101	The Control Breaks Entry contains duplicate control break data names.	The second name is ignored.	

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔRE102	A Line Control Entry has a line code starting with other than H,T or SH, ST.	The entry is ignored.	
ΔRE103	The same level of control break is associated with more than one line or series of the H or T class.	The second Line Control Entry is ignored.	
ΔRE104	The line code given in the Line Control Entry does not match any of the lines defined for the report.	The Line Control Entry is ignored.	
ΔRE105	A Line Control Entry has a line code for a series member. Line control must be at the series level in this case.	The entry is ignored.	
ΔRE106	A Line Control Entry has a control break data-name which was not given in the Control Breaks Entry.	The entry is ignored.	
ΔRE107	The Line Control Entry specifies FINAL, but FINAL was not established in the Control Breaks Entry.	The Line Control Entry is ignored.	
ΔRE108	An H or SH line has a Line Control Entry with a control break data-name of FINAL.	The entry is ignored.	

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔRE109	A LINE CONTROL section does not contain a Control Breaks Entry, or the Control Breaks Entry is not the first entry in the section.	The Line Control Entries are bypassed.	
ΔRE110	A line number given in a Line Number Entry is greater than the number given in a LINES/PAGE statement.	The Line Number Entry is accepted with the number given.	
ΔRE111	A Line Number Entry lists an alphabetic code that was not used in a Line Image Entry.	That entry in the list is ignored.	
ΔRE112	An alphabetic code in the "pre" column of the Line Image Entry was not defined in the Line Number Entry.	A zero is assumed for pre slew value.	
ΔRE113	A Line Section Entry has an undefined line code.	The entry is ignored.	
ΔRE114	A Line Section Entry has a series line code.	The entry is ignored.	
ΔRE115	A Line Section Entry is made for a line which already has been given a Line Section Entry.	The second entry is ignored.	

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔRE116	The Line Image Entry contains an invalid line code.	The data description of the line is compiled (with line code as record name) but the line is ignored in the Report Writer procedure.	
ΔRE117	The first defined PH or PF line contains a "PRE" character that is not numeric or alphabetic (other than E).	The character is ignored. Although E as a pre-slew for the first PH is redundant rather than illegal, the message prints and can be ignored by the programmer.	
ΔRE118	PH or PF lines after the initial one contain a "PRE" character which is non-numeric.	The character is ignored.	
ΔRE119	A PH or PF line has a non-numeric character for "POST".	The character is ignored.	
ΔRE120	The first RH or RF contains a "PRE" character other than numeric or alphabetic.	The character is ignored.	
ΔRE121	An RH or RF line after the initial one contains a "PRE" character other than numeric or E.	The character is ignored.	
ΔRE122	An RH or RF line contains a "POST" character of other than numeric or E.	The character is ignored.	

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔRE123	A D,H or T line contains an invalid "PRE" character. A numeric or alphabetic value is allowed.	The character is ignored.	
ΔRE124	A D, H or T line contains an invalud "POST" character. Only a numeric or alphabetic (not E) is allowed.	The character is ignored.	
ΔRE125	A character was placed under "PRE" for an S line.	The character is ignored.	
ΔRE126	A character was placed under "POST" for a S line.	The character is ignored.	
ΔRE127	The 1st member of a series (H,D or T) has an illegal character under "PRE". This must be a character which is non-numeric or non-alphabetic.	The character is ignored.	
ΔRE128	A member of a series (H,D or T), other than the first one, has a non-numeric character under "PRE".	The character is ignored.	
ΔRE129	A member of a series (H,D or T), other than the last one, has a non-numeric character under "POST".	The character is ignored.	

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔRE130	A last member of a series has a character under "POST" other than numeric or E.	The character is ignored.	
ΔRE131	A series header is followed by no members.	The series header is ignored.	
ΔRE132	A series is defined with only one member.	The entries are accepted as a series.	
ΔRE133	H or T lines were defined but given no Line Control Entry.	The data description of the line is compiled (with line code as record name) but the line is ignored in the Report Writer procedure.	
ΔRE134	In the Control Breaks Entry, the word FINAL is included but is not the last name in the list.	It is accepted as if it were written last.	
ΔRE135	A Line Image Entry was given the same line code as a previous entry.	The second entry is ignored.	
ΔRE199	ACC or COUNT names have exceeded the limit of 50 unique names per report; on previous source error messages caused a compiler stop of unknown origin.	Program halt.	Correct any other errors. Shorten field names in ACCs or COUNTs. If error persists, notify the Computer Department.

DATA DIVISION

Error Code, right justified with blank fill	Δ	Δ	Δ	+	Δ	Relative Position	Major Heading	Minor Heading

Relative Position

Is the position of the card containing information found in error relative to the major or minor entry. If the message contains both a major and minor entry, the position is relative to the minor entry.

Major Heading

The last named Section or File. For example: "WS" or "FILE~NAME~1" respectively.

Minor Heading

The last named Record. For example: REC~1.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔΔΔ001	An illegal character has been found.	The character has been deleted.	See Language Structure and Data Division chapters for allowable characters in data names, file descriptions and data images.
ΔΔΔ002	A Report Section control or accumulation field cannot be found in the sections of the Data Division or an accumulation field is not described as numeric.	The field has been ignored.	Correct the source program and recompile.
ΔΔΔ003	This Input File has multiple record descriptions. Each record must have a control key. Some or all record descriptions do not have a control key.	The records without a control key are flagged; unpacking is generated but there is no path to reach it unless you "Enter General Assembly Program".	See Data Division, FILE SECTION, FILE DESCRIPTION. Make the necessary changes to the record descriptions that do not have a control key.
ΔΔΔ005	The record description indicates that fields are separated by commas. The file must be assigned to the card reader, but Environment Division has this file assigned to another I/O medium.	The I/Ø medium has been accepted and compilation continued.	Correct the source program to assign this file to card reader, or delete "fields separated by commas" entry.
ΔΔΔ006	1. An element does not have the most significant or least significant position assigned. 2. An element is larger than the source field described.	1. The size has been set to 1 character by the compiler. 2. The element has been given the same size as the field by the compiler.	Correct the source program and recompile.

ΔΔΔ007	1. A conditional or field literal is described as greater than 120 characters.	1. The compiler truncates at 120 characters.	1. If data is over 83 characters long and used in a move (implied or actual) care must be taken to ensure that leading characters in source and destination are in the same character position of the computer word. Otherwise, the program may run incorrectly.
	2. An output literal does not have a terminating quote.	2. Compiler assumes terminating quote at end of card.	

ΔΔΔ008	Error in construction of Array Section, True-False Section, or Integer Section may be one of the following:		
	1. Repeated field has been found within a repeated group.	1. The field was treated as a single entry by the Compiler.	1. See Data Division "ARRAY SECTION"
	2. No entry in the Data Division (Input Files, Working Storage, Constant Section) has been found for an entry described in the Array Section.	2. The entry has been ignored by the Compiler and the mode of computation is assumed to be fixed point.	
	3. An entry described in the Array Section does not have the dimension specified.	3. The Compiler treats the array as a list using the number of repeats as the dimension.	
	4. The dimensions of an array and the number of repeats do not agree.	4. The Compiler treats the array as a list using the number of repeats as the dimension.	
	5. No entry in the Data Division (Input Files, Working Storage Section, Constant Section) has been found for a qualified entry in the Array Section, True-False Section, or Integer Section.	5. The entry was ignored by the Compiler.	
	6. A True-False entry was found that contains an improper data image.	6. The entry was ignored by the Compiler.	6. See Data Division "TRUE-FALSE" SECTION".
	7. An integer entry was found that contains an improper data image.	7. The entry was ignored by the Compiler.	7. See Data Division "INTEGER SECTION".
	8. A field in the *Common-Storage Section is not a repeated entry.	8. The entry was ignored by the Compiler.	8. Correct the source program and recompile.
ΔΔΔ009	An entry in the Type column cannot be identified.	The entry was ignored by the Compiler. This will cause the computer to stop at the end of DATA DIVISION run.	See Data Division chapter for allowable entries on the Type column.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔΔΔ010	1. A tape file FD sentence does not have the phrase "LABEL RECORDS OMITTED", but the label record was not found as the first record of the file description.	1. The Compiler assumes that label records are omitted.	1. See FILE SECTION, FILE DESCRIPTION.
	2. No terminating quote (") character on a LABEL~ IDENT was found.	2. The first nine characters following the first quote (") are used as the IDENT by the Compiler.	2. See Data Division, TAPE LABELS.
ΔΔΔ011	1. An illegal entry has been found in the "FORMAT", "BINARY", or "JUSTIFY" column.		1. Check the indicated entry to find the type, then refer to the General Compiler Forms and Data Division chapters for legitimate entries.
	2. The "BINARY" column contains an entry when the recording mode from the FD entry indicated BCD. This is conflicting information.	2. The Compiler assumes that the data is binary.	
	3. An entry in a higher Type has a conflicting entry.	3. The entry from the higher Type was used by the Compiler.	3. Check the indicated line of the source program Data Division against higher level entries to determine conflict.
ΔΔΔ012	1. This source name was not unique.	1. The source name was accepted.	1. See Language Structure, QUALIFIERS.
	2. Termination source name cannot be found.	2. The entry was ignored by the Compiler.	2. Correct the source program and recompile.
ΔΔΔ013	This file name was not assigned in the Environment Division.	This will force the compiler to stop at the end of the Data Division run.	Correct the source program and recompile.

ΔΔΔ014	1. Input block size specified was too small.	1. The Compiler accepts the specified block size.	
	2. The entry between parentheses "()" was equal to zero.	2. It has been set equal to one by the Compiler.	2. See Data Division, DATA IMAGE.
	3. No Data Image was given.	3. The Compiler assumes a numeric 2-word entry.	3. See Data Division, DATA IMAGE.
	4. There is a 1 or 2 in the format column and no scale was given.	4. A scale of 0 is assumed.	4. See Data Division, DATA IMAGE.
	5. A numeric entry is described as greater than 11 characters.	5. The compiler accepts the entry as described, but object program will produce inconsistent results.	5. See Data Division, DATA IMAGE.
ΔΔΔ017	The source program Data Division has exceeded the internal table.	This message is accompanied by a typewriter message "WTV", and the computer is halted in a loop.	See Data Division, OVERFLOW CONDITION.

PROCEDURE DIVISION AND PROCEDURE DIVISION ANALYZER

Error Code, right justified with blank fill	ΔΔΔ	SENTENCE~NAME	Δ	+	Δ	Relative Position

Sentence Name + Relative Position

If the error occurred in a named sentence, the sentence is identified by its Sentence Name and a relative position of 000. If the error occurred in an unnamed sentence, the unnamed sentence is identified by its position relative to the last named sentence. For example: the first unnamed sentence after a sentence named COMPUTE~1 is identified by "COMPUTE~1 + 001".

Checking Errors in Procedure Division

A source program error in the Procedure Division usually causes either incorrect coding to be produced or no coding to be produced in the corresponding part of the object program. Furthermore, a source program error can throw the translation process out of phase, thereby causing additional error messages and incorrect coding to be produced. This may cause additional error messages to be printed in subsequent compiler runs. The additional error messages may be unfounded (no basis in the source program). The compiler always attempts to continue or to get back in phase to continue processing so that it can at least detect as many source program errors as possible even though the object program may not run correctly until all errors are corrected. In summary, it should be noted that one error may cause more than one error message. When an error list is produced, the user should examine the sentences in question keeping in mind the possible causes for the error message (as shown on the following pages). It would be wise to check the sentences before and after the questionable sentence as well. When all apparent errors are corrected, recompilation should be attempted. On recompilation the unfounded error messages should not recur.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔEA001	An undefined character code was found, possibly because of a mispunched card.	The entry in question is presumed to be complete and not contained in the GECOM vocabulary. Processing continues, the entry is treated as a non-vocabulary entry.	See CHARACTERS, Page III-1.
ΔEA002	A defined character was illegally used.	The entry in question is presumed to be complete and not contained in the GECOM vocabulary. Processing continues, the entry is treated as a non-vocabulary entry.	See DATA NAMES, Page III-2.
ΔEA004	A word reserved for the compiler vocabulary was used as a data name, sentence name, etc., or in some other out-of-context manner.	The entry in question is presumed to be complete and not contained in the GECOM vocabulary. Processing continues, the entry is treated as a non-vocabulary entry.	See Appendix A.
ΔEA005	A qualifier was associated with an un-qualifiable entry.	The entry in question is presumed to be complete and not contained in the GECOM vocabulary. Processing continues, the entry is treated as a non-vocabulary entry.	See QUALIFIERS, Page III-4.
ΔEA006	More than one verb was found in a single sentence. It is possible that a period was missing and two sentences ran together or perhaps a GECOM vocabulary word was used as a data name, section name, etc.	The entry in question is presumed to be complete and not contained in the GECOM vocabulary. Processing continues, the entry is treated as a non-vocabulary entry.	See Appendix A; see VERBS, Chapter VI.
ΔEA007	One of the words POSITIVE or NEGATIVE was used in a sentence other than a conjunction with the verb IF or perhaps one was used illegally within an IF statement.	The entry in question is presumed to be complete and not contained in the GECOM vocabulary. Processing continues, the entry is treated as a non-vocabulary entry.	See Page V-21, Option 4.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔEB001	A character other than a blank or hyphen was found in card column 7.	Processing continues and all information is accepted as found.	See CONVENTIONS, Page IV-1.
ΔEB003	The use of right and left parentheses was such that the total number of left parentheses did not equal the total number of right parentheses at some point where they should be equal. This unbalancing of parentheses can occur in arithmetic expressions, ASSIGNMENT, and logical IF.	Processing continues and all information is accepted as found.	See EXPRESSIONS, Page III-8; see ASSIGNMENT, Page VI-9; see LOGICAL EXPRESSIONS, Page III-11.
ΔEB004	An arithmetic expression contained more than 50 operations.	Processing continues and all information is accepted as found.	See EXPRESSIONS, Page III-8.
ΔEB005	A procedure name was used to name more than one sentence or section.	Processing continues and all information is accepted as found.	
ΔEB006	The receiving field of an ADD or MOVE sentence was a pure numeric name and thus appeared to be a constant. Perhaps an alphabetic character was omitted.	Processing continues and all information is accepted as found.	See DATA NAMES, Page III-2.
ΔEB007	The procedure names in an ALTER sentence were not paired, viz. there was an odd number of names.	Processing continues and all information is accepted as found.	See ALTER, Page VI-8.
ΔEB008	An un-named GO sentence had no object procedure name. The sentence must be named if no point of departure follows the verb GO.	Processing continues and all information is accepted as found.	See GO, EXAMPLE 1, Page VI-17.

ΔEB009	The verb GO of a GO - DEPENDING sentence had no object procedure name.	Processing continues and all information is accepted as found.	See GO, Option 2, Page VI-17.
ΔEB010	A pure numeric name was used where a numeric constant is illegal.	Processing continues and all information is accepted as found.	
ΔEB011	No entry was found after DEPENDING.	Processing continues and all information is accepted as found.	See GO, Option 2, Page VI-17.
ΔEB012	The word TYPEWRITER was not used when more than one entry appeared in a WRITE sentence.	Processing continues and all information is accepted as found.	See WRITE, Option 1, Page VI-38.
ΔEB013	One of the words POSITIVE or NEGATIVE was preceded by a relation.	Processing continues and all information is accepted as found.	See IF, Option 4, Page VI-19.
ΔEB014	There were too many names in the operand portion of one of the entries under an ENTER verb.	The names were accepted as qualifiers to the field. Processing continues and all information is accepted as found.	See ENTER, Page VI-12.
ΔEB015	There was more than one entry between VARY and FROM in a VARY sentence.	Processing continues and all information is accepted as found.	See VARY, Page VI-35. NOTE: Compilation beyond this point is unsafe inasmuch as part of the compiler coding may have been destroyed.
ΔEB016	There were too many entries after the UNTIL of a VARY sentence.	Processing continues and all information is accepted as found.	See VARY, Page VI-35.
ΔEB017	During the breakdown of an IF sentence, it was not clear where the object program control should go if the condition being tested were true. Perhaps a procedure name is missing or the construction of a logical IF is not logical.	Processing continues and all information is accepted as found.	See IF, Page VI-19.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔEB018	A literal constant was somehow being tested with an arithmetic expression. This may have been through an implied relation in a multiple IF sentence.	Processing continued and all information is accepted as found.	See CONSTANTS, Page III-2.
ΔEB019	There were unreferenced or undefined procedure names preceding the printing of this error code.	A check is made at the conclusion of each section and at the end of the program. The compiler takes no action other than to indicate this condition. Processing continues and all information is accepted as found. Care must be taken that a procedure name within a section is not referenced outside the section or that a referenced name lies outside the section.	See SECTIONS, Page VI-1.
ΔEB020	A PERFORM sentence referred to a sentence name.	Processing continues and all information is accepted as found.	See PERFORM, Page VI-29.
ΔEB021	A decision table header does not specify the number of conditions, actions, or rows.	Results of this error are unpredictable. Processing will probably continue into the TABSOL run where it will stop with an error message indicating an excessive number of errors, table limits exceeded, or compiler error.	
ΔEB022	"BEGIN" statement missing from a section.	Processing continues, but incorrect coding will be produced.	
ΔEB23	Both READING and WRITING specified in a READY sentence.	Processing continues. The last named option is used.	See READY verb.

ΔEB024	An EXIT sentence name differs from the name of its corresponding VARY sentence.	Compiler generates a branch instruction with undefined address. Processing continues.	
ΔEB025	A minus sign was used within a MOVE sentence	Processing continues and information is accepted as found.	Remove minus sign from MOVE sentence and recompile.
ΔEB026	The order of EXITs in nested VARYs is not innermost to outermost.	Processing continues and information is accepted as found.	
ΔEB027	A LOAD verb was used and the object program was not ASSIGNed to tape or DSUS.	Processing continues, but incorrect coding may be produced.	See OBJECT-COMPUTER sentence and LOAD verb.
ΔEB028	A LOAD verb was used in a segment not designated as a MAIN segment.	Processing continues, but incorrect coding may be produced.	See OBJECT-COMPUTER sentence and LOAD verb.
ΔEC001	An entry was found to exceed the 12 character limit.	The entry up to the point the error was detected is deleted and processing continues.	See DATA NAMES, Page III-2.
ΔEC002	Two names were used without an arithmetic operator, relation, GECOM key word, etc., between them to suggest the appropriate or intended connection.	Many sentences are legal that contain the condition described at the left and no error indication is given in those cases. The entry up to the point the error was detected is deleted and processing continues.	
ΔEC003	An exponent had more than two characters or else it contained a character other than one of the set 0-9.	The entry up to the point the error was detected is deleted and processing continues.	See NUMERIC CONSTANTS, Page III-2.

ΔEC004	The number of conditions, actions, or rows in a decision table header is nonnumeric.	Processing is as explained for message EB021.	
ΔEC005	There is an illegal entry in a STOP sentence.	The entry is deleted and processing continues.	See STOP verb.
ΔEC006	More than one STOP RUN was found.	The RUN option is deleted and processing continues.	See STOP verb.
ΔED001	One of the characters, ", @, |, $, *, /, %,), was misused.	Processing on the sentence stops and is resumed on the next sentence.	
ΔED002	A subscript was used on an entry that cannot be subscripted.	Processing on the sentence stops and is resumed on the next sentence.	See SUBSCRIPTS, Page III-8.
ΔED003	A subscript was used that contained no entries.	Processing on the sentence stops and is resumed on the next sentence.	See SUBSCRIPTS, Page III-8.
ΔED004	A subscript was used that had more than 3 dimensions.	Processing on the sentence stops and is resumed on the next sentence.	See SUBSCRIPTS, Page III-8.
ΔED005	One of the characters +, -, (, ., =, ~, was misused.	Processing on the sentence stops and is resumed on the next sentence.	
ΔED006	Too many fields were named in an ADD sentence or after a GO TO DEPENDING or COPYING.	Processing on the sentence stops and is resumed on the next sentence.	See ADD, Page VI-6; GO, Option 2, Page VI-29; READ, Option 3, Page VI-30.
ΔED007	More than one receiving field was named in an ADD-GIVING sentence.	Processing on the sentence stops and is resumed on the next sentence.	See ADD, Page VI-6.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔED008	One of the relations (GR, LESS, EQ, etc.) was used illegally.	Processing on the sentence stops and is resumed on the next sentence.	See RELATIONAL EXPRESSIONS, Page III-8; IF, Option 2, Page VI-19.
ΔED009	The PERFORM - USING named more fields than the INPUT part of the section or the PERFORM-GIVING named more fields than the OUTPUT part of the section.	Processing on the sentence stops and is resumed on the next sentence.	See PERFORM, Page VI-29.
ΔED010	Entries were made that have unknown meanings. For example, a field name after the word ALL has no meaning to the problem or the compiler.	Processing on the sentence stops and is resumed on the next sentence.	
ΔED011	A GENERATE or TERMINATE sentence references an undefined report-name or a GENERATE sentence references an undefined detail-name.	The sentence is ignored and processing continues.	
ΔED012	A NOTE verb has been detected between a decision table header and the beginning of a table.	Processing on the sentence stops and is resumed on the next sentence.	
ΔEE001	An error of unknown origin has occurred. The error may be a machine failure, a compiler error, or an unforeseen source program condition.	Recovery action is impossible so operation halts.	An attempt should be made to recompile the program and if the same error occurs, proper documentation should be sent to the GE Computer Department.
ΔEE002	The problem being compiled has more sentence names than the compiler is able to handle.		The source program should be partitioned or the number of sentence names reduced.
ΔEF001	The parentheses used in an arithmetic expression in a VARY sentence are unpaired (unbalanced).	The entire statement is deleted and processing continues with the next sentence.	

ΔEF002	A character other than a blank or one of the set 0-9 was used following a period/ decimal point.	The entire statement is deleted and processing continues with the next sentence.	
ΔEF003	A numeric constant containing a decimal fraction was used in an ADVANCE sentence.	The entire statement is deleted and processing continues with the next sentence.	See ADVANCE, Page VI-7.
ΔEF004	More than one name was used between ADVANCE and TOP.	The entire statement is deleted and processing continues with the next sentence.	See ADVANCE, Page VI-7.
ΔEF005	A VARY sentence has no procedure name.	The entire statement is deleted and processing continues with the next sentence.	See VARY, Page VI-35.
ΔEF006	Two procedure names appear to be adjacent. This may be caused by the naming of a NOTE sentence or by errors indicated by preceding messages.	Processing continues with the next sentence.	
ΔEG001	The sentence was terminated before a verb was found or else during a time when more information was required. Perhaps a period/decimal point was misused.	An attempt to terminate the sentence is made, but should it fail, processing is discontinued and resumed on the next sentence.	

ΔEG002	A GENERATE or TERMINATE sentence occurs where no Report Section is contained in the Data Division.	An attempt to terminate the sentence is made, but should it fail, processing is discontinued and resumed on the next sentence.	
ΔEH001	A character code which is undefined in GECOM was used.	The character is deleted and processing continues.	
ΔEH002	The character code octal 77 is an internal compiler code and must not be used in the source language.	The character is deleted and processing continues.	

TABSOL RUN

Error Code	ΔΔ	Table Name	Δ	Row Number	ΔΔΔ	Column Number	ΔΔΔΔΔ	Memory Location Where Error Detected

See "Checking Errors in Procedure Division", page IV-14 of this manual.

When the TABSOL run detects apparent errors in the source program, it usually attempts to skip over the questionable input and continue compilation. Ordinarily, this leads to omission of object coding or incorrect object coding. This may have no significant effect on the object program, e.g., if a noise word is skipped. The compiler makes the most likely assumption about the questionable input. This could result in a correct object program from a faulty source program. If compilation results in error messages, the programmer should check the source program against these messages taking into consideration the probable causes, and actions (or lack of actions) taken by the compiler. Programmer should determine if an attempt to run the object program is worth the chance. In the final analysis, all errors should be corrected and the source program recompiled, to produce clean documentation.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ERR001	An illegal character or word has been detected.	Compilation continues ignoring the character or word.	
ERR002	The first character after column 7 on a card is not a vertical line.	The character found will be used. Object coding may be correct.	
ERR003	A card is not terminated by a vertical line.	The compiler assumes that the vertical line was found. Object coding may be correct.	
ERR004	Illegal use of words or characters in a primary row.	The column is ignored.	
ERR005	A blank column has been found in a secondary row.	The column is ignored. Object program may be correct.	
ERR006	An illegal combination has been detected in a primary and secondary row.	The column is ignored.	
ERR007	Multiple verbs or illegal use of verbs.	The column is ignored.	
ERR008	Multiple operands not separated by a logical or relational operator.		
ERR009	An illegal exponent or an alphabetic character has been found after a decimal point.	The column is ignored.	
ERR010	An illegal qualifier or subscript has been found.	The column is ignored.	

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ERR011	Unpaired parentheses detected.	The column is ignored.	
ERR012	Table limits have been exceeded.	Compilation has stopped and cannot be continued.	Examine other error messages. It is possible that this may have been caused by other source program errors.
ERR013	An excessive number of errors has been detected.	Compilation has stopped and cannot be continued.	Correct errors and recompile.
ERR014	A word has been found which is greater than 30 characters long.	The first 30 characters are discarded, and the remaining characters are used.	
ERR015	Illegal use of relational operators, e.g., two consecutive operators.	Column is ignored.	
ERR016	An OPEN verb is not followed by the word INPUT or OUTPUT.	The column is ignored.	
ERR017	An incorrect conditional statement has been found.	The condition is ignored.	
ERR018	An incorrect action statement has been found.	The action is ignored.	
ERR019	Illegal use of NOT detected.	The column is ignored.	

ERR020	Compiler error.	Compilation has been stopped and cannot continue.	Try recompilation from the beginning. If error occurs again, report to the GE Computer Department sending copies of the source program deck, typewriter log, and printer listing.
ERR021	Illegal relational expression detected.	The column is ignored.	
ERR023	Table dimensions do not agree with number of entries found in table.	All entries exceeding table dimensions are deleted; error is noted, and compilation continues.	Correct errors and recompile.

INTERNAL LANGUAGE TRANSLATOR

Error Code, right justified with blank fill	ΔΔΔ	SENTENCE~NAME	Δ	+	Δ	Relative Position	ΔΔΔ	Major Heading	Minor Heading

Sentence Name + Relative Position

If the error occurred in a named sentence, the sentence is identified by its Sentence Name and a relative position of 000. If the error occurred in an unnamed sentence, the unnamed sentence is identified by its position relative to the last named sentence. For example: the first unnamed sentence after a sentence named COMPUTE~1 is identified by "COMPUTE~1 + 001".

Major Heading

The data name that was in error, if any.

Minor Heading

The qualifier of the data name, if any.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔILΔΔΔ	This name has been used illegally in a procedure sentence.		Check the sentence against the format for the particular verb as shown in Chapter VI of GECOM II Reference Manual. The usual error is an output file name used in place of the record name, or a field name used instead of a file name. A file name or record name may have been used instead of a field name.
ΔIL001	In multiple record type files, fields with the same name, position and description in each record are assigned the same symbolic name if the field name is not qualified. This message occurs if such a field has been referenced previously with a qualifier.	Processing continues. Incorrect object coding may be produced.	If necessary, change source program and recompile.
ΔIL002	A move to more than one record or group has been attempted, where the number of fields in any of the records or groups involved is greater than 25.	Entry was deleted.	Make the necessary source program corrections and recompile.
ΔIL003	Incorrect READ UNTIL or CHAIN UNTIL sentence. (Neither field is in the file to be read or chained; both fields are in the file to be read or chained; more than 4 READ UNTIL or CHAIN UNTIL sentences for same file, etc.	Processing continues. Incorrect object coding may be produced.	Correct source program and recompile if necessary.
ΔIL004	An attempt has been made to READ UNTIL on a file defined as output.	Entry was deleted.	Make necessary source program corrections and recompile.
ΔIM000	An illegal move was attempted.	Entry was deleted.	Make the necessary corrections in the source program and recompile.

ΔIX001	A name that was not an array was indexed.	The dummy symbolic name "[*]" was used by the compiler instead of the original name. This causes an "undefined symbol" flag to appear in the object program listing. Only a repeated field entry may be indexed. The entry was deleted.	Make the necessary corrections in the source program and recompile.
ΔME001	Mixed modes on moves have been attempted, i.e., a record to a field, a field to a record, or a group to a field. Data movement is only provided for records and fields of the same format.	The incorrect entry was deleted.	Make the necessary corrections in the source program and recompile.
ΔME002	An attempt was made to move data into a constant field.	The entry was deleted.	Make the necessary correction in the source program and recompile.
ΔPR001	An attempt was made to advance a file which is not assigned to a printer.	The dummy symbolic name "[*]" was used by the compiler instead of the original name. This causes an "undefined symbol" flag to appear in the object program listing.	Make the necessary corrections in the source program and recompile.
ΔUDΔΔΔ	An undefined source name has been encountered.		
ΔUNSAN	An array name has not been subscripted.	The first entry in the array will be used.	

OUTPUT ANALYZER

Error Code, right justified with blank fill	ΔΔΔ	OUTPUT	Δ	Δ	Major Heading	Minor Heading

Major Heading

The data name in OUTPUT that was in error, if any.

Minor Heading

The qualifier of the data name, if any.

SPECIAL FORMAT FOR ERROR CODE IDIM:

ΔIDIMΔ	Δ	Δ	FILEΔXX	Δ	Δ	RECORDΔYYY	ΔΔΔ	FIELDΔZZZ

"XX" represents the file number of the entry in question.

"YYY" represents the record number within that file number of the entry in question.

"ZZZ" represents the field number within that record of the entry in question.

ERROR MESSAGE	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔIDIMΔ	An illegal data image was encountered in the Output Section which is incompatible with the source. Message shows file number, record number within that file, and field number within that record for the entry in question.	The entry is deleted.	Make the necessary corrections to the source program and recompile.
ΔIDNAM	An illegal data name was encountered in the Output Section.	The entry is deleted.	Make the necessary corrections to the source program and recompile.
ΔPLOVF	A record in a file assigned to the printer exceeds 120 characters.		Make the necessary corrections to the source program and recompile.
ΔUDΔΔΔ	An undefined data name was encountered in the Output Section.	The name was not defined within the Data Division.	Make the necessary correction to the source program and recompile.

ARITHMETIC ANALYZER, SIMPLE-IF ANALYZER, AND MOVE-EXCHANGE ANALYZER

Error Code, right justified with blank fill	ΔΔΔ	SENTENCE~NAME	Δ	+	Δ	Relative Position

Sentence Name + Relative Position

If the error occurred in a named sentence, the sentence is identified by its Sentence Name and a relative position of 000. If the error occurred in an unnamed sentence, the unnamed sentence is identified by its position relative to the last named sentence. For example: the first unnamed sentence after a sentence named "COMPUTE~1" is identified as "COMPUTE~1 + 001".

DE and FL Prefixes

The numeric error codes are prefixed by the letters DE or FL. All error codes prefixed by DE are listed first in numeric order followed by those prefixed by an FL. DE and FL are two categories of errors which are explained below.

DE	Delete:	The compiler has found an error with respect to the GECOM language specifications. The statement or phrase was deleted. The resulting object program will not be complete with respect to the problem definition. The error must be corrected and the problem recompiled.
FL	Flag:	The compiler has found an apparent error with respect to the GECOM language specifications. Action was taken which was the best possible action which could be attempted at the time. This action, however, may have caused an erroneous object program with respect to the problem definition. The author must review all FL errors carefully. In most cases the condition causing the error will have to be corrected and the problem recompiled.

ERROR MESSAGE	VERB	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔDE122	"IF"	Mixed numeric and alphanumeric fields have been named. The numeric field is larger than the alphanumeric field.	The statement has been deleted.	Make the necessary corrections to the source program and recompile.
ΔDE167	ALL VERBS	Only figurative constants or literals were named in a source statement.	The statement was deleted.	See appropriate verb sentence format. Make the necessary corrections to the source program and recompile.
ΔDE170	"MOVE"	A numeric constant was named as a destination field.	The statement has been deleted.	See "MOVE" verb sentence format. Make the necessary corrections to the source program and recompile.
ΔDE171	"MOVE"	An alphanumeric literal was named as a destination field.	The statement has been deleted.	See "MOVE" verb sentence format. Make necessary corrections to the source program and recompile.
ΔDE172	"MOVE"	No source field(s), or no proper source field(s) were named.	The statement has been deleted.	See "MOVE" verb sentence format. Make the necessary corrections to the source program and recompile.
ΔDE173	"MOVE"	No destination field(s), or no proper destination field(s) named.	The statement has been deleted.	See "MOVE" verb sentence format. Make the necessary corrections to the source program and recompile.
ΔDE175	"MOVE"	The destination field named was in a file label.	The statement has been deleted.	Check field name used. Make the necessary corrections to the source program and recompile.
ΔDE177	"MOVE"	The source was an array and the destination was a field (or element).	The statement has been deleted.	Check data description of destination. Make the necessary corrections to the source program and recompile.
ΔDE181	"MOVE"	The source was an input field; the destination was an array. This may indicate omission of a subscript on the array name.	The statement has been deleted.	Check source field description. Make the necessary corrections to the source program and recompile.

ΔDE182	"EXCH-ANGE"	Records, groups, or elements have been named.	The statement has been deleted.	See "EXCHANGE" verb sentence format, Page VI-16. Make the necessary corrections to the source program and recompile.
ΔDE184	"EXCH-ANGE"	Only one field, or only one proper field was named.	The statement has been deleted.	See "EXCHANGE" verb sentence format, Page VI-16. Make the necessary corrections to the source program and recompile.
ΔDE185	"EXCH-ANGE"	The fields named did not have identical data descriptions.	The statement has been deleted.	See Convention 1 under "EXCHANGE" verb, Page VI-16. Make the necessary corrections to the sourc program and recompile.
ΔDE190	"MOVE"	The source is a record or group; the destination was a record or group. The source and destination did not have the same number of fields.	The statement has been deleted.	See Convention 12 under "MOVE" verb, Page VI-22. Make the necessary corrections to the source program and recompile.
ΔDE201		An incorrect parameter code has been encountered as input to Arithmetic Analyzer. This condition can only be caused by compiler or machine error.	Processing cannot continue.	Equation has been deleted. Recompile.
ΔDE202		Unmatched parenthesis in subscript. Too many right or left parenthesis.	The statement has been deleted.	Make necessary corrections and recompile.
ΔDE204		Equal sign was found in subscript.	The equation was deleted. "Possible error generated".	Make the necessary corrections to source program and recompile.
ΔDE206		No left parenthesis was found after array name.	The equation was deleted.	Make the necessary corrections to source program and recompile.

ERROR MESSAGE	VERB	MEAMING	COMPILER ACTION	PROGRAMMER ACTION
ΔDE210		A True-False variable appeared to the right of an equal sign and was involved in an arithmetic operation with a floating point variable.	The equation was deleted.	Make the necessary corrections to source program and recompile.
ΔDE214		Too many right parentheses used in an expression.	The equation was deleted.	Make the necessary corrections to source program and recompile.
ΔDE218		A subscript is of higher dimension than the array, e.g., a two dimensional subscript for a one dimensional array.	The subscript has been deleted.	Correct the source program and recompile
ΔDE219		A single quantity followed by a left parenthesis. Probably a missing "*" e.g. A(B+C) instead of A*(B+C).	The statement has been deleted.	Make the necessary corrections to source program and recompile.
ΔDE300	"MOVE"	The source was an element; and the destination was an array.	The statement has been deleted.	Check data description of source element. Make the necessary corrections to the source program and recompile.
ΔDE301	"MOVE"	A MOVE BCD numeric input field to BCD numeric input field with "E" was used.	The statement has been deleted.	Check data descriptions of source and destination fields. Make the necessary corrections to the source program and recompile.

ΔDE302	"MOVE"	A MOVE BCD numeric input field with "E" to BCD numeric input field without "E" was used.	The statement has been deleted.	Check data descriptions of source and destination fields. Make the necessary corrections to the source program and recompile.
ΔDE303	"MOVE"	A MOVE BCD numeric input field with "E" to BCD numeric input field with "E" was used and the source and destination had different data descriptions.	The statement has been deleted.	Check data descriptions of source and destination fields. Make the necessary corrections to the source program and recompile.
ΔDE304	"MOVE"	A MOVE fixed point numeric field or constant to an alphanumeric element or field has been used. The source and destination had an unequal number of characters in the data images.	The statement has been deleted.	See Convention 4 under "MOVE" verb, Page VI-22.

ΔDE305	"MOVE"	A MOVE fixed point numeric field or constant to a BCD numeric input field with "E" has been used.	The statement has been deleted.	See Convention 4 under "MOVE" verb, Page VI-22. Check data description of destination field. Make the necessary corrections to the source program and recompile.
ΔDE306	"MOVE"	A MOVE floating point numeric field to an alphanumeric field has been used. The source and destination have an unequal number of characters in their data images.	The statement has been deleted.	See Convention 4 under "MOVE" verb, Page VI-22.
ΔDE307	"MOVE"	A MOVE alphanumeric field to a floating point numeric field has been used. The source and destination had an unequal number of characters in their data images.	The statement has been deleted.	See Convention 5 under "MOVE" verb, Page VI-22. Make the necessary corrections to the source program and recompile.
ΔDE308	"MOVE"	A MOVE alphanumeric element, field, or literal to a fixed point numeric field has been used. Source and destination had an unequal number of characters in their data images.	The statement has been deleted.	See Convention 5 under "MOVE" verb, Page VI-22. Make the necessary corrections to the source program and recompile.
ΔDE309	"MOVE"	A MOVE alphanumeric element, field, or literal to a true-false variable was specified.	The statement has been deleted.	See Convention 5 under "MOVE" verb, Page VI-22. Check data description of source field. Make the necessary corrections to the source program and recompile.
ΔDE310	"MOVE"	MOVE BCD numeric input field to a true-false variable was specified.	The statement has been deleted.	Check data description of source field. Make the necessary corrections to the source program and recompile.

ERROR MESSAGE	VERB	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔDE311	"MOVE"	A MOVE BCD numeric input field with "E" to a fixed point numeric field was specified.	The statement has been deleted.	Check data description of source field. Make the necessary corrections to the source program and recompile.
ΔDE312	"MOVE"	A MOVE BCD numeric input field with "E" to a true-false variable was specified.	The statement has been deleted.	Check data description of source field. Make the necessary corrections to the source program and recompile.
ΔDE313	"MOVE"	The source was a field and the destination was an array in a MOVE statement. The source field and array field had different data descriptions.	The statement has been deleted.	Check data descriptions of source and destination. Make the necessary corrections to the source program and recompile.
ΔDE314	"MOVE"	Source was a field, element, or array in an input file area; destination was a record or group.	The statement has been deleted.	See Convention 12 under "MOVE" verb, Page VI-23. Make the necessary corrections to the source program and recompile.
ΔDE315	"WRITE" OPTION 1	A fixed point numeric field with "E" has been named in a WRITE statement.	The statement has been deleted.	Check data description of field. Make the necessary corrections to the source program and recompile.
ΔDE316	"MOVE"	Source was a record or group; destination was a field.	The statement has been deleted.	See Convention 12 under "MOVE" verb, Page VI-23. Make the necessary corrections to the source program and recompile.

ΔDEBIG	"MOVE"	A MOVE statement exceeded the limit of 50 names in the combined source and destination. Also, an IF statement or an arithmetic statement has exceeded its limits.	The statement has been deleted.	Make the necessary corrections to the source program and recompile. Write several statements instead of one statement.
ΔDEXXX	ALL VERBS	Compiler Error: An improper tag has been used in the internal compiler language indicating an undetected error in some previous compiler phase.	The statement was deleted.	Check sentence format under appropriate verb. Make the necessary corrections to the source program and recompile.

ΔDEXYX	ALL VERBS	An error in a generator class or number assignment indicating an undetected error in some previous compiler phase.	The statement was deleted.	Check sentence format under appropriate verb. Make the necessary corrections to the source program and recompile.
ΔDEZZZ		Too many constants have been used in the Procedure Division, i.e., constants which are not described in the Constant Section.	The compiler has assigned the dummy symbolic name "[*]" for the last procedural constant encountered.	The constant should be given a field name and placed in the Constant Section. Then it can be referred to by name in the Procedure Division without exceeding the limit.
ΔFL100	"ADD" "SUBTRACT" "MULTIPLY" "DIVIDE"	An alphanumeric field (or element) has been named in an arithmetic statement.	The dummy symbolic field name "[*]" was substituted by the compiler for the original field (or element) name. This causes an "undefined symbol" flag to appear in the object program listing.	See appropriate verb function. Check data description of field (or element) in error. Make the necessary corrections to the source program and recompile.
ΔFL121	"IF"	Mixed floating point numeric and fixed point numeric fields have been named.	Object coding produced converts the fixed point field to floating point and performs the comparison.	Check data descriptions of the two fields being compared. If necessary, correct the source program and recompile.
ΔFL122	"IF"	Mixed numeric and alphanumeric fields have been named.	Object coding produced converts the fixed point field to alphanumeric and performs the comparison.	Check data descriptions of two fields being compared. If necessary, correct the source program and recompile.
ΔFL123	"IF"	Two alphanumeric fields have been named that are not the same size (i.e., unequal number of characters).	The object coding produced uses the size of the smaller field to perform the comparison.	See Convention 5 under "IF". Make the necessary corrections to the source program and recompile.
ΔFL124	"IF"	Only one alphanumeric field has been named	The object coding produced tests the six left most characters of the field.	Check data description of field. See Option 4 under "IF" sentence. Make the necessary corrections to the source program and recompile.

ERROR MESSAGE	VERB	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔFL130	"GO"	More than one sentence name appeared without a "depending on" field.	Object coding produced will "GO TO" the first sentence named.	See Option 2 under "GO" sentence format, Page VI-17. If necessary, correct the source program and recompile.
ΔFL131	"GO"	The "depending on" field named was a floating point numeric.	The object coding produced converts the field to fixed point numeric and uses the integer portion only.	Check data description of field. See Convention 2 under the "GO" verb, Page VI-17. If necessary, correct the source program and recompile.
ΔFL132	"GO"	The "depending on" field named was alphanumeric.	The object coding produced uses a constant of value "1" instead of the original field.	Check data description of field. See Convention 2 under the "GO" verb, Page VI-17. Make the necessary corrections to the source program and recompile.
ΔFL133	"GO"	The "depending on" field named was fixed point numeric, but not an integer.	The object coding produced uses only the integer portion of the field.	Check data description of field. See Convention 2 under the "GO" verb, Page VI-17. If necessary, correct the source program and recompile.
ΔFL140	ALL VERBS WITH SUB-SCRIPTS	An alphanumeric field has been named as a subscript.	The object coding produced uses a constant of value "1" instead of the original field.	Check data description of field used as a subscript. See ARRAY SECTION, Page V-32. Make the necessary corrections to the source program and recompile.
ΔFL141	ALL VERBS WITH SUB-SCRIPTS	An absolute numeric subscript has been used with an inconsistent exponent value (e.g., "2E-1").	The object coding produced uses a constant of value "1" instead of the original subscript.	Check format of absolute subscript. See ARRAY SECTION, Page V-32. Make the necessary corrections to the source program and recompile.
ΔFL150	"ADVANCE"	A floating point numeric field has been named to control slewing of printer paper.	The object coding produced converts the field to fixed point numeric, and only uses the integer portion.	Check data description of field. See Convention 1 under "ADVANCE" verb, Page VI-7. If necessary, correct the source program and recompile.

ΔFL151	"AD-VANCE"	A True-False variable has been named to control slewing of printer paper.	The True-False variable is used by the object program with no alteration.	See Convention 1 under "AD-VANCE" verb, Page VI-7. Check True-False Section of source program. If necessary, correct the source program and recompile.
ΔFL152	"AD-VANCE"	An alphanumeric field has been named to control slewing of printer paper.	The object coding produced uses a constant of value "1" instead of the original field.	Check data description of field. See Convention 1 under "ADVANCE" verb, Page VI-7. If necessary, correct the source program and recompile.
ΔFL153	"AD-VANCE"	A fixed point numeric field has been named to control slewing of printer paper. The field was not an integer.	The object coding produced uses the integer portion only.	See Convention 1 under "AD-VANCE" verb, Page VI-7. If necessary, correct the source program and recompile.
ΔFL154	"AD-VANCE"	An error has occurred in internal compiler language generator number.	Object coding produced slews one line.	Check File Name used. If necessary, correct the source program and recompile.
ΔFL160	"READ" OPTION 1	A field named in a READ CONSOLE SWITCHES statement was not a True-False variable.	The object coding produced places the appropriate switch setting in the first computer word storage of the original field.	See Convention 7 under "READ" verb, Page VI-31. Make the necessary corrections to the source program and recompile.
ΔFL165	ALL VERBS	Only constants were named in a source statement.	Object coding produced stores constants according to computation mode in the Environment Division.	See sentence format under the appropriate verb, Chapter VI. Make the necessary corrections to the source program and recompile.
ΔFL166	"ADD" "SUB-TRACT" "MUL-TIPLY" "DIVIDE"	Only constants were named in an arithmetic statement with no receiving field named.	The compiler has assigned the dummy symbolic field name "[*]" for the receiving field. This causes an undefined symbol flag to appear in the object program listing.	See appropriate verb sentence format. Make the necessary corrections to the source program and recompile.

ERROR MESSAGE	VERB	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔFL174	"MOVE"	The source field was larger than the destination field.	The object coding produced truncates the source field according to the data image of the destination field.	See Conventions under "MOVE" verb. If necessary, correct the source program and recompile.
ΔFL178	"MOVE"	The source and the destination used are arrays but the arrays are of different size. The size of the smaller array determines the number of fields moved.	The object coding produced moves the source array to the destination array without regard to column, row, or plane description of the destination array.	Check the Repeat column in data descriptions. Check the Array Section in the source program. Make the necessary corrections to the source program and recompile.
ΔFL179	"MOVE"	The source was a numeric field; the destination was a numeric array. The source field and the array field have different data descriptions.	The object coding produced fills the array with the source field without the adjustment normally performed on a MOVE.	Check data description of the source field. See Conventions under "MOVE" verb. Make the necessary corrections to the source program and recompile.
ΔFL180	"MOVE"	The source was a field (or element); the destination was an array. The source field was smaller in size than the array field.	The object coding produced fills the array with the source field without regard to separate array fields (i.e., the array block is filled as if it were a single field).	Check data description of the source field. Make the necessary corrections to the source program and recompile.
ΔFL182	"MOVE"	A MOVE statement was encountered with mixed alphanumeric and numeric fields. The source field was smaller in size (i.e., number of characters in the data image).	The object coding produced performs the move with blank fill. In this case, the numeric field is stored as a BCD numeric and not binary.	Check field data descriptions. If necessary, correct the source program and recompile.

ΔFL187	"WRITE" OPTION 1	An array was named in a WRITE statement. The size of the array exceeded 84 characters.	The object coding produced types only the first 84 characters of the array.	If necessary, correct the source program and recompile.
ΔFL190	"MOVE"	The source, a record (or group) is larger than the Destination Record (or Group).	The object coding produced moves only the number of fields in regard to the number of Destination fields.	Check data description of Destination Record or Group. Make the necessary corrections to source program and recompile.
ΔFL200		A constant has been used in a sentence with a field which has an erroneous data description.	The compiler has assigned the undefined name "[*]" in place of the constant.	Check data description of field. Correct source program and recompile.
ΔFL203		Too many right parentheses.	Last right parenthesis is deleted from equation and processing is continued.	If necessary, correct the source program and recompile.
ΔFL205		A non-numeric field has been encountered in an expression.	If the number of characters is less than 4, the field will be handled as though it were an integer. If the number of characters is 4 or more, the field will be handled as though it were a fixed point field at a scale of 38	If necessary, correct the source program and recompile.
ΔFL208		A second non-numeric array was encountered where the first non-numeric array was not an FL205 error.	The second non-numeric array was deleted, expression ended, and processing continued.	Make the necessary corrections to the source program and recompile.
ΔFL209		A non-numeric array was found to have a zero word size (zero characters).	It was set to a word size of one and processing continued.	Make the necessary corrections to source program and recompile.

ERROR MESSAGE	STOP	MEANING	COMPILER ACTION	PROGRAMMER ACTION
ΔFL211		Mixed modes of arithmetic were present, i.e., floating point and fixed point to the right of an equal sign involved in an arithmetic operation.	Mode of incorrect variable was assumed to be that of previously decided mode and processing continued.	Correct the source program and recompile.
ΔFL212		A constant negative subscript has been found.	The subscript has been accepted as is.	If necessary, correct the source program and recompile.
ΔFL213		Too many left parentheses in an expression.	Right parentheses were added to the end of the expression until a zero level reduction was reached.	If necessary, correct the source program and recompile.
ΔFL215		An array had a zero value for its first dimension.	It was set to one and processing continued.	Correct the source program and recompile.
ΔFL216		A three dimensional array had a zero value for its second dimension.	The second dimension was set to one and processing continued.	Make the necessary corrections to the source program and recompile.
ΔFL217		A subscript is of lower dimension than the array, e.g., a one dimensional subscript for a two dimensional array.	The subscript has been accepted as is.	If necessary, correct the source program and recompile.

ERROR MESSAGES IN THE REFORMER PHASE

All error messages printed by the Expression Generator during the Reformer phase indicate an expression has been deleted. In some cases, some instructions have already been generated and will appear in the Edited List, but in the majority of cases no instructions will show, as the entire expression will have been deleted before any generation of coding was begun. The majority of the messages can only be caused by machine error producing "incorrect" information as input to the Expression Generator.

Format of Expression Generator Error Messages:

INTERNAL CODE	SENTENCE NUMBER	SEQUENCE NUMBER	CONSTANT
XXXXXX	XXX	XXX	PEX

The internal code is meaningful only to the compiler implementors. It should be ignored unless errors continue to recur. In this case these codes should be reported to the GE Computer Department for diagnostic purposes.

The sentence number is the number assigned internally by the compiler to the sentence in question. It is printed on the right side of the Edited List opposite the sentence.

The sequence number is a number assigned internally by the compiler to a part of the sentence in question. It is a relative number, but it is usually "000". It is not printed on the Edited List and is significant only to the compiler implementors.

The constant "PEX" is always printed as the last part of an Expression Generator error message. When a PEX error message occurs, examine the sentence in question. It can be found in the Edited List by its sequence number which is printed opposite the sentence on the right side of the list. If the error is not obvious, see the Internal Codes listed below. Correct the source program as necessary and recompile. If the error occurs again, report it to the GE Computer Department. Include internal code and all other pertinent information.

Internal Code	Meaning
10552	Expression is too large for input area. It must be divided into two or more expressions.
10554	Expression overflows entry table. It must be divided into two or more expressions. The table can handle 225 entries of four words each (total of 900 words). To compute the number of entries required for an expression use the following rules: 1. Everything to the left of the equals sign is treated the same as information to the right. 2. Each field, function, sign, open, or close parenthesis requires one entry. 3. Each single dimensional subscripted field requires fourteen entries.

4. Each field with a two dimensional subscript requires twenty-two entries.
5. Each field with a three dimensional subscript requires thirty-two entries.
6. If a subscript itself is an expression, apply rule 2 to the expression within the subscript.
7. "X**Y" requires seven entries. If "Y" is an expression, apply rule 2.

The entry table size is subject to change.

10727	Error in subscript dimensions (too few, too many, etc.).
11204	An illegal exponent configuration.
11240	An undefined function name.
11306	Multi-dimension array which does not appear in the Array Section.

V. EDITED LIST

PURPOSE

The last run of GECOM (the EDITOR) produces the object program and a listing documenting the compilation. The Edited List is prepared on the on-line printer. It can be used as a final record of a successful compilation or as a working paper for correcting an unsuccessful compilation. The object program is produced either as a card deck or a magnetic tape file according to the contents of the ASSIGN clause of the Environment Division.

Figure 8 below illustrates the GE-225 High Speed Printer.

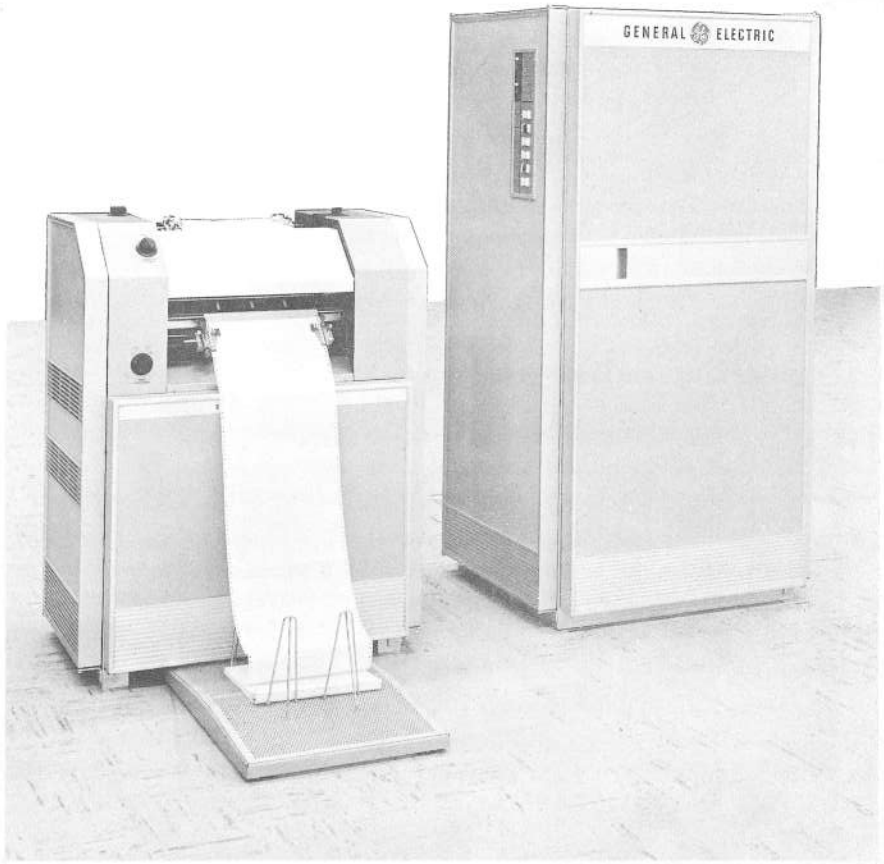


Figure 8. GE-225 High Speed Printer

SWITCH OPTIONS

Two console switches control the preparation of the object program:

Switch 1 down: If the object program is punched on cards, required subroutines are included with the object program. Switch 1 should be down whenever the object program requires the linkage subroutines AP2 or MI2 since these routines are generated by the compiler and are variable. A linkage subroutine is required whenever an API OPEN entrance is given in the Environment Division.

If the object program is written onto magnetic tape, required subroutines are included with the object program regardless of the setting of switch 1.

Switch 5 down: Object program is not punched out or written on tape.

Three console switches control the preparation of the Edited List.

Switch 2 down: Input-output, Report Writer, and GECOM common coding is listed. Since this coding is usually not of interest, it is not listed unless switch 2 is set.

Switch 3 down: Editor is to be rerun.

Switch 4 down: No Edited List is printed.

Inconsistent setting of switches is detected. If switches 1 and 5 are both down, and the object program is assigned to cards, the message "SWITCHES" is typed. The switch settings should then be corrected and switch 0 toggled. Similarly, if switches 2 and 4 are down, the switch setting is indicated as inconsistent (by the typeout "SWITCHES") and should be corrected. If switch settings are found to be inconsistent after an opportunity to correct them has been provided, the lowest numbered switch controlling each preparation is assumed: 1 is assumed if both 1 and 5 are down, and 2 is assumed if both 2 and 4 are down.

If only switches 4 and 5 are down, neither listing nor object program are prepared.

RERUN

When switch 3 is down, the Editor may be rerun to provide other copies of the Edited List and/or the object program. At the end of the compilation, the Editor option switches may be reset to specify new listing or object program preparation instructions. Set the option switches and then toggle switch zero to order rerun (switch 3 should be put to normal unless another rerun is desired).

CONTENTS

A description is given below of the Edited List. The Edited List, obtained when neither switch 2 nor 4 is set, is described first. The effect of setting switch 2 is then discussed.

Page Heading

Each page is headed with the PROGRAM-ID, the DATE COMPILED, and the name of the AUTHOR (if these are supplied in the Identification Division). In addition, a page number is printed.

Other Headings

Several parts of the Edited List are distinguished by means of headings. Three major parts are the source listing, the object listing, and the object information. Within each major part, further distinctions are indicated by minor headings. When a part is displayed in table form, column headings are used.

Part Printing

A description of each part that is printed is given below. The major and minor headings are indicated on the left.

Source Listing

IDENTIFICATION DIVISION: The Identification Division is printed without change.

ENVIRONMENT DIVISION: The Environment Division is printed without change.

DATA DIVISION: The sections of the Data Division are printed differently.

Report Section: In printing the Report Section, column headings taken from the GECOM Report Description form are used. The line identification part (col. 1-27) of line image entries is printed on one line and the actual line image (col. 28-147) is printed on the next line. This format displays the line image in the same positions on the listing as will be obtained in the report. The report definition entries are printed as punched: col. 1-80 on one line, col. 81-147 on the next line.

True-False, Integer, and Array Sections: These sections are printed without change.

File, Common Storage, Working Storage, and Constant Sections: These sections are printed beneath column headings taken from the GECOM Data Division form. An additional column is inserted between the columns for Sequence Number and Type. In this column, GAP symbols are printed for data entries referenced in the Procedure Division. For a type FD entry, the last two digits of the symbol are the assigned file number. For a type R entry, the last two digits are the assigned record number within the file. For all other types of entries, the alphanumeric characters are the GAP symbols assigned to the entries. If this column is blank, no symbol has been assigned.

Tacit Data Division Entries: A list of tacit Date Division entries is also printed. It gives the GAP symbol, name, and format type for each data name mentioned in the True-False, Integer, or Array sections but not described elsewhere in the Data Division and/or for special entries, such as "LINE~COUNT," which are supplied automatically. The size of each dimension is also given for arrays.

PROCEDURE DIVISION: The procedure division is printed without change except for the addition of an internal sequence number which also appears in the listing of the object program.

Object Listing

GAP Symbolic Octal Location: The symbol table formed during the assembly of the object program is listed. It gives the octal location assigned to each GAP symbol. By means of this table and the Data Division table, the octal location assigned to a data entry can be found. The symbol for the data entry is obtained first, then the octal location for that symbol.

Procedure Coding: The assembly input produced for each procedure sentence and the assembly output for that input are printed beneath the sentence. The sentence is printed with its internal sequence number, as in the listing of the Procedure Division. The assembly input and output are listed in the format of the GAP listing: octal output on the left, symbolic input on the right. The octal output is a five digit address followed by the contents (seven digits) of that address.

Storage Reservations and Procedure Constants: After the object coding is printed for all procedure sentences, the working area and constants referred to in this coding are printed. Printing is in GAP listing format.

Object Information

Names and Sizes of Segments and Subroutines Required: A table is printed giving information about each segment or subroutine required by the object program. The GAP symbol and name of each required segment is provided.

If segment being compiled has multiple entrances the entry names are listed.

When an OVERLAY-SEGMENTED object program has been assigned to the DSU, the following information is listed:

1. The GAP symbol associated with the LOAD function.
2. The order in which the overlay segments must be submitted to the RAB function of BRIDGE II.
3. The overlay segment name.
4. The directory table address (in octal) for the overlays at object execution time. (See BRAT CD225E2.005R.)

For required subroutines, the GAP symbol, (which is also the name of the subroutine), and the size of the subroutine are given. The size is the number of locations in the subroutine, in decimal.

If a segment or subroutine has not been supplied with the object program, an asterisk is appended to the GAP symbol. When switch 1 is down or if the object program has been assigned to magnetic tape, the asterisk indicates that the subroutine was not found on the master GECOM tape.

If a subroutine was copied from the master tape and one or more of its records were found to have checksum errors, a plus sign is appended to the GAP symbol. In this case, the copy on the master tape may be defective.

If an invalid subroutine name is specified, the name is printed followed by the legend "UNKNOWN". This indicates a bad compilation. After five names are printed, any other invalid names are ignored.

Memory Use: A table is printed giving the number of locations used by the object program including:

1. Number of words in subroutines.
2. Number of words required for loader and common constants. (This entry must be adjusted by the user whenever the API or Octal Correction module is included in the MCML II Loader.)
3. Number of words required for vector table at load time. Note that the user may "hide" this memory requirement by assigning his vector table to an input/output buffer area through the type 1 card for the MCML II Loader. The compiler assigns the vector table area immediately below the last field literal described in COMMON~STORAGE or to $(08159)_{10}$ if no COMMON~STORAGE is given. No object program memory is lost if there is room for the vector table in COMMON~STORAGE below the last field literal described in COMMON~STORAGE.
4. Number of words in main program.
5. Number of words in COMMON~STORAGE.

These numbers are totaled to indicate the amount of storage required by the object program. If the total exceeds the amount specified for the amount specified for the object computer, a warning is printed. If the total does not include locations in required segments, whose sizes are not known, a warning to include these amounts is printed.

If the object computer is described as having a 16K memory, the amounts, totals, and warnings are printed separately for the lower and upper 8K.

Errors Affecting Object Production: On a separate page after the edited list, a table of any unrecoverable tape errors is printed. The table gives the number of times an unrecoverable tape error has occurred, the controller and handler numbers, and an indication of how the tape is used for each tape contributing to the preparation of the object program.

Switch 2 Option: If switch 2 is down, the following information is included in the object listing:

1. Any location assignments for GECOM common constants and report writer coding when present, are printed before the listing of the procedure coding.

2. Input-output coding is printed after the procedure coding, before the storage reservations and procedure constants.

The location assignment listing consists of instructions to the assembler to assign locations to constants made available to all object programs. The report writer coding listed is generated from the Report Section. The listing of input-output coding consists of the subroutines called in response to READ, WRITE, and similar sentences, and of file tables used by these subroutines. All listings are in GAP listing format.

AS A DEBUGGING TOOL

The following steps should be taken before running a GECOM produced object program. These checks prevent waste of computer time and aid the GECOM user in isolating source language and clerical errors.

Using the Compiler Error Messages

During compilation the source program undergoes a thorough analysis. The compiler notes obvious violations of the language specifications and possible logical or clerical errors by printing a message. The user should validate each message by cross referencing the listing of the source program and the error message manual. Gross errors should be corrected by making the necessary source program changes and recompiling. If the user is satisfied that the Procedure sentence or Data Division entry in question is allowable, the object program may be tried.

Desk Checking the Source Program

The Edited List begins with a print out of the source program as written by the user. In many cases this is the first time the user sees the source program as a printed document. Each Division should be checked.

The Environment Division

1. Object-Computer

   Check object computer configuration. Note that floating point hardware should be listed in the configuration if it is available to the object program even though computation mode is not floating point. The efficiency of the object program could be affected.

   Check object program assignment. In particular, note that an object program should not be assigned to the card punch.

2. File-Control

   Check assignment of all files as follows:

   - Each file in the Data Division should be assigned to a hardware medium.
   - All files under file control should appear in the Data Division.
   - Check all tape and plug assignments.

An error in the Environment Division probably necessitates a recompilation.

The Data Division

1. Check True-False, Array, and Integer sections to make sure that all true false field names, array names, and integer names are listed in the appropriate section.
2. Check file descriptions for proper field sequence.
3. Check qualification of output fields.
4. Check content and format of all constants.
5. Check working storage records and groups for field sequence and total length.
6. Check conditionals to make certain that values have been assigned.

The Procedure Division

1. A check should be made for a PROGRAM-ID name if this program is to be used as a segment.
2. Check housekeeping routines for the following:
   a. Advance printer to top of page if used as an output medium.
   b. Set true-false variables to initial value.
   c. Set switches to initial sentence-name if necessary.
   d. Read console switches to set true-false variables.
   e. Clear work areas to initial contents if necessary.
   f. Open files.
3. Check all sections for a BEGIN and END sentence.
4. Check sequence numbers, if used, to insure that cards have not been shuffled prior to compilation.
5. The program should logically end with a STOP sentence.
6. Check ALTER sentences for proper sentence-names and to make sure that only GO TO Option 1 sentences are being altered.

7. Check data-names for proper and adequate qualification.

8. Make sure that no vocabulary words were used as sentence-names or data-names.

Using the Merged Listing of the Source Program and Generated Coding

Some error flags may appear in this area. A flag consists of a letter "U", "M", "A", or "O" to the right of a line of generated coding. An error flag usually relates to a compiler printed error message for the corresponding sentence. If it does not, check for misspelled sentence names, an unnamed GO TO sentence, errors in sentence format, or vocabulary names used as data-names or sentence-names.

If the user becomes familiar with GE-225 instructions, he may check the contents of the generated coding for a given procedure sentence. Verbs usually produce the same kind of coding.

All symbolic names appearing in the generated coding (except input/output) consist of three characters derived from the numbers zero (0) through nine (9) and letters "A" through "Z".

Common object program constants are used throughout the generated coding. See Appendix D of the GECOM-II REFERENCE MANUAL for their description and symbolic names.

See Appendix F of the GECOM-II REFERENCE MANUAL for a description of "GECOM Input/Output Symbolic Name Assignment."

VI. OBJECT PROGRAM OPERATING INSTRUCTIONS

Before an object program can be released for production running, the programmer must prepare operating instructions for the run. The following information is presented to aid the programmer in this documentation.

SETUP INSTRUCTIONS

Prepare setup instructions from the Environment Division of the source program. These instructions include:

- Object program input unit. (See Object~Computer sentence*.)
- Specific hardware assignments for input and output files. (See File~Control and DSU~CONTROL sentences*.)
- Memory dump tape unit(s). (See I~O~Control sentence*.)

LOADING INSTRUCTIONS

GECOM object program loading instructions may be considered standard, in which case it would not be necessary to repeat these instructions in every run book. Otherwise, loading instructions may be extracted from the following paragraphs.

Object Programs on Cards

If the required subroutines were not punched out as part of the object program during compilation, several operations must be performed before the object program can be executed:

- Consult the Edited List to determine the library subroutine requirements.
- Remove the Constant Cards and transfer cards from the end of each segment object deck and place the segment(s) behind the main program in any order. (If desired, subroutines may be placed between or behind segments because the order is not important.)
- Place the MCML II Loader with the appropriate modules and a type 1 card at the front of the deck. (See CD225B1.006R.)
- If memory is to be cleared before loading the object program, place a memory clear card in front of MCML II.

*See GECOM REFERENCE MANUAL, Chapter VII.

The sequence of the deck when placed in the card reader should then be as shown in Figure 9.

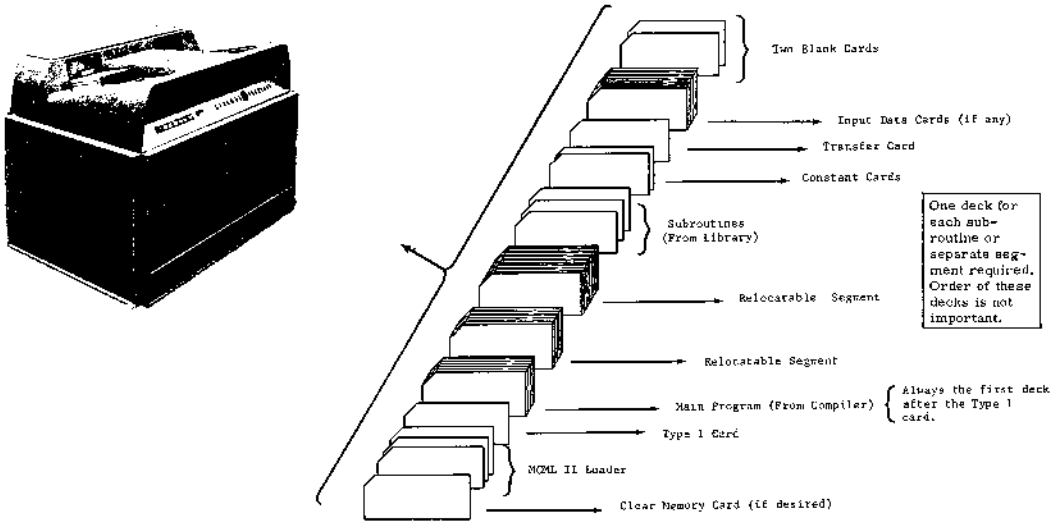


Figure 9. GECOM Object Program Deck

Note that the Compiler (in order to decrease compilation time) does not place the loader in the object program when the object program is assigned to the card reader. The user may further decrease compilation time by directing the compiler, through console switch 1, not to punch out the object program subroutines. The GE Computer Department supplies one copy of the loader to each installation using GECOM. A Library of the object program subroutines is contained on the compiler systems tape. Each installation can establish its own punched card library of these subroutines by punching out any or all of the subroutines from the tape by use of the Selected Punching from Library Instruction Tape (SPLIT) routine. (See Chapter VIII of this manual.

If the object program is assigned to cards and console switch 1 was down during compilation, the compiler automatically will punch out the required subroutines after the main program and before the transfer card. It is not necessary for the user to obtain subroutines from a card file.

The MCML II must be placed in front of the main program as illustrated in Figure 9 before the object program may be executed. (The CONSTANTS cards must be immediately before the transfer card.)

MCML II Loader

All of the decks are loaded by the MCML II Loader. The Type 1 card required by MCML II is prepared by the user to indicate the desired loading options (print memory map, include date, etc.). For additional information on loading options, operating instructions, etc., see MCML II Loader, (CD225B1.006R).

Object Programs on Magnetic Tape

Ordinarily, object program card decks are processed by the BRIDGE II Service System, (CD225J1.001), to place the program on magnetic tape. If a total object program contains no SEGMENTS, it can be placed directly onto a BRIDGE II compatible magnetic tape by the compiler. A magnetic tape object program producted by GECOM is in the following sequence:

1. Beginning Tape Label (BTL) record (24 word record).
2. Sequential Run Locator (one record of "n" words).
3. Run Index (40 word record of zeros).
4. End-of-File Mark.
5. Run Label (24 word record containing PROGRAM-ID as run name).
6. Bootstrap Routine (one record of "n" words).
7. MCML II Loader ("n" card image records).
8. MCML II Transfer Record (one card image record).
9. Type 1 Parameter Card (one card image record).
10. Object program including required library subroutines ("n" card image records).
11. Sequential Run Locator (one record of "n" words).
12. Run Index (40 word record containing PROGRAM-ID of object program).
13. End-of-File Mark.
14. Instruction Tape Fence (24 word record).
15. End-of-File Mark.

Loading Object Programs from Tape

If the object program was placed on magnetic tape by the compiler, the tape format will contain one card image per block. The MCML II Loader and the Type 1 card image precede the program on tape. The compiler sets the 8K indicator and the AAU indicator in the Type 1 card image according to the Environment Division entries. When the compiler produces the tape object program, the API and Octal Corrections modules of MCML II are set to "No", i.e., the Type 1 card image produced by the compiler will have "ONO" in columns 7-9 and "PNO" in columns 10-12.

The object program is loaded by using the standard START and/or SCC cards. See BRIDGE II, (CD225J1.001).

GECOM Common Constants

The GECOM Common Constants must be the last portion of the program loaded. Whenever the compiler punches the subroutines or places the object program on magnetic tape it places the Common Constants just before the transfer card. The constants are in a pseudo-subroutine called CON.

> Note: Although the format of the CON subroutine is relocatable, the constants still load into the absolute locations 00252_8 - 00377_8. Thus, they overlay part of the MCML II Loader. No header cards may be processed after the CON subroutine has been loaded. If the CON deck has not been loaded when the transfer card is reached, a missing subroutine stop occurs.

Absolute Coding

The MCML II Loader treats every card or card image following the Common Constants as absolute coding. This coding must be assembled with GAP II in the relocatable format. It is treated as absolute coding, and no relocation of either location or address is made. If BRIDGE II is used to make corrections to an object program on tape, corrections could be of this type. (Corrections can be inserted just before the transfer card on tape and thus, behind the CON card images.)

Absolute Transfer

If the regular transfer card is removed and an absolute transfer card substituted, MCML II transfers to the 14-bit address in the transfer card. Index register 1, Group Zero, then contains the 14-bit address of the regular GECOM object program entrance.

When it loads an object program from tape, MCML II passes on an indication of tape and plug number in the A and Q registers. When it loads an object program from cards, it sets A to zero before exiting. If the user substitutes an absolute transfer card, it is his responsibility to reset the A and Q registers before going to the regular GECOM object program entrance.

MCML II Transfer Table Location

The MCML II transfer table requires 6N+2 locations where there are N entrances in the program. It is ordinarily constructed immediately below the lowest addressed Field Literal in Common Storage. Thus, only those quantities in Common Storage above this address are preserved. The table may be placed elsewhere in memory, for example, in a tape input/output buffer. In order to move this table area, an octal address is punched into columns 25-30 of the Type 1 card.

The address should be the highest available address since the table is developed in a downward direction from this address. It is made even by MCML II. Before MCML II relinquishes control, it zero clears the transfer table.

VII. OBJECT PROGRAM MESSAGES

SOURCE PROGRAM OPERATIONS

The programmer is responsible for preparation of instructions for operations which he has introduced via READ (option 1), STOP, and WRITE (option 1), and ENTER GAP sentences. The programmer should consult the source program listing and provide the operator with instructions for all "read control switch" loops and messages initiated by such sentences.

GECOM PRODUCED OBJECT PROGRAM MESSAGES

All possible messages which may be compiled automatically into an object program are listed on the following pages. The messages, meanings, and actions are as specific as possible for such a general list. The programmer may extract messages from this list which are appropriate to his program, and then tailor the messages, meanings, and actions

Note that the compiler assigns a two digit file number starting at "00" to each data file in order of appearance in the source program Data Division. These file numbers are used in certain object program messages to designate the file to which the message refers. In the following list, the letters "f" and "n" are used in the appropriate messages to indicate positions in which a file number may appear. In preparing operating instructions the programmer may treat such messages in either of the following ways:

1. Show the messages, meanings, and actions in the general format and wording in which they appear in the following list, but provide the operator with a legend cross referencing compiler assigned file numbers to user assigned file names and hardware units.

2. Show each message, meaning, and action for each file for which the message may occur, replacing the letters "f" and "n" in the messages with specific file numbers and showing the actual file names and hardware units under meaning.

The compiler also produces conventional label messages in object programs. The programmer can and should show specific label messages in his operating instructions. (See "PxΔTyΔ" in the following list. Also see GET PROGRAMMING CONVENTIONS, for further information on tabbing, color coding, etc.)

In addition, the programmer should merge into his list of GECOM produced object messages all messages which he has created through use of the STOP and WRITE verbs. Also, if TAL (loader emitted by BRIDGE II RAB function) or GAL (CD225B1.013R) loaders or DSU/SIOS (CD225E8.000) are used with the object program, their halts and messages should be included in the programmer's list.

OBJECT PROGRAM MESSAGES

MESSAGE	MEANING	OPERATOR ACTION
CC`xxxxxx	1. Input card count. xxxxxx represents number of cards read from input card file. (Typed at end-of-file.)	1. Remove cards from reader. Check card count if known.
	2. Output card count. xxxxxx represents number of cards punched in output card file. (Typed at end-of-file.)	2. Remove cards from punch.
	3. Printed output line count. xxxxxx represents number of lines printed on listing. (Typed at end-of-file.)	3. Remove listing from printer.
CFSfnS	fn is a file number 1. If fn is an input file, a READ of a DSU file has been given without the necessary previous seek.	1. Remove program.
	2. If fn is an output file, a WRITE for a DSU file has been given, and there is not room for the record in the output buffer.	2. Remove program.
CLE	Program is trying to close an input card file that is in a closed status. Error in logic.	Toggle switch zero to ignore. Otherwise remove program.
CLOfnS	Program is trying to perform an I/O operation on a closed file. fn is the file number. Error in logic which cannot be bypassed.	Remove program.
CRE	Read error on the last card in the hopper. This card must be read again.	Empty the read station. Place the card that was read in error on the read station followed by the remainder of the unread cards, start, toggle switch zero.
END	Termination of run. May be followed by message of programmer origin.	Label and remove output. Setup for next run. If program is on magnetic tape, toggle switch zero to read in sequential run locator. If program is on cards, toggle switch zero to cause the program to attempt to read a binary card into absolute zero and transfer control to that location. The binary card should be the loader for the next program.

ΔFD	Any one of the following: 1. Optional file designated in preceding message is not present in this run. 2. Input tape label forced. The actual reel number, file identification, and creation date contained on the tape identified in the preceding message have been substituted for the expected values in memory. 3. Block or record count forced for input tape designated in preceding message.	 1. None. Program continues without optional file. 2. None. Program accepts tape and continues. 3. None. Program continues as if counts agreed.
FNQfnS	fn is a file number. Program is attempting to execute a seek on DSU file fn which is not in OPEN status.	Remove program.
GEN	An attempt has been made to release data for a report (see GENERATE vert) which has previously been TERMINATED.	Toggle switch zero to reinitialize the report and continue processing.
ITE\\xxxxxx	Input tape error cannot be corrected and no AFTER error section is available. xxxxxx is the block count. Preceding message identifies tape handler, plug, and type of error.	Toggle switch zero to ignore the block. Otherwise remove program.
MREfnS	fn is a file number. A DSU error on file fn cannot be corrected after five attempts and no AFTER error section is available.	Remove program or toggle switch 0 to continue (with a high probability of erroneous results).
ΔMT	Either of the following: 1. Processing of input tape designated in preceding message (PxTy) is completed. 2. Label check failure. Label in preceding message is on tape. Label in following message is expected.	 1. Mount next reel and toggle switch zero. 2. a. To repeat check only, toggle switch zero. b. If wrong tape is mounted, mount correct tape, then toggle switch zero. c. To force acceptance of label, set switch 19 down, then toggle switch zero. The actual label will be substituted for the expected values in memory.

MESSAGE	MEANING	OPERATOR ACTION
NCKfn\xxxxxx	No match on control key. fn is the file number. xxxxxx is the block count.	Toggle switch zero to ignore the block; otherwise, remove program.
NO	Either of the following: 1. Label check failure. Label in preceding message is on tape. This message is followed by an "MT" message with the expected label. 2. Write error on memory dump for rerun point identified in preceding message.	 1. See "MT" message. 2. a. If dump is at the end of a reel, no action; program continues. However, dump cannot be used for later rerun. b. If dump is on a separate rerun tape, the rerun point counter is increased by one and another dump is attempted. This procedure is repeated until a successful dump is written or until end-of-tape. If end-of-tape is reached, the only possible action is to mount a new rerun tape and rerun from point of last successful dump.
OCEfnS	fn is a file number. Program is trying to CLOSE a DSU file fn which is already in CLOSed status or is trying to OPEN DSU file fn which is already in OPEN status.	Toggle switch zero to ignore the OPEN or CLOSE operation. Otherwise, remove program.
OFC	Program is trying to close a printer or punch file which is in a closed status. Error in logic.	Toggle switch zero to ignore. Otherwise, remove program.
OFO	Program is trying to open a printer file which is in open status. Error in logic.	Toggle switch zero to ignore. Otherwise, remove program.
OPE	Program is trying to open an input card file which is in open status. Error in logic.	Toggle switch zero to ignore. Otherwise, remove program.
OPFfnS	A file designated as optional is ready to be opened. fn designates the file number. If file is absent, AFD will be typed on the same line.	If file is present for this run, set switch 19 down and toggle switch zero. If not present, toggle switch zero without setting switch 19.
OPO	Program is trying to open a punch file which is in open status. Error in logic.	Toggle switch zero to ignore. Otherwise, remove program.

Px\Ty\	This message may occur alone, with label messages, or as part of other coded messages. The code messages are listed separately with Px\Ty\. 1. Px\Ty\ alone: a. An input tape without labels is expected on handler y on plug x. b. An output tape without labels has been completed and is now rewinding on handler y on plug x. 2. Px\Ty\ with label: a. Indicated label has been read from tape on handler y on plug x. If this is not the expected label, message will be followed by "NO". b. Output tape with the indicated label has been completed and is now rewinding on handler y on plug x.	1. a. None, if tape is mounted. Otherwise, mount tape. b. Remove output tape at end of rewind. Mount blank tape on same handler. If this file is assigned to a single handler, toggle switch zero to continue. 2. a. None, if label is correct. Otherwise, see "NO" and "MT" messages. b. Remove output tape at end of rewind. Mount blank tape on same handler. If this file is assigned to a single handler, toggle switch zero to continue.
Px\Ty\BL	The block count recorded on an end label of an input tape on handler y on plug x does not agree with the count kept during the run.	Toggle switch zero to ignore. Otherwise, remove program or rerun from last rerun point prior to starting processing of this reel.
Px\Ty\CLE	Program is trying to close a tape file on handler y on plug x which is already closed. Error in logic.	Toggle switch zero to ignore. Otherwise, remove program.
Px\Ty\E4 Px\Ty\E5 Px\Ty\E6	Parity error on handler y, plug x. MOD error on handler y, plug x. I/O Buffer error on handler y, plug x.	<u>Input Tape</u>: To try to reread, set switch 19 down and toggle switch zero; otherwise, toggle switch 0 to continue processing until an attempt is made to use the bad record. If an "AFTER ERROR" section is present, no further action required; if not present ITE message will type (See ITExxxxxx).

MESSAGE	MEANING	OPERATOR ACTION
		Output Tape: Toggle switch zero to cause skipping of tape and retrying the write. Program will alternately skip 3 inches of tape and retry writing 5 times until either the write is successful or 60 inches of tape have been skipped. If unsuccessful, the whole operation described above may be repeated by again toggling switch zero. To mount a new output tape, change tapes, set switch 19 down and toggle switch zero. The error tape will have two tape marks following the last good record. Labels will not be put on the end of the tape in error, or on the beginning of the new tape. Otherwise, rerun from last rerun point prior to start of creation of this reel. Do not use same tape.
Px Ty E7	Tape controller error, handler y, plug x: 1. Tape in "LOCAL" 2. Unit not dialed. 3. Multiple handlers dialed to same number. 4. Ring out on an output tape. 5. No tape on handler. 6. Tape was rewinding when controller received the instruction.	Correct the error condition, reset controller, and toggle switch zero. The command will be reexecuted.
Px Ty ELR	Ending label read error on tape handler y, plug x.	Set switch 19 down, to indicate END OF FILE, on 19 normal to indicate END OF REEL and toggle switch zero.
Px Ty LRE	Beginning label read error on tape handler y, plug x.	Toggle switch zero to try reading label again or set switch 19 down and toggle switch zero to skip label. When the label is skipped, the label on the next reel will give an error message since reel number is not updated.
Px Ty MOP	Program is attempting to open a file of a multifile tape on handler y, plug x, when another file of the tape is in an open status. Error in logic.	Remove program or toggle switch zero to continue with no positioning of tape.
Px Ty MT	Processing of current input reel on handler y, plug x, is completed.	Mount next reel and toggle switch zero.

Px⎵TyΔOLE	Output label write error on tape handler y, plug x.	Toggle switch zero to continue without a label write, or set switch 19 down and toggle switch zero to try the label write again after skipping 3 inches of tape. Otherwise, if tape is at beginning of reel, a new blank tape may be mounted.
Px⎵TyΔOPE	Program is trying to open a tape file on handler y, plug x, that is in an open or locked status.	Toggle switch zero to ignore. Otherwise, remove program.
Px⎵Ty⎵RC	The record count recorded on the end label of an input tape on handler y, on plug x, does not agree with the count kept during the run.	Toggle switch zero to ignore. Otherwise, remove program or rerun from last rerun point prior to starting processing of this reel.
zzz⎵⎵⎵Px⎵Uy⎵ EN⎵aaaaaa	Error on DSU file name zzz, plug x, unit y, cannot be corrected after five attempts. aaaaaa is the DSU address in octal. n is one of the following error types: 1. permanent parity error 2. input/output error. An invalid instruction or instruction sequence was issued to the DSU controller. 3. unit error. A DSU unit error cannot position itself correctly after five attempts.	If source program specified an "AFTER ERROR" section, no action is required. If no "AFTER ERROR" section was specified, an "MRE" message will be typed out.(See "MREfnS".)
RFRfnS	fn is a file number. A seek for DSU file fn was followed by a second seek without an intervening READ or WRITE.	Toggle switch zero to allow second seek to override first seek. Otherwise, remove program.
RP⎵xxx	Rerun point has just been made. xxx represents the rerun point number. If a write error occurs on dumping memory a "NO" message will be typed.	None, if dump is successful. Otherwise, see "NO" message.

VIII. OBJECT PROGRAM SUBROUTINES

The Object Coding produced by the GENERAL COMPILER contains references to segments of coding which are basic to many programs. These segments have been put in such a form that they may be referenced one or more times throughout the program with the coding appearing only once. These segments of coding are called subroutines. An example of such a subroutine might be a fixed point to floating point conversion routine.

To reduce compilation time, these subroutines which may be common to many different object programs are not generated during compilation. The compiler simply assumes that the subroutines are present in the object program and refers to them through calling sequences. However, the subroutines must be included in the object program before execution. A list of the required subroutines is produced by the Editor as part of the edited listing.

The required subroutines may be placed in the object program through any one of the following options:

1. If the object program is punched on cards and console switch 1 is up during compilation, the compiler will only punch the main program and its transfer card. The subroutines will not be punched. In order to execute the object program, the user must consult the edited list to determine which subroutines are required, then obtain the required subroutines from the library card file and place them in the object program deck after the main program and before the transfer card. (See Chapter VI, OBJECT PROGRAM OPERATING INSTRUCTIONS of this manual for more details on loading.)

2. If the object program is punched on cards and console switch 1 is down during compilation, the compiler will automatically punch out the required subroutines after the main program and before the transfer card. It is not necessary for the user to obtain subroutines from a card file.

3. If the object program is written onto magnetic tape, the compiler will automatically include the required subroutines in the object program on tape.

The Selected Punching from Library Instruction Tape (SPLIT) routine enabling each installation to establish its own punched card library from those subroutines contained in the GECOM subroutine library is described on Page VIII-4.

The GECOM relocatable subroutines are organized into three sections: Arithmetic, Move/Convert, and Procedural. A general description, a list of subroutines and a legend precede each of the three sections. In addition an alphabetic index of all of the Object Program Subroutines is given on Page VIII-2.

SUBROUTINE INDEX

The following four subroutines are called by some of the above subroutines in the GECOM Subroutine Library: ZSC, ZOT, ZCB, and ZOR.

SPLIT

SPLIT - Selected Punching from Library Instruction Tape Routines

PURPOSE

To punch all or selected object program subroutines from the GECOM system tape library without executing a compilation to establish or update a punch card library of such subroutines.

MINIMUM HARDWARE CONFIGURATION

8,192 word memory

Card Reader

Card punch

Printer on plug 6

Magnetic tape handler 1 on plug 1

INPUT CARDS

To select particular subroutines for punching, a name card must be punched for each subroutine desired except as noted in the next paragraph. The format of the name card is as follows:

Columns 1-3	#OV	
Columns 4-12	Left justified BCD name of desired subroutine. Note that some subroutines have more than one entrance name. Only the subroutines name may be used in a name card, and the entire subroutine is always punched. For example, if the FAD subroutine is desired, the FLT subroutine must be requested. (See list on page VIII-6.) Unused columns in the name field should be blank. At present, only three characters are required for subroutine names, but nine columns are allowed for possible expansion and possible inclusion of segments in the library.	
Columns 13-15	PCH	To cause punching of subroutines.
Columns 16-18	PRT	To cause octal print of subroutines.
Columns 19-80	Blank	

If a name card is followed by a card containing PCH in columns 13-15 and all other columns are blank, all subroutines following the last one named will be punched out.

Example:

```
#OVFLOΔΔΔΔΔΔPCH
#OVSQTΔΔΔΔΔΔPCH
#OVΔΔΔΔΔΔΔΔΔΔPCH
```

Causes punching of FLT, SQT and all subroutines following SQT.

SPLIT

The sequence of the input deck should be as follows:

1. If all subroutines are to be punched:
   a. One subroutine name card containing #OVABSΔΔΔΔΔΔPCH in columns 1-15.
   b. One card contains PCH in columns 13-15 with all other columns blank.
   c. A standard end-of-file card.
   d. Two blank cards.
2. If subroutines are to be punched selectively:
   a. Subroutine name cards in alphabetic order.
   b. Optional card with "PCH" in columns 13-15 and all other columns blank to punch all remaining subroutines.
   c. A standard end-of-file card.
   d. Two blank cards.

OUTPUT CARDS

A label card is punched immediately preceding each subroutine. Columns 1-12 of the label card contain the BCD name of the following subroutine. These label cards are used for identification purposes only and should be removed prior to placing a subroutine in an object program.

Subroutine decks are punched in standard relocatable binary format as described in Appendix C of the GECOM-II REFERENCE MANUAL.

OPERATING INSTRUCTIONS

Mount GECOM system tape on handler 1, plug 1.

Place input deck behind SPLIT deck, and load.

SPLIT

LIST OF OBJECT PROGRAM SUBROUTINES ON GECOM SYSTEM TAPE

NAME	ENTRANCES	NAME	ENTRANCES
ABS	ABS	ZBB	ZBB
AP1	APO, APC	ZBN	ZBN, ZBR
AP2	APO, APC	ZCB	ZCB, ZBC, ZSG, ZOT
ART	ART	ZFF	ZFF, ZGG
AXP	AXP	ZFL	ZFL
EXP	EXP	ZFN	ZFN, ZRF
FLT	FAD, UFA, FSU, UFS, FMP, FDV	ZFR	ZFR
FLX	FLX	ZIP	ZIP, ZAP
FXL	FXL	ZLX	ZLX
FXP	FXP	ZMV	ZMV
LOG	LOG, LN	ZMW	ZMW
MI1	MIP, MIO, MIC	ZNB	ZNB, ZNR
MI2	MIO, MIC	ZNE	ZNE
MI3	MIP, MIO, MIC	ZNF	ZNF
MTF	MTF	ZNN	ZNN, ZIN
MTR	MTR	ZNP	ZNP
PBS	PBS	ZOR	ZOR
PIN	PIN, POS	ZOT	ZOT
POG	POG, FLN	ZPR	ZPR
PQT	PQT	ZRE	ZRE
PRT	PRT	ZRP	ZRP
PXP	PXP	ZSC	ZSC
RC*	RC*	ZSF	ZSF
RCS	RCS	ZSS	ZSS
RLC	RLC	ZUA	ZUA
RL*	RL*	ZUR	ZUR
RPT	RPT	ZUS	ZUS
*RC	*RC	ZWM	ZWM
SIN	SIN, COS	ZXL	ZXL
SQT	SQT	ZZA	ZZA
TYP	TYP	ZZB	ZZB
TY*	TYA	ZZN	ZZN
ZAM	ZAM, ZAR	CON	False subroutine; entrance

MISCELLANEOUS SUBROUTINES

The subroutines below may be produced from a GECOM compilation, but they also exist outside of GECOM object programs. For a description of the API Executive see CD225J4.000; for DSU/SIOS. (MIO) see CD225E8.000: for *RC see Run Completion, BRIDGE II, CD225J1.001.

Subroutine Name	Description	Length
AP1	API Executive	
AP2	Linkage to API Executive (when API Open is given in Environment Division)	4
MI1	DSUS/SIOS for 1 unit, 1 plug	450
MI2	Linkage to DSUS/SIOS (when MIO entrance is given in Environment Division)	4
MI3	DSUS/SIOS for multiple units and/or multiple plugs	650
*RC	Run Completion Routine	84

ARITHMETIC SUBROUTINES

Calling Sequences to Arithmetic Library Routines

Conventions

All entries to the subroutines are made on Index Register 1.

All subroutines preserve the settings of index 2 and index 3, if they have occasion to use them.

The calling sequences to a subroutine may vary in length from 1 word to 4 words.

Binary points not specified in FXP calling sequences are assumed to be 38.

Legend

BP1	Binary Point of first variable in calling sequence.
BP2	Binary Point of second variable in calling sequence.
Arg1	First variable in calling sequence (Argument 1).
Arg2	Second variable in calling sequence (Argument 2).
AAU	Auxiliary Arithmetic Unit.
CPU	Central Processing Unit.

List of Arithmetic Subroutines is Order of Appearance in Manual

FXP	Fixed Point Package
AXP	Fixed Point Package (AAU version)
FLT	Floating Point Package
FLX	Floating Point to Fixed Point (Conversion Routine)
FXL	Fixed Point to Floating Point (Conversion Routine)
ABS	Absolute Function (Floating Point AAU)
ART	Arctangent Function (Floating Point AAU)
EXP	Exponential Function (Floating Point AAU)
LOG	Log to Base 10 Function (Floating Point AAU)
LN	Log to Base e Function (Floating Point AAU)
SIN	Sine Function (Floating Point AAU)
COS	Cosine Function (Floating Point AAU)
SQT	Square Root Function (Floating Point AAU)
PBS	Absolute Function (Floating Point Package)
PRT	Arctangent Function (Floating Point Package)
PXP	Exponential Function (Floating Point Package)
POG	Log to Base 10 Function (Floating Point Package)
PLN	Log to Base e Function (Floating Point Package)
PIN	Sine Function (Floating Point Package)
POS	Cosine Function (Floating Point Package)
PQT	Square Root Function (Floating Point Package)

FXP

PURPOSE

The GECOM Fixed Point Package is used to perform double-precision arithmetic. In the package, all operands are normalized before an operation takes place. The FXP pseudo accumulator (2 words) is normalized after each operation.

USAGE

Calling Sequences

There are 28 calling sequences to FXP; five each for add and multiply, six each for subtraction and division and six possible stores. The general calling sequence is as follows:

Operation	Operand	Index	Remarks
DLD	Argument 1	X	Optional
SPB	FXP	1	Fixed
XXX	Argument 2	Y	Present but Variable
OCT	OBP2BP1		Optional
XXX	Designates the operation code which is used to decide which calling sequence is being used.		

After a double-precision operation is performed, the result (in pseudo A-Q) may be used in the next operation. If it is not, it is stored into cell blocks of 3 words, (designated (01, (02, etc.) from which it will be used later. The first two words of a block contain the normalized number, and the third word contains the binary point of the number. If the number is zero, the third word contains the illegal binary scale $(2000000)_8$. FXP tests all operands for zero, and makes an exit when zero is encountered. Thus, it is seen that any single operation may involve a variety of operands.

Division and subtraction, for example, have the possibilities of:

1. 2 word (variable B. P.) GECOM Variable - 2 word (V. B. P.) GECOM Variable
2. 2 word (variable B. P.) GECOM Variable - 3 word temporary ex. (01
3. 4 word temporary (01 - 3 word temporary ex. (02
4. 4 word temporary (01 - 2 word (V.B.P.) GECOM Variable
5. 4 word Pseudo A-Q - 2 word (V.B.P.) GECOM Variable
6. 4 word Pseudo A-Q - 3 word temporary (03

where as addition and multiplication, being commutative, regard cases 2. and 4. above as identical.

If the operand is in a 3-word temporary (designated (01) cell or in the pseudo A-Q, the binary point is known. If this is not the case, the 4th word in the general calling sequence is necessary and contains the BP of one or both of the operands as required. If division by zero is attempted, the result is set to the largest possible positive fixed point number. Likewise, if the final result of any series of calculations is larger in magnitude than can be stored at the requested binary point, the largest positive number which the requested scale can accommodate is stored.

FXP

The GAP operation code, beginning with the letter Z, is used. The 2 octal digits in the operation code following the character Z are inserted into bits S, 1-4 and the instruction is treated as an LDA. Thus, Z01 or ADD will place the same bit configuration into bits S, 1-4 of the instruction.

All possible calling sequences follow:

	Operation	Operand	Index	Remarks
1.	SPB	FXP	1	Ex.) Pseudo A-Q + A,X
	Z00	Arg2	X	either Arg. in PAQ - other not in temporary.
	OCT	OBP2000		
2.	DLD	Arg1	X	Ex.) A,X + B,Y
	SPB	FXP	1	Both 2 word operands
	Z01	Arg2	Y	
	OCT	OBP2BP1		
3.	DLD	Arg1	X	Ex.) A,X - B,Y
	SPB	FXP	1	Both 2 word operands
	Z02	Arg2	Y	
	OCT	OBP2BP1		
4.	SPB	FXP	1	3 word Pseudo A-Q to CPU
	Z03	BP1 (decimal)		A-Q at specified B.P.
5.	SPB	FXP	1	3 word temporary to hardware
	Z04	(01		A-Q, normalized and BP put into
				address of location 3,1
6.	DLD	(01		
	SPB	FXP	1	Ex.) (01/B,Y
	Z05	Arg2	Y	Arg1 is a temporary cell and
	OCT	OBP2000		Arg2 is a 2 word operand.
7.	SPB	FXP	1	3 word Pseudo A-Q to CPU A-Q
	Z06	0		and normalized.
8.	DLD	(01		
	SPB	FXP	1	Ex.) (01 - B,Y
	Z07	Arg2	Y	
	OCT	OBP2000		
9.	SPB	FXP	1	Ex.) Pseudo A-Q - B,Y
	Z10	Arg2	Y	
	OCT	OBP2000		
10.	DLD	Arg1	X	Ex.) A,x + (01 or Reverse
	SPB	FXP	1	
	Z11	(01		
	OCT	0000BP1		
11.	DLD	Arg1	X	Ex.) A,X - (01
	SPB	FXP	1	
	Z12	(01		
	OCT	0000BP1		

FXP

	Operation	Operand	Index	Remarks
12.	SPB Z13	FXP (01	1	3 word Pseudo A-Q to 3 word (01
13.	SPB Z14	FXP 0	1	Floating Point variable in C.P.U. A-Q 3-word normalized variable in pseudo A-Q.
14.	DLD SPB Z15 OCT	Arg1 FXP Arg2 OBP2BP1	X 1 Y	Ex.) A,X * B,Y
15.	DLD SPB Z16 OCT	Arg1 FXP Arg2 OBP2BP1	X 1 Y	Ex.) A,X/B,Y
16.	SPB Z17	FXP 0	1	3 word Pseudo A-Q to C.P.U. A-Q. Normalized and BP put into address of location 3,1.
17.	SPB Z20	FXP (01	1	Ex.) Pseudo A-Q + (01
18.	DLD SPB Z21 OCT	Arg1 FXP (01 0000BP1	X 1	Ex.) A,X * (01 or Reverse
19.	DLD SPB Z22 OCT	Arg1 FXP (01 0000BP1	X 1	Ex.) A,X/C01
20.	SPB Z23	FXP (01	1	Ex.) Pseudo A-Q - (01
21.	SPB Z26	FXP (01	1	Ex.) Pseudo A-Q / (01
22.	SPB Z27	FXP (01	1	Ex.) Pseudo A-Q * (01

FXP

	Operation	Operand	Index	Remarks	
23.	SPB	FXP	1	Ex.)	Pseudo A-Q * B,Y
	Z30	Arg2	Y		
	OCT	OBP2000			
24.	DLD	(01		Ex.)	(01 + (02
	SPB	FXP	1		
	Z31	(02			
25.	DLD	(01		Ex.)	(01 - (02
	SPB	FXP	1		
	Z32	(02			
26.	SPB	FXP	1	Ex.)	Pseudo A-Q / B,Y
	Z33	Arg2	Y		
	OCT	OBP2000			
27.	DLD	(01		Ex.)	(01 * (02
	SPB	FXP	1		
	Z35	(02			
28.	DLD	(01		Ex.)	(01/(02
	SPB	FXP	1		
	Z36	(02			

CONVENTIONS

Memory locations used: 413

SUBROUTINES REQUIRED: None

AXP

AXP - Fixed Point Package for AAU Machine

PURPOSE

This version of the GECOM fixed point package saves memory space by using the AAU for multiplies and divides. It is not significantly faster than FXP, however.

USAGE

See FXP

CONVENTIONS

Memory locations used: 382.

SUBROUTINES REQUIRED: None

FLT

FLT - Floating Point Package

PURPOSE

To simulate the function of the AAU in performing floating point arithmetic operations. Input is found and output is left in floating point mode in the A and Q registers of the CPU.

USAGE

Calling Sequence

	Operation	Operand	Index	Remarks
1.	DLD	Arg1	X	Performs a floating point addition of Arg1, X to Arg2, Y and leaves the floating point sum in hardware (not AAU) A-Q. Result is normalized.
	SPB	FAD	1	
	DLD	Arg2	Y	
2.	DLD	Arg1	X	Same as above but subtraction is performed.
	SPB	FSU	1	
	DLD	Arg2	Y	
3.	DLD	Arg1	X	Same as above but multiplication is performed.
	SPB	FMP	1	
	DLD	Arg2	Y	
4.	DLD	Arg1	X	Same as above but division is performed.
	SPB	FDV	1	
	DLD	Arg2	Y	
5.	DLD	Arg1	X	Performs a floating point addition of Arg1, X to Arg2, Y and leaves the floating point sum in hardware (not AAU) A-Q. Result is left unnormalized.
	SPB	UFA	1	
	DLD	Arg2	Y	
6.	DLD	Arg1	X	Same as above but subtraction is performed.
	SPB	UFS	1	
	DLD	Arg2	Y	

Note: If Arg1 is already in the A and Q registers of the central processing unit, the initial DLD instruction will not be present.

CONVENTIONS

Memory locations used: 440

SUBROUTINES REQUIRES: None

FLX

FLX - Floating Point to Fixed Point Conversion

PURPOSE

To convert a floating point number located in the A and Q registers of the CPU to a fixed point number left in the same registers.

USAGE

Calling Sequence (BP1 is binary point of output)

Operation	Operand	Index
SPB	FLX	1
LDA	BP1	
XXX		Normal Return

CONVENTIONS

Incorrect scaling (trying to convert a floating point number to a fixed point number which will result in truncation of most significant bits) yields a zero in the A and Q registers of the CPU.

Memory locations used: 32.

SUBROUTINES REQUIRED: None

FXL

FXL - Fixed Point to Floating Point Conversion

PURPOSE

To convert a fixed point number located in the A and Q registers of the CPU to a floating point number left in the same registers.

USAGE

Calling Sequence (BP1 is Binary Point of Input)

Operation	Operand	Index
SPB	FXL	1
LDA	BP1	
XXX		Normal Return

CONVENTIONS

Memory locations used: 45

SUBROUTINES REQUIRED: None

ABS

ABS - Absolute Function

PURPOSE

To convert a floating point number located in the Auxiliary Arithmetic Unit A register to a positive floating point number left in the same register.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ABS	1
XXX		Normal Return

CONVENTIONS

Memory locations used: 8

SUBROUTINES REQUIRED: None

ART

ART - Arctangent Function

PURPOSE

To determine the Arctangent (in radians) of any argument X. Y = ART X where X as input and Y as output are both in the floating point mode. Input is found and output is left in the AAU A register.

METHOD

A 15th degree polynomial approximation is used to compute Arctangent X.

$$\text{ART } X = \frac{\pi}{4} + \sum_{i=0}^{7} C_{2i+2} \left(\frac{x-1}{x+1} \right)^{2i+1}$$

All floating point arithmetic operations are performed using AAU instructions.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ART	1
XXX		Normal Return

CONVENTIONS

Memory locations used: 74

SUBROUTINES REQUIRED: None

EXP

EXP - Exponential Function

PURPOSE

To compute the result of e being raised to any input. $Y = e^X$ where X as input and Y as output are both in floating point mode. Input is found and output is left in the AAU A register.

METHOD

A 9th degree polynomial approximation of 2^X is used to calculate e^X of any number.

All floating point arithmetic operations are performed using the AAU instructions.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	EXP	1
XXX		Normal Return

CONVENTIONS

Memory locations used: 98

SUBROUTINES REQUIRED: None

LOG
LN

LOG - Logarithmic Function - Base 10
LN - Logarithmic Function - Base e

PURPOSE

To take the log (base 10 or base e) of any positive arguments. Y = LOG X or Y = LN X where X as input and Y as output are both in floating point mode. Input is found and output is left in the AAU A register.

METHOD

A 7th degree polynomial approximation is used to calculate the logarithm to the base 10 or base e of any positive number. All floating point arithmetic operations are performed using AAU instructions.

USAGE

Calling Sequences

	Operation	Operand	Index
base 10	SPB	LOG	1
	XXX		Normal Return
base e	SPB	LN	1
	XXX		Normal Return

CONVENTIONS

A negative or 0 argument results in the largest floating point number possible being left as the result.

Memory locations used: 97

SUBROUTINES REQUIRED: None

SIN
COS

SIN - Sine Function
COS - Cosine Function

PURPOSE

To determine the Sine or Cosine of any argument X (in radians). Y = SINE X or Y = COSINE X where X as input and Y as output are both in the floating point mode. Input is found and output is left in the AAU A register.

METHOD

A 9th degree polynomial approximation is used to compute Sine X. The argument must be in radians and is reduced to the proper quadrant equivalent before computation of the function. COS X is computed from SIN X = COS ($\pi/2$ - x) (in radians).

$$\text{SIN}\ \left(\frac{\pi}{2}X\right) = ((((C9X^2+C7)\ X^2+C5)\ X^2+C3)\ X^2+C1)\ X+X$$

All floating point arithmetic operations are performed using AAU instructions.

USAGE

Calling Sequences

	Operation	Operand	Index
Sine	SPB	SIN	1
	XXX		Normal Return
Cosine	SPB	COS	1
	XXX		Normal Return

CONVENTIONS

Memory locations used: 97

SUBROUTINES REQUIRED: None

SQT

SQT - Square Root Function

PURPOSE

To take the square root of a floating point argument which is located in the AAU A register and leave the result in the AAU A register.

METHOD

A Newton-Raphson iteration is applied four times to compute the square root of any positive number:

i.e. $Y_{i+1} = 1/2 \left(\frac{ARG}{Yi} + Yi\right)$

where Yi is the initial approximation

All floating point arithmetic operations are performed using the AAU instructions

USAGE

Calling Sequence

Operation	Operand	Index
SPB	SQT	1
XXX		Normal Return

CONVENTIONS

A negative argument results in the largest floating point number possible being left as the result.

Memory locations used: 50

SUBROUTINES REQUIRED: None

PBS

PBS - Absolute Function

PURPOSE

To convert a floating point number located in the central processing unit A and Q registers to a positive floating point number left in the same registers.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	PBS	1
XXX		Normal Return

CONVENTIONS

Memory locations used: 23

SUBROUTINES REQUIRED: None

PRT

PRT - Arctangent Function

PURPOSE

To determine the Arctangent (in radians) of any argument X. Y = Art X where X as input and Y as output are both in the floating point mode. Input is found and output is left in the A and Q registers of the central processing unit.

METHOD

A 15th degree polynomial approximation is used to compute Arctangent X.

$$\text{ART } X = \frac{\pi}{4} + \sum_{i=0}^{7} C_{2i+2} \left(\frac{x-1}{x+1}\right)^{2i+1}$$

All floating point arithmetic operations are performed using FLT (floating point package).

USAGE

Calling Sequence

Operation	Operand	Index
SPB	PRT	1
XXX		Normal Return

CONVENTIONS

Memory locations used: 102

SUBROUTINES REQUIRED: FLT

PXP

PXP - Exponential Function

PURPOSE

To compute the result of e being raised to any argument. $Y = e^X$ where X as input and Y as output are both in floating point mode. Input is found and output is left in the central processing unit A and Q registers.

METHOD

A 9th degree polynomial approximation of 2^X is used to calculate e^X of any number.

All floating point arithmetic operations are performed using FLT (floating point package).

USAGE

Calling Sequence

Operation	Operand	Index
SPB	PXP	1
XXX		Normal Return

CONVENTIONS

Memory locations used: 107

SUBROUTINES REQUIRED: FLT

POG
PLN

POG - Logarithmic Function - Base 10
PLN - Logarithmic Function - Base e

PURPOSE

To take the log (base 10 or base e) of any positive argument. Y = LOG X or Y = LN X where X as input and Y as output are both in the floating point mode. Input is found and output is left in the A and Q registers of the central processing unit.

METHOD

A 7th degree polynomial approximation is used to calculate the logarithm to the base 10 or base e of any positive number. All floating point arithmetic operations are performed using FLT (floating point package).

USAGE

Calling Sequences

	Operation	Operand	Index
base 10	SPB	POG	1
	XXX		Normal Return
base e	SPB	PLN	1
	XXX		Normal Return

CONVENTIONS

A negative or 0 argument results in the largest floating point number possible being left as the result.

Memory locations used: 114

SUBROUTINES REQUIRED: FLT

PIN
POS

PIN - Sine Function
POS - Cosine Function

PURPOSE

To determine the Sine or Cosine of any argument X (in radians). Y = Sine X or Y = Cosine X where X as input and Y as output are both in the floating point mode. Input is found and output is left in the A and Q registers of the central processing unit.

METHOD

A 9th degree polynomial approximation is used to compute Sine X. The argument must be in radians and is reduced to the proper quadrant equivalent before computation of the function. Cosine X is computed from --

$$\text{SIN } X = \text{COS } (\pi/2 - X) \quad \text{(in radians)}$$

$$\text{SIN } \left(\frac{\pi}{2} X\right) = ((((C9X^2+C7)\ X^2+C5)\ X^2+C3)\ X^2+C1)\ X+X$$

All floating point arithmetic operations are performed using FLT (floating point package).

USAGE

Calling Sequences

	Operation	Operand	Index
Sine	SPB	PIN	1
	XXX		Normal Return
Cosine	SPB	POS	1
	XXX		Normal Return

CONVENTIONS

Memory locations used: 117

SUBROUTINES REQUIRED: FLT

PQT

PQT - Square Root Function

PURPOSE

To take the square root of a floating point argument which is located in the A and Q registers of the central processing unit and leave the result in the same registers.

METHOD

A Newton-Raphson iteration is applied four times to compute the square root of any positive number.

i.e. $Yi+1 = 1/2 \left(\frac{ARG}{Yi} + Yi\right)$

where Yi is the initial approximation

All floating point arithmetic operations are performed using FLT (floating point package).

USAGE

Calling Sequence

Operation	Operand	Index
SPB	PQT	1
XXX		Normal Return

CONVENTIONS

A negative argument results in the largest floating point number possible being left as the result.

Memory locations used: 50

SUBROUTINES REQUIRED: FLT

MOVE/CONVERT SUBROUTINES

Calling Sequences to Object Program Move/Convert Subroutines

Individual descriptions of the GECOM object library routines follow Legend below. No attempt has been made to describe the "inner" routines that are not used independently. These are ZSC, ZOT, ZCB, and ZOR.

Conventions

1. Input or source address is assumed to be in index register 3 unless the calling sequence denotes the input address.
2. Output or destination address is assumed to be in index register 2 unless the calling sequence denotes an output address.
3. All subroutines preserve the settings of index 2 and index 3.
4. All entrances to the subroutine are made index 1.
5. The calling sequence to a subroutine may vary in length by 1 depending on whether an input or output address is specified.
6. Bit positions not specified in the calling sequences must be zero.

Legend

D	decimal point indicator. 1 = an actual point in the BCD field. If both fields are BCD (input and output), the indicator specifies the output condition. 0 = no actual point.
C	comma indicator. 0 = no commas in the output field. 1 = 1 comma, 10 = 2 commas.
I	input indicator. 0 = input address is part of calling sequence. 1 = input address is in index register 3.
Ø	output indicator. 0 = output address is part of calling sequence. 1 = output address is in index 2.
SCP	Input starting character position (1, 2, 3).
SCPOT	output starting character position (1, 2, 3).
NCI	number of input characters.
NCØ	number of output characters.
FS	input field size, i.e., number of BCD information characters not counting decimal point or separate sign positions.

FSØT — output field size not counting decimal point, separate signs, or edit characters.

DS — input decimal scale = the number of characters before the decimal point (assumed or actual) not counting a separate sign position.

DSØT — output decimal scale = the number of characters before the decimal point (assumed or actual) not counting separate sign positions or edit characters.

BS — input binary scale.

BSØT — output binary scale.

ECP — input ending character position (1, 2, 3).

EDIT — edit type code:

0000 = no edit	0110 = * fill
0001 = zero suppression	0111 = complete zero suppression
0010 = floating $	1000 = Floating -
0011 = fixed $	1001 = Floating +
0100 = fixed $ and zero suppression	1010 = fixed $ and complete zero suppression
0101 = fixed $ and * fill	

NEC — number of characters to edit, if editing, = number of edit symbols + 1 that appear in the Data Division. (Do not add +1 for floating $ or sign.)

IN2 — a value equal to 15_{10} - number of input integer positions BEFORE FULL STOP including any separate leading sign. EXPRESS IN OCTAL.

ØT2 — a value equal to 15_{10} - number of output integer positions BEFORE FULLSTOP including fixed $ and separate leading sign. EXPRESS IN OCTAL.

TS — output type sign indicator:

0000 - no sign	0100 - trailing -	1000 - trailing R
0001 - leading +	0101 - leading I	1010 - trailing CR
0010 - trailing +	0110 - trailing I	1100 - trailing DB
0011 - leading -	0111 - leading R	

NSW — the number of words increment between one input field and the next field; that is, the increment from the word containing the SCP of a field to the word containing the SCP of the next field.

NSW1 — NSW1 = NSW from fields 1-2, 4-5, 7-8, ...

NSW2 — NSW2 = NSW from fields 2-3, 5-6, 8-9, ...

NSW3 — NSW3 = NSW from fields 3-4, 6-7, 9-10, ...

NSWØT — the number of increment words between the SCP of one output field and the SCP of the next output field.

NSWØT1	NSWØT1 = NSWØT from fields 1-2, 4-5, 7-8, ...
NSWØT2	NSWØT2 = NSWØT from fields 2-3, 5-6, 8-9, ...
NSWØT3	NSWØT3 = NSWØT from fields 3-4, 6-7, 9-10, ...
SCP1	input starting character position for fields 1, 4, 7, ...
SCP2	input starting character position for fields 2, 5, 8, ...
SCP3	input starting character position for fields 3, 6, 9, ...
ECP1	input ending character position for fields 1, 4, 7, ...
ECP2	input ending character position for fields 2, 5, 8, ...
ECP3	input ending character position for fields 3, 6, 9, ...
SCPØT1	output starting character position for fields 1, 4, 7, ...
SCPØT2	output starting character position for fields 2, 5, 8, ...
SCPØT3	output starting character position for fields 3, 6, 9, ...

List of Move/Convert Subroutines in Order of Appearance in Manual

ZBN	Binary to BCD
ZZB	Repeated Binary to BCD
ZNN	BCD Numeric to BCD Numeric with Editing (One or Both Fields Packed)
ZZN	Repeated BCD Numeric Moves with Editing (Both Input and Output Unpacked)
ZUS	Repeated BCD Moves with Editing
ZFF	Floating Point to Free Form Numeric BCD
ZNE	Repeated Floating Point to BCD Free Form Numeric
ZFN	Floating Point to Numeric with E Description

ZRE	Repeated Floating Point to Numeric with E
ZNB	BCD to Binary
ZRP	Repeated BCD to Binary
ZUA	Word Moves
ZMV	Hardware Move
ZMW	Repeated, Non-sequential Word Moves Using the MOVE Hardware
ZWM	Repeated, Non Sequential Word Moves
ZIP/ZAP	Character Move
ZNF	BCD to Floating Point
ZFR	Repeated Numeric and Numeric with E to Floating Point
ZNP	BCD to Floating Point
ZPR	Repeated BCD to Floating Point
ZBB	Binary to Binary
MTR	Truncate and/or Round Binary Fixed Point Numbers
ZAM	BCD to BCD
ZZA	Repeated BCD to BCD
ZUR	Repeated BCD to BCD
ZLX	Repeated Move Floating Point Binary to Fixed Point Binary Numbers
ZXL	Repeated Move Fixed Point Binary to Floating Point Binary Numbers
RPT	Repeated Move
ZSS or ZSF	Fields Separated by Commas
MTF	Move to True-False Variables
ZFL	Move and Fill

ZBN

ZBN - Binary to BCD

PURPOSE

To convert a one word or two word binary number to BCD with provision for editing.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZBN	1

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		O	T	2				SCPOT						NCO					
	EDIT				NEC						D	C		TS					I
					NW1			FSOT-DSOT						38-BS					
INPUT ADDRESS, IF NEEDED																			

NW1 - number of binary words. 001 = 1 word, 000 = 2 words.

Example:

Move a one word binary field (binary scale of 15) at SORCE to DEST with a description of $ZZZ9.99 and a starting character position of 2.

Operation	Operand	Index	Remarks
LDA	DEST		
LDX	*-1	2	Destination address into index 2
SPB	ZBN	1	
OCT	0122010		
OCT	0404400		
OCT	0010227		
LDA	SORCE		Input address

CONVENTIONS

Memory locations used: 87

SUBROUTINES REQUIRED: ZCB, ZSC, ZOR

ZZB

ZZB - Repeated Binary to BCD

PURPOSE

To convert repeated binary fields to BCD as in ZBN.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZZB	1

Parameter words as in ZBN, followed by:

S	1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19
		SCPOT1	NSWOT3	SCPOT3	NSWOT2	SCPOT2	NSWOT1
Number of Fields							

Example:

Move two word binary fields (binary scale of 19) to BCD fields with a description of 9999.9999-. The input address is in index 3 and the destination address is in index 2. The move is to be done 16 times. The first output starting character position is 3.

Operation	Operand	Index
SPB	ZZB	1
OCT	0133012	
OCT	0000421	
OCT	0000423	
OCT	0332314	
DEC	16	

CONVENTIONS

Memory locations used: 51

SUBROUTINES REQUIRED: ZBN and its associated routines.

ZNN

ZNN - BCD Numeric to BCD Numeric with Editing.

PURPOSE

To move a BCD numeric field to BCD numeric with editing.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZNN	1

S	1	2 3 4 5 6 7	8 9 10	11 12 13	14 15 16 17 18 19
		IN2	SCP	ECP*	NCI
		OT2	SCPOT		NCO
	EDIT	NEC		D C	TS

* only if trailing sign

Example:

Input description 9(5)V99R SCP is 1. Input address is index 3.
Output description +999,999.999 SCPOT is 3. Destination address in index 2.

Operation	Operand	Index
SPB	ZNN	1
OCT	0121210	
OCT	0103014	
OCT	0000504	

CONVENTIONS

Memory locations used: 94

SUBROUTINES REQUIRED: ZCB

ZZN

ZZN - Repeated BCD Numeric Moves with Editing. (One or Both Fields Packed)

PURPOSE

To move repeated BCD numeric fields.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZZN	1

Parameter words as in ZNN followed by:

S	1 2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19
	SCP1	*ECP1	SCP3	*ECP3	SCP2	*ECP2
	SCPOT1		SCPOT3		SCPOT2	
	NSWOT3	NSW3	NSWOT2	NSW2	NSWOT1	NSW1
NUMBER OF FIELDS						

* only if input has trailing signs

Example:

Input description +9 SCP is 2. Input address is in index 3.
Output Description 99+ SCPOT is 1. Destination address is in index 2.

Operation	Operand	Index	Remarks
SPB	ZZN	1	
OCT	0152002		
OCT	0151002		
OCT	0000004		
OCT	0203010		
OCT	0101010		
OCT	0111011		
DEC	15		15 fields to move

CONVENTIONS

Memory locations used: 68

SUBROUTINES REQUIRED: ZNN and its associated routines.

ZUS

ZUS - Repeated BCD Moves with Editing (Both Input and Output Unpacked)

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZUS	1

Parameter words as in ZNN followed by:

S	1	2 3 4	5 6 7	8 9 10 11 12 13 14 15 16 17 18 19
		NSWOT	NSW	NUMBER OF FIELDS

Example:

Input description-unpacked, 999+ SCP is 3. Input address in index 3.

Output description-unpacked, 9999I SCP is 2. Destination address in index 2. Move 5 fields

Operation	Operand	Index
SPB	ZUS	1
OCT	0143304	
OCT	0122005	
OCT	0000030	
OCT	0220005	

CONVENTIONS

Memory locations used: 37

SUBROUTINES REQUIRED: ZNN and its associated routines.

ZFF

ZFF - Floating Point to Free Form Numeric BCD.

PURPOSE

To convert a two word floating point number to BCD numeric with provisions for editing.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZFF	1

S	1	2 3 4 5 6 7	8 9 10	11 12 13 14 15 16 17	18	19
		OT2	SCPOT	NCO		
	EDIT	NEC		D C TS		I
FSOT-DSOT						
INPUT ADDRESS, IF NEEDED						

Example:

Convert a floating point to BCD format 999999.9+ where output starting character position is 3. Index 3 contains the input address. Index 2 contains the destination address.

Operation	Operand	Index
SPB	ZFF	1
OCT	0113011	
OCT	0000411	
OCT	0000001	

CONVENTIONS

Memory locations used: 60

SUBROUTINES REQUIRED: ZCB, ZSC

ZNE

ZNE - Repeated Floating Point to BCD Free Form Numeric.

PURPOSE

To convert repeated floating point fields to BCD as in ZFF.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZNE	1

Parameter words for ZFF followed by:

S 1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19
	SCPOT1	NSWOT3	SCPOT3	NSWOT2	SCPOT2	NSWOT1
NUMBER OF FIELDS TO MOVE						

Example:

Move 27 fields to unpacked destination description of -9999. Source address is FLTY, destination address in index 2.

Operation	Operand	Index
SPB	ZNE	1
OCT	0122005	
OCT	0000014	
OCT	0000000	
LDA	FLTY	
OCT	0222222	
DEC	27	

CONVENTIONS

Memory locations used: 49

SUBROUTINES REQUIRED: ZFF and its associated routines.

ZFN

ZFN - Floating Point to Numeric with E Description

PURPOSE

To convert a two word floating point number to the BCD E description.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZFN	1

S	1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18	19
		OT2		SCPOT	NCO			
			NCE	ETS	D	TS		I
INPUT ADDRESS, IF NEEDED								

ETS - type sign of exponent. 000 = +, 001 = -.

NCE - number of characters in exponent description. (1 or 2).

Example:

Convert the floating point number at RIGGED. Destination description is 1.99999E+9 with a starting character of 1. The destination address is in index 2.

Operation	Operand	Index
SPB	ZFN	1
OCT	0161011	
OCT	0010424	
LDA	RIGGED	

CONVENTIONS

Memory locations used: 118

SUBROUTINES REQUIRED: ZCB, ZSC

ZRE

ZRE - Repeated Floating Point to Numeric with E.

PURPOSE

To convert repeated floating point numbers to BCD numeric as in ZFN.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZRE	1

Parameter words as in ZFN followed by:

S	1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19
		SCPOT1	NSWOT3	SCPOT3	NSWOT2	SCPOT2	NSWOT1
NUMBER OF FIELDS TO MOVE							

Example:

Convert 3 floating point fields (address in index 3) to a destination (in index 2) description of +9.99999E-99, designated as unpacked. (i.e., SCPOT-2)

Operation	Operand	Index
SPB	ZRE	1
OCT	0152013	
OCT	0021405	
OCT	0232323	
DEC	3	

CONVENTIONS

Memory locations used: 46

SUBROUTINES REQUIRED: ZFN and its associated routines.

ZNB

ZNB - BCD to Binary

PURPOSE

To convert a BCD field to one or two word binary.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZNB	1

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A		D	O	L	S	C	P	ECP*			STS			NCI				**	
DS-FS																			
BSOT-38																			
OUTPUT ADDRESS, IF NEEDED																			

** Not including a separate trailing sign
* Only if there is a trailing sign

STS - Separate Trailing Sign indicator (not an overpunch).
001 = Separate Sign; 000 = No Separate Sign

A - Number of Binary words. 0 = 2, 1 = 1.

L - Input leading Sign Indicator. 0 = No, 1 = Yes.

Example:

Convert a BCD field (address in index 3) with the description 9999V9R to a two word binary number (scale of 19) and store at ENDIT. The input starting character position is 2.

Operation	Operand	Index
SPB	ZNB	1
OCT	0221006	
OCT	3777776	
OCT	3777755	
LDA	ENDIT	

CONVENTIONS

Memory locations used: 343

SUBROUTINES REQUIRED: ZSC

ZRP

ZRP - Repeated BCD to Binary

PURPOSE

To convert repeated BCD fields to binary as in ZNB.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZRP	1

Parameter words as in ZNB, followed by:

S 1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19
	SCP1	ECP1*	SCP3	ECP3*	SCP2	ECP2*
				NSW3	NSW2	NSW1
NUMBER OF FIELDS TO MOVE						

* only if there is a trailing sign on input.

Example:

Convert a BCD image of -99999.999 repeated 13 times to one word binary (scale of 19) and store starting at T1SIT. The input starting character is 2. The input starting address is in index 3.

Operation	Operand	Index
SPB	ZRP	1
OCT	2520012	
OCT	3777775	
OCT	3777755	
LDA	T1SIT	
OCT	0201030	
OCT	0000343	
DEC	13	

CONVENTIONS

Memory locations used: 69

SUBROUTINES REQUIRED: ZNB and its requirements.

ZUA

ZUA - Word Moves

PURPOSE

To move a specified number of words, with provision for handling leading and trailing characters.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZUA	1

S 1 2	3 4	5 6	7 8 9 10 11 12 13 14 15 16 17 18 19
	NL	NT	NUMBER OF FULL WORDS TO MOVE

NL - Number of extra leading characters.
00 = none, 01 = 1, 10 = 2.

NT - Number of extra trailing characters.
00 = none, 01 = 1, 10 = 2.

Example:

Move a field of description +999.9999 to a field of same description; both have a SCP of 2. Output address is in index 2; input address is in index 3.

Operation	Operand	Index
SPB	ZUA	1
OCT	0220002	

CONVENTIONS

The number of full words to move must be ≥1. See ZIP/ZAP.

Memory locations used: 88.

SUBROUTINES REQUIRED: ZIP

ZMV

ZMV - Hardware Move

PURPOSE

To move a specified number of words using the MOV instruction, with provision for handling leading and trailing characters.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZMV	1

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
			NL		NT		NO.			OF	FULL	WORDS		TO	MOVE				

NL - number of extra leading characters. 00=none, 01=1, 10=2.

NT - number of extra trailing characters. 00=none, 01=1, 10=2.

Example:

Move a field of description 999999.999 - to a field of the same description; both have an SCP of 1. Output address is in index 2; input address is in index 3.

Operation	Operand	Index
SPB	ZMV	1
OCT	0020003	

CONVENTIONS

The number of full words to move must be ≥1. See ZIP/ZAP.

Memory locations used: 42

SUBROUTINES REQUIRED: ZIP

ZMW

ZMW - Repeated, Non-sequential Word Moves Using the MOVE Hardware

PURPOSE

To do repeated word moves where the fields are displaced by some fixed amount.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZMW	1

Parameter word for ZMV followed by:

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	NSWØT										NSW								
NUMBER OF FIELDS TO MOVE																			

CONVENTIONS

Memory locations used: 30

ZWM

ZWM - Repeated, Non Sequential Word Moves

PURPOSE

To do repeated word moves where the fields are displaced by some fixed amount.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZWM	1

Parameter words for ZUA followed by:

S	1	2 3 4 5 6 7 8 9 10	11 12 13 14 15 16 17 18 19
		NSWOT	NSW
NUMBER OF FIELDS TO MOVE			

CONVENTIONS

Memory locations used: 32

ZIP/ZAP

ZIP/ZAP - Character Move

PURPOSE

To move from a source field to a destination field where the starting character positions are the same and NCWOT ≤ NCW.

USAGE

Calling Sequence

The calling sequence depends on the starting character positions and the number of characters to move.

SCP	No. of Characters	Calling Sequence		
		Operation	Operand	Index
1	1	LDA	ZER	
		SPB	ZIP	1
1	2	LDA	Z01	
		SPB	ZIP	1
2	1	LDA	Z02	
		SPB	ZIP	1
2	2	LDA	Z03	
		SPB	ZIP	1
2	3	LDA	Z03	
		SPB	ZAP	1
		LDA	ZER	
2	4	LDA	Z03	
		SPB	ZAP	1
		LDA	Z01	
3	1	LDA	Z04	
		SPB	ZIP	1
3	2	LDA	Z04	
		SPB	ZAP	1
		LDA	ZER	
3	3	LDA	Z04	
		SPB	ZAP	1
		LDA	Z01	

Whenever ZAP is used, index register 2 and 3 are incremented by 1 before return is made.

Note that ZUA is used for starting character positions and number of characters not mentioned above.

CONVENTIONS

Memory locations used: 32

SUBROUTINES REQUIRED: None

ZNF

ZNF - BCD to Floating Point

PURPOSE

To convert BCD numeric with E and free form BCD numeric to floating point, using the floating point hardware.

USAGE

Calling Sequence

The calling sequence takes two forms - one for numeric with E, and one for free form numeric.

1. Example using numeric with E:

S	1	2	3	4	5 6 7	8 9 10	11 12 13	14 15 16 17 18 19
	B	D	O	L	SCP	ECP*	STS	NCM
					C	SEC	EC	FLD
OUTPUT ADDRESS, IF NEEDED								

* only if there is a trailing input sign

where:

B - always 1.

L - 1 = leading input sign

STS - 001 = a separate trailing sign, i.e., a separate sign position at the end and not an overpunch.

NCM - number of characters in mantissa, including any decimal point or separate leading sign.

C - always 001.

SEC - starting character position of the exponent sign (1,2,3).

EC - number of exponent characters (1,2).

FLD - 1 minus the field size as a positive (absolute) value where field size is the number of mantissa information characters, not including separate leading sign or decimal point.

Example:

Convert a field with the description -9.99999E+99 to floating point. The input address is in index 3; the input SCP is 2. The output address is in index 2.

Operation	Operand	Index
SPB	ZNF	1
OCT	1720010	
OCT	0011205	

ZNF

2. Example using free form numeric:

Operation	Operand	Index
SPB	ZNF	1

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	B	D	O	L	S	C	P	ECP*			STS			NCI**					
DS-FS																			
OUTPUT ADDRESS, IF NEEDED																			

* only if trailing sign

** not including a separate trailing sign

Example:

Convert a field with a description 9999V99+ to floating point and store at FLOATY. The input address is in index 3, and the starting character position is 1.

Operation	Operand	Index
SPB	ZNF	1
OCT	1011106	
OCT	3777776	
LDA	FLOATY	

CONVENTIONS

Memory locations used: 142

SUBROUTINES REQUIRED: FXL, ZNB, ZSC

ZFR - Repeated Numeric and Numeric with E to Floating Point

PURPOSE

To convert repeated fields from BCD to floating point as in ZNF.

USAGE

Calling Sequence

1. Example using numeric with E.

Operation	Operand	Index
SPB	ZFR	1

Parameter words as in ZNF, followed by:

S	1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19
		SCP1		SCP3		SCP2	
		SEC1	NSW3	SEC3	NSW2	SEC2	NSW1
NUMBER OF FIELDS TO MOVE							

where SEC1, SEC2, SEC3 are the first, second, and third starting character positions of the exponent signs.

Example:

Convert 5 sequential fields of the description RV99999E-9 to floating point. The input starting character position is 3. The input address is in index 3 and the output address is IDELA.

Operation	Operand	Index
SPB	ZFR	1
OCT	1130006	
OCT	0013105	
LDA	IDELA	
OCT	0301020	
OCT	0321323	
DEC	5	

ZFR

2. Free form numeric.

Operation	Operand	Index
SPB	ZFR	1

Parameter words as in ZNF, followed by:

S · 1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19
	SCP1	ECP1*	SCP3	ECP3*	SCP2	ECP2*
		NSW3		NSW2		NSW1
NUMBER OF FIELDS TO MOVE						

* only if trailing sign - otherwise zeros

Example:

Convert 7 fields with a description of +99.9999 to floating point. The output address is in index 2, input address in index 3 and the starting character position is 1.

Operation	Operand	Index
SPB	ZFR	1
OCT	1710010	
OCT	3777774	
OCT	0102030	
OCT	030302	
DEC	7	

CONVENTIONS

Memory locations used: 75

SUBROUTINES REQUIRED: ZNF and its requirements.

ZNP

ZNP - BCD to Floating Point

PURPOSE

To convert BCD numeric with E, and free form BCD numeric to floating point, using the floating point package.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZNP	1

Parameters as in ZNF.

CONVENTIONS

Memory locations used: 146

SUBROUTINES REQUIRED: FLT, FXL, ZNB, ZSC.

ZPR

ZPR - Repeated BCD to Floating Point

PURPOSE

To convert repeated BCD fields to floating point as in ZNP.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZPR	1

Parameter words as in ZNF, followed by parameter words as in ZFR.

CONVENTIONS

Memory locations used: 75

SUBROUTINES REQUIRED: ZNP and its requirements.

ZBB

ZBB - Binary to Binary

PURPOSE

To move a one or two word binary number to a one or two word binary number with provision for changing field size, decimal scale, and binary scale.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZBB	1

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
		FS						DS						BS						
		FSOT						DSOT						BSOT						
		S	I	O		M	N	NUMBER OF FIELDS TO MOVE												
INPUT OR OUTPUT ADDRESS, IF NEEDED																				
		NSWOT						NSW												(only if S is 1)

where:

M - number of input words indicator. 0 = 1 word, 1 = 2 words.

N - number of output words indicator. 0 = 1 word, 1 = 2 words.

S - 1 if special non sequential repeats (requires the NSW word in the calling sequence).

Example:

Move a one word input binary field (address in index 3) to PROMP as two words. The input has a binary scale of 15 and a description of 999V9. The output scale is 21 with the description 999999V99.

Operation	Operand	Index
SPB	ZBB	1
OCT	0040317	
OCT	0100625	
OCT	0210001	
LDA	PROMP	

CONVENTIONS

Memory locations used: 84

SUBROUTINES REQUIRED: MTR

MTR

MTR - Truncate and/or Round Binary Fixed Point Numbers

PURPOSE

To truncate the integer portion and/or truncate or round the fraction portion of a fixed point binary number.

METHOD

MTR is entered with the fixed point binary number in registers A and Q.

MTR exits with the truncated and/or rounded fixed point binary results in registers A and Q.

The calling sequence to MTR contains the field size, decimal scale, and binary scale of both the input and desired output binary numbers.

1. Field size is the number of characters representing the maximum range of the number. Decimal point and sign are not present in the field size count.

2. Decimal scale is the number of characters representing the maximum value of the binary number's integer portion.

3. Binary scale is the scale at which the number is being expressed.

MTR adjusts the input number to the output number's representation with rounding of the fraction portion of the input number if desired.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	MTR	1
OCT	ABBCCDD	
OCT	OEEFFGGG	
Return from MTR		

where: A equals "1" if the fraction portion of the input number is to be rounded by MTR.
A equals "0" if no round is to be performed.
BB is input number's field size.
CC is input number's decimal scale.
DD is the input number's binary scale.
EE is output number's field size.
FF is output number's decimal scale.
GG is output number's binary scale.

MTR

If CC> FF, the integer portion of the input number is truncated to the size of the output number's decimal scale.

If BB-CC>EE-FF and A = 1, the fraction portion of the input number is rounded to the size of the output number's fraction portion.

If BB-CC>EE-FF and A = 0, the fraction portion of the input number is truncated to the size of the output number's fraction portion.

The output number is always expressed at the binary scale as represented by GG.

Examples:

1. INPUT 9999V99
   OUTPUT 999V99

Calling Sequence

Operation	Operand	Index
SPB	MTR	1
OCT	0060423	
OCT	0050323	

result: Input number is truncated 1 significant digit on integer portion.

2. INPUT 9999V9999
   OUTPUT 999V999

Calling Sequence

Operation	Operand	Index
SPB	MTR	1
OCT	0080423	
OCT	0060323	

result: Input number is truncated 1 significant digit on both the integer and fraction portions.

3. INPUT 9999V9999
   OUTPUT 999V999

MTR

Calling Sequence

Operation	Operand	Index
SPB	MTR	1
OCT	1080423	
OCT	0060323	

result: Input number is truncated 1 significant digit on the integer portion and rounded 1 significant digit on the fraction portion.

CONVENTIONS

Both input and output numbers are assumed contained in 2 computer words. Under this configuration, binary scale 19 indicates the integer portion lies in the 1st word and the fraction lies in the 2nd word. Adjustments from binary scale 19 for fixed point binary numbers indicate the position of the binary point within the 2 words.

Memory locations used: 186

SUBROUTINES REQUIRED: ZSC

ZAM

ZAM - BCD to BCD

PURPOSE

To move an alphanumeric BCD field with provision for increasing the field size. Blanks are added if the destination field is longer than the source field.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZAM	1

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		SCP									NCI < 83								
				I	O	SCPOT					NCO < 83								
INPUT OR OUTPUT ADDRESS, IF NEEDED																			

Example:

Move a 60 character field from SUNNY (starting character position 1) to a 60 character field whose starting character position is 2. The destination address is in index 2.

Operation	Operand	Index
SPB	ZAM	1
OCT	0100074	
OCT	0120074-	
LDA	SUNNY	

CONVENTIONS

Memory locations used: 80

SUBROUTINES REQUIRED: ZCB or ZOT.

ZZA

ZZA - Repeated BCD to BCD

PURPOSE

To move repeated alphanumeric fields as in ZAM. One or both fields packed.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZZA	1

Parameter words as in ZAM, followed by:

S 1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19
	SCP1	SCPOT1	SCP3	SCPOT3	SCP2	SCPOT2
	NSW3		NSW2		NSW1	
	NSWOT3		NSWOT2		NSWOT1	
NUMBER OF FIELDS TO MOVE						

Example:

Source field has a starting character position of 2 and is 5 characters long. Destination field is 7 characters with a starting character position of 1. Move 6 fields. The input and output addresses are in indexes 3 and 2.

Operation	Operand	Index
SPB	ZZA	1
OCT	0200005	
OCT	0310007	
OCT	0213312	
OCT	0020102	
OCT	0030202	
DEC	6	

CONVENTIONS

Memory locations used: 73

SUBROUTINES REQUIRED: ZAM, ZCB.

ZUR

ZUR - Repeated BCD to BCD

PURPOSE

To move repeated unpacked fields to fields of a different size.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZUR	1

Parameter words as in ZAM, followed by:

S 1 2 3 4 5 6 7	8 9 10 11 12 13	14 15 16 17 18 19
	NSW	NSWOT
NUMBER OF FIELDS TO MOVE		

Example:

Move 37 fields of 16 characters, unpacked to fields of 12 characters, unpacked. The source and destination addresses are in index registers 3 and 2.

Operation	Operand	Index
SPB	ZUR	1
OCT	0100020	
OCT	0310014	
OCT	0000504	
DEC	37	

CONVENTIONS

Memory locations used: 41

SUBROUTINES REQUIRED: ZAM, ZCB.

ZLX

ZLX - Repeated Move Floating Point Binary to Fixed Point Binary Numbers

PURPOSE

To move a list of floating point binary numbers "n" fields in length to a list of fixed point binary numbers "n" fields in length.

METHOD

Use is made of FLX GECOM library convert routine.

Each floating point binary field is converted to its fixed point binary equivalent.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZLX	1

S	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
	NUMBER OF FIELDS TO MOVE
	0 I O | NSWOT | M | OUTPUT BINARY SCALE
	INPUT OR OUTPUT ADDRESS, IF NEEDED

where: M - number of output words. 1 = 1 word; 0 = 2 words.

Example:

Move a list of floating point binary numbers 25 in length to a list of fixed point binary numbers 25 in length. Output numbers stored in 2 words. Output binary scale is 19. Destination address is in index register 2.

Operation	Operand	Index	
SPB	ZLX	1	
OCT	0000031		
OCT	0110023		
LDA	SOURCE		(Address of source or input list - 1st word).

CONVENTIONS

Output numbers must be 1 or 2 word fixed point binary stored consecutively.

Input numbers must be 2 word GE-225 floating point format numbers stored consecutively.

Output binary scale is the scale at which the output number is expressed.

Number of fields to move should be greater than 1. Use FLX if move is for only 1 field.

Memory locations used: 52

SUBROUTINES REQUIRED: FLX

ZXL - Repeated Move Fixed Point Binary to Floating Point Binary Numbers

PURPOSE

To move a list of fixed point binary numbers "n" fields in length to a list of floating point binary numbers "n" fields in length.

METHOD

Use is made of FXL GECOM library convert routine.

Each fixed point binary field is converted to its floating point binary equivalent.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZXL	1

S	1	2 3 4	5 6 7 8	9 10	11 12 13 14 15 16 17 18 19
	NUMBER OF FIELDS TO MOVE				
	0	I 0	NSW	M	INPUT BINARY SCALE
	INPUT OR OUTPUT ADDRESS, IF NEEDED				

where: M - number of input words. 1 = 1 word; 0 = 2 words.

Example:

Move a list of fixed point binary numbers 25 in length to a list of floating point binary numbers 25 in length. Input numbers stored in 2 words. Input binary scale is 19. Destination address is in index register 2.

Operation	Operand	Index
SPB	ZXL	1
OCT	0000031	
OCT	0110023	
LDA	SOURCE	(Address of source or input list - 1st word)

CONVENTIONS

Input numbers must be 1 or 2 fixed word point binary stored consecutively.

Output numbers must be 2 word GE-225 floating point format numbers stored consecutively.

Input binary scale is the scale at which the input number is expressed.

Number of fields to move should be greater than 1. Use FXL if move is for only 1 field.

Memory locations used: 52

SUBROUTINES REQUIRED: FXL

RPT

RPT - Repeated Move

PURPOSE

To move a source field n computer words in length to a destination field m computer words in length.

METHOD

The source field is moved into the destination field as many times as it will fit starting on the left (high order) end of the destination field. The source field is truncated if necessary on the right (low order) end for the last placement in the destination field.

The source and destination are considered an integral number of computer words in length.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	RPT	1
LDA	SOURCE	A
DEC	XXXXX	
STA	DESTINATION	B
DEC	YYYYY	

where: SOURCE is the symbol assigned to the source field.

DESTINATION is the symbol assigned to the destination field.

XXXXX is the size of the source field in number of computer words.

YYYYY is the size of the destination field in number of computer words.

A is the index register modifying the source address. May be 2 or 3. If not used index should be blank.

B is the index register modifying the destination address. May be 2 or 3. If not used index should be blank.

CONVENTIONS

Memory locations used: 24 words

ZSS or ZSF

ZSS or ZSF - Fields Separated By Commas

PURPOSE

To unpack fields separated by commas.

USAGE

Generated Coding

Symbol	Operation	Operand	Index	Remarks
FNZRN	SPB	ZSS or ZSF	1	(ZSS if package floating point; ZSF if hardware floating point)
	DEC	STARTING COLUMN-1		For all but the first card of multiple card records, skipping over any control key
	DEC	STARTING COLUMN-1		
	LDA	FNTXT		
	BRU	FNW or BRU FNW	1	
	LDA	FNTCP		
	[Segments of calling sequence, by field, as described below.]			
	BRU	FNTXT		

1. NUMERIC TO FIXED POINT BINARY

2 3 4	5 6 7	8 9 10 11 12 13	14	15 16 17 18 19
0 0 1	RI	BS-38 (ABSOLUTE)	S	DS-FS
OUTPUT ADDRESS				
NUMBER OF REPEATS, IF NEEDED				

2. NUMERIC TO FLOATING POINT

2 3 4	5 6 7	8 9 10 11 12 13	14	15 16 17 18 19
0 1 0	RI		S	DS-FS
OUTPUT ADDRESS				
NUMBER OF REPEATS, IF NEEDED				

ZSS
or
ZSF

3. ALPHA TO ALPHA

2 3 4	5 6 7	8 9 10	11 12 13	14 15 16 17 18 19
0 1 1	RI		NXC	NO. STORAGE WDS.
OUTPUT ADDRESS				
NUMBER OF REPEATS, IF NEEDED				

RI - Repeat indicator. 000 - no repeats; 001 - repeats

S - Sign of DS-FS. 0 = plus; 1 = minus

NXC - Number of characters in last storage word. 000 = 3, 001 = 1, 010 = 2.

CONVENTIONS

Memory locations used: ZSS - 412
ZSF - 410

MTF - Move to True-False Variable

PURPOSE

To move a field or Array to a True-False variable or True-False Array.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	MTF	1

S 1	2 3 4	5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
	I	*NW NO. REPEATS IN INPUT FIELD
	O	*NWOT NO. REPEATS IN OUTPUT FIELD
LDA		ADDRESS, IF NEEDED

*NW = No. words in Source Field

*NWOT = No. words in Destination Field

Example:

Move an Array with 2 word fields repeated 10 times to a True-False Array repeated 10 times. Input address is in XR 3. Output address is in XR 2.

Operation	Operand	Index
SPB	MTF	1
OCT	0120012	
OCT	00110012	
LDA	0	

CONVENTIONS

Memory locations used: 71.

ZFL

ZFL - Move and Fill

PURPOSE

To fill a given number of words (partial or integral) with a given number of words (partial or integral).

USAGE

Calling Sequence

Operation	Operand	Index
SPB	ZFL	1

S 1	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19					
	#WORDS IN INPUT					
	*I/O	SCP	ECP		SCPOT	ECPOT
	NO. REPEATS IN OUTPUT					
	# WORDS IN OUTPUT					
LDA	ADDRESS IF NEEDED					

*I/O = 1 If input address in calling sequence.
= 2 If output address in calling sequence.
= 3 If neither input nor output in calling sequence.

Example:

Move a 4 word field containing 9 characters, SCP = 3, ECP = 1 to 3 word field SCP = 1 ECP = 3, repeated 5 times. Input address is in XR 3. Output address is in XR 2.

Operation	Operand	Index
SPB	ZFL	1
OCT	0000004	
OCT	0331013	
OCT	0000005	
OCT	0000003	
LDA	0	

CONVENTIONS

Memory locations used: 114.

PROCEDURAL SUBROUTINES

Procedural Subroutines are directly related to the GECOM verbs and their options.

List of Procedural Subroutines in Order of Appearance in Manual

RCS	Read Control Switches
RC*	Read Control Switches (API environment)
RLC	Read Next Runs' Loader and Transfer Control to Zero
RL*	Read Next Runs' Loader and Transfer Control to Zero (API environment)
TYP	Type
TY*	Type (API environment)

RCS
or
RC*

RCS - Read Control Switches

RC* - Read Control Switches (API Environment)

PURPOSE

To read the console control switches and save the switch settings after the operator has placed the switches in normal position.

METHOD

Upon entering RCS or RC*, the computer halts in a read control switch loop.

When switch zero is put down, the setting of the switch is saved.

When switch zero is put back in normal position, exit from RCS with the switch settings placed in register A is triggered. Following Programming Conventions, put switches 1 to 19 in normal position before putting switch zero in normal position.

Upon exiting from RCS or RC*, register A contains the settings of switches 1 to 19.

Bit 1 of Register A is on, if switch one was depressed for the read loop. Bit 2 is on, if switch 2 was depressed, etc.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	RCS or RC*	1

CONVENTIONS

Memory locations used: RCS - 12; RC* - 14

Return from RCS or RC* is made to the instruction following the SPB.

RLC or RL*

RLC - Read Next Runs' Loader from the Card Reader and Transfer Control to Zero.

RL* - Read Next Runs' Loader from the Card Reader and Transfer Control to Location Zero. (API environment)

PURPOSE

To load sequential runs into the card reader simultaneously and to automatically call in the next run under program control.

METHOD

Upon entrance to RLC or RL*, the computer enters a control switch loop. The operator toggles switch zero to read in the next run if it is present in the card hopper.

The next binary card present in the hopper must be a loader.

RLC or RL* reads in the loader card performing error checks. If an error occurs, the operator must backspace the card reader and toggle switch zero to attempt a reread of the loader.

Upon a successful read, RLC or RL* transfers control to location zero.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	RLC or RL*	1

No return is made to the instructions following the "SPB"

CONVENTIONS

Memory locations used - 23 words.

"CRE" is typed if an error occurs when reading the loader.

SUBROUTINES REQUIRED: RLC requires RCS, TYP
RL* requires RC*, TY*

TYP or TY*

TYP - Type

TY* - Type when in API environment

PURPOSE

To type one word or a list of words with or without a carriage return.

USAGE

Calling Sequences

The routine has two entrances, TYP or TY* and TYA. TYA does the actual typing and TYP or TY*. feeds the information to TYA one word at a time. TYA is entered with the 3 BCD characters to be typed in register A.

1. Calling sequence for TYA:

Operation	Operand	Index	Remarks
SPB	TYA	3	Register A contains 3 BCD characters to be typed.

2. Calling sequence for TYP or TY*:

Operation	Operand	Index
SPB	TYP or TY*	3

The second word in the calling sequence indicates the following:

Sign Bit On - Type a list of words. The address portion specifies the number of words to type. The third word required in the calling sequence gives the beginning address of the list.

Sign Bit Off - Type the 3 BCD characters in position 2-19 of this word.

Position one
Bit On - No carriage return after typing.

Position one
Bit Off - Carriage return after typing.

Examples:

	Operation	Operand	Index	Remarks
1.	SPB	TYP or TY*	3	
	ALF	ERR		Type one word (ERR) and return the carriage.
2.	SPB	TYP or TY*	3	
	OCT	1255151		Type one word and do not return the carriage.
3.	SPB	TYP or TY*	3	Type 6 words starting at "LIST" and return the carriage.
	OCT	2000006		
	LDA	LIST		
4.	SPB	TYP or TY*	3	Type 6 words starting at "LIST" and do not return the carriage.
	OCT	3000006		
	LDA	LIST		

CONVENTIONS

Memory locations used: TYP - 54; TY* - 60.

RLT

RLT - Read Sequential Run Locator Transfer Control to it.

PURPOSE

To allow sequential runs to be called off a program tape under program control.

METHOD

Upon entering RLT, the computer halts in a control switch loop. The operator toggles switch zero to continue with the next run.

RLT searches the specified program tape for a sequential run locator, which is identified by the constant "RUN 00 LINKAGE" in the first four words of the physical record. RLT searches only forward.

Conventional error checks are performed to insure a properly read record. If an error occurs, RLT tries five times to reread the record. If unsuccessful after five trys, the computer is stopped in a control switch loop. If switch zero is toggled, RLT will attempt to read the record again.

On a successful read, the sequential run locator is read into location zero and RLT transfers control to location five.

USAGE

Calling Sequence

Operation	Operand	Index
SPB	RLT	1
OCT	OAABBCC	

where: AA is the octal tape number for GE-225 instructions.

BB	AA
00	01
01	02
02	04
03	10
04	21
05	22
06	24
07	30

BB is the handler the program tape is mounted on. (high order zero)

CC is the plug number the program tape handler is attached to (high order zero).

CONVENTIONS

Memory locations used - 90 words.

All typewriter messages noting tape read errors follow the standards specified under Programming Conventions.

SUBROUTINES REQUIRED: RCS, TYP

TYP or TY*

TYP - Type

TY* - Type when in API environment

PURPOSE

To type one word or a list of words with or without a carriage return.

USACE

Calling Sequences

The routine has two entrances, TYP or TY* and TYA. TYA does the actual typing and TYP or TY* feeds the information to TYA one word at a time. TYA is entered with the 3 BCD characters to be typed in register A.

1. Calling sequence for TYA:

Operation	Operand	Index	Remarks
SPB	TYA	3	Register A contains 3 BCD characters to be typed.

2. Calling sequence for TYP or TY*:

Operation	Operand	Index
SPB	TYP or TY*	3

The second word in the calling sequence indicates the following:

Sign Bit On - Type a list of words. The address portion specifies the number of words to type. The third word required in the calling sequence gives the beginning address of the list.

Sign Bit Off - Type the 3 BCD characters in position 2-19 of this word.

Position one
Bit On - No carriage return after typing.

Position one
Bit Off - Carriage return after typing.

Examples:

	Operation	Operand	Index	Remarks
1.	SPB ALF	TYP or TY* ERR	3	Type one word (ERR) and return the carriage.
2.	SPB OCT	TYP or TY* 1255151	3	Type one word and do not return the carriage.
3.	SPB OCT LDA	TYP or TY* 2000006 LIST	3	Type 6 words starting at "LIST" and return the carriage.
4.	SPB OCT LDA	TYP or TY* 3000006 LIST	3	Type 6 words starting at "LIST" and do not return the carriage.

CONVENTIONS

Memory locations used: TYP - 54; TY* - 60.