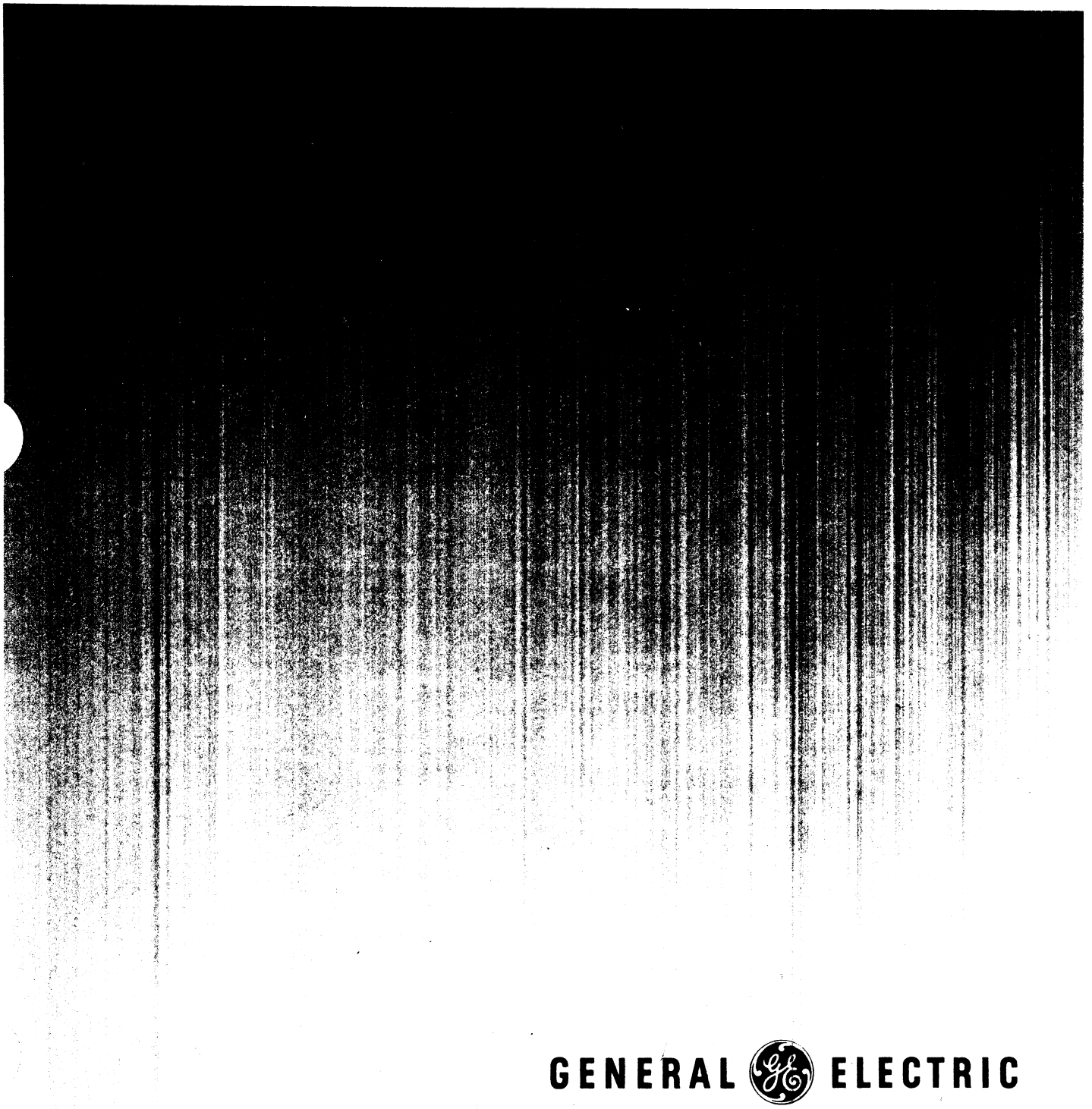
GENERAL ELECTRIC
COMPUTERS

GE-200 Series GECOM-II Reference Manual



GENERAL ELECTRIC

CPB – 1108

GE-200 SERIES
GECOM-II

COBOL COMPATIBLE

REFERENCE MANUAL

Program Number
CD225H1.005

September 1965

GENERAL ELECTRIC
COMPUTER DEPARTMENT

PREFACE

This Reference Manual (together with the GE-200 Series GECOM-II Operations Manual, CPB-1109) covers the COBOL Compatible version of GECOM-II, Program No. CD225H1.005. This manual was previously issued with the program number serving also as the publication identification number. The current edition, which supersedes all earlier editions, is identified with the new publication number, CPB-1108, on the cover. The program number remains unchanged.

Comments on this publication may be addressed to Technical Publications, Computer Department, General Electric Company, P. O. Box 2961, Phoenix, Arizona, 85002.

CONTENTS

APPENDICES

ILLUSTRATIONS

1. INTRODUCTION

This manual is intended to familiarize the programmer, who has a working knowledge of compilers, with the language of the GECOM-II system. Two supporting manuals about the GECOM system are available: Introduction to GECOM, CPB-230, which provides a broad description of the GECOM system, and the GECOM-II Operations Manual, CPB-1109, which covers the operating instructions for the compiler.

The GECOM system translates a source language into a machine language program. GECOM is composed of two elements: the source language or the language in which the program is written, and the compiler which translates the source language into an object program ready for execution on the GE-200 Series. This manual is primarily concerned with the source language. However, mention of the compiler is necessary in certain cases because, to a large extent, the specifications of the language determine the broad area of the compiler.

GECOM source language is based primarily on COBOL (Common Business Oriented Language) and ALGOL (International Algorithmic Language). Of these two, COBOL was selected as a base language because it achieves a "nearly natural" language which satisfies the needs of a broad spectrum of data processing applications. Boolean expressions, floating point arithmetic, and the ability to express complex equations were taken from ALGOL and incorporated into the language structure of COBOL to accommodate the needs of more technical applications. Therefore, with the present version of GECOM, the programmer may state his problem in one, two, or a combination of two, languages.

In concept, a COBOL source program facilitates the statement of a problem for computer solution. Recognizing that there are four levels of program preparation, the GECOM source program is correspondingly divided into four parts, or divisions. Each division represents a separate and independent level of program preparation. For example:

Program Preparation		GECOM Division
1. Identification of program	=	Identification
2. Description of computer configuration	=	Environment
3. Description of data	=	Data
4. Steps or set of procedures	=	Procedure

The Identification Division labels the source program. The programmer may include the date written, the program title, his own name, and any other information necessary for computer program documentation.

The Environment Division indicates the equipment being used to run the object program. Among the many items that may be mentioned for a particular computer are: memory size, number of tape units, hardware switches, printers, etc. Also described here are those aspects of a file relating directly to hardware. Because this division deals entirely with the specifications of the computer being used, it is largely computer-dependent.

The Data Division uses file and record descriptions to describe the files of data that the object program is to manipulate or create, together with the individual logical records which comprise these files. The characteristics or properties of the data are described in relation to a standard data format rather than an equipment-oriented format. Therefore, this division is to a large extent computer-independent.

The Procedure Division indicates the steps that the programmer wishes the computer to follow. These steps are expressed in terms of meaningful English words and sentences. This aspect of the overall system is often referred to as the "program"; in reality it is only part of the total specification of the problem solution (the program), and is insufficient, by itself, to describe the entire problem. This is true because repeated references must be made--either explicitly or implicitly--to information appearing in the other divisions. Concepts of verbs to denote action, and sentences to describe procedures, are basic, as is the use of conditional statements to provide alternative paths of action.

MACHINE REQUIREMENTS

The source computer configuration for GECOM is as follows:

1. GE-200 Series central processor with 8192-word memory and typewriter
2. Card reader
3. Card punch
4. On-line high-speed printer
5. Four to six tape units, depending on needs of the user
6. One to six tape controllers

Object computer configuration--The GECOM-II compiler will compile an object program for any standard GE-200 Series configuration.

ORGANIZATION OF MANUAL

This manual is organized to give a logical presentation of GECOM II. The four Divisions are discussed in the order in which they may be most meaningful to the reader. The reference guide below outlines the contents of each chapter.

1. INTRODUCTION
2. GLOSSARY--defines certain words as to their usage in the GECOM System.
3. LANGUAGE STRUCTURE--describes how words are formed in GECOM and how they may be combined to form sentences.
4. USING THE GENERAL COMPILER FORMS--tells how to enter letters or symbols on the GECOM forms.
5. DATA DIVISION--explains how to describe the input/output information which is contained in files and used in the processing of a problem.

6. PROCEDURE DIVISION--describes the verbs used to carry out the procedures in a given problem.

7. ENVIRONMENT DIVISION--describes the functions centralizing the aspects of a problem which are dependent upon the physical characteristics of the GE-200 Series.

8. IDENTIFICATION DIVISION--shows how to label the source program.

9. DATA MANIPULATION--gives the programmer a better understanding of his object program so he may write a more efficient data description in his source program.

10. THE USE OF GECOM TO OBTAIN EFFICIENT OBJECT PROGRAMS--gives rules and techniques independent of the compiler itself which should be followed to produce efficient programs.

ACKNOWLEDGMENT

"This publication is based 'in part' on the COBOL System developed in 1959 by a committee composed of government users and computer manufacturers. The organizations participating in the original development were:

Air Materiel Command, United States Air Force
Bureau of Standards, Department of Commerce
David Taylor Model Basin, Bureau of Ships, U. S. Navy
Electronic Data Processing Division, Minneapolis-Honeywell Regulator Company
Burroughs Corporation
International Business Machines Corporation
Radio Corporation of America
Sylvania Electric Products, Inc.
Univac Division of Sperry-Rand Corporation

"In addition to the organizations listed above, the following other organizations participated in the work of the Maintenance Group:

Allstate Insurance Company
Bendix Corporation, Computer Division
Control Data Corporation
DuPont Corporation
General Electric Company
General Motors Corporation
Lockheed Aircraft Corporation
National Cash Register Company
Philco Corporation
Standard Oil Company (N. J.)
United States Steel Corporation

"This COBOL-61 manual is the result of contributions made by all of the above-mentioned organizations. No warranty, expressed or impiled, is made by any contributor or by the committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

"It is reasonable to assume that a number of improvements and additions will be made to COBOL. Every effort will be made to insure that the improvements and corrections will be made in an orderly fashion, with due recognition of existing users' investments in programming. However, this protection can be positively assured only by individual implementors.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures and the methods for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein: FLOW-MATIC (Trade-mark of Sperry-Rand Corporation), Programming for the UNIVAC (R) I and II, Data Automation Systems © 1958, 1959, Sperry-Rand Corporation; IBM Commercial Translator, Form No. F 28-8013, copyrighted 1959 by IBM, FACT, DSI 27A5260-2760, sopyrighted 1960 by Minneapolis-Honeywell, have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

"Any organization interested in reproducing the COBOL report and initial specifications in whole or in part, using ideas taken from this report or utilizing this report as the basis for an instruction manual or any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention 'COBOL' in acknowledgment of the source, but need not quote this entire section."

2. GLOSSARY

The definitions in this glossary pertain to the usage of these words in this manual.

Array	A list of values.
Array Name	Name representing all values in an array.
BCD	Binary Coded Decimal--a system of representing any character of the character set of the computer by a group of binary digits.
Beginning-File Label	A label block which identifies the contents of each file of a multifile tape. It is block 0, the first block of the file.
Binary Numeric (external)	Any numeric that exists in one- or two-word binary form.
Binary Numeric (internal)	Any numeric used in an arithmetic operation or an IF statement.
Block	A group of logical records read from, or written on, tape as one physical tape record.
Block Size	The number of words in a block.
Buffer	Storage locations (set aside in memory) used to compensate for a difference in rate of flow of information when transmitting information from one device to another.
Common Storage	Memory allocated for data required for processing during execution of more than one segment of a program.
*Common Storage	Upper 8k memory allocated to repeated numeric fields.
Element	A subordinate section of a BCD field. May be of any size not to exceed field length.
End of File	The point following reading or writing of the last physical data record of a file.
End-of-file Label	A unique set of characters that follows every end-of-file mark.
End-of-tape Label	A unique set of characters that follows the mark of every intermediate reel (all but the last) of a multireel file.
Expression (arithmetic)	A sequence of variables, numbers and/or mathematical functions connected by symbols of arithmetic operations.

Expressions (logical)	A combination of conditional names, relational expressions and arithmetic expressions connected by the logical AND, OR (Inclusive), and NOT (Exclusive).
Expressions (relational)	Any expressed or implied comparison of two field names, element names, literals, or arithmetic expressions.
Field	Units of data.
Field Literal	A literal used only for input, Working Storage, and Common Storage.
Figurative Constant	A name representing specific values.
File	A set of logical records.
Fixed Point	A number including an actual or assumed decimal point either between digits or following them (1.23, 123. or 123.0).
Floating Point	A number expressed as a whole number and fraction, and a power of ten ($1.287*10^{-2}$).
Generated Field	A field which is generated as a result of calculations and is not input to the program.
Group (of fields)	A named set of data similar to a record but beneath it in rank.
*Group (of fields)	Equivalent to a logical record for input/output purposes.
Integer (as used in this manual)	Indicates a number of not more than five digits which does not contain an actual decimal point.
Key Words	A vocabulary which has special meaning for GEneral COMpiler and, therefore, should not be used as data names by the programmer.
Literal	A string of characters forming a constant made up from the character set of the GE-200 Series.
Logical Record	Any consecutive set of related information within a physical record.
Multifile Tape	A tape containing more than one file.
Multireel File	A file that extends over more than one tape reel.
Multitape File	Same as multireel file.
Nonstandard Data	Data not conforming to GECOM internal binary scaling.
Numeric Constant	May consist of numerals 0-9; the plus sign (+), the minus sign (-) and the letter E, delineating the exponent in floating point.
Object Program	A program in machine language (output from the compiler).

Packed Data (BCD)	Data entered into the computer without regard to the GE-200 Series word length into which it will be placed.
Physical Tape Record	Information contained between successive tape gaps.
Procedure	An action which the programmer desires the computer to carry out.
Qualifier	Data names used in conjunction with other nonunique data names to make them unique.
Record	A logical record.
Record Size	The number of words in a record.
Section	Ordered sets of sentences having a common function.
Segment	Subprograms which are compiled and tested independently. Two or more are subsequently loaded together and executed as a total program.
Sentence	Describes a computer procedure to be followed.
Source Program	The English language program written for the General Compiler (GECOM).
Subscript	Method of identifying or selecting a particular value in an array of values.
Tape Mark	A special character (001111) that signifies either end of file, or end of tape, depending upon the label block following it.
Throughput Fields	Fields which are not referenced or operated on in the Procedure Division but are moved from input to output.
Truncation	The dropping of either least-significant or most-significant characters. This occurs when forcing a field to conform to the receiving image.
Unpacked Data (BCD)	Data so arranged that it may be read into integral GE-200 Series word lengths.
Word (as applied to computers)	A set of characters which is moved as a unit by the computer. A word may be data or instructions.
Working Storage	That part of computer memory set aside by the programmer for intermediate processing of data.
Zero Suppression	Special editing performed only on numeric fields when leading characters become zero.

3. LANGUAGE STRUCTURE

The GECOM language, like most languages, is a body of words with a set of conventions for combining these words to express meanings. Its structure or syntax closely resembles English grammar, and its body of words may be appropriately termed a vocabulary. This section shows how words are formed and how they may be combined to express a computational process.

CHARACTERS

The basic units of the language are the characters used to form words and symbols. The GECOM character set includes:

Alphabetics	A, B, C, ..., Z
Numerals	0, 1, 2, ..., 9

and the special characters shown in Figure 1. Special characters are presented in more detail as their use is encountered.

Character	Meaning	Hollerith	GE-200 HSP
Δ	Space or Blank	Space	Space
.	Period and Decimal Point	12-3-8	.
,	Comma	0-3-8	,
"	Quotation Mark	3-8	#
-	Hyphen	5-8	_ (underscore)
(	Left Parenthesis	0-5-8	(
)	Right Parenthesis	0-6-8	)
+	Addition and Plus Sign	12	+
-	Subtraction and Minus Sign	11	-
*	Multiplication	11-4-8	*
/	Division	0-1	/
=	Assignment	6-8	=
	Decision Table Column Delimiter	12-4-8	Space

Figure 1. Special GECOM Characters

WORDS

Words fall into one of two vocabulary categories:

1. Words used by the compiler
2. Words used by the programmer

The programmer's vocabulary consists mostly of arbitrary names given to his data.

The compiler's vocabulary, on the other hand, is predetermined and is used only to form sentences and descriptive phrases.

These two categories of words are illustrated by the following sentence:

GET~RECORD. READ MASTER~FILE RECORD.

Here, the words READ and RECORD belong to the vocabulary of the compiler. The words GET~RECORD and MASTER~FILE belong to that of the programmer since he has freedom to choose names of sentences and data files.

Appendix A lists the compiler vocabulary; the programmer should avoid using these words when choosing names of sentences and data.

DATA NAMES

Data names are words representing data (files, records, fields, constants, etc.) and are arbitrarily assigned by the programmer. They are formed from the following characters:

Letters	A, B, C, ..., Z
Numerals	0, 1, 2, ..., 9
Hyphen	~

The programmer should choose data names that

1. Do not exceed 12 characters.
2. Do not begin or end with a hyphen.
3. Do not contain imbedded spaces.
4. Do contain at least one letter.
5. Do not consist of all numerals or contain the letter E, since the letter E is used to to indicate the exponent in floating point notation.

Since data names represent or stand for data, they must be defined in the Data Division. It is here that the physical characteristics of data are completely described.

EXAMPLES:

A276B
27AB6
SIGMA
GROSS~PAY

PROCEDURE NAMES

In addition to data names, the programmer may name sentences and sections of sentences. These names or words are called procedure names. Procedure names are formed from the character set using the same rules used to form data names. However, unlike data names, procedure names may be wholly composed of the numerals. A procedure name consisting only of numerals does not have numeric value, for example, 26 and 026 are not the same procedure name. In this case, leading zeros are significant, and a part of the name.

EXAMPLES:

S~44
SENTENCE~44
A26
ABC
26

CONSTANTS

Values associated with data names generally change during the actual running of the compiled program. It is for this reason that they are sometimes called variables. A constant, as opposed to a variable, is a specified value and does not change within the scope of a program. A constant may be one of three kinds: literal, numeric, or figurative.

A literal constant is a string of characters made up from the character set of the GE-200 Series. Literals must be enclosed in quotation marks to set them apart from data names and other words of the source language. All spaces within a literal are interpreted as part of the literal.

Literal constants do not have numeric value and cannot be used in arithmetic calculations.

When a literal is used in the Procedure Division, it must not contain more than 30 characters or include an imbedded quotation mark. Literals described in the Data Division may be 120 characters long.

Numeric constants may be written as:

Integers	230
Fixed Point	230.1, 0.08
Floating Point	2.301E+2

A numeric may consist of the numerals 0-9, the plus sign (+), the minus sign (-), the decimal point(.), and the letter E, which in the floating point notation delimits the exponent.

Excluding the plus and minus signs and decimal point, fixed point numbers must not exceed 11 digits, and integers must not exceed 5 digits.

A numeric written in the floating point notation may be used only in floating point computation. Its exponent must not exceed ± 75 in value and its mantissa may consist of 9 or fewer digits, one of which is to the left of the decimal point. The plus or minus sign, the decimal point, and the delimitor E are not considered digits.

Numerics enclosed in quotation marks lose numeric value and are treated as literal constants.

Figurative constants are special names which represent specific values. They are:

ZERO(S)	ONE(S)	FOUR(S)	SEVEN(S)
ZEROES	TWO(S)	FIVE(S)	EIGHT(S)
SPACES	THREE(S)	SIX(ES)	NINE(S)

and their values are 0, Δ, 1, 2, ..., 9. The actual value of a figurative constant depends on the manner in which it is used in the Procedure Division. Their plurals do not represent more than one character. Rules governing their use are given in Chapter 9, "Data Manipulation."

The characters designated by Hollerith 2 - 8, 0 - 2 - 8, and 0 - 7 - 8 (octal 12, 72, and 77 respectively) are prohibitive in literals due to their special use by the compiler in the scan of source language statements.

CONDITIONAL NAMES

Conditional names are names assigned to each possible value of a numeric or alphanumeric field or element. For example, an employee's type of pay may be represented on a punched card as 1 if the employee is salaried, or 0 if the employee is paid hourly.

If the programmer wishes to use the PAY field as a conditional field, he would describe it in the Data Division as follows:

Type	Data Name	Data Image
F	PAY	9
C	SALARIED	1
C	HOURLY	0

To the compiler this means the data name PAY is a field (Type Code - F) and it is numeric and consists only of a single character (Data Image - 9). The data names SALARIED and HOURLY are conditional names of the PAY field (Type Code - C) and stand for the values 1 and 0 listed under Data Image.

The programmer is now free to use the conditional names SALARIED and HOURLY in procedure sentences. For example, the sentence

IF SALARIED GO TO ....

instructs the compiler to provide the coding for testing the PAY field to determine if it equals the numeral 1. If the PAY field does contain a 1, the GO TO path will be followed. If it does not,

the sentence after the IF sentence will be executed next. The same effect can be accomplished by

IF PAY EQUALS 1 GO TO ....

Conditional names in reality are a convenient means for stating relational expressions (see Expressions) and may be used only in sentences which permit the use of relational expressions.

Conditional names must conform to the rules governing the formation of data names.

TRUE-FALSE FIELDS

There is a class of variables which, either through usage or definition, may assume only the numerals 1 or 0. The value 1 is said to be their true state and the value 0 their false state. The words END FILE of the READ sentence (see Chapter 6 "Procedure Division") is such a variable. When the OPEN sentence is executed, the END FILE clause is set to its false state and remains set until the file's end-file condition is encountered. When the end file is encountered, the END FILE clause is then set to its true state.

The programmer is now free to interrogate the state of the END FILE clause by

IF END FILE OF MASTER GO TO ....

Variables having truth values are termed True-False variables. The END FILE variable is a convenience provided by the compiler; the programmer may also formulate his own true-false variables by merely listing them under the heading True-False Section in the Data Division. These variables may be named according to the rules given for data names and used only in conditional expressions and assignment.

True-false variables may not be mixed with floating point variables on the right side of an expression.

QUALIFIERS

Qualifiers are data names used in conjunction with other nonunique data names to make them unique. Every name in a source program must be unique. Either the name itself is unique, or the name exists within a hierarchy of names, such that the name can be made unique by mentioning one or more names in the hierarchy. When used in this way, the higher names are called qualifiers, and the process is called qualification. With each use of a name, enough qualification should be mentioned to make the use unambiguous, but it is not necessary to mention all possible levels of qualification unless they are needed for uniqueness. Note, however, that a name--except a record or a star group (*G.) name--which is unique within all sections except the output files need not be qualified in the output files. This absence of qualification is permitted in the Procedure Division and in the output record description, because output data other than records are not referenced in the Procedure Division. For faster compilations, it is recommended that the minimum necessary amount of qualification be used. A file name is the highest level qualifier available for a data name, but need not be used if a qualifier at a lower level is sufficient for uniqueness.

In order to minimize the creation of otherwise unnecessary qualifiers, the following abbreviations may be used to qualify data assigned to storage as indicated below:

Abbreviation	Storage Assignment
CONSTANT	Constant Section
CS	Common Storage or *Common Storage Section
WS	Working Storage Section

Note that the words CONSTANT~SECTION, COMMON~STORAGE, *COMMON~STORAGE and WORKING~STORAGE may not be used as qualifiers. Four basic rules should be used for qualification:

1. A qualifier should exist outside (above) the same it is qualifying. It should be preceded by the word OF in the Procedure Division.

2. A name may not appear at two levels in a hierarchy so that it would appear to qualify itself.

3. If a data or condition name appears more than once in the Data Division of a program, it must be qualified in all references occurring in the Procedure Division (except as noted above).

4. All file names must be unique.

EXAMPLES:

Two Input files might be described as:

```
FD   FILE~A
 R   RECORD~A
 F   TAX
 F   EARNINGS


...
FD   FILE~B
 R   RECORD~B
 F   TAX
 F   PAY
```

If a reference is made to the name TAX in the Procedure Division, TAX must be qualified. If the user wishes to specify the TAX field in FILE-A, the qualification may be any of the following forms:

TAX OF RECORD~A ...

TAX OF FILE~A ...

TAX OF RECORD~A OF FILE~A ...

Tax of FILE-B would be qualified in a similar manner. Since the file name is the highest level of qualification, and file names are unique names, it becomes convenient to use file names as qualifiers in order to avoid possible errors.

ARRAYS

A list of values x_1, x_2, x_3, ... x_{10} may be given a name, for example, X. This name is the array name and represents all values in the array. The values are consecutive and each value may be referenced by a subscript; in the case of x, a subscript ranging from 1 through 10. The array name is a field name which has an entry in the repeat columns.

The physical description of the values of X might appear in the File Section of the Data Division as:

Type	Data Name	Repeat	Data Image
F	X	10	999V99

To reference a value of X, the list is described as

x_i (which would appear as X(I) on the Sentence Form)

where:

x = name given to the 10 values of the list.

i = subscript name denoting the relative position of each value of the list.

If i = 1 and the value x_i is referenced, the first value of the list is used. If i = 7 when x_i is referenced, the seventh value of the list is used. The above array of values may be thought of either as a list or as an array with a single subscript. An array may be multidimensional; it may have more than one subscript.

A list of values, such as the following

$A_{1,1}$ $A_{1,2}$ $A_{1,3}$ $A_{1,4}$

$A_{2,1}$ $A_{2,2}$ $A_{2,3}$ $A_{2,4}$

$A_{3,1}$ $A_{3,2}$ $A_{3,3}$ $A_{3,4}$

may be given a name; for example, A. In this case the array would be referenced as

$A_{i,j}$

where:

A = name given to the 12 values of the array.

i = subscript name denoting the row of the array.

j = subscript name denoting the column of the array.

If i = 1 and j = 4, and the value $A_{i,j}$ is referenced, the fourth value in the first row will be used ($A_{1,4}$). If i = 3 and j = 2, the second value in the third row will be used ($A_{3,2}$).

In no case may an array extend beyond three dimensions; that is, it may not have more than three subscripts. Array names, when defined in the array Section as one, two, or three dimensional, must be referenced with one, two, or three subscripts, respectively. The exceptions to this appear in the MOVE and EXCHANGE verbs explained in the Procedure Division.

A three-dimensional array of three rows, four columns, and three planes (3, 4, 3) may be thought of as the above two dimensional array repeated three times. See Figure 2.

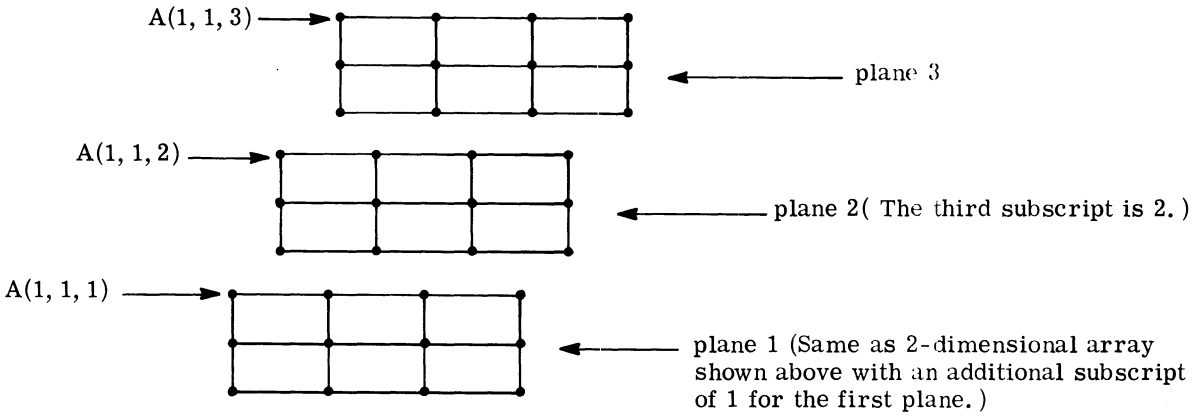


Figure 2. A Three-Dimensional Array

Here, the data name A is an array name and fields of the array would be referenced as $A_{I,J,K}$

All two- and three-dimensional arrays must be listed in the Array Section as:

ARRAY SECTION. A(3, 4) for the two dimensional array, or
ARRAY SECTION. A(3, 4, 3) for the three dimensional array.

There are two types of arrays allowable in GECOM.

Homogeneous Arrays

Homogeneous arrays are lists of data as described above. The list consists of a finite number of items with identical data descriptions. The items are stored consecutively in the File Section, Working Storage, Common Storage and/or *Common Storage.

Nonhomogeneous Arrays

Nonhomogeneous arrays are lists of data described in a nonhomogeneous, or repeated grouping, manner. In this case, a number of data items (numeric and/or alphanumeric -- in any mix) are described as a group. The group is then repeated a finite number of times. An example of this array as a repeated group is shown on the following page.

EXAMPLE:

Type	Data Name	Format	Repeat	Data Image
G	INVOICE	U	100	
F	NUMBER			XXXXX
F	QTY			999
F	UNIT~PRICE			999V99

This repeated group may be pictured as:

$NUMBER_1$	QTY_1	$UNIT\sim PRICE_1$
$NUMBER_2$	QTY_2	$UNIT\sim PRICE_2$
.	.	.
.	.	.
.	.	.
$NUMBER_{100}$	QTY_{100}	$UNIT\sim PRICE_{100}$

For use in procedure sentences, the object program processes the repeated group as a set of homogeneous one-dimensional arrays: NUMBER, QTY, UNIT~PRICE, each consisting of a list of 100 values.

The reference to a field within the nonhomogeneous group is a reference to the field-name qualified by the subscripted group-name.

In the above example, the quantity of the fifth listed invoice would be obtained by referring to

QTY of INVOICE (5) ...

If a group is repeated, none of the fields within a group may be repeated. (See Chapter 9, "Data Manipulation" for a more detailed discussion on the use of repeated groups.)

SUBSCRIPTS

Subscripts are a method of identifying or selecting a particular value in an array of values. To subscript an array, the array must be defined in the Array Section of the Data Division except for a field in a one-dimensional array with fixed point computation mode. The values must be described as having been repeated in the File, Working Storage, Common Storage, or *Common Storage Sections.

Subscripts may be written as arithmetic expressions containing other subscripted arrays. They may be nested to a maximum depth of 10 in any one sentence.

The mode of a set of subscripts (within parentheses) must remain constant, except that fixed point numbers and integers may be mixed.

EXAMPLES:

```
ABC(R + L)
K (A - B * C, L(I, J), X)
RATE (T + L, D - 4)
A(I(N + M(J * K(B/C))), X(D * E * F), P)
```

In the second example, I and J must be in the same mode of arithmetic as (A - B * C), L, and X because they are all parts of the subscript of K.

In the third example, (T + L) and (D - 4) must both be either floating point or one may be fixed point and one an integer.

In the last example, as in the second, all parts of the subscript of the array name, A, must be in the same mode. All of the first subscript; I, N, M, J, K, B/C; as well as all of the second subscript; X, D * E * F; and the third subscript P must be in the same mode of arithmetic. If the above subscripted A were added to:

```
Q(G, H(R * S), T(W + V))
```

the Q must be the same mode as A, but the subscript of Q may be in a different mode than the subscript of A.

EXPRESSIONS

An arithmetic expression is a sequence of variables (data names), numbers (numeric literals), and/or mathematical functions connected by symbols representing the arithmetic operations add, subtract, multiply, divide, and exponentiation. Arithmetic expressions are evaluated from left to right, and indicated operations are performed in the order given in Figure 3.

Operation	Symbol
Function and Exponentiation	SIN, COS, etc. **
Multiplication and Division	* /
Addition and Subtraction	+ -

Figure 3. Priority of Arithmetic Operations

When parentheses are used, this priority may be overridden. The expression is evaluated from the innermost to the outermost set of parentheses.

Conventions

I = A**B. may be used as stated.

1. If B is an integer from 2 to 9, then A may be anything.
2. If A is a single nonsubscripted variable, then B may be anything.
3. If A is not a single, nonsubscripted variable, then B must be an integer from 2 to 9.

If any of the above three cases do not hold, then the exponentiation must be expressed as:

I = EXP(LN A*B).

The available functions appear in Figure 4.

Function	Symbol
Sine	SIN
Cosine	COS
Arctangent	ATAN
Square Root	SQRT
Exponential	EXP
Common Logarithm	LOG
Natural Logarithm	LN
Absolute Value	ABS

Figure 4. Available Functions

All functions are always calculated in floating point arithmetic (even though the mode of computation is fixed point). If the AAU is indicated in the Environment Division, the floating point of the AAU will be used to calculate functions. Otherwise the floating point package will be called in.

The arguments for sine and cosine must be expressed in radians. The results of arc tangent will be in radians.

EXAMPLES:

FED~TAX = (GROSS~PAY - (NUM~DEP * 13.0)) * 0.18.

The value of FED~TAX is obtained when data is substituted for the variables (data names).

YTD~FICA = YTD~FICA + (CURR~FICA = GROSS~PAY * 0.3).

GROSS-PAY is multiplied by 0.3, stored in CURR-FICA and added to YTD-FICA to complete the computation.

Relational expressions are any expressed or implied comparison of two field names, element names, literals, or arithmetic expressions. Relational expressions are connected by any of the relations shown in Figure 5 and are evaluated from left to right.

Relation	Abbreviation
Exceeds	GR
Greater than	
Not Greater than	NGR
Less than	LS
Not less than	NLS
Equal to Equals	EQ
Not Equal to Unequal to	NEQ
NOT POSITIVE	No Abbreviations
NOT NEGATIVE	
NOT ZERO	

Figure 5. Relational Expressions

Logical expressions provide a convenient method for obtaining truth values. They are formed by combining true-false variables and relational expressions with the logical operators AND, OR (Inclusive), and NOT (Exclusive). For instance, if P and Q are a combination of

- True-false variables
- Relational expressions
- Logical expressions

their truth value is obtained according to Figure 6 shown on the following page.

P	Q	Not P	P and Q	P or Q
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

Figure 6. Truth Values

EXAMPLES:

1. IF PAY NOT GREATER THAN 4800 ...

2. IF K1 + K2 EQUALS K3 ...

   Note the combination of arithmetic expression (K1 + K2) and the relational expression (EQUALS).

3. IF TIME OF EXP~FILE NOT LESS THAN 40 ...

A logical expression is any combination of conditional names and relational expressions connected by the logical AND, OR (Inclusive), and NOT (Exclusive) and may be an arithmetic expression. Logical expressions are evaluated from left to right with the logical AND having precedence over the logical OR. Parentheses may be used to establish precedence. There is no limit on the number of logical operators in a logical expression. Nine or less levels of parentheses may be used.

EXAMPLES:

1. IF A - B NOT LESS THAN 50.0 AND R * G EXCEEDS 272.0 ...
2. IF EXPERIENCED OR GRADUATE ...
3. IF NOT MARRIED AND AGE GR 30 OR CODE~3 ...
4. IF AGE GR 30 AND (GRADUATE OR EXPERIENCE EXCEEDS 10) AND LEVELS NLS 7 AND CLASS NOT EQUAL K ...

Note the logical expression in Example 4 may be an employee benefits test. The parentheses establish precedence such that the logic of the expression would be:

1. If the employee's age is greater than 30, and
   he is a graduate, or
   his experience exceeds 10, and
   his level is not less than 7, and
2. If his class (classification) does not equal K, then:
3. The employee has met all the requirements of the logical expression.

Note: the NOT in Examples 1 and 4 is part of a relational expression; the NOT in Example 3 is a logical expression.

4. USING THE GENERAL COMPILER FORMS

This chapter describes the sentence form and the Data Division form and indicates the letters and symbols which may be entered on these forms. Both of these forms are organized to facilitate the writing of the program, and the keypunching of the source deck cards. Samples of these forms are included at the end of this chapter.

Each line of both forms represents the 80-column card into which the information contained on each line is punched.

Both forms reserve columns 1-6 for sequence numbers. However, when writing General Assembly Program instructions (see ENTER verb), sequence numbers should be written in columns 75-80. Use of sequence numbers is optional. When they are used, a sequence check of the source program may be obtained during compilation by depressing console switch 18. Sequence numbers should be in ascending numerical sequence from Identification Division through Procedure Division. Numbers should be zero filled, if necessary, instead of blank filled. Blank columns are assumed to be greater than zero.

The sentence form (Figure 7) on which is written the Identification, Environment, and Procedure Divisions has heavy lines at columns 12 and 16 to facilitate indentation of sentences that are continued onto the following line (and consequently punched in another card). It is suggested that:

1. Sentence-names start in column 8.
2. Unnamed sentences start in column 12.
3. Continuation lines should be indented to column 16.

Observing these conventions for indentation of continued sentences makes it easier to follow a program both on the form itself and on the Edited List.

The Data Division (Figure 8) form, on which is written all sections of the Data Division, is blocked into several columns to aid the user in placing the numbers and characters necessary for describing data in the correct position on the form so they will appear in the correct card column when keypunched.

CONVENTIONS

1. Each division title and each section title (in the Data Division) is contained on a separate line and is terminated by a period. The titles start in Column 8.

2. All sentences are terminated by a period followed by at least one space.

3. Commas are optional EXCEPT in subscripts of multidimensional arrays.

4. Words are separated by at least one space. This space is optional when words are bound by +, -, *, /, **, (), =, and ,.

5. A hyphen (~) is placed in column 7 whenever a word is split at column 80, or when more than one qualifier is used in output files. A letter C in column 7 (in the Procedure Division and Data Division) indicates a comment card, the contents of which will be printed on the Edited List.

6. A decimal point followed by a blank is illegal. The expression 12.Δ + A * 4. is illegal because the compiler interprets the decimal points followed by blanks as ending the sentence.

7. All sentence-names are terminated by a period, and followed (on the same line) by the sentence itself.

8. Subscripts (including arithmetic expressions used as subscripts) must be enclosed in parentheses, and must be separated by commas when a multidimensional subscript is used. The parentheses may be immediately adjacent to the array name or may be separated from the array name by a space.

DATA DIVISION FORM

The entries for the various columns are as follows:

Heading	Columns	Entries	Manual Chapter	Reference Heading
Type	8-9	FD	5	File Description
		ΔR	5	Record
		*G	5	*Group
		ΔG	5	Group
		ΔF	5	Field
		ΔC	5	Conditional
		ΔE	5	Element
		FL	5	Field Literal
		ΔL	5	Literal
		ΔΔ	5	FILL, Constant Section, Common~Storage, and *Common~Storage
Data Name	11-22	any legal data name	3	Data Names
		blanks (for literals) FILL	5	Physical Characteristics
Qualifier	24-35	any data-name needed for uniqueness in the output files	3	Qualifiers
Format	37	P (or blank) U S 1 2 ,	5	See Specific Level

Heading	Columns	Entries	Manual Chapter	Reference Heading
Repeat	39-41	maximum of 3 numeric blank (If number is greater than 999, col. 42 may be used.)	5	See Specific Level
Binary	43	B (or any character) blank	5	See Specific Level
Justify	45	L R blank	5	See Specific Level
MS and LS	49-53	numerics	5	Elements
Data Image	55-80		5	Data Image Entries

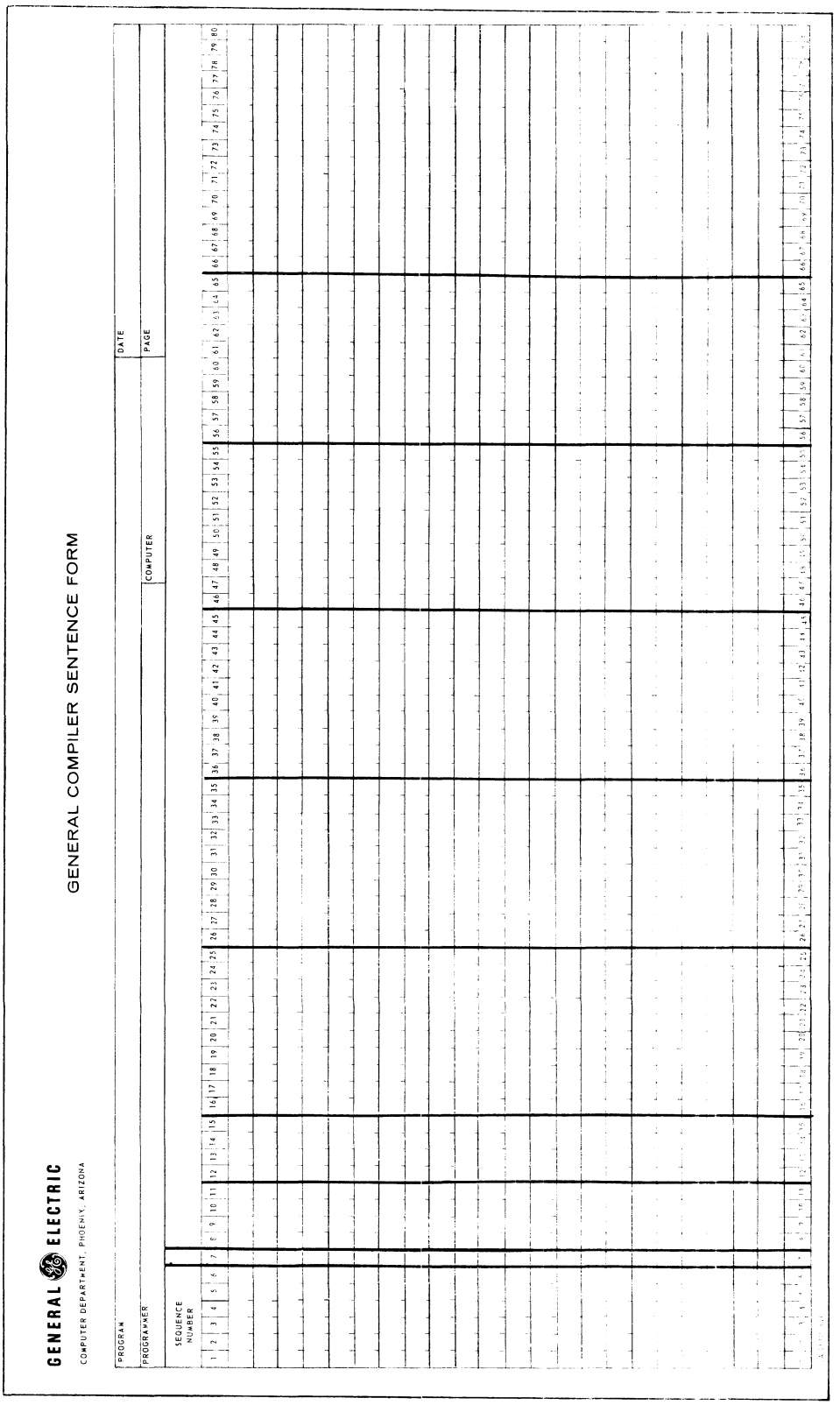


Figure 7. **GECOM** Sentence Form

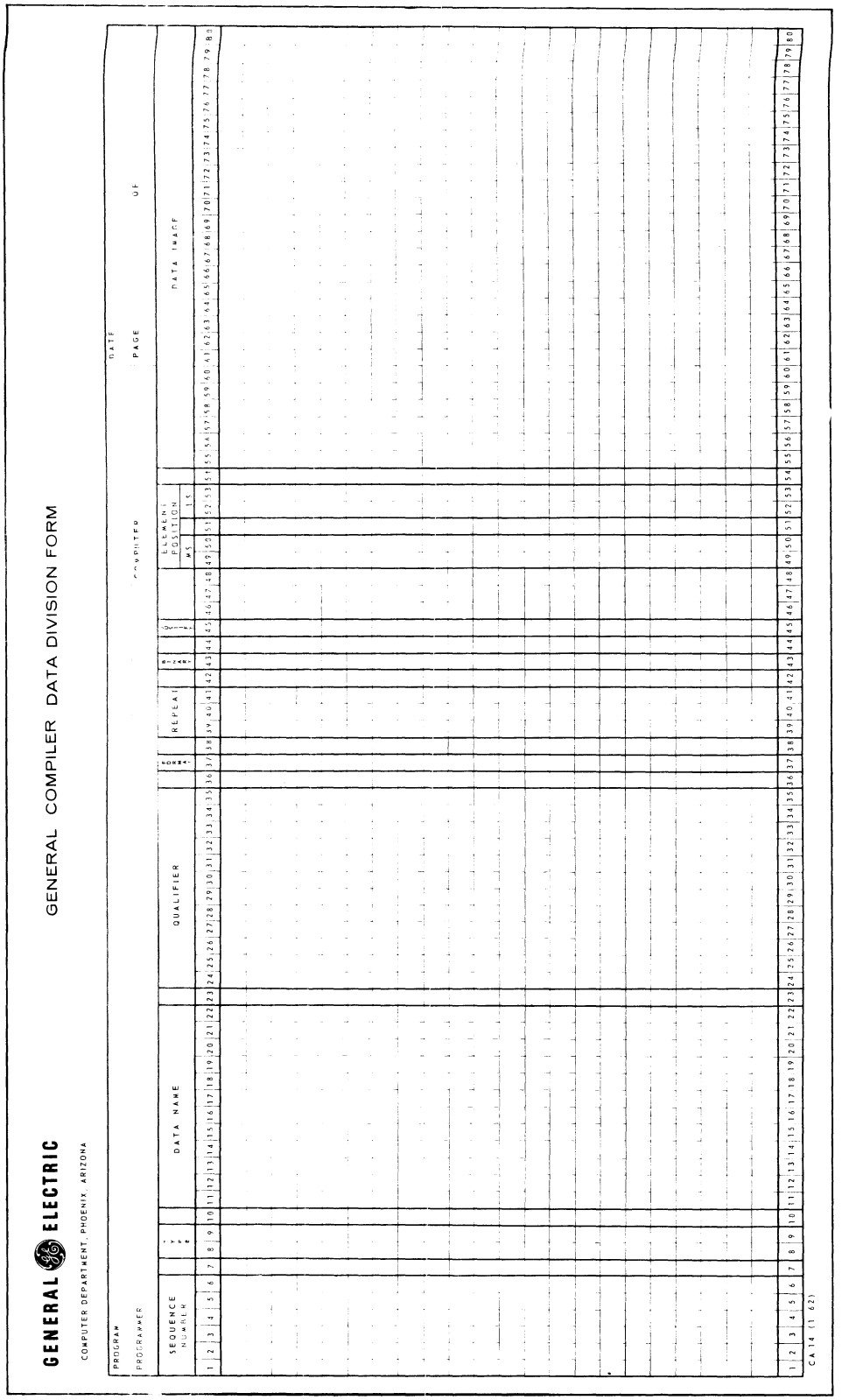


Figure 8. GECOM Data Division Form

5. DATA DIVISION

BASIC CONCEPTS

The Data Division contains the information which relates to the files. This information may be the input to or output from internal memory as well as that information which is generated during the processing of the input data. In addition, the constants used during the processing are defined. Data may be in the form of lists, tables, or arrays.

Sets of data pertaining to a subject are called "files" because the data contained in a file (section) of a source program is analogous to the data contained in an office file cabinet. For example, one cabinet might hold a set of invoices. As the cabinet may be labeled, so may a data file be labeled. The label of the data file, however, contains more information about the contents of the file. Each invoice might be considered one complete record within the file, but portions of the invoice might be grouped within the record. Two groups might be considered in this case; customer information, and the transaction itself. The specific items within these groups are called fields: name, address, city, state; and date, stock number, quantity, item price, total. In some circumstances it may be expedient further to divide the data into subfields which are called elements. Thus, an item such as data consisting of three elements--day, month, and year--may be contained in one field.

Levels of data are more completely described in Figure 9, starting with the highest level.

FILE	An overall description of many sets of data. It is the quantity of data referenced by READ.
RECORD	A named set of data described by the individual items below it. It is the quantity of data that is referenced in WRITE, and made available by READ.
*GROUP (of fields)	A group, which may be referenced by WRITE and made available by READ. Its name may be used as a qualifier.
GROUP (of fields)	Similar to a record but beneath it in rank, the data is described by the fields below it.
FIELD	Units of data which constitute a record or group. It is described in the Data Image columns.
ELEMENT	A position within a field. (Its description is contained within that of the field.)

Figure 9. Levels of Data

This structure may be illustrated on the following page.

TRANSACTIONS	FILE
INVOICE	RECORD
Customer Information	Group
Name	Field
Address	"
City	"
State	"
Sales	Group
Date	Field
Day	Element
Month	"
Year	"
Quantity	Field
Stock Number	"
Item Price	"
Total	"

After organizing the data into usable groupings, it is sometimes advantageous to consider many of these records as an entity. This is accomplished by specifying a given number of words as a block. The compiler then recognizes that these records are to be treated as a unit under specified circumstances.

A block is a physical record that contains more than one logical record. A block is treated as an entity by the READ sentence. Grouping many logical records into one physical record (for read purposes) is called blocking records and is accomplished by a notation in the file description of the File Section of the Data Division. Blocking permits reading into memory many short records with one READ sentence, thereby saving tape reading time. (The start/stop time of tape movement is a considerable portion of tape reading time.)

If records are not blocked, a logical record is equivalent to a physical record. However, when blocks are utilized, a physical record becomes a group of logical records and a READ sentence causes the next logical record to be made available to the programmer.

PHYSICAL CHARACTERISTICS

The Data Division describes the physical and conceptual characteristics of data as they appear on the external media. Physical characteristics include the mode in which data is recorded, the grouping of logical records, the indication of data with specialized functions, etc. Conceptual characteristics entail the specific description of each item of the data.

The recording mode of data refers to the mode (binary or decimal) of the data as it exists on the external media.

Computation mode must not be confused with recording mode which refers to mode of data externally. Internally, the computation may proceed in either fixed point or floating point.

A control-key is a field within a record or a *group containing a (literal) value which is used to identify the data description of that record or *group. The compiler must be able to identify the record or *group to assign it to the correct memory location according to the data images given by the programmer. The control-key is needed when there is more than one record (of *group) of different data descriptions within one file. The READ verb, though it refers to a file, actually makes available only the next logical record or group; therefore, the control-key is needed to indicate the record with which it is working.

Packed data is data that is entered into the computer without regard to the GE-200 Series word length into which it will be placed. The compiler will unpack the data before it is used.

Unpacked data is data that is so arranged that it may be read directly into integral GE-200 Series word lengths. The alphanumeric fields are left-justified and space filled, and numeric fields are right-justified and zero filled.

Data Name	Data Image	
F	9(2)	(Numeric field)
Q	A(3)	(Alphabetic field)
X	X(2)	(Alphanumeric field)
Y	9	(Numeric field)

Packed data as entered on a card:

1	2	3	4	5	6	7	8
F	F	Q	Q	Q	X	X	Y

Unpacked data on the card:

1→3	4→6	7→9	10→12
△FF	QQQ	XX△	△△Y

(Three BCD characters will occupy one GE-200 Series word.)

When fields are moved from an unpacked input area to an output area, the moves are done as full words. Therefore, caution should be used in laying out an output record which requires data from unpacked input to ensure that the data can be moved as full words.

The data-name FILL indicates to the compiler that the number of digits shown in the Data Image columns of the Data Division are not used in the Procedure Division and may be ignored by the compiler. This allows existing data tapes to be read and those portions not pertinent to the program to be disregarded. The pertinent items of data may be named and described (as alphanumeric, alphabetic, numeric, or blanks) as they appear.

EXAMPLE 1. Record on Tape:

TYPE	DATA NAME	DATA IMAGE
R	ACCTS~REC	
F	NAME	X(30)
F	NUMBER	9(6)
F	STREET	X(20)
F	QTY~ORDERED	9(3)
F	ITEM	X(20)
F	SUB~TOTAL	999V99
F	TOTAL	99999V99

If the same file were used to check the inventory, (that is, only the items and quantities sold are needed) the same record of the file might have the following description:

EXAMPLE 2. Record in Data Division of Program:

R	ACCTS~REC	
	FILL	X(56)
F	QTY~ORDERED	9(3)
F	ITEM	X(20)
	FILL	X(12)

In Example 2, the second name, FILL, replaces the NAME and ADDRESS fields of Example 1: The NAME and ADDRESS fields occupy 56 digits (30 + 6+ 20) and are alphanumeric; therefore, the FILL of Example 2 is described as 56 alphanumeric characters, X(56). The same reasoning applies to the fourth field in Example 2 which replaces the SUB-TOTAL and TOTAL fields of Example 1.

The actual data which was read in and described as FILL in a record or group of the input files is not supplied to the record or group of the output files unless only the incoming record is listed with qualifications (no fields are described), in output files.

EXAMPLE 3:

```
OUTPUT FILES
FD OUT1 ... .
 R INA     IN1
```

```
INPUT FILES
FD IN1 ....
  F INA
  F IA          9(5)
    FILL        X(10)
  F 2A          X(3)
    FILL        A(20)
  F 3A          X(4)
```

In Example 3, the actual data read in from tape and described as FILL is moved to the output files.

EXAMPLE 4:

```
OUTPUT FILES
FD  OUT1 ....
 R  OUTA
 F  1A
    FILL                B(10)
 F  2A
    FILL                B(20)
 F  3A


INPUT FILES
FD  IN1 ....
 R  INA
 F  1A                  9(5)
    FILL                X(10)
 F  2A                  X(3)
    FILL                A(20)
 F  3A                  X(4)
```

In Example 4, new FILL (blanks) is supplied.

COMMA-SEPARATED FIELDS

A comma in Column 37 at the record or *group level indicates that the input card fields are separated by commas. The data image columns for these fields are written normally with the following exceptions:

1. The scaling factor, P, may not be used.
2. All type-of-sign indicators (+, -, T, I, R) should be leading signs.
3. A description using E (a number in floating point) must contain an actual decimal in the fraction portion. The assumed decimal, V, is illegal.
4. The data description is limited to eight characters described as FILL preceding the first field of the record.

If a control-key is used, it must be of fixed length and must always appear as the first field of the *group or record. If another record type is introduced in another file, a new starting column may be used.

If the record requires more than one card, the control-key field (followed by a comma) must be in each card. However, the control-key fields after the one on the first card of a record are not placed in Process Storage and no description should be given for them in the Data Division.

A field may not be split between cards; the last field on the card must be followed by a comma. The unused columns must be left blank, and the next field on the following card started in the same column as the first field of the record.

Each data card must have at least one field. This necessitates at least one comma which delimits the field. There are as many commas as there are fields; that is, the last field of a record (or *group) must be followed by a comma.

The starting columns of the intermediate cards must be the same as that on the first card. The starting column may be any column from 1-9. If a FILL field appears before the first data field, the starting column is determined by the number of FILL characters. For example, if the data description is FILL 9(6), field-name 9(8), the data must start in Column 7 of all data cards. (FILL in this example might account for a six-digit sequence number on the data cards.)

If data in the field is to remain the same as previously read in, a comma following the comma delimiting the preceding field is all that is necessary.

For example:

XXXX,, YYYY,

causes XXXX to be read into field-1, field-2 remains the same, and YYYY is read into field-3.

JUSTIFICATION

Justification as indicated in the Justify column refers to justification of unpacked fields (as described by data image) including zeros and blanks and not the justification of the most significant characters of the fields. Therefore, a field described as X(2) and appearing on the input medium as 2ΔΔ is right-justified to Δ2Δ because the field is 2Δ, the third blank being the FILL which causes it to be unpacked.

Unpacked alphanumeric and alphabetic data are assumed left-justified and unpacked numeric data are assumed right-justified.

NONSTANDARD DATA

The numbers 1 and 2 (used in conjunction with binary data) are means of indicating to the compiler that the data on the input/output media are not standard GECOM data and exist in another form. The 1 and 2 indicate the word length is one or two binary words, respectively. Once inside the computer, the word is represented in the standard binary form (two word lengths).

The letter S for scale must be included in the data image if a 1 or 2 appears in column 37. (For example, 99V99S5 in the data image columns with a 1 in the column 37 would indicate that the field is contained in one word with a binary point of 5.) Note that when S is used in this manner, it describes the input or output or external binary scale. Once involved in a computing process, the field is carried in standard binary scale (internal scale).

Disc storage unit (DSU) addresses use bits 2-18 of the address word. The decimal equivalent of the higher DSU addresses is six characters in length. A special symbol, M, as the first character of a Data Image allows the DSU address fields to be at a scale of 18 and to have a data description of 9(6).

Absolute DSU address fields are described as Binary, 1 word, with a data description of M 9(6). The address field remains at a scale of 18 throughout the object program. Decimal DSU addresses have a description of M9(1) to M9(6). The field is converted to a binary number with a scale of 18.

The letter, M, preceding the image of a BCD numeric input field forces an internal scale of 18. An M preceding the image of an input binary field without the letter, S, assumes an external scale of 18 and assigns an internal scale of 18. S can be used with M to assign nonstandard internal scales. An M image on a BCD input field description along with an S value forces an internal scale equal to the S value. For example, the description M9999.9S13 for a BCD field would indicate that the internal binary scale is to be set at 13.

If an input field is binary (one or two words) and S is used with M, then the external scale is assumed to be at the S value and the internal binary scale is set to the S value. On output, M is meaningful only if the field is binary, in which case the binary scale is assumed to be 18 unless overridden by S.

TAPE LABELS

Labels are records for tape files only. Card files, DSU files, and printer output cannot have label records. Note that label record descriptions but not data record descriptions may be given for a JOURNAL~TAPE file, (See Environment Division, DSU~CONTROL sentence.) The tape label records are 24-word records the contents of which identify a tape (or file, in the case of a multifile tape).

If no label records appear on tape, the LABEL RECORDS ARE OMITTED clause must be included in the FD sentence. If this clause does not appear, the compiler expects the first record under the FD entry to be a beginning-tape-label or a biginning-file-label record.

There are four types of labesl: (See Figure 10 for formats.)

1. Beginning-tape label (BTL) is the first record on any tape except a multifile tape.
2. Beginning-file label (BFL) is a record which precedes the file of each multifile tape.
3. End-tape label (END REEL) is a record which follows the last valid data record (and the tape mark) on intermediate reels of a multireel file. This is not used on multifile tapes.
4. End-of-file label (END FILE) is a record which appears once only after the last data record (and the tape mark(on the last reel of a file. On a multifile tape, this record appears after each file.

The recording mode of tape labels is always binary regardless of the recording mode clause.

The formats of these records are:

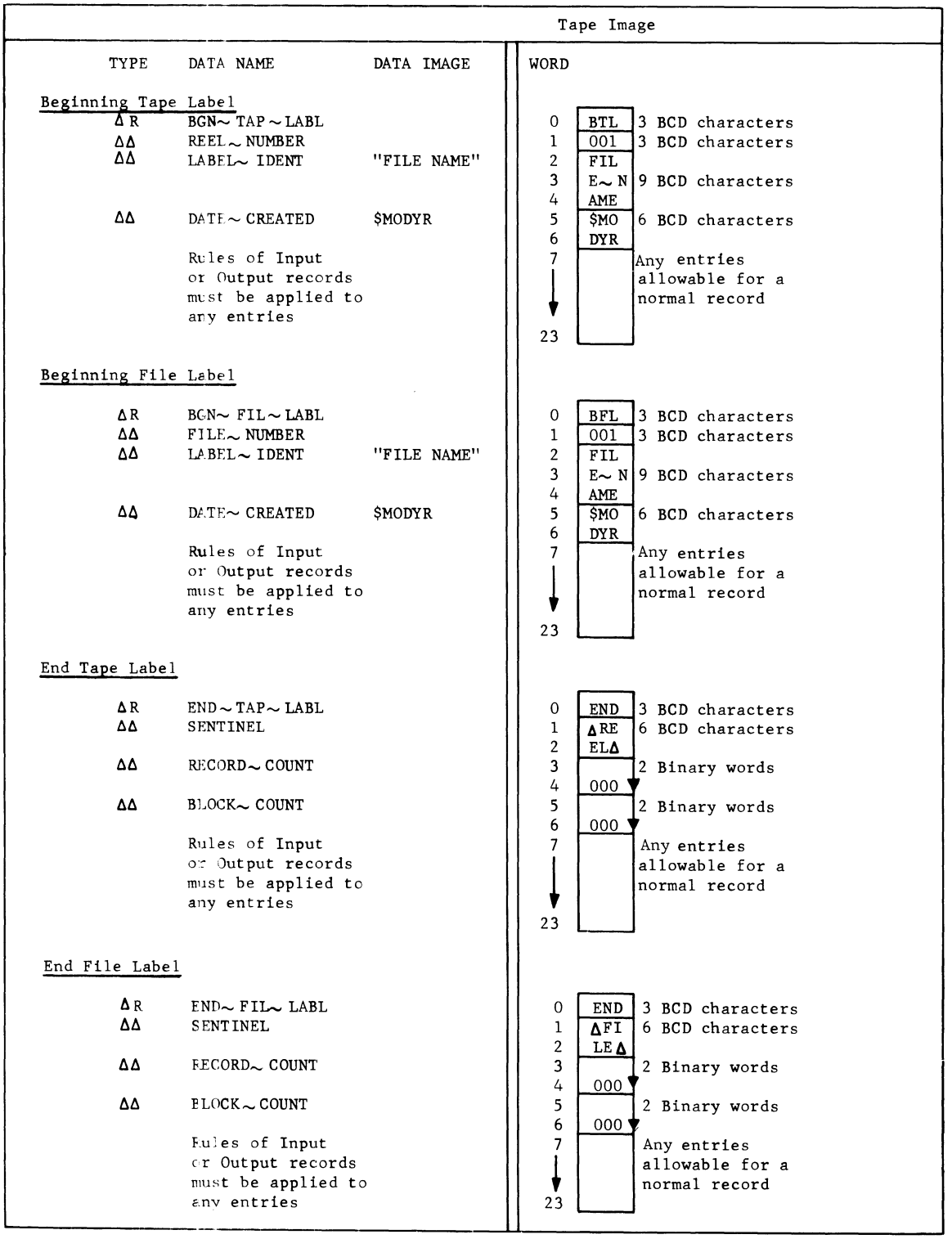


Figure 10. Formats for Tape Labels

All beginning and ending label records must immediately follow the File Description.

The use of a beginning label necessitates the presence of an ending label.

The ending labels are supplied by the object program.

The ending label (or labels) may be described if access to the contents is required.

The following combinations of labels are always used:

1. If a beginning-tape label is used, all intermediate reels will have an end-tape label and the last reel will have and end-file label.

2. A beginning-file label is used only on multifile tapes. Each file is terminated by an end-file label.

3. The last label appearing on a multifile tape is the end-file label of the last file on the tape. (A multifile tape consists of one reel.)

A tape mark precedes every ending label.

All that is necessary for any label record entry is an R in column 9, and the assigned name of the label record. No other entry is necessary unless access to the contents of the label is required. If the standard fields within the label record are used, the names are entered in the data name field without an entry in the type columns.

REEL~NUMBER, automatically started with the number 1, is generally not used, and therefore is not listed. If it is to be referenced, it must be listed directly beneath the tape label record entry. The name, REEL~NUMBER, is entered in the data-name columns. Type and data image columns are left blank.

LABEL~IDENT is normally entered within the record to allow comparison of the LABEL~IDENT in the Object Program file table to the LABEL~IDENT on the tape. The data image columns contain a maximum of nine BCD characters (in quotation marks).

If DATE~CREATED is not entered in the record, the date is automatically entered into the label from the GECOM generated Object Program.

If date symbols are not used, the Object Program expects to find the six BCD date characters in locations $(1076)_8$ and $(1077)_8$.

If DATE~CREATED is entered on the Data Division form, it must be entered in the beginning-tape label of all tape files using label records. The entry in the data image columns is a $ followed by a maximum of five characters. The entry may be in quotation marks but they are not needed.

Field names represent words 7-23 of the label record and may contain any information the programmer wants in the label record. Only those standard names of fields to which access is required need be listed under the label record except in the case of DATE~CREATED which must be listed in all label records if listed in any. These standard names must appear in the record prior to any nonstandard (programmer assigned) names. Nonstandard names follow the last mentioned standard name.

The standard names in the end label records generally are not listed, but may be entered if the programmer wishes to consult the contents of any word of the label. If used for comparison to another number, it must be remembered that RECORD~COUNT and BLOCK~COUNT are retained as binary numbers and must be compared to binary numbers. If these standard names are entered in the record, the type and data image columns are left blank.

Additional information may be added to the end labels in the same manner as described for the beginning labels.

EXAMPLE 1:

```
FD  TAPE~FILE 1, RECORDING MODE IS BINARY.
 R  BGN~TAP~LABL
    LABEL~IDENT              "FILE~NAME"
    DATE~CREATED             $MODYR
 F  RUN~FOR                  A(23)
 F  PURGE~DATE               X(6)
 R  END~TAP~LABL
 F  HASH~COUNT               9(5)
 R  END~FIL~LABL
 F  GRAND~HASH               9(11)
 R  NORMAL1
 F  A                        X(5)
```

EXAMPLE 2:

```
FD  TAPE~FILE 2, BLOCK SIZE IS 270 WORDS.
 R  BGN~TAP~LABL
    LABEL~IDENT              "FILE~NAME"
 R  NORMAL2
 F  A                        9(5)
 .
 .
```

Example 1 illustrates the File Section of a program that uses a dating routine to produce a dated program.

In Example 2, the GECOM Systems Tape date is supplied.

The various levels of data description are described on the following pages.

FUNCTION

The function of the file description is to identify the physical and conceptual characteristics of the data contained in the input and output files which are used in the object program. The file description describes the physical structure of the named file; the mode of the incoming and outgoing data, the size of blocks when the data is blocked, information pertaining to the label records, and the name of the control-key when records of different types are involved.

SENTENCE FORMAT

FD file-name-1, [RECORDING MODE IS [18~BIT] BINARY]

[, BLOCK {SIZE IS / CONTAINS} integer-1 WORDS]

[, LABEL RECORD(S) {IS / ARE} OMITTED]

[, NO END OF BLOCK SENTINEL]

[, CONTROL~KEY IS field-name-1] [, {NO~SET / ZERO~SET} BUFFER]

[, SEQUENCED ON field-name-2, field-name-3 ...] [, PROCESS FILE]

CONVENTIONS:

1. The RECORDING MODE clause indicates that data on the external media are not in the decimal mode. The [18~BIT] BINARY option may be specified only for tape files. It is used to permit reading and writing of tapes in the 18-bit binary mode. If the data are both binary and decimal, the binary mode must be indicated. The RECORDING MODE of a JOURNAL~TAPE file (see Environment Division, DSU~CONTROL sentence) is always binary. Consequently, the RECORDING MODE need not be specified for a JOURNAL~TAPE file.

2. The BLOCK SIZE clause indicates the number of words contained in the block. When the physical record (that data existing between tape gaps) is a logical record, there is no need for the BLOCK SIZE clause; it will, in fact, produce additional coding which will unnecessarily increase object running time. If a BLOCK SIZE clause is included in a program and the actual block size of an input tape does not agree with the stated block size, difficulties may be encountered. See Chapter 9, "Data Manipulation," for conventions relative to this situation.

3. The LABEL RECORD clause is used only when label records do not appear. Note that label record descriptions may be given for a JOURNAL~TAPE file (see Environment Division, DSU~CONTROL sentence), but data record descriptions may not be given for a JOURNAL~TAPE file. The clause may be omitted when describing files not assigned to magnetic tape.

4. The CONTROL-KEY clause applies only to input files. The control-key of each record must be identified by the same name, regardless of its position within the record. It is mentioned only for input files.

   It is recommended that the control-key field appear in the same position relative to the beginning of each record or *group and that it have the same data description but a different literal value as the control-key in the other records or *groups within the file. The literal value will appear in quote marks in the data image column. Do not qualify a control~key field when using it in PROCEDURE sentence if the control~key field is assigned to process storage.

   If there are multiple record formats in a file and no CONTROL~KEY clause is given:

   a. The compiler will print an error message.
   b. Unpacking coding will be generated for all record types.
   c. When a READ is executed, the first record type unpacking will be executed. There is no path to the other record type unpackings except via the ENTER GAP sentence.

   If there are multiple record formats in a file and a CONTROL~KEY clause is given, but not all record types have a control-key:

   a. The compiler will print an error message for each record type that does not have a control-key.
   b. Unpacking coding will be generated for all record types.
   c. Control-key coding (entered on each READ) can only reach the unpacking for those records which have control-keys. There is no path to the other record type unpackings except via the ENTER GAP sentence.

5. A hyphen (-) is placed in column 7 when a word of the file description is split at column 80.

6. The field-name-2 in the SEQUENCED clause represents the major key, field-name-3 represents the next lower key, etc. This clause is used for documentation purpose only.

7. The ZERO~SET buffer option applies to output files only. This is used to clear the entire buffer to zeros. All character positions (FILL) will be zero.

   The NO~SET buffer option applies to output files only. This is used to prevent clearing the output buffer. All undescribed character positions (FILL) will be assumed to contain miscellaneous characters. Process files assume NO~SET.

   If neither buffer setting option is used, the entire buffer will be cleared to blanks. All undescribed character positions (FILL) will be blank.

8. The NO...SENTINEL clause is used when end-of-block sentinels are not to be added to blocked output records when the buffer is released to the DSU. Ordinarily, sentinels would not be desired when a DSU blocked output file is released and the buffer is being shared with a blocked input file of the same description. For example, if there are five records to a block, the

third record is updated, and the block is ready to be released. (that is, no updating is done for the fourth and fifth records), a RELEASE may be given without giving READS and WRITES for the fourth and fifth records. However, no sentinel is desired on the RELEASE since it would destroy the first word of the fourth record. Any sentinel following the fifth record would, of course, be recorded on the DSU when the buffer is released.

When the NO...SENTINEL clause is used, the programmer can write his own sentinel by writing a one-word record of all 1 bits if he wishes to override the NO SENTINEL indication. This would be done when a new output block (additions) is being created.

9. An output file may be designated as a Process file. This ability is now included in GECOM for the COBOL Translator to give the user the same output file philosophy that COBOL provides. The user is responsible for building the output records through the use of MOVE, ARITHMETIC, or ASSIGNMENT statements. He may MOVE to output fields, groups, or assigned statements. A field that has been placed in the output record area may still be used in procedure statements; that is, the output area is used as working storage. At WRITE time, any fields that have been assigned Process storage will be automatically moved to the output area.

   When a DSU file is designated as a Process file, the programmer must ready that file before causing its data to be moved to output. (See READY verb.)

EXAMPLES

1. FD FILE~A, RECORDING MODE IS BINARY.
2. FD FILE~77, BLOCK CONTAINS 200 WORDS.
3. FD PAY~FILE, LABEL RECORDS ARE OMITTED, CONTROL~KEY IS FLAG.
4. FD MASTER~FILE, RECORDING MODE IS BINARY, BLOCK CONTAINS 500 WORDS, CONTROL~KEY IS INDICATOR.
5. FD TRANS~FILE, BLOCK CONTAINS 700 WORDS, CONTROL~KEY IS REC~CODE.
6. FD PRINT~FILE, RECORDING MODE IS 18~BIT BINARY, BLOCK CONTAINS 336 WORDS.
7. FD TAPE~FILE RECORDING MODE IS BINARY ZERO~SET BUFFER.
8. FD OUTTER, PROCESS, BLOCK CONTAINS 157 WORDS.

INPUT AND PROCESS OUTPUT ENTRIES

TYPE - R

DATA NAME

Any legal data-name.

QUALIFIER - not used.

FORMAT

P	Assumes all levels below (within this record) to be packed with the exception of binary numerics.
U	Assumes all levels below (within this record) to be unpacked.
,	File is on cards. All fields are delimited by commas.
Δ	Assumes data to be packed. An entry at a lower level takes precedence.

REPEAT - not used.

BINARY

B or any character	Assumes all levels within the record having numeric descriptions (9) to be in the standard GECOM binary form unless format column entry at a field level indicates nonstandard binary data. This does not alter nonnumeric data.
Δ	Assumes BCD data. Entries at lower levels take precedence.

JUSTIFY

L	Assumes unpacked BCD numeric data to be left justified and zero filled.
R	Assumes all alphanumeric and alphabetic unpacked data to be right justified and blank filled.
Δ	Assumes all BCD numeric data to be right justified and zero filled, all alphabetic and alphanumeric data to be left justified and blank filled. An entry at a lower level takes precedence.

DATA IMAGE - not used.

NONPROCESS OUTPUT ENTRIES

TYPE - R

DATA NAME

Any legal data-name.

QUALIFIER - the file name of this record name to force implied move of entire record to output. An output record which is a direct reflection of a working-storage record must be qualified by WS even when the name is unique.

FORMAT

Used only if description of data differs from that in Input Files, Working-Storage, or Common-Storage.

P	Forces all levels below (within this record) to be packed with the exception of binary numerics.
U	Forces all levels below (within this record) to be unpacked.
Δ	Assumes the entry from which the implied move is made.

REPEAT - not used.

BINARY

B or any character	Forces all lower levels of the record having a numeric data description (9) to be in the standard binary form unless the format column entry at the field level indicates nonstandard binary data. This does not alter nonnumeric data.
Δ	Assumes the entry from which the implied move is made.

JUSTIFY - used only when output format differs.

L	Forces unpacked BCD numeric data to be left justified and zero filled.
R	Forces unpacked alphanumeric and alphabetic data to be right justified and blank filled.
Δ	Assumes the entry from which the implied move is made.

DATA IMAGE - not used.

INPUT ENTRIES

TYPE - *G

The group of fields must be followed by another *group or be the last entry in the record in order to delimit the group. This entry must be preceded by a record entry.

DATA NAME

Any legal data-name.

QUALIFIER - not used.

FORMAT

P	Assumes all levels below (within this *group) to be packed with the exception of binary numerics. This entry takes precedence over any entry at record level.
U	Assumes all levels below (within this *group) to be unpacked. This entry takes precedence over any entry at the record level.
Δ	Assumes data to be packed. This entry takes precedence over any entry at the record level.

REPEAT - not used.

BINARY

B or any character	Assumes all levels within the group having a numeric description (9) to be in the standard GECOM binary form unless format column entry at a field level indicates nonstandard binary data. This does not alter nonnumeric data.
Δ	Assumes BCD data. An entry at a lower level takes precedence.

Any entry at the *group level takes precedence over any entry at the record level.

JUSTIFY

L	Assumes unpacked BCD numeric data to be left justified and zero filled.
R	Assumes all alphanumeric and alphabetic unpacked data to be right justified and blank filled.
Δ	Assumes all BCD numeric data to be right justified and zero filled, and all alphabetic and alphanumeric data to be left justified and blank filled. An entry at a lower level takes precedence.

Any entry at *group level takes precedence over any entry at the record level.

ELEMENT POSITION - not used.

DATA IMAGE - not used.

PROCESS OUTPUT ENTRIES

*Groups are not allowed in Process output files.

NONPROCESS OUTPUT ENTRIES

TYPE - *G

DATA NAME - Any legal data-name.

QUALIFIER - The name of the file which is the source of the *group in order to force implied movement of the entire *group to output.

FORMAT

P	Forces all levels below (within this *group) to be packed with the exception of binary numerics. The group entry takes precedence over any entry at the record level.
U	Forces all levels below (within this *group) to be unpacked with the exception of binary numerics. This entry takes precedence over any entry at the record level.
△	Assumes the entry from which the implied move is made.

REPEAT - not used.

BINARY

B or any character	Forces all lower levels of the group having a numeric data description (9) to be in the standard binary form unless the format column entry at the field level indicates nonstandard binary data. This does not alter nonnumeric data.
△	Assumes the entry from which the implied move is made.

Any entry at this level takes precedence over any entry at the record level.

JUSTIFY

L	Forces BCD numeric data to be left justified and zero filled.
R	Forces alphanumeric and alphabetic data to be right justified and blank filled.
△	Assumes the entry from which the implied move is made.

Any entry at this (*G) level takes precedence over any entry at the record level.

ELEMENT POSITION - not used.

DATA IMAGE - not used.

INPUT AND PROCESS OUTPUT ENTRIES

TYPE - G

A group may be delimited in one of two ways:

1. The group of fields may be followed by another group or may be the last entry in the record.

2. The terminate (type T) may be used.

DATA NAME - any legal data-name.

QUALIFIER - not used.

FORMAT

P	Assumes this group to be packed if there are no conflicting entries at a higher level. This is meaningless for binary numerics.
U	Assumes all levels below (within this group) to be unpacked if no conflicting entry appears at a higher level.
△	Assumes data to be packed.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

REPEAT

The repeat field may contain blanks or one to three numerics indicating the number of times the group is repeated. If a fourth numeric is required, column 42 may be used. Repeated groups are restricted to a maximum of seven computer words per group.

BINARY

B or any character	Assumes all levels within the group having a numeric description (9) to be in the standard GECOM binary form unless format column entry at a field level indicates nonstandard binary data. This does not alter non-numeric data.
△	Assumes BCD data. Entries at lower levels take precedence.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

JUSTIFY

L	Assumes unpacked BCD numeric data to be left justified and zero filled.
R	Assumes all alphanumeric and alphabetic unpacked data to be right justified and blank filled.
△	Assumes all BCD numeric data to be right justified and zero filled, all alphabetic and alphanumeric data to be left justified and blank filled. An entry at a lower level takes precedence.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

ELEMENT POSITION - not used.

DATA IMAGE - not used.

NONPROCESS OUTPUT ENTRIES

TYPE - G must be present in the type columns.

Note: Use of G in output forces implied movement of all fields in the source group to output. If the source fields are listed under the group in the output description, the fields will be repeated in output.

DATA NAME - Any legal data-name.

QUALIFIER - A record name if necessary for uniqueness.

FORMAT

P	Forces all levels below (within this group) to be packed with the exception of binary numerics.
U	Forces all levels below (within this group) to be unpacked with the exception of binary numerics. This entry takes precedence over any entry at the record level.
Δ	Assumes the entry from which the implied move is made.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

REPEAT - not used.

BINARY

B or any character	Forces all lower levels of the group having a numeric data description (9) to be in the standard binary form unless the format column entry at the field level indicates nonstandard binary data. This does not alter nonnumeric data.
Δ	Forces BCD data output. Entries at lower levels take precedence. Any entry at record or group level takes precedence over the blank.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

JUSTIFY

L	Forces unpacked BCD numeric data to be left justified and zero filled.
R	Forces unpacked alphanumeric and alphabetic data to be right justified and blank filled.

Δ Assumes the entry from which the implied move is made.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

ELEMENT POSITION - not used.

DATA IMAGE - not used.

FILE SECTION
TERMINATE

INPUT AND PROCESS OUTPUT ENTRIES

TYPE - T

DATA NAME

Name corresponding to a previous group name. The T entry delimits the specified group. Groups may be nested and overlapped by use of the type T entry.

Note: In Working Storage, type T may be used in the same manner to define overlapping or nested group.

Type T cards may only be used in programs generated by the COBOL Translator.

QUALIFIER - not used.

FORMAT - not used.

BINARY - not used.

JUSTIFY - not used.

ELEMENT POSITION - not used.

DATA IMAGE - not used.

INPUT AND PROCESS OUTPUT ENTRIES

TYPE - F (or blank when FILL is used.)

DATA NAME - any legal data-name or FILL.

QUALIFIER - not used.

FORMAT

P	Assumes this field to be packed if there is no conflicting entry at a higher level. This is meaningless for binary numerics.
U	Assumes this field to be unpacked if there is no conflicting entry at a higher level.
1	Assumes one-word binary numeric data. A scaling factor must be supplied in the data image columns.
2	Assumes nonstandard two-word binary numeric data. A scaling factor must be supplied in the data image columns.
S	The preceding image is to be repeated for this entry. S cannot be used when the preceding image contains a 1 or 2.
Δ	Assumes data to be packed.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

REPEAT

The repeat entry may be blanks or one to three numerics indicating the number of times the field is repeated. If a fourth numeric is required, column 42 may be used.

BINARY

B or any character	Assumes the field to have a numeric description (9) and to have standard GECOM binary form unless the format column indicates nonstandard binary. This does not alter nonnumeric data.
Δ	Assumes BCD data. Entries at higher levels take precedence over a blank.

Any entry at this level takes precedence over a blank at a higher level.

JUSTIFY

L	Assumes unpacked BCD numeric data to be left justified and zero filled.
R	Assumes all alphanumeric and alphabetic unpacked data to be right justified and blank filled.
Δ	Assumes all BCD numeric data to be right justified and zero filled, all alphabetic and alphanumeric data to be left justified and blank filled.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

DATA IMAGE

A data image must be entered for every field. Output fields are limited to 127 characters each. (See "Data Image Entries.")

NONPROCESS OUTPUT ENTRIES

TYPE - F

DATA NAME - Any legal data-name.

QUALIFIER

Qualification as necessary for uniqueness: record name and/or file name; for more than one level of qualification, a hyphen (~) must be in column 7 of the continued qualifier entry; and the type must be blank, and the qualifier entry must be in columns 24-35.

FORMAT

Not necessary unless different from Input or Working-Storage files.

P	Forces this field to be packed if there is no conflicting entry at a higher level. This is meaningless for binary numerics.
U	Forces this field to be unpacked if there is no conflicting entry at a higher level. This is meaningless for binary numerics.
Δ	Assumes the entry from which the implied move is made.
1	Assumes one-word binary numeric data. A scaling factor must be supplied in the data image columns.
2	Assumes nonstandard two-word binary numeric data. A scaling factor must be supplied in the data image columns.
S	The preceding image is to be repeated for this entry. S cannot be used when the preceding image contains a 1 or 2.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

REPEAT - not used.

BINARY

B or any character	Forces field (which has numeric description (9)) to be in the standard GECOM binary form.
Δ	Forces BCD data output. Entries at higher levels take precedence over the blank.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

JUSTIFY

L	Forces BCD numeric data to be left justified and zero filled.
R	Forces alphanumeric and alphabetic data to be right justified and blank filled.
Δ	Assumes the entry from which the implied move is made.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

ELEMENT POSITION - not used.

DATA IMAGE

Not necessary unless different from input entries. Output fields are limited to 127 characters each. (See "Data Image Entries.")

INPUT AND PROCESS OUTPUT ENTRIES

TYPE - E

DATA NAME - any legal data-name.

ELEMENT POSITION

MS	The position of the most significant digit of the element within the alphanumeric field.
LS	The position of the least significant digit of the element within the alphanumeric field.

NO OTHER ENTRIES are made for elements.

NONPROCESS OUTPUT ENTRIES

TYPE - E (for documentation purposes only).

DATA NAME - any legal data-name (for documentation purposes only).

NO OTHER ENTRIES are made for elements.

1. The element is ignored and appears only in the field in which it is contained.
2. If the element itself is required, it must be renamed a field in the output files (the type column may be ΔΔ or ΔF).

Note: An element must be alphanumeric.

INPUT AND PROCESS OUTPUT ENTRIES

TYPE - C

DATA NAME - any legal data-name.

DATA IMAGE

1. The value of the conditional may be described as:

literal constant	"230"
fixed point	230.1
integer	230
floating point	+2.301E+2

2. The value assigned to the conditional name must be consistent with the description of the field or element containing the conditional value.

3. The field may not be a repeated entry or part of a repeated group.

NO OTHER ENTRIES are made for conditional names.

NONPROCESS OUTPUT ENTRIES

Conditional names may not be used in nonprocess output files.

EXAMPLES OF CONDITIONAL NAMES

GENERAL ELECTRIC — GENERAL COMPILER DATA DIVISION FORM

COMPUTER DEPARTMENT PHOENIX ARIZONA

PROGRAM
PROGRAMMER
COMPUTER
DATE
PAGE OF

SEQUENCE NUMBER		DATA NAME	QUALIFIER	LEVEL			REPEAT					ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
	F	PAY												9
	C	SALARIED												1
	C	HOURLY												0
	F	TYPE												XXXX
	C	SPECIAL												"AAAA"
	C	ALTERNATE												"CCCC"
	E	SUBTYPE~1										01	02	
	C	USE												"01"
	C	NOUSE												"02"
	E	SUBTYPE~2										03	04	
	C	YES												"F3"
	C	NO												"NO"
	F	VALUE												19V99
	C	COND~1												10.05
	C	COND~2												-10.05
	C	COND~3												+00.00

GENERAL ELECTRIC

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER DATA DIVISION FORM

PROGRAM

PROGRAMMER

COMPUTER

DATE

OF

SEQUENCE NUMBER		DATA NAME	QUALIFIER		REPEAT			ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
	F	FLOAT~1								+9.99E-99
	C	FLPTCOND~1								+2.30E-02
	C	FLPTCOND~2								+0.01E-02
	C	FLPTCOND~3								+0.04E-02

INPUT AND PROCESS OUTPUT ENTRIES

Literals are not used in input files or process output files.

NONPROCESS OUTPUT ENTRIES

TYPE - L

DATA NAME - not used.

REPEAT - not used.

OTHER COLUMNS

The same rules that apply to fields apply to literals.

DATA IMAGE

1. When writing literals which are continued to the next line of the data image columns, the literal must start on the succeeding lines in column 55, a hyphen (~) must be placed in col. 7.

2. Literals are used in output fields to generate Hollerith characters for headings. The most efficient method of writing literal headings entails writing "one" literal consisting of 120 characters. The literal may be continued on as many succeeding lines as are necessary. A hyphen must appear in column 7 of the continuing lines. Quotation marks must precede the first character and follow the last character only.

3. Literals may be described as:

literal constant	"230"
fixed point	230.1
integer	230
floating point	+2.301E+2

4. Literals enclosed in quotation marks are in BCD mode. Numerics are converted to binary. Thus, it is more efficient to use quotation marks if practical.

EXAMPLES OF LITERALS:

GENERAL ELECTRIC

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER DATA DIVISION FORM

PROGRAM

PROGRAMMER

COMPUTER

DATE

PAGE OF

SEQUENCE NUMBER		DATA NAME	QUALIFIER		REPEAT			ELEMENT POSITION MS	ELEMENT POSITION LS	DATA MACRO
	L									" HEADING
	~									OF REPORT "
	L									"1234567890123456789012345
	~									6789012345678901234567890 1
	~									2345678901234567890 1234"
	L									+230.1
	L									-230
	L									+2.301E+02
	L									"ABCDEF"BB 'GHIJKLM"BB"NOP
	~									QRST"BB"UVWXYZ"

CARD IS 62

INPUT ENTRIES

TYPE - FL

DATA NAME - any legal data-name.

QUALIFIER - not used.

FORMAT

P	Assumes this field to be packed if there is not conflicting entry at a higher level. This is meaningless for binary numerics.
U	Assumes this field to be unpacked if there is no conflicting entry at a higher level.
1	Assumes one-word binary numeric data. A scaling factor must be supplied in the data image columns.
2	Assumes nonstandard two-word binary numeric data. A scaling factor must be supplied in the data image columns.
S	May not be used.
Δ	Assumes data to be packed.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

REPEAT

The repeat field may contain blanks or one to three numerics indicating the number of times the field literal is repeated. If a fourth numeric is required, column 42 may be used.

BINARY

B or any character	Assumes the field to have a numeric description (9) and to have standard GECOM binary form unless the format column indicates nonstandard binary. This does not alter nonnumeric data.
Δ	Assumes BCD data. Entries at higher levels take precedence over a blank.

Any entry at this level takes precedence over a blank at a higher level.

JUSTIFY

L	Assumes unpacked BCD numeric data to be left justified and zero filled.
R	Assumes all alphanumeric and alphabetic unpacked data to be right justified and blank filled.
Δ	Assumes all BCD numeric data to be right justified and zero filled, all alphabetic and alphanumeric data to be left justified and blank filled.

Any nonblank entry at this level takes precedence over a blank at a higher level.

Any nonblank entry at a higher level takes precedence over any entry at this level.

DATA IMAGE

1. The field literal may be described as:

literal constant	"230"
fixed point	230.1
integer	230
floating point	+2.301E+2
integer	230K1

When described as a literal constant, it may not be used in arithmetic statements.

2. The field literal is ordinarily used only for a control-key field. When used for this purpose, the contents of the input field are compared to the literal entered in the data image columns to determine the type of the incoming record. A control-key field may not be repeated.

3. If a field literal is used in any input field other than the control-key field, the description listed in the data image columns will be treated as an image and the data in the incoming field will be used according to the image. A field literal may not be more than 83 characters in length, when enclosed in quotation marks.

4. Except as indicated above, the compiler processes field literal entries as field entries.

PROCESS AND NONPROCESS OUTPUT ENTRIES

Field literals are not used in output.

EXAMPLES OF FIELD LITERALS:

TYPE	DATA NAME	REPEAT	DATA IMAGE
	A field literal described as:		
FL	FIELD~1		"ABCD"
	is treated as a field and assumes an image of:		
F	FIELD~1		XXXX
	A field literal described as:		
FL	FIELD~2		230.1
	is treated as a field and assumes an image of:		
F	FIELD~2		999.9

A repeated field literal is described as follows:

GENERAL ELECTRIC — GENERAL COMPILER DATA DIVISION FORM

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

PROGRAM | COMPUTER | DATE
PROGRAMMER | | PAGE | OF

SEQUENCE NUMBER		TYPE	DATA NAME	QUALIFIER	SECTION	REPEAT	BLANK	SIGN	JUSTIFY	ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
		FL	FIELD~A			003						"ABCD"
		FL										"EFGH"
		FL										"IJKL"
		FL	FIELD~B			003						+230.1
		FL										+230.2
		FL										+230.3
		FL	FIELD~C			003						-230
		FL										-231
		FL										-232
		FL	FIELD~D			003						+2.301E+2
		FL										+2.302E+2
		FL										+2.303E+2
		FL	FIELD~E			003						230K1
		FL										230K2
		FL										230K3

CARD FORM 1 62

FUNCTION

The function of data image entries is to show a detailed picture and the general characteristics of data.

CONVENTIONS

1. The data image consists of any allowable combination of symbols required to describe data.

2. USE codes have been assigned to symbols to show in which parts of the Data Division the various symbols may be used in data images. USE codes with the corresponding parts of the Data Division are as follows:

CODE	MEANING
1	NONPROCESS OUTPUT FILES
2	PROCESS OUTPUT FILES
3	INPUT FILES
4	WORKING~STORAGE SECTION
5	COMMON~STORAGE SECTION
6	*COMMON~STORAGE SECTION
7	CONSTANT SECTION

3. The following operational symbols do not represent character positions of the data:

Symbol	Meaning	See Convention Number	USE Code
E	Indicates the point of separation of the mantissa from the characteristic of a floating point number.	5	1, 2, 3, 4, 5, 6, 7
K	May be used instead of symbol for actual or assumed decimal point. Numeric data described with a K are interpreted as integers for all manipulations. See Data Manipulation, using K in Data Descriptions.	7	1, 3, 4, 5, 6, 7
M	May be used as leading character of image only. See explanation under Data Division, Nonstandard Data.	7	1, 2, 3, 4, 5, 6, 7
P	Indicates zeros to be applied to the data to obtain the true value of the data. May be used to indicate right or left scaling (99PPP or . PPP99).	7	1, 2, 3
S (followed by an integer)	Indicates the binary point location. See Data Division, Nonstandard Data. See Data Manipulation, Use of Scaling Factor.	7	1, 2, 3, 4, 5, 6, 7
V	Assumed decimal point position.	6	1, 2, 3, 4, 5, 6, 7
(n)	An integer enclosed in parentheses following a symbol indicates the number of consecutive occurrences of the symbol (for example, A(3)X(2) and AAAXX are equivalent.)		1, 2, 3, 4, 5, 6

4. The following symbols represent character positions of the data:

Symbol	Meaning	See Convention Number	USE Code
A	Corresponding data position contains an alphabetic character, A-Z or blank. (This is not meant to indicate leading and/or trailing blanks adjacent to numeric data.)		1, 2, 3, 4, 5, 6
X	Corresponding data position contains one of the characters comprising the character set. (May be used to reflect data containing a combination of alphabetic and numeric characters.)		1, 2, 3, 4, 5, 6
9	Corresponding data position contains a numeric character, 0-9. The data image of a numeric field may never exceed 11 numeric symbols.	7 8 9	1, 2, 3, 4, 5, 6
I	Corresponding data position contains a numeric character (0-9), with a 12-row overpunch (+) when the data is positive, or an 11-row overpunch (-) when the data is negative. If no overpunch, the data is assumed positive. May only appear as the leading or trailing symbol of the data image.	7, 10	1, 2, 3
R	Corresponding data position contains a numeric character (0-9), with a 11-row overpunch (-) when the data is negative or no overpunch when the data is positive. May only appear as the leading or trailing symbol of the data image.	7. 10	1, 2, 3
T	Corresponding data position contains a minus sign (-) when the data is negative or the most significant character of the data when positive. May only appear as the leading symbol of the data image. See Data Division, Comma Separated Fields.	7	3
+	Corresponding data position contains a plus sign (+) when data is positive or a minus sign (-) when data is negative. If no sign, data is assumed positive. May only appear as the leading or trailing symbol of the data image.	7, 8, 9	1, 2, 3, 4, 5, 6, 7
-	Corresponding data position contains a minus sign (-) when data is negative and blank when positive. May only appear as the leading or trailing symbol of the data image.	7, 8, 9	1, 2, 3, 4, 5, 6, 7
.	Corresponding data position contains an actual decimal point.	6	1, 2, 3, 4, 5, 6, 7
,	Corresponding data position contains a comma.	9, 10	1, 2, 3

Symbol	Meaning	See Convention Number	USE Code
$	Corresponding data position contains a dollar sign. May only appear as the leading symbol of the data image.	9, 10	1, 2, 3
Z	Zero suppression; represents a numeric data position the contents of which will be suppressed (replaced by a blank) when the value is zero.	9, 10	1, 2, 3
*	Asterisk fill; represents a numeric data position the contents of which will be replaced by an asterisk when the value is zero.	9, 10	1, 2, 3
$$ ++ --	Floating Dollar Sign or Floating Report Sign. A sequence of two or more $'s, or two or more +'s, or two or more -'s represents numeric data positions in which the least significant (rightmost) leading zero is replaced by the indicated sign, and preceding zeros (if any) are replaced by blanks.	9, 10	1, 2, 3
CR	Credit sign; represents two data positions, the contents of which are CR when the data value is negative, or blank otherwise.	8, 10	1, 2, 3
DB	Debit sign; represents two data positions, the contents of which are DB when the data value is negative, or blank otherwise.	8, 10	1, 2, 3

5. A floating point data image must be in the form:

$$\left\{\begin{matrix}+\\-\end{matrix}\right\}\left\{\begin{matrix}9\\R\\I\end{matrix}\right\}\left\{\begin{matrix}.\\V\end{matrix}\right\} \; 9 \; (8) \; E \; \left\{\begin{matrix}+\\-\end{matrix}\right\} \; 9 \; (2)$$

The 8 and 2 indicate the maximum number of digits.

6. A numeric data image may contain only one decimal point position (assumed or actual). Numeric data which contains an actual decimal point may be used in computations.

7. The image of numeric data to be used in computations may contain the symbols:

+ - I R (T) 9 . V E P M S K

8. The image of numeric data not to be used in computations may contain the symbols $, *, Z CR, and DB in addition to those shown in Convention 7.

9. Only one type of suppression may be specified in a data image.

   a. The data positions to which suppression applies are represented by:

      One or more Z symbols (zero suppression)
      One or more * symbols (asterisk fill)
      Two or more $ signs (floating dollar sign)
      Two or more + signs (floating plus sign)
      Two or more - signs (floating minus sign)

   b. Numeric data positions to the right of the decimal point position (must be actual decimal point) in a "suppressed" data image, may be represented by the symbols 9 or Z but not a mixture. Any Z to the right of the decimal point indicates that all data positions must be set to blank when the data value is zero.

   c. If the data image contains floating signs ($$, or ++, or --), one sign is to be inserted, even if no suppression takes place. For this reason, the total number of symbols shown must be one greater than the maximum number of numeric data positions. The appropriate sign is placed in the rightmost position actually suppressed.

   d. Zero suppression (Z) or asterisk fill (*) may be specified in a data image with a dollar sign ($) in the leading position, for example, $*** or $ZZZ. Otherwise, the sequence of suppression symbols must occupy the leftmost positions of the data image.

   e. Comma symbols may optionally appear within the sequence of suppression symbols, but no other symbols may intervene. Commas are used to set off the integral portion into three-character sequences, counting to the left from the decimal position. The object program spreads the adjacent data digits apart one position and inserts a comma. If leading zeros are suppressed up to or beyond a comma position, the comma is suppressed, and its position receives the appropriate suppression character. If any commas are used, the proper number must be supplied in the correct position in the data description field.

EDITING EXAMPLES

SOURCE AREA		RECEIVING AREA	
DATA IMAGE	DATA VALUE IN MEMORY	DATA IMAGE	EDITED DATA
9(5)	45678	$ZZ,ZZ9.99	$45,678.00
9(3)V99	45678	$ZZ,ZZ9.99	$ 456.78
9(3)V99	00067	$ZZ,ZZ9.99	$ 0.67
9(3)V99	00004	$ZZ,ZZZ.99	$ .04
9(5)	00000	$ZZ,ZZZ.ZZ	
V9(5)	12345	$ZZ,ZZ9.99	$ 0.12
9(5)	12345	$**,**9.99	$12,345.00
9(5)	00045	$**,***.99	$****45.00
9(5)V99	(-)0000003	$**,***.ZZCR	$******.03CR
9(5)V99	0000000	$**,***.ZZCR	
9(5)	67890	$$$,$$9.99	$67,890.00
9(3)V99	67890	$$$,$$9.99	$678.90
9(5)	00000	$$$,$$9.99	$0.00
9(5)V99	(-)0000003	$$$,$$$.ZZDB	$.03DB
9(5)V99	0000000	$$$,$$$.ZZ	
V9(5)	67890	$$$,$$9.99	$0.67
9(5)V	(-)56789	ZZZZ9.99-	56789.00-
9(5)	(+)56789	ZZZZ9.99-	56789.00
9(5)	(+)56789	ZZZZ9.99+	56789.00+
9(5)	(-)56789	ZZZZ9.99+	56789.00-
99V9(3)	(-)56789	------.99	-56.78
9(5)V99	(+)0056789	+++,+++.99	+567.89
9(5)	(-)00567	ZZZZZ.99-	567.00-
9(5)	(-)56789	$$$$$.99CR	$56789.00CR
9(5)	(+)56789	$$$$$.99CR	$56789.00
$99,999.99	$00,123.45	$ZZ,ZZZ.99CR	$00,123.45
99,99CR	098.76CR	***.99DB	098.76CR
ZZZZZZ	3 21	ZZZZZZ	321

10. Editing is not allowed in the Working-Storage, Common-Storage, or *Common-Storage except in GECOM source programs that have been generated by the COBOL Translator.

FUNCTION

The function of the array Section is to define the size and number of the subscripts. The actual number of times a data-name is to appear is noted in the repeat columns after the data-name. A description of the subscripts in the Array Section reflects the arrangement of the array (as pictured by the programmer). Thus, A(10), B(5, 10), C(2, 4, 3) indicates A is a list of 10 items, B is a matrix of 50 items arranged in 5 rows with 10 items to each row, and that C is a three-dimensional matrix consisting of 2 rows of 4 columns each, repeated on 3 planes.

CONVENTIONS

1. Array names (and their subscripts) of one-dimensional arrays (lists) need not be described in the Array Section, but may be included for reference purposes. Arrays of two and three dimensions must be described.

2. The names should be qualified if necessary.

3. No more than three dimensions may be ascribed to any homogeneous array.

   Data names listed in the Array Section with their dimensions are described as repeated fields elsewhere in the Data Division or may be "tacit" entries if computation mode is floating point. (See GECOM II Operations Manual, Edited List, Tacit Data Division Entries.)

4. Only one subscript may be ascribed to a nonhomogeneous array, (a repeated group of fields).

FUNCTION

A constant is any data whose value does not change within the program. Any such value may be listed in the Constant Section and named so that it may be referred to by name rather than writing out its value each time it is used. The other advantage of naming and listing the values of constants is that in case of a change of assigned value, only the entry in the Constant Section need be changed; the necessity of changing the actual value several times within the Procedure Division is eliminated.

CONVENTIONS

1. The only permissible entry in the Type columns is FL. (See File Section, Field Literal, Input Entries). If no entry is made in the Type columns, FL is assumed.

2. Repeated field literals may be included in the Constant Section.

3. The use of the Format, Binary, and Justify columns is necessary only when the entry is to be used in an output record with no change to be applied.

4. The mode of computation is implied by the data image.

5. The actual value of the constant is entered in the Data Image columns. Assumed (V) decimal points may not be shown in constant data images. The symbol M may be used as the first character in data images. See explanation under "Nonstandard Data."

EXAMPLES OF CONSTANTS

GENERAL ELECTRIC — GENERAL COMPILER DATA DIVISION FORM

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

PROGRAM
PROGRAMMER — COMPUTER

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	SIGN	REPEAT	BLANK	JUSTIFY	BINARY	ELEMENT POSITION	DATA IMAGE
	CONSTANT SECTION.									
	FL	CON1								"A"
	FL	CON2								100
	FL	CONARAY			005					1
	FL									2
	FL									3
	FL									4
	FL									5
		CONHEADS			003					"HEADING FOR REPORT ONE "
	~									"HEADING FOR REPORT TWO"
	~									"HEADING FOR REPORT THREE "
		CON3								1234.5
		CON4								-3.1
		CON5								333E33
	FL	CON6								123.45
	FL	CON7								+10
		CON8								-10
		CON9								"ABCDEF HIJK"
	FL	CON10								"ALL KINDS OF CONSTANTS"

CA 14 3 62

FUNCTION

The function of the Integer Section is to identify fields (such as subscripts) referred to in the Procedure Division which may have integral values, and to identify fields to be carried internally as integers.

CONVENTIONS

1. This section is necessary only when the computation mode is floating point to identify fields to be carried internally as integers. Regardless of computation mode, fields in Common Storage and *Common Storage which are to function as integers must be declared in this section.

2. Fields described as integers in the Working Storage and Constant Sections need not be mentioned in the Integer Section. The data image defines the internal storage mode in these sections.

3. When computation mode is floating point, input file fields requiring process storage will be stored internally as floating point numbers unless names in the Integer Section.

TRUE ~ FALSE SECTION

FUNCTION

This section specifies those fields that assume only two values: the decimal value 1 which represents a TRUE condition, and the decimal value 0 which represents a FALSE condition.

CONVENTIONS

1. These data-names may or may not appear elsewhere in the Data Division. Contained in the true-false section are those data-names which are used to direct the logical flow, or which are used as a two-way switch. A true-false field must be numeric and only one character in length.

2. A true-false data-name may be used in any procedure having data-names as operands, and may be a field with qualification if necessary.

3. Care must be exercised in the Procedure Division that the value of the true-false data-name is set before it is tested. When tested with an IF sentence, a value of 1 causes control to be passed to the sentence indicated in the IF sentence; if the value if 0, control passes to the sentence following.

4. When setting or resetting the value of true-false data-names, multiple assignment is permissible. However, do not mix assignments of true-false data-names and data-names. (See Example No. 3).

5. List all control console switches that appear in the Procedure Division READ, Option 1, statments. The control console switches are to be listed in order from console switch number 1 through switch N, where N is the highest numbered console switch referenced in the READ statements.

EXAMPLES

1. TRUE~FALSE SECTION. FLAG~1, SWITCH~B.
   IF FLAG~1 GO TO SENTENCE~22.

   When FLAG - 1 = 1, control is transferred to Sentence - 22.
   When FLAG - 1 = 0, the next sentence is executed.

2. NEW~GROSS = GROSS - SWITCH~B*FICA.

   When SWITCH - B = 0, no deduction will be made from GROSS.

3. Flag - 1 = Flag - 2 = Flag - 3 = 0. (Correct)
   Flag - 1 = Flag - 2 = Data-Name - 1 = Data-Name - 2 = 0. (Incorrect)

FUNCTION

The Working-Storage Section provides for the allocation of memory to data required for intermediate processing during execution of a segment of a program; that is, data saved from one record to another, or intermediate results.

CONVENTIONS

1. When computation mode is fixed point, any intermediate results generated in the Procedure Division as well as any conditional names or field literals which are not described in the Input, Common Storage, *Common Storage, or Constant Sections must be described so the storage location of the appropriate size may be assigned at compilation time. This is accomplished by a Working Storage Section.

2. When the computation mode is floating point, it is not necessary to describe (intermediate) calculated data with a Working-Storage Section. The compiler generates the Working Storage for this data as standard (double-length) floating point fields. Working Storage in this case is used to store any incoming data that must be available for use throughout the program. When the computation mode is floating point, the compiler assumes that any field without a data image has a floating point description. No error messages are printed.

3. Data in the Working-Storage Section need not be grouped into records and files, but may be when it is convenient to do so (as when moving input to a holding area record-by-record).

4. Following are types of data which may be described in the Working-Storage Section:

   R - See File Section, Record, Input and Process Output Entries,
   *G - See File Section, *Group, Input Entries,
   G - See File Section, Group, Input and Process Output Entries,
   F - See File Section, Field, Input and Process Output Entries,
   E - See File Section, Element, Input and Process Output Entries,
   C - See File Section, Conditional Names, Input and Process Output Entries,
   FL - See File Section, Field Literal, Input Entries,
   T - See File Section, Terminate, Input and Process Output Entries.

5. The use of the Format, Binary, and Justify columns is necessary only when the entry is to be used in an output record with no change to be applied.

6. The Working-Storage abbreviation WS must be used as a qualifier when necessary for uniqueness. An output record which is a direct reflection of a Working-Storage record must be qualified by WS even when the name is unique.

7. The mode of computation is implied by the data image.

8. A field literal in the Working-Storage Section causes the field to be initially set to the value shown in the data image. The value may be changed by placing a new value in the field during processing.

9. Regardless of computation mode, any alphabetic or alphanumeric data must be described.

10. The symbol M may be used as the first character of a data image. See explanation under "Nonstandard Data."

EXAMPLES OF WORKING-STORAGE

GENERAL ELECTRIC
COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER DATA DIVISION FORM

PROGRAM
PROGRAMMER
COMPUTER

SEQUENCE NUMBER	LEVEL	DATA NAME	QUALIFIER	REPEAT	ELEMENT POSITION AS	ELEMENT POSITION LS	DATA IMAGE
	WO	WORKING~STORAGE SECTION.					
	FL	FIELD~1					"ABCD"
	FL	FIELD~A		003			"ABCD"
	FL						"EFGH"
	FL						"IJKL"
	F	FIELD~4		004			999.9
	F	FIELD~5					-999.9
	G	GROUP~1					
	F	FIELD~6					XX
	F	FIELD~7					999
	G	GROUP~2					
	F	FIELD~8					A(20)
	F	FIELD~9		009			9(9)
	R	WSREC~1					
	F	FIELD~10					99
	F	FIELD~11					X(10)
	F	FIELD~12					9(5)
	F	FIELD~13					A(13)
	F	FIELD~14					9
	C	RED					1
	C	PINK					2
	C	GREEN					3
	F	FIELD~15					XX
	C	COPPER					"CP"
	C	LEAD					"LD"

CA 14 (1) 621

FUNCTION

To provide for the allocation of memory to data required for processing during execution of more than one segment of a program.

CONVENTIONS

1. Data are described in the same manner as in the Working-Storage Section.

2. Data described in the Common-Storage Section will be assigned memory addresses relative to the fixed address specified in the "Begin Common~Storage" clause in the Environment Division. If this address is not specified, the initial address will be 4063 for a 4k memory or 8159 for 8k and 16k memories. The assignment takes place in descending order with entries assigned to even locations. The number of words assigned to a data name is dependent on its description.

3. When two or more relocatable segments share Common Storage, the data image entries in each Common-Storage section must be identical and in the same sequence. It is suggested that data names be identical for clearness and documentation.

4. Following are types of data which may be described in the Common-Storage Section:

   R - See File Section, Record, Input Entries,
   *G - See File Section, *Group, Input Entries,
   G - See File Section, Group, Input Entries,
   F - See File Section, Field, Input Entries,
   C - See File Section, Conditional Names, Input Entries,
   FL - See File Section, Field Literal, Input Entries.

   NOTE: Elements may not be used in Common Storage.

5. The use of the Format, Binary, and Justify columns is necessary only when the entry is to be used in an output record with no change to be applied.

6. The Common-Storage abbreviation CS must be used as a qualifier when necessary for uniqueness.

7. The mode of computation is implied by the data image.

8. A field literal entry in the Common-Storage Section will cause the field to be initially set to the value shown in the data image. The value may be changed by placing a new value in the field during processing.

9. Common and *Common Storage are not preblanked for alphanumeric or prezeroed for binary data. For the purpose of comparisons, it is necessary to insure that any unused character positions in Common Storage words are preset to the value (blank or zero) desired. This may be accomplished by moving SPACES as explained by convention 8 of the MOVE verb. The move of SPACES is done as full words.

EXAMPLES OF COMMON~STORAGE

GENERAL ELECTRIC
COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER DATA DIVISION FORM

PROGRAM
PROGRAMMER
COMPUTER

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	REPEAT	ELEMENT POSITION	DATA WORD
	C	COMMON~STORAGE SECTION.				
	FL	FIELD~1				"ABCD"
	FL	FIELD~A		003		"ABCD"
	FL					"EFGH"
	FL					"IJK "
	F	FIELD~4		004		999.
	F	FIELD~5				-999.9
	G	GROUP~1				
	F	FIELD~6				XX
	F	FIELD~7				999
	G	GROUP~2				
	F	FIELD~8				A(2)
	F	FIELD~9		009		9(9)
	R	CSREC~1				
	F	FIELD~10				99
	F	FIELD~11				X(10)
	F	FIELD~12				9(5)
	F	FIELD~13				A(13)
	F	FIELD~14				9
	C	RED				1
	C	PINK				2
	C	GREEN				3
	F	FIELD~15				XX
	C	COPPER				"CP"
	C	LEAD				"LD"

CAT# 11-621

FUNCTION

The *Common-Storage Section provides for the allocation of upper 8k memory to data required for processing during execution of one or more segments of a program.

CONVENTIONS

1. Object Computer memory size must be 4 modules (16k).

2. Data described in the *Common-Storage Section is assigned memory addresses relative to the fixed address specified in the BEGIN *COMMON~STORAGE clause in the Environment Division.* If this address is not specified, the initial address will be 8190 of upper 8k memory. The assignment takes place in descending order with entries assigned to even locations.

3. When two or more relocatable segments share *Common Storage, the data image entries in each *Common-Storage Section must be identical and in the same sequence. It is suggested that data names be identical for clearness and documentation.

4. Repeated numeric fields are the only type of data that may be described. See File Section, Field, Input Entries.

5. The use of the Format, Binary and Justify columns is necessary only when the entry is to be used in an output record with no change to be applied.

6. The *Common-Storage abbreviation CS must be used as a qualifier when necessary for uniqueness.

7. The mode of computation is implied by the data image.

EXAMPLES OF *COMMON~STORAGE

Type	Data Name	Repeat	Data Image
F	Field~1	003	999
F	Field~2	003	999V99
F	Field~3	003	+999.99
F	Field~4	003	+999K99
F	Field~5	003	+9.99999999E+99

* The lower bank of 8192 words is referred to as "lower 8k," and the upper bank of 8192 words is referred to as "upper 8k."

OVERFLOW CONDITION

In processing the Data Division a data-name table is created. This table consists of data-name in the Report Array, True-False, Integer, File, Working-Storage, Common-Storage, *Common-Storage, and Constant Sections. The data-name table is used to match data names referenced in the Procedure Division with those described in the Data Division. Since the table is fixed in memory, its capacity may be exceeded before all data-names in the Data Division are processed. When this occurs, the table is considered to be in an "overflow" state. The overflow limit is reached when the compiler processes approximately 200 data-names. The sequence for processing and entering data-names in the table is as follows:

1. File and record-names from the Report Section, and special field names (such as ACC. COUNT) created.
2. The file-names and record-names in the output files portion of the File Section.
3. All the data-names in the input files portion of the File Section.
4. All the data-names in the Working-Storage Section.
5. All the data-names in the Common-Storage Section.
6. All the data-names in the *Common-Storage Section.
7. All the data-names in the Constant Section.
8. All the data-names in the Array Section that do not appear in other sections of the Data Division and then only if mode of computation is floating point.
9. All the data-names in the True-False Section that are not described in other sections of the Data Division.
10. All the data-names in the Integer Section that are not described in other sections of the Data Division.

Overflow is indicated with a typewriter message of WTV and a printer message of 017 with an indication of where in the Data Division the overflow occurred. Indication is as follows:

Data Division Section	Relative Position	Indicator
FILE	+XXX	File-name being processed, record-name being processed.
WORKING~STORAGE	+XXX	WS, record-name, if any.
COMMON~STORAGE	+XXX	CS, record-name, if any.
*COMMON~STORAGE	+XXX	CS
CONSTANT	+XXX	CONSTANT
REPORT SECTION		Report-name or reports file name.

To anticipate and to avoid overflow, the programmer may indicate overflow points by inserting parameter cards in the Data Division prior to compilation. These parameter or Type D cards cause the compiler to terminate the data-name table, write it on tape and begin building a new table from the data-name following the Type D card. The Type D card must contain D in columns 7, 8, and 9. It may have a sequence number in columns 1 through 6 and comments in columns 11 through 80. Comments may not be continued on next card.

Since Type D cards terminate entries in the data-name table and avoid the overflow error halt, they must be inserted after approximately every 250 data-names appearing in the Data Division. Their exact place of insertion is determined by counting the data-names in the Data Division and inserting a Type D card every 250 data-names (approx.) at the following points:

DATA DIVISION SECTION	INSERTION POINT
File	Immediately prior to an FD statement.
Working Storage	Immediately prior to a record or group where the group is not part of a record or immediately prior to a field when there are no records or groups.
Common Storage	Same as Working Storage.
*Common Storage	Immediately prior to a field.
Constant	Immediately before any field literal that is not part of an array.
Procedure Division	Immediately prior to the Procedure Division sentence if the Computation Mode is floating point and the data-names in the Array, True-False, and Integer Sections are not described in the File, Working-Storage, Common-Storage, *Common-Storage, or the Constant Section.

Inserting Type D cards does not appreciably increase compilation time.

REPORT SECTION

Type D cards are not allowed in the Report Section.

If the overflow occurred on a single report, inspect the number and length of unique ACC and COUNT names. A limit of 30 names is allowed. Shortening of these names (including FOR clauses) may allow more than fifty names and eliminate the overflow.

If the overflow occurred on a file of several reports for deferred printing, the reports may be assigned to two or more output tapes instead of one. This requires only the addition of an RFD entry in the Report Section and an assign clause in the Environment Division. ACC or COUNT names may also influence the overflow of combined reports in the manner described above for a single report.

If these solutions do not solve the problem, please forward documentation of the source program to the Computer Department for analysis.

6. PROCEDURE DIVISION

PURPOSE

The Procedure Division specifies the steps the programmer wishes the computer to follow. These steps are expressed in meaningful English words and sentences using ordinary verbs to denote the actions to be taken in the sentences.

ORGANIZATION

The Procedure Division is written on the GECOM Sentence Form and is identified by the heading:

PROCEDURE DIVISION.

The heading is written beginning in Column 8 and is terminated by a period. No other information may appear on the line containing the heading.

The Procedure Division is composed of two parts: a body of sentences called the main program and sentences grouped into Sections, which act like subroutines or subprograms. In preparing the Procedure Division for compilation, Sections, if used, are to precede the main program.

SECTIONS

Sections are ordered sets of sentences having a common function and needing to be executed from more than one place in the main program. The programmer is free to partition a program into sections as he chooses. When doing so, however, he must prepare them as follows:

```
{section-name} SECTION.

Head   [ INPUT data-name-I1 data-name-I2 ...   .
         OUTPUT data-name-O1 data-name-O2 ...  .
         NOTE ...   .                          ]

       BEGIN.

Body   [   One or more procedure sentences.    ]

       END section-name SECTION.
```

The "section-name" is used to identify and to make reference to the section. It is like a sentence name and may be formed from the characters and conventions used to form sentence names. The word SECTION alerts the compiler of the section's presence.

The information following the section's name is partitioned into a "head" and a "body."

The body of a section is treated as a "closed procedure" or subroutine and may be executed only by the Perform sentence. A GO TO sentence or clause cannot be used as a transfer to a section. The word BEGIN acts as the entrance point to the section and the word END as its exit point. Since a section results in a closed body of coding, no GO TO sentence or clause (as a result of the IF sentence) may transfer control to sentences or other sections outside of a section. To provide a common exit point, the word END may be given a sentence name for transferring from within the section to a sentence in the main program which is always the sentence following the executing Perform sentence.

However, a section may contain Perform sentences which execute other sections. This effect is referred to as "nested" sections or subroutines.

Since subroutines often require input data before execution and yield output data as a result of their execution, the section head serves to accommodate this function. The data-names following the word INPUT are data-names used in the sentences of the section body requiring values before the section can be executed. Those data-names following the word OUTPUT indicate the results of executing the section. To get data to and from a section, the programmer may use the following option of the Perform sentence:

PERFORM {section-name} SECTION USING

data-name-1 data-name-2 ... GIVING

data-name-n1 data-name-n2 ... .

In using this option, the value of data-name-1 is moved to data-name-I1 of the section, data-name-2 to data-name-I2, etc., before the section body is executed. When the data transfer is completed, the body of the section is then "performed." After the sentences of the body are executed, for example, upon reaching the exit END, but before transferring control to the main program, data-name-O1 is moved to data-name-n1 of the Perform sentence, data-name-O2 to data-name-n2, etc. It is only when this latter data transfer is completed that control reverts to the main program.

The data descriptions of the data-names appearing in the Perform sentence must be compatible to those given for the input/output data-names in the section. The data-names of Perform, Section, Common Storage, and *Common Storage may be:

Record names	Array names
Group names	Element names
Field names	True-False variables

described in either input files or working storage. Data names of the USING clause and those listed as output in the section may be constants. In no case can any of the data names listed as input or output of a section be subscripted elements of an array. However, the USING and GIVING clauses of the Perform may contain subscripted fields.

When a section is executed by the **PERFORM...USING...GIVING** option, the section must appear before the **PERFORM** sentence in the Procedure Division.

SEGMENTS

Segments are subprograms which are compiled and tested independently and subsequently loaded together and executed as a total program. Thus, a user may decide to break up a large complex program into several parts or segments, write each one as a separate source program, and compile and test each segment independently, thereby overlapping programming and checkout time. Another use of segments is to facilitate the writing of common subroutines (installation oriented) in source language to be compiled and tested once and then included in many programs.

All source programs processed by the GECOM-II system and later systems are compiled as segments. However, a total program can still be compiled and executed as a complete entity.

Segments are executed from other segments by means of the PERFORM verb with the following format:

PERFORM {segment-name} SEGMENT.

The PROGRAM-ID contained in the performed segment's Identification Division is used as the segment-name.

A segment's END PROGRAM sentence is treated as the exit of the segment. The END PROGRAM sentence may be named so that control may be transferred to the exit point from within the segment. When a segment is executed, the exit is set and control is transferred to that part of the object program corresponding to the first sentence in the segment's Procedure Division. If there are no logical control transfers within the segment, sentences are executed in sequence down through the exit. Only the main segment may contain the logical end of the program (see STOP verb).

Segments may contain Sections. Segments may be nested in the same manner as Sections. However, Sections contained within a Segment may not be performed from outside of the Segment.

At times, it may be desired to perform a set of procedures embedded within the main (control) segment or within some other segment from outside of the segment containing the procedures. The procedures to be performed may be written much like a section. However, instead of being preceded by section-name SECTION

and followed by:

END {section-name} SECTION.

such a set of procedures should be preceded by

{procedure-name} SEGMENT.

and followed by:

END {procedure-name} SEGMENT.

Note that the procedure-name specified should not be the PROGRAM-ID (which is used when it is desired to execute an entire segment). All of the rules for sections apply except that the USING...GIVING options may not be employed. The set of procedures is compiled as a section, but the procedure-name may be specified in a PERFORM sentence in the main segment or any other segment.

Communication from segment to segment can be accomplished by data in Common and/or *Common Storage. The descriptions of Common and/or *Common Storage should be identical for all segments in total program. It should also be kept in mind that Working Storage and other Process Storage areas are compiled separately for each segment and only Common and *Common Storage may be shared by two segments.

The continuity of the object program depends upon procedures contained in the main segment. The main segment should contain the input/output housekeeping routines and the logic to determine when data should be read or written. Certain input/output procedures may sometimes be included in individual functional segments rather than in the main segment; examples are high-speed printer and punched card output routines. Usually, however, the buffers and housekeeping routines for input/output files should be in the main segment, particularly for magnetic tape and DSU files.

Suppose, for example, that segment A reads a file from tape controller plug 1, and segment B writes a file on tape plug 1. Both files are buffered to overlap input/output with computing. Whenever A starts a read, it resumes processing in parallel with the tape operation. If the controller should detect a tape error or end of file, the signal must be communicated to the input routine in A. But if B needs to issue a write before the next read from A, then B will detect the error indications. They are of no use to B, and it is impractical to have B pass them back to A. Thus A loses information it requires, while B receives information it cannot interpret.

The solution is to have a single input/output supervisory routine in the main segment, and let both A and B call to it through transfer vectors. Not only is the error control problem thereby eliminated, but a memory saving also results; it is unnecessary to have two or more copies of a sizable input/output package in memory at once. The conclusions would be the same whether A reads and B writes or both read or write; and the equivalent problem with two files both open on the same disc storage unit or plug is even more serious.

The problem just discussed may be termed that of shared peripheral devices. Another type of problem results from shared logical files. Suppose, for example, that segments A and B jointly process a card input file, and that either A or B may determine when another read is needed. It is clearly desirable to have a centralized card input subroutine to which both A and B have access. The solution is either to include the subroutine in the main segment, or else to make it a segment in itself, and in either case to permit A and B to call it through transfer vectors.

In general, a functional segment should not include the buffers and housekeeping routine for any file unless--

1. It is opened, processed, and closed exclusively within the segment.
2. Other segments do not need access to its buffers.
3. It does not share a magnetic tape controller or a disc storage unit or controller with a file processed by another segment.

In a segmented GECOM program, all magnetic tape and/or DSU files should be described in the Data and Environment Divisions of the main segment, and all OPEN, READ, WRITE, CHAIN, READY, RELEASE, and CLOSE sentences referring to these files should appear in its Procedure Division. Isolated punched card or high-speed printer files may be processed in individual functional segments.

To test a particular segment, it is necessary to execute its object coding in conjunction with the object coding of the main segment or a dummy segment written only for test purposes to make input test data available to the segment and to produce output test results.

During compilation of any segment, object program subroutines can be punched out optionally by the Editor phase. If this option is exercised for more than one segment, duplicate subroutines would be included in the total program. However, most of the subroutines required by the total object program are required by the segment containing the input/output functions. It is therefore recommended that object program subroutines be punched out only with the main segment (or with a dummy segment for test purposes). The names of subroutines required in each segment (contained in the Edited List of each segment) can then be checked against those in the main segment to determine if there are any omissions. If any additional subroutines are required, they can be selected from the library and placed in the object program deck.

After testing all segments and obtaining any required additional subroutines, the total object program should be consolidated as follows:

1. Remove the constant cards and transfer card from the end of each segment object deck.

2. Place segment object decks behind main segment object deck and subroutines in any order. If desired, subroutines could be placed between or behind segments. Order is important only when overlay segments have been compiled. (See OVERLAY SEGMENTATION.)

3. Place any one of the constant decks and transfer cards at the end of the complete deck.

4. Depending on the object program loading medium and format, perform one of the following:

   a. To be loaded from cards--place Multicapability Modular Loader II (MCML II), CD225B1.006R, with a type 1 card in front of complete deck, and two blank cards at end of complete deck.

   b. To be loaded from magnetic tape in relocatable format--place MCML II Loader with a type 1 card in front of complete deck. Prepare BRIDGE II control cards and make a BRIDGE II run to place program on magnetic tape. (See CD225J1.001.)

   c. To be loaded from magnetic tape in absolute format--perform a BRIDGE II RAB run on the deck. The RAB control card indicates whether the user is supplying an absolute loader, or wishes one to be emitted by RAB. (See BRIDGE II, RAB function, CD225J1.001.)

When the object program is loaded, control is transferred to the first (main) segment.

OVERLAY SEGMENTATION

An object program segmented for overlays consists of a "Control Load" plus one or more "overlay segments" (overlays). The Control Segment (main segment) is brought into memory at the beginning of the program execution, and it remains in memory throughout execution of the entire object program. The Control Segment consists of one or more ordinary segments and the subroutines they require. The overlays are not brought into memory with the Control Segment; instead, they are brought in individually, under control of explicit LOAD sentences in the Control Segment. Each overlay consists of one or more ordinary segments.

The object of overlay segmentation is to use the same memory area for two or more sets of procedures which are not needed at the same time. In this way, it is possible to subdivide very large programs so that all of the procedures needed at a given time can be contained in memory, even though the entire program might greatly exceed memory if it all had to be contained at once.

The following conditions characterize the simplest application of overlay segmentation:

1. The Control Segment:

    a. Contains all input and output procedures
    b. Contains all subroutines it requires, plus all other subroutines shared by the overlays
    c. Contains logical procedures to determine when each overlay is needed
    d. Loads each overlay into memory when it is needed
    e. Performs each overlay as required

2. The overlays:

    a. Contain all procedures except those just described in the Control Segments
    b. Communicate only with the Control Segment and Common and/or *Common Storage, not with each other
    c. Occupy memory only one at a time (when another is needed, it is loaded over the old one)
    d. Are located in memory immediately after the Control Segment and thus may use all of memory up to the beginning of Common or *Common Storage.

In a more complicated application, it might be desired to hold two or more overlays in memory at once and to replace them selectively with other overlays as required, and to let overlays perform each other. To accomplish this, the programmer may use an Environment Division option to relocate each overlay an arbitrary number of words away from the end of the Control Segment. For example, assume overlays A and B are to be in memory together, and that A requires 600 (or fewer) words of memory. (The amount of memory required by A is determined from an early compilation of it, or else estimated on the basis of the programmer's experience with object program memory requirements.) The programmer should instruct GECOM to RELOCATE B BY 600 WORDS. This causes B to be positioned 600 words after the last location used by the Control Segment, so that A can fit in between. It is then entirely arbitrary whether A or B is loaded first, or whether one is actually loaded immediately after the other. Assuming

that C is another overlay, and that it, like A, requires no more than 600 words, and must be in memory with B but not with A, C can replace A when it enters memory, as follows: The programmer RELOCATES B BY 600 WORDS, but does not RELOCATE A or C; and the processing sequence in the Control Segment might be:

```
      .
      .
      .
LOAD A.
      .
      .
      .
LOAD B.
NOTE A AND B ARE BOTH IN MEMORY.
      .
      .
      .
(use A and B with PERFORM sentences)
      .
      .
      .
LOAD C.
NOTE A HAS NOW BEEN REPLACED BY C.
      .
      .
      .
(use C and B with PERFORM sentences)
```

This framework can be extended to permit several overlay segments to be held in memory at once. In effect, the Control Segment is automatically given exactly the amount of memory it requires; and the remaining memory is divided into portions of fixed size by suitable RELOCATE clauses. The general rule is that if two overlays must be in memory at the same time, then one of them must be relocated by enough words to contain the other. If n overlays must be in memory at the same time, the second must be relocated by enough words to hold the first, the third relocated by enough words to hold the first two, and so forth.

When the object program is operated, the Control Segment will be loaded into memory at the beginning, but no overlays will be loaded with it. Thus, the Control Segment must load each overlay before performing it the first time. Should an overlay be performed before it is loaded (or after another has been loaded over it), unpredictable results will occur. It is, therefore, necessary for the programmer to plan the dispatching of the overlays very carefully.

The Control Segment is in memory throughout execution of the overlay segmented program, but each overlay is in memory only part of the time. Each time an overlay is reloaded, it enters memory with all of its initial conditions restored. Thus, any GO statements previously altered (within the overlay) are reset to their initial values; and all storage areas other than Common or *Common are reset to initial values for Field Literals and contain unpredictable data for everything except Field Literals. Common and *Common Storage area contents are preserved except that Field Literals are reset to initial values.

Similarly, any data file processed in an overlay must be opened, completely processed, and closed by sentences in the overlay before the overlay is replaced by another LOAD sentence. It is not possible to open a file in an overlay, process part of it, replace the overlay with a different overlay, and then load the original overlay back into memory to finish processing and close the file, because all of the information about the file is lost when the overlay is reloaded, and it is assumed that the file is not yet opened. If a file is needed again after reloading the overlay, it must be opened again. It is recommended that the user define all files and do all opens and closes in the Control Segment. Files processed in overlays are subject to the same rules as files processed in segments. (See Segments.)

In programs utilizing 16k memories, the PLACE clause in the Environment Division permits the main procedural portion of the object program and *Common Storage to be stored in upper memory, while common constants, file tables, subroutines, and buffers are stored in lower memory. Normally, the compiler assumes that the lower and upper portions of the object program should be started at standard beginning locations in memory (576_{10} and 8192_{10}, respectively). For overlays, the normal assumption is that the lower and upper portions should follow immediately after the Control Segment in memory. However, as described above, the programmer may utilize the RELOCATE clause of the Environment Division to override the standard assumptions. The following example illustrates how to use this option to permit more than one overlay to be in memory at once:

Assume that the memory requirements of a Control Segment and its overlays correspond to the following table.

Program	LOWER Memory	UPPER Memory
Control Segment	3024_{10}	2008_{10}
OVERLAY-A	500_{10}	1000_{10}
OVERLAY-B	600_{10}	1500_{10}
OVERLAY-C	500_{10}	1000_{10}

1. When the execution of the object program begins, memory will be used as follows:

   Location 576 - 3599 -- Control Segment (Lower)
   Location 8192 - 10199 -- Control Segment (Upper)
   All other -- Not in use

2. When the Control Segment has executed a LOAD OVERLAY-A sentence, memory will be used as follows:

   Location 576 - 3599 -- Control Segment (Lower)
   Location 3600 - 4099 -- Overlay-A (Lower)
   Location 8192 - 10199 -- Control Segment (Upper)
   Location 10200 - 11199 -- Overlay-A (Upper)
   All other -- Not in use

3. Assume that the Environment Division for Overlay-B said, RELOCATE BY 500 WORDS IN LOWER MEMORY, RELOCATE BY 1000 WORDS IN UPPER. Then, after the Control Segment has loaded Overlay-A and also has loaded Overlay-B, memory will be used as follows:

```
Location 576   -  3599 -- Control Segment (Lower)
Location 3600  -  4099 -- Overlay-A (Lower)
Location 4100  -  4699 -- Overlay-B (Lower)
Location 8192  - 10199 -- Control Segment (Upper)
Location 10200 - 11199 -- Overlay-A (Upper)
Location 11200 - 12699 -- Overlay-B (Upper)
All other              -- Not in use
```

4. Assume that Overlay-B has been relocated as above, and that the Control Segment has loaded Overlay-A, then loaded Overlay-B, and then loaded Overlay-C. Memory will be used as follows:

```
Location 576   -  3599 -- Control Segment (Lower)
Location 3600  -  4099 -- Overlay-C (Lower)
Location 4100  -  4699 -- Overlay-B (Lower)
Location 8192  - 10199 -- Control Segment (Upper)
Location 10200 - 11199 -- Overlay-C (Upper)
Location 11200 - 12699 -- Overlay-B (Upper)
All other              -- Not in use
```

The main use of the RELOCATE option is to permit overlays to share memory, as the above example shows. In addition, it could be used on the Control Segment to set aside memory for an external executive routine, or for other purposes of similar nature. Individual overlays can be placed entirely in lower memory if desired, even though a 16k memory is in use. The programmer has explicit control over all such placing and relocating, and with the control goes the responsibility to arrange things so as never to destroy needed information or procedures.

Despite the fact that they are compiled separately, each overlay shares the subroutines of the Control Segment. If the overlay includes subroutines which already appear in the Control Segment, they will automatically be dropped, instead of entering memory. The opposite is not true; the Control Segment must contain all of its own subroutines. Overlays which are in memory together can share common subroutines--and thus save memory--if and only if the common subroutines are contained in the Control Segment. The programmer can enhance this economic effect by deliberately placing all common subroutines in the Control Segment, even though they might not be used by it.

To understand how duplicate subroutines are automatically dropped instead of being loaded into memory, it is necessary to consider the steps involved in preparing an overlay segmented object program. The programmer writes separate GECOM source programs for the Control Segment and each overlay, and compiles and perhaps even tests them separately. Finally, he collects them all together, following the instructions in the BRIDGE II Manual (CD225J1.001). In doing so, he may take care to place all common subroutines with the Control Segment. Now the entire object program is processed by the BRIDGE II RAB (Relocatable to Absolute) converter, which supplies linkage between the Control Segment and all of its overlays, and writes the final version to a specified magnetic tape handler. As it processes the program, RAB checks the subroutines included with each overlay against those included with the Control Segment. When RAB finds

duplicates, it links the overlay to the Control Segment subroutines and skips past the duplicate copy without writing it out. Thus, even though duplicate subroutines might be present in the input to RAB (that is, in the collection of Control Load and overlay object programs), they will not be physically present in the output from RAB, which represents the actual running program. RAB does not eliminate duplicate subroutines between overlays, since it cannot assure that Overlay-A and its subroutines will necessarily be in memory when Overlay-B needs the subroutine. As has been mentioned above, the programmer may accomplish this economy by including all common subroutines with the Control Segment.

The BRIDGE-to-DSU Absolute Load Translator (BRAT), CD225E2.005R, may be used to transform the overlay segmented object program from the output magnetic tape onto disc storage. When the overlays have been assigned to disc storage, the Control Segment Edited List will show the overlay segment names in the order in which they must be introduced to BRAT. Each overlay is also assigned a directory table address. At object time the DSU address of an overlay to be loaded will be found in that overlay's directory table position.

When the overlays segments are on magnetic tape, each LOAD procedure begins with a search for the named segment. The programmer can optimize tape searching time by arranging the segments in ascending sequence by name. The LOAD routine reads the next segment name from tape. If the desired segment name is less than the one read, searching begins in a backward direction; otherwise searching proceeds in a forward direction. If either end of the program is reached, the direction of searching is reversed and searching continues.

Communications between an overlay and the Control Segment are always accomplished by PERFORM sentences and by data in Common or *Common Storage. The descriptions of Common and *Common Storage should be identical for the source programs of the Control Load and all of the overlays. One overlay can perform another only if the Control Load also performs it. However, the PERFORM sentence from the Control Load need never be executed. It is only necessary that the PERFORM sentence appear in the Control Load Procedure Division to establish the necessary transfer vector linkage.

A set of procedures contained within a Control Segment or within an overlay segment may be performed from outside of the segment containing the procedures in the same manner as described under "Segments." However, one overlay cannot perform a set of procedures contained in another overlay unless the Control Segment contains a perform (not necessarily executed) of the same procedures in order to establish the necessary linkage.

SEGMENT AND SUBROUTINE TABLE DESCRIPTION

This table is comprised of three parts.

1. The count for the Segment portion entries and the count for the Subroutine portion entries.

2. Five-word Segment names, not to exceed fifty, left in memory for the Reformer Overlay of the Reformer Phase. The first word is the internal symbol assigned by the compiler and attached to the four-word name assigned by the source programmer.

3. One word Subroutine names, not to exceed fifteen if fifty Segment names were used, placed into the table by direction of the Generator Overlays during the Reformer Phase.

MEMORY LOCATION	Discussion of Table Item Entries
5464	Count of Segments, Overlays, and Multiple Entry Points contained in the table. Maximum number of names allowed is fifty. Over fifty will cause typed messages to occur at beginning of Reformer Overlay specifying, XXX SEG TBL OVF. Look for SEG * TOO MANY on printer listing for error notice.
5465	Count of Subroutines names called by this compilation and placed in table entered here during Reformer Phase.
5466	Three types of entries, five words each, may be made to the Segment table beginning at this location. (a) Perform Program-ID Segment The names used for Program-ID are always entered to the table first. These names will be used in generating vectors for executing the Segment. The first word left of the four-word Program-ID name is the assigned internal symbol. The sign and one bit of this word will be off to indicate Perform type entries. (b) Procedure-Name Segment These names are always entered in the table after Perform types. They are known as Embedded Segments and represent multiple entry points into their containing Segment. Their internal symbol word will have the one bit on. (c) Load Program-ID Segment These names, known as Overlays, always enter in the table after the Perform and/or Embedded Segment type entries. Their internal symbol word will have the sign bit on.
6060	The CON Subroutine name is inserted into the first available word adjacent to the last word of the last Segment, Multiple Entry, or Overlay name placed into the table. The Reformer will always make this entry to the table during the Reformer Phase. All other Subroutine names entered to the table by the Reformer are directed by the generators. If fifty Segment Table names are used, only fifteen subroutines may be generated; otherwise, compilation will give unpredictable results.

NOTATIONS IN SENTENCE FORMATS

The following notations are used in this manual to facilitate presentation of the sentence formats.

KEY WORDS	are underlined, upper case words Key words are required to complete the meaning of the sentence. They must be correctly spelled.
NOISE WORDS	are upper case words (not underlined). Noise words are optional. If they are used they must be correctly spelled.
Operands	are lower case words. They indicate the types of operands supplied by the user.
{Choices}	are enclosed in braces. The programmer should select one entry from those shown within a set of braces.
[Options]	are enclosed in brackets. The programmer may include or omit these entries. In some cases, options have been separated into individual numbered formats.

VERB FORMATS

The GECOM verb formats are described in alphabetical order on the following pages. The list below groups the verbs into their respective categories.

ARITHMETIC	ADD SUBTRACT MULTIPLY DIVIDE ASSIGNMENT
INPUT-OUTPUT	ADVANCE READ WRITE OPEN CLOSE CHAIN LOAD READY RELEASE
PROCEDURE-BRANCHING	GO ALTER PERFORM
DATA MOVEMENT	MOVE EXCHANGE
CONDITIONAL	IF VARY
ENDING	STOP
COMPILER DIRECTING VERBS	ENTER
EXPLANATORY (not compiled)	NOTE

ADD

FUNCTION

The ADD verb adds two quantities and stores the sum in either the last-named field or the specified field.

SENTENCE FORMAT

```
ADD  { numeric-1    }  [ TO  ]  { numeric-2    }
     { field-name-1 }  [ AND ]  { field-name-2 }

     [GIVING field-name-3]        [ROUNDED]

     [IF SIZE ERROR GO TO sentence-name-1].
```

CONVENTIONS

1. If the GIVING option is not present, the last-named field receives the result.
2. Decimal points do not appear in stored fields, and are used only to properly align data before execution of an arithmetic operation.
3. Only a numeric may be used. If a sign (+ or -) is included, it must appear as the most significant character of the numeric.
4. ROUNDED may be used to round off the result before it is stored in the receiving field. If the receiving field is in floating point mode or if the operands are all integers, then rounding is ignored.
5. The SIZE ERROR option may be used to truncate the most significant digits of a number.

EXAMPLES

1. ADD 0.5, RATE OF PAY~FILE GIVING TOTAL.
2. ADD TOTL~RECVD, ON~HAND~QTY. (The result is stored in ON~HAND~QTY.)
3. ADD VALUE~1 OF FILE~A, VALUE~2 OF FILE~B GIVING VALUE~3 OF FILE~C, IF SIZE ERROR GO TO ERROR~RTN.

FUNCTION

The ADVANCE verb slews the printer paper.

SENTENCE FORMAT

ADVANCE file-name [{ integer LINES / field-name LINES / TO TOP OF PAGE }] .

CONVENTIONS

1. Field-name must contain an integer less than or equal to 128. If it is greater than 128, only 128 lines will be advanced.

2. If the options are omitted, the printer page is advanced one line. Therefore, the number of lines specified should be greater than one.

3. When a printer is used, the compiler automatically generates a Line-Count field. The programmer is free to use this field name to interrogate the position of the printer page.

4. TOP OF PAGE clause is defined as "Line-Count equals zero."

5. TOP OF PAGE clause may be used as a conditional name of Line-Count. For example,

   IF TOP OF PAGE OF file-name GO ... .

   is the same as

   IF LINE~COUNT OF file-name EQUALS 0 GO ... .

   If only one printer is used, it is not necessary to qualify LINE-COUNT or TOP OF PAGE.

6. The WRITE verb automatically slews the printer page one line after writing the print line and increases the Line-Count field by one. Any ADVANCE sentence causes additional slewing.

7. LINE-COUNT and TOP OF PAGE clauses must not be described in the Data Division.

8. The OPEN verb does not advance a printer file to TOP OF PAGE.

EXAMPLES

1. ADVANCE PAY~REGISTER 20 LINES.

2. ADVANCE TRANS~FILE X LINES.

   Note that the value contained in field X must be an integer.

3. ADVANCE OUTPUT~FILE TO TOP OF PAGE.

FUNCTION

The ALTER verb modifies a predetermined sequence of operations.

SENTENCE FORMAT

ALTER sentence-name-1 TO PROCEED TO sentence-name-2
[, sentence-name-1 TO PROCEED TO sentence-name-4...] .

CONVENTIONS

1. Sentence-name-1 and sentence-name-3 are names of GO sentences as defined under Option 1 of the GO verb.

2. After a GO sentence has been altered, it will continue to go to the changed destination on every execution until it is altered again.

EXAMPLES

1. ALTER SENT~25 TO PROCEED TO SENT~33.

   Effect of SENT-25 before execution of the ALTER sentence:

   SENT~25. GO TO ERROR~RTN.

   Effect of SENT-25 after the execution of the ALTER sentence:

   SENT~25. GO TO SENT~33.

2. ALTER 777 TO PROCEED TO S~8, SENTENCE~52 TO PROCEED TO SENT~A.

   Effect of sentences 777 and SENTENCE-52 before the execution of the ALTER sentence:

   777. GO TO SENT~7A2.

   SENTENCE~52. GO. (Object program will halt in a loop at this point if this sentence has not been altered prior to execution.)

   Effect of sentences 777 and SENTENCE-52 after the execution of the ALTER sentence:

   777. GO TO S~8.
   SENTENCE~52. GO TO SENT~A.

FUNCTION

The ASSIGNMENT verb evaluates an arithmetic expression and assigns the result of the evaluation to a specified field.

SENTENCE FORMAT

true-false variable = { true-false variable / arithmetic expression / 0 / 1 }

field-name-1 [ROUNDED] = { field-name-2 / arithmetic expression / numeric-1 } [IF SIZE ERROR GO TO sentence-name-1] .

CONVENTIONS

1. An arithmetic expression is a sequence of variables (fields), numbers and functions connected by symbols representing the arithmetic operations add, subtract, multiply, divide and exponentiation.

2. In fixed point evaluations, decimal points are aligned according to the data description of the result variable.

3. The ROUNDED option may be used to round off the result before it is stored in the leftmost receiving field only. If the receiving field is in floating point mode or if the operands are all integers, then rounding is ignored. Note: Operands cannot be all integer.

4. Equal signs may be placed in any position in an expression (except a subscript) providing the receiving field is a single variable (not an expression). Equal signs cannot appear in a subscript (See Examples 6 and 9.)

5. The size error option may be used to truncate the most significant digits of a number.

EXAMPLES

1. GROSS~PAY OF PAY~FILE ROUNDED = HRS~WORKED * RATE - WEEKLY~TAX.

2. QTY~ON~HAND = OLD~QTY + NO~RECVD - QTY~SHIPPED.

3. AVG~INCREASE = (END~PAY - START~PAY) / END~PAY.

4. X = A + B - (C * 54.5) * SIN (A - R23) / 22.35.

5. Z ROUNDED = ABS (RATE * R - SQRT (B2-K))-M ** 3.

ASSIGNMENT
(cont.)

6. YTD~FICA = YTD~FICA + (CURR~FICA = GROSS~PAY *0.03).

7. X = Y = Z = A + B * C/E + SIN (ALPHA).

8. A ROUNDED = B = C = SIGMA (I) + PHI (J).

   NOTE: Only the result stored in field A may be rounded.

9. X = A + B + (C (X) = D + F) is legal

   X = A + B = (C + D) is illegal (an expression equals an expression).

10. W=A+B*C IF SIZE ERROR GO TO OUT~RANGE.

CHAIN

FUNCTION

The CHAIN verb searches an input chained file assigned to disc storage for a record or block which satisfies a specified relation.

SENTENCE FORMAT

CHAIN file~name~1 UNTIL { field~name~1 / element~name~1 / constant~1 } { IS [NOT] GREATER THAN / IS [NOT] LESS THAN / IS [NOT] EQUAL TO / IS UNEQUAL TO / EQUALS / EXCEEDS }

{ field~name~2 / element~name~2 / constant~1 } [USING field~name~3 FOR CHAINING]

[, IF END OF CHAIN GO TO sentence name 1] .

CONVENTIONS

1. A READY sentence for file~name~1 must be issued before the CHAIN ... UNTIL sentence.

2. The Procedure Division may not contain more than four CHAIN UNTIL sentences per file. Note that only one constant may be used in an UNTIL clause. The input file field or element being compared must be in the same position relative to the beginning of each logical record assigned to the file; it must be the same size and mode as the field, element, or constant with which it is being compared. If records are blocked, the condition of the first logical record of the block determines whether the block is accepted, or whether a chain is made to another record block.

3. If a file has one or more CHAIN sentences referencing it, the USING clause must be specified in at least one of the CHAIN sentences for that file. If more than one CHAIN sentence for a file has a USING clause, all CHAIN sentences for that file must have a USING clause.

4. If a file has one or more CHAIN sentences referencing it, the END OF CHAIN clause must be specified in at least one of the CHAIN sentences for that file. Transfer is made to the sentence named when an end of chain is reached and the specified relation is not satisfied. If different sentence names are employed in END OF CHAIN clauses for the same file, every CHAIN sentence for that file must have an END OF CHAIN clause.

5. The field named in the USING clause must contain an absolute DSU address at object time. The field-name may not be subscripted. The field must be an input field contained in all records of the file being chained. The mode and size of the field must be the same in each record description and the field must be in the same position relative to the beginning of each logical record assigned to the file.

6. The abbreviations for relations may be used instead of the English words shown in the sentence format.

7. If bit 0 of the field named in the USING clause is 1, the record block is considered as vacant. The contents of the field are negated to develop the address of the next record block whenever bit 0 is 1. No comparison is made in the vacant record.

8. If bits 0-18 of the field named in the USING clause are all zero, the record block is assumed to be the last one in the chain. If the record block does not satisfy the relation, a transfer is made to sentence-name-1. If bits 0-18 of the field named in the USING clause are all 1-bits, it is assumed that the record block is vacant and that the end of chain has been reached; a transfer is made to sentence-name-1. Note: Bit 19 is not checked, but it must not be a 1-bit unless bits 0-18 are also 1-bits.

9. The contents of the field named in the USING clause of the READY sentence preceding the CHAIN sentence is updated each time a chaining operation is done. When the relation is satisfied, the address of the record block which satisfied the relation is in the field named in the READY USING clause. A read of the file obtains the first logical record. If the file is not blocked, a READY clause must precede the READ sentence.

10. The CHAIN sentence may not be issued for files described by a SEQUENTIAL clause.

11. One of the fields or elements named in the UNTIL clause must be contained in the chained file. Both fields or elements named in the UNTIL clause should not be in the chained file.

EXAMPLE:

1. FILE NOT BLOCKED:

```
READY DSU-IN FOR READING USING DSU-ADDR FOR ADDRESSING.
CHAIN DSU-IN UNTIL STOCK-NO EQUALS TRAN-STOCK USING FIELD-INDIC FOR
CHAINING, IF END OF CHAIN GO TO ERROR-CHAIN.

READY DSU-IN FOR READING USING DSU-ADDR
FOR ADDRESSING.
READ DSU-IN.
```

2. BLOCKED FILE:

```
READY DSU-IN FOR READING USING DSU-ADDR FOR ADDRESSING.
CHAIN DSU-IN UNTIL....
READ DSU-IN.
```

CLOSE

FUNCTION

The CLOSE verb terminates the processing of both input and output reels and files, with optional rewind and/or lock.

SENTENCE FORMAT

CLOSE file-name-1 WITH [NO LOCK] [NO REWIND] [,file-name-2 . . .] .

CONVENTIONS

1. A CLOSE file-name sentence must be executed only once for a given file unless the file has been reopened. A CLOSE file-name sentence will initiate the final closing conventions for the specified file and release its data area.

2. If the NO LOCK option is used on a tape file, the tape will be rewound.

3. If the NO REWIND option is used on an output tape file, the tape will be positioned at the end of the file after the end-of-file conventions are executed.

4. If the NO REWIND option is used on an input tape file, the tape will remain at its current position whether that position is the end of the file or not, unless it is a multifile tape in which case it is advanced to end-of-file.

5. If neither a NO LOCK nor NO REWIND option is specified for a tape file, the tape will be rewound and locked to prevent the tape from being read or written upon. LOCK is accomplished by programming, not hardware.

6. If the same tape file is opened or closed more than once, the NO LOCK option should be used.

7. When an input card file is closed, the current card count will be typed whether the end-of-file card has been detected or not. Any remaining cards in the reader will not be read.

8. When an output card file is closed, the card count will be typed. No end-of-file card is punched.

9. The NO LOCK and NO REWIND options may not be used for DSU files.

10. A JOURNAL~TAPE file (See Environment Division, DSU~CONTROL sentence) need not be closed. If it is not closed, no label or tape mark is written.

EXAMPLE

1. CLOSE PAYROL~FILE WITH NO LOCK, MASTER~FILE WITH NO REWIND, EMPLOY~FILE.

   Note: PAYROL-FILE is rewound (if a tape file) with no lock, MASTER-FILE is closed but remains positioned at its present point and EMPLOY-FILE is rewound and locked.

FUNCTION

The DIVIDE verb divides one number into another and stores the result in the last-named field or the specified field.

SENTENCE FORMAT

```
DIVIDE { numeric-1    }          INTO { numeric-2    }
       { field-name-1 }               { field-name-2 }
    [GIVING          field-name-3]  [ROUNDED]
    [IF SIZE ERROR GO TO sentence-name-1] .
```

CONVENTIONS

1. If the GIVING option is not present, the last-named field receives the result.
2. Decimal points do not appear in stored fields, and are used only to properly align data before execution of an arithmetic operation.
3. Only a numeric may be used. If a sign (+ or -) is included, it must appear as the most significant character of the numeric.
4. The ROUNDED option may be used to round off the result before it is stored in the receiving field. If the receiving field is in floating point mode or if the operands are all integers, then rounding is ignored.
5. The SIZE ERROR option may be used to truncate the most significant digits of a number.
6. The SIZE ERROR option does not check for division by zero.

EXAMPLES

1. DIVIDE NUMBER INTO TOTAL GIVING AVERAGE.
2. DIVIDE 100.0 INTO K2H GIVING VALUE OF FILE~16 ROUNDED.
3. DIVIDE A26 INTO R17K.

   NOTE: The contents of R17K will be divided by the contents of A26, and the result will be stored in R17K.

ENTER

FUNCTION

The ENTER verb allows the programmer to insert General Assembly Program coding into a GECOM source program. It is envisioned that General Assembly Program coding within GECOM source programs will be used primarily for functions more of a machine-oriented level than GECOM sentences. For example, the use of the ENTER verb might allow General Assembly Program coding to sense or manipulate bits, or to tailor GECOM-produced coding to a particular need. (In the latter case, the user must be thoroughly familiar with GECOM object coding and object data organization.)

The ENTER verb specifies departure from the normal sequence of procedures in order to execute General Assembly Program (GAP) coding appearing elsewhere in the Procedure Division.

SENTENCE FORMAT

ENTER GAP [AT GAP-symbol].

(A set of General Assembly Program instructions)

END [.]

CONVENTIONS

1. General Assembly Program coding may appear anywhere in the GECOM Procedure Division. It may be preceded and/or followed by GECOM procedure sentences. GECOM does not change the relative position of General Assembly Program coding within the Procedure Division. A given GECOM source program may contain any number of ENTER sentences.

2. The AT option indicates execution of General Assembly Program instructions appearing elsewhere in the Procedure Division. The General Assembly Program instructions must not follow an ENTER sentence using the AT option. The GAP symbol must appear in Columns 1 to 6 of a line of General Assembly Program coding under another ENTER sentence.

   When the AT option is used, an SPB (using index register 1) to the designated symbol is generated. Index register-1 may be used as an exit parameter. The END instruction should not appear after an ENTER sentence with the AT option.

3. If the ENTER sentence is referenced by a GO statement, the ENTER sentence must be named. When the GO verb is used to execute an ENTER sentence, a BRU to the first General Assembly Program instruction under the ENTER sentence is generated.

4. An ENTER sentence is executed in line if it is not logically skipped by a preceding GECOM procedure sentence or ENTER sentence.

5. The source program name of the ENTER sentence, if present, is converted to a three-character symbolic name. This name is assigned to the first General Assembly Program instruction under the ENTER sentence. The first instruction may also have a Symbol, and in this case an EQU is generated in order to make the assignment.

6. General Assembly Program instructions must be punched in the following card format and may be written on the General Compiler Sentence Form:

   Columns 1 to 6 are reserved for the Symbol.
   Columns 8 to 10 are reserved for the GAP operation code.
   Column 11 has a special purpose under ENTER (see below).
   Columns 12 to 74 are reserved for the operand and index register designation.
   Columns 75 to 80 are reserved for sequence numbers.

7. The Symbol is written using conventions described in the GE-200 Series Programming Reference Manual, with the exception that it must not begin with a numeric (0 to 9) and may be six characters or less but must not be three as this is reserved for GECOM three-character symbolic names.

8. The OPERATION is written using conventions described in the GE-200 Series Programming Reference Manual.

9. Column 11 of the instruction format denotes the type of operand which follows in Columns 12 to 74.

   - A P in Column 11 designates a GECOM sentence-name as an operand. The sentence-name is written following the normal rules for GECOM sentence-names. It is converted to a three-character symbolic name, and this name is used as the operand in the object program symbolic coding.

   - A D in Column 11 denotes a GECOM data-name in Columns 12 to 74. The data-name must be qualified if necessary and must appear in the GECOM Data Division. The data-name is written following the normal rules for GECOM data-names. It is converted to a three-character symbolic name, and this name is used as the operand in the object program symbolic coding. No edit is performed on the operation code to insure proper manipulation or use of the data. The conventions governing storage of GECOM object program data are described in Chapter 9, Data Manipulation.

   - An L in Column 11 denotes a library subroutine name in Columns 12 to 19. All the subroutines available from the GECOM library are described in the GECOM-II OPERATIONS MANUAL. The unique name assigned to a particular library subroutine must be used when referring to that subroutine. No edit is performed on the instructions following the L-type operand to insure a proper calling sequence to the designated subroutine. The subroutine name is replaced with a relative address to a transfer vector table. The subroutine name is included in a program card produced by the compiler for object program loading, and is also included in the list of subroutines printed by the EDITOR. Comments may appear in Columns 32-74. Subroutine names may not be longer than 3 BCD characters in length.

   - A blank or any other character in Column 11 denotes a General Assembly Program operand in Columns 12 to 19. The operand is written using conventions described in the GE-200 Series Programming Reference Manual. It is transferred intact to the object program symbolic coding. Comments may appear in Columns 32-74.

10. Index registers may be designated in the General Assembly Program instructions. For P- and D-type operands, the index register preceded by a blank or comma (,) must follow the last character of the operand. For L-type and General Assembly Program operands, the index register must be in Column 20. If no index register is specified, Column 20 must be blank. The index register is transferred to the object program symbolic coding with no edit.

11. Instructions with P- or D-type operands must not contain comments.

12. P- and D-type operands do not terminate with a period (.).

13. P- and D-type operands must not be subscripted.

14. The last General Assembly Program instruction under an ENTER sentence must be an END card. END must appear in Columns 8 to 10. No coding is produced for an END instruction; therefore, all exits from the General Assembly Program coding must precede the END instruction. A possible exit is to "fall through" to the next ENTER sentence or GECOM procedure sentence.

15. Symbols must be unique in all General Assembly Program coding present in a GECOM source program.

16. A list and description of constants and work areas always present in GECOM-produced object programs is included in Appendix D to this manual. These constant and work area names may be used in General Assembly Program instructions as operands. Constants must not be altered, since GECOM-produced routines rely on their exact content. Work areas may be used but only as temporary storage. Upon exiting from General Assembly Program coding to GECOM procedure sentences or library subroutines, the contents of work areas will undoubtedly be destroyed.

17. Card read and punch areas are available to entered General Assembly Program coding if these areas are not being used by the GECOM-produced coding. The card areas used by GECOM-produced coding are forty-one (41) words long. If API hardware is not specified in the Environment Division, the card read areas start at locations 128_{10} and 256_{10}. (The latter area is used only if reading is buffered.) If API hardware is not specified in the Environment Division, the card punch areas start at locations 384_{10} and 512_{10}. (The latter area is used only if punching is buffered.) If API hardware is specified in the Environment Division, the card read areas start at 256_{10} and 384_{10} and the card punch areas at 512_{10} and 640_{10}.

18. Magnetic tape commands should not appear in General Assembly Program coding if the tape controller is used in the coding produced from source language sentences. When a select is given in the General Assembly Program coding, error conditions are reset. Therefore, certain error conditions could be reset prior to detection by the compiler-produced coding.

19. REM cards may be used as in General Assembly Program. The contents of REM cards are not scanned. Any combination of input characters is permissible. The contents will be printed on the Edited List.

20. General Assembly Program symbolic names may not be equated (EQU) to GECOM source data names.

21. General Assembly Program symbolic names may be equated (EQU) to the GECOM common constants but not to GECOM input/output symbolic names (see Appendix F). If General Assembly Program symbolic names are equated to GECOM input/output symbolic names, an EQU error message will be typed by the Assembler (General Assembly Program) phase of the compiler.

22. The following pseudo-instructions and mnemonic operations codes may not be used under ENTER GAP:

EJT (Pseudo)	Eject Printer Paper
*LAC	Load A Register from C Register
*LCA	Load C Register from A Register
LST (Pseudo)	List
MAL (Pseudo)	Multiple Alphanumeric
NAM (Pseudo)	Print Name or Title on Each Page
NLS (Pseudo)	No List
PAL (Pseudo)	Multiple Alphanumeric for Printer with Print Line Indicator
PLD (Pseudo)	Punch Loader Cards
RAW	Read After Write Check (DSU)
RCM	Read Cards Mixed
RRD	Read from Disc Storage Unit
SEQ (Pseudo)	Check Card Sequence Numbers
WRD	Write on Disc Storage Unit

*This instruction is an optional feature.

In addition, instructions for certain hardware, such as the DATANET and 300-lpm printer may not be used.

Although the mnemonic codes listed above are not recognized by the compiler, the programmer may utilize the listed functions through the pseudo-instruction OCT (octal).

EXAMPLE 1:

GENERAL ELECTRIC
COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER SENTENCE FORM

PROGRAM
PROGRAMMER
COMPUTER
DATE
PAGE

SEQUENCE NUMBER

```
        EVERB.  ENTER GAP.
        LDA     XYZW
        STAD    CALC OF A~FILE
        LDA     NOWRD
SAND    STAD    LAND OF B~FILE
        SUB     ONE
        BZE
        BRU     HERE
SAND2   LDX     ONE     1
        LDAD    ENTRY
        STAD    A~ARRAYD , 1
        BRU     *+2
XYZW    BSS     2
NOWRD   DEC     375
        STA     XYZW
        END.
                .
                .
                .
                GO TO EVERB.
                .
                .
                .
                ENTER GAP AT SAND2
```

EXAMPLE 2:

GENERAL ELECTRIC
COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER SENTENCE FORM

PROGRAM
PROGRAMMER
COMPUTER
DATE
PAGE

SEQUENCE NUMBER

```
                TO MVEDNO.
                .
                .
                .
        MVEDNO. ENTER GAP.
START   INX     1       1
        LDA     1
        STO     EXIT
        LDX     ZER     1
        LDAD    DRAW~NO OF INPUT~FILE,1
        STA     SAVDNO  1
        INX     1       1
        BXL     5       1
        BRU     *-4
        DLDD    PART~NO OF INPUT~FILE
        DST     SAVPNO
EXIT    BRU     *
SAVDNO  BSS     5
SAVPNO  BSS     2
        END.
                .
                .
                .
                ENTER GAP AT START.
```

FUNCTION

The EXCHANGE verb transposes the contents of two fields.

SENTENCE FORMAT

EXCHANGE { field-name-1 / array-name-1 } [,] { field-name-2 / array-name-2 } .

CONVENTIONS

1. The data images of field-name-1 and field-name-2 must be identical.
2. Field-name-1 and/or field-name-2 may be subscripted.
3. Two arrays may be exchanged by using the array field names without subscripts. The arrays must have the same size, number of fields, and the corresponding fields must have identical data images.
4. Element names are not permitted in an EXCHANGE statement.

EXAMPLES

1. EXCHANGE VELOCITY (I), VELOCITY (J).
2. EXCHANGE CODE~A, CODE~B.
3. EXCHANGE OLD~TAX, NEW~TAX.
4. EXCHANGE ARRAY~A, ARRAY~B.

GO

FUNCTION

The GO verb enables the program to depart from the normal sequence of procedures.

SENTENCE FORMAT

Option 1:

GO TO [sentence-name] .

Option 2:

GO TO sentence-name-1, sentence-name-2, [sentence-name-3 . . . sentence-name-n]

DEPENDING ON { field-name / RECORD OF file-name } .

CONVENTIONS

1. In Option 1, if a GO sentence is to be altered, it may be named. The name of the GO sentence is referred to by the ALTER verb in order to modify the sequence of execution of the program. If the destination sentence name is omitted, the compiler will insert a halt loop in the object program. Therefore, a GO sentence without a destination should be referenced by an ALTER sentence before the first execution of such a GO sentence. After a GO sentence has been altered, it will continue to go to the changed destination until it is altered again.

2. In Option 2, the field name must have a positive integral value. The branch will be the 1st, 2nd, ..., 30th sentence name as the value of the field is 1, 2, ..., 30. If the value is zero, or exceeds 30 (that is, the number of sentences named) the next sentence in normal sequence will be executed. Field-name may be subscripted.

3. In Option 2, using RECORD OF file-name, control is transferred to sentence-name-1 if the current record of the specified file is the type described in the first record description after the FD entry for that file; control is transferred to sentence-name-2 if the current record is the type described in the second record description, etc. The type record is determined by testing the control-key.

EXAMPLES

1. SENT~18. GO.

   (Note if this sentence is not referenced by the ALTER verb before it is executed, the object program will halt in a loop.)

2. GO TO SENTENCE~7.

3. GO TO SENT~B, SENT~77, SENT~A, SENT~64 DEPENDING ON CODE.

4. GO TO SENT-1, SENT-2, SENT-3, DEPENDING ON RECORD OF TRANS-FILE.

NOTE: If the records in TRANS-FILE in the preceding example are described in the following order:

```
FD TRANS-FILE.

 R RECORD-0.

 R RECORD-1.

 R RECORD-2.
```

Control will be transferred as indicated below:

CURRENT RECORD	CONTROL TRANSFERRED TO
RECORD - 0	SENT-1
RECORD - 1	SENT-2
RECORD - 2	SENT-3

IF

FUNCTION

The IF verb transfers control to the specified sentence if the stated condition is satisfied (true) or to the next sentence if the stated condition is not satisfied (false).

SENTENCE FORMAT

Option 1:

```
IF [NOT] { record-name
           conditional-name
           END OF FILE OF file-name-1
           TOP OF PAGE [OF] file-name-2
           true-false date-name }

    GO TO sentence-name-1.
```

Option 2 (relational expressions):

```
IF { field-name-1
     element-name-1
     constant-1
     arithmetic-expression-1 }

   { IS [NOT] GREATER THAN
     IS [NOT] LESS THAN
     IS [NOT] EQUAL TO
     IS UNEQUAL TO
     EQUALS
     EXCEEDS }

   { field-name-2
     element-name-2
     constant-2
     arithmetic-expression-2 }

    GO TO sentence-name-1.

[ , IF { [NOT] GREATER THAN
         [NOT] LESS THAN
         [NOT] EQUAL TO
         UNEQUAL TO
         EQUALS
         EXCEEDS } { field-name-3
                     element-name-3
                     constant-3
                     arithmetic-expression-3 } ]

    GO TO sentence-name-2 [, IF...] .
```

Option 3 (logical expressions):

```
IF { conditional-name-1
     relational-expression-1 }

   { AND
     OR }

   { conditional-name-2
     relational-expression-2 }
```

$$\left[\left\{\begin{matrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{matrix}\right\}\left\{\begin{matrix}\text{conditional-name-3}\\ \text{relational-expression-3}\end{matrix}\right\}\ \cdots\ \left\{\begin{matrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{matrix}\right\}\left\{\begin{matrix}\text{conditional-name-20}\\ \text{relational-expression-20}\end{matrix}\right\}\right]$$

, GO TO sentence-name-1.

Option 4 (Tests):

$$\underline{\text{IF}}\ \left\{\begin{matrix}\text{field-name-1}\\ \text{arithmetic-expression-1}\end{matrix}\right\}\ \text{IS}\ \left[\underline{\text{NOT}}\right]\left\{\begin{matrix}\underline{\text{POSITIVE}}\\ \underline{\text{NEGATIVE}}\\ \underline{\text{ZERO}}\end{matrix}\right\}$$

, GO TO sentence-name-1.

CONVENTIONS

1. The abbreviations for relations may be used instead of the English words shown in the above formats.
2. A quantity is positive only if it is greater than zero. A quantity is negative only if it is less than zero. The value zero is considered neither positive nor negative.
3. Fields may be subscripted.
4. Alphanumeric fields that are being compared must have the same data description.
5. In Option 1, the named record must be an input record with a control key. It must be qualified by a file-name unless the record-name is unique. If the current record is of the type corresponding to the named record (as determined by testing the control key), control will be transferred to sentence-name-1.

EXAMPLES

Option 1

1. IF MALE, GO TO 789.
2. IF NOT END OF FILE OF MASTER~FILE, GO TO SENTENCE~4.
3. IF TOP OF PAGE OF STOCK~FILE, GO TO PRINT~HEAD.
4. IF SHIP~RECORD OF TRANS~FILE, GO TO SHIPMENT.
5. IF FLAG~1 GO TO SENTENCE~22.

Option 2

1. IF LINE~COUNT EQ 58 GO TO ADVANCE~PAGE.

2. IF PART~NUMBER OF MSTR~INVNTRY IS LESS THAN PART~NUMBER OF TRANSACTIONS GO TO WRITE~MASTER, IF EQUAL GO TO UPDAT~MASTER, IF GREATER GO TO NEW~RECORD.

3. IF WEEKLY~FICA OF MASTR~PAYROL + ANNUAL~FICA OF MASTR~PAYROL EXCEEDS 144.00 GO TO COMP~WK~FICA.

4. IF TRANSACT~COD EQUALS 1 GO TO SHIPMENT, EQUALS 2 GO TO RECEIPT, EQUALS 3 GO TO CHANGE, EQUALS 4 GO TO ADDITION, EQUALS 5 GO TO DELETE.

Option 3

1. IF SHIPMENT AND QTY~ON~HAND OF MSTR~INVNTRY IS LESS THAN QTY~SOLD OF TRANSACTIONS, GO TO BACK~ORDER.

2. If A + B - C EQ Z * Y AND P GR Q GO TO XYZ.

Option 4.

1. IF ADJUSTED~PAY OF MASTR~PAYROL IS NEGATIVE, GO TO ADJUSTMENT.

2. IF QTY~ON~HAND OF MSTR~INVNTRY IS ZERO GO TO REORDER.

FUNCTION

The LOAD verb brings a designated overlay segment into the central processor memory for subsequent execution.

SENTENCE FORMAT

LOAD segment-name SEGMENT.

CONVENTIONS

1. The overlay segment to be loaded must be in absolute form on magnetic tape or disc storage at object time. The input hardware name assigned in the Environment Division for the specified segment must be the same as that assigned for the main segment. (See the RAB function of BRIDGE II, CD225J1.001, and BRAT, CD225E2.005R.)

2. LOAD sentences may be given only in the main segment.

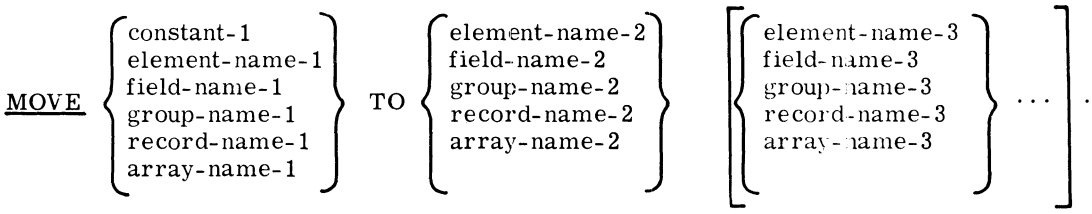
3. The overlay segment-name given in the LOAD sentence must be identical to the name given when the segment is converted from relocatable to absolute by RAB.

4. Communications between a loaded segment and the main segment and other loaded segments are always accomplished by PERFORM sentences and by data in Common or *Common Storage. (See Overlay Segmentation and the PERFORM verb.)

FUNCTION

The MOVE verb transfers a constant or the contents of an element, field, group, or record to one or more other elements, fields, groups, or records.

SENTENCE FORMAT



CONVENTIONS

1. A numeric constant or numeric field being moved is aligned in accordance with the decimal point of the destination field with truncation or zero fill on either end as required. (See Example 1.)

2. A nonnumeric element or nonnumeric field being moved is left justified with space fill on the right if the destination element or field is larger than the source data. (See Example 2.)

3. A nonnumeric element or nonnumeric field, or a literal being moved will be left justified and truncated on the right if the destination element or field is smaller than the source data. The compiler gives a warning unless the source is a literal. (See Example 3.)

4. A numeric constant or numeric field being moved to a nonnumeric element or nonnumeric field must have the same number of characters in its data image as the destination element or field. (See Example 4.)

5. A nonnumeric element or nonnumeric field or a literal being moved to a numerid field must have the same number of characters in its data image as the destination field. (See Example 5.)

6. A literal consisting of a single character being moved to a nonnumeric element or nonnumeric field fills the destination with the character specified. (See Example 6.)

7. A literal consisting of more than one character being moved to a nonnumeric element or nonnumeric field is placed repeatedly in the destination position (starting at the leftmost character position). If the literal does not fit the destination position an integral number of times, it is truncated on the right for the last placement. (See Example 7.)

8. A figurative constant except "SPACE (S)" being moved to a numeric field fills the destination positions described by nines (9's) in the data image. When the figurative constant "SPACES(S)" is moved to a numeric field, the destination positions described by nines (9's) in the data image are filled with zeros. Spaces would violate the data description of numerics. (See Example 8.)

9. A figurative constant being moved to a nonnumeric element or nonnumeric field fills each position of the destination. (See Example 9.)

10. If a constant is moved to a record or group, the receiving area cannot contain more than fifty (50) fields.

11. Fields may be subscripted.

12. Movement of numerics, literals, figurative constants, or fields to groups or records is treated as separate MOVE sentences--one for each field present in the group or record. Therefore, the group or record fields must satisfy the above rules for moving numerics, literals, figurative constants, or fields to fields.

13. The compiler provides for the movement of only one record or group of fields to other records or groups of fields of the same size and format. Any other movement can be accomplished by moving elements and/or fields, and/or the implied movement under the WRITE verb. Note that it is unnecessary to specify data movement to output.

14. A numeric field may be moved to a Working Storage or input field with editing. If an edited field is moved to a numeric field, all editing characters are ignored except that the sign and decimal scale of the source field influence the conversion process. If a field containing editing characters is moved to a field containing editing characters, it is treated as a BCD alphanumeric move.

15. Arrays may be referenced by using the array field name without a subscript. In this case, the array is treated as a string of fields and may be used as a source and/or destination. If the source is an array, the destination must be an array. The compiler provides for moving a literal, a numeric, a figurative constant, or a field to an array. The array is filled under the conventions governing literals, numerics, figurative constants, or field to field movement. In moving an array to an array, the source array is moved to the destination array under the conventions governing field-to-field movement.

16. See Chapter 9 - Data Manipulation for a detailed discussion of movement of repeated groups.

EXAMPLES

The following fields are assumed to have the indicated images in the examples below:

FIELD NAME	DATA IMAGE
TOTAL	999999V
FACTOR	9999V99
PART~NUMBER	A(10)
SAVE~AREA	A(6)
MULTIPLIER	999V9
PT~NUMBER	A(10)

1. MOVE 1000 TO FACTOR.
   RESULTS: 1000V00

   MOVE 300 TO FACTOR.
   RESULTS: 0300V00

   MOVE 1.235 TO FACTOR.
   RESULTS: 0001V23

MOVE MULTIPLIER TO FACTOR.
RESULTS: 024V5 0024V50

MOVE FACTOR TO MULTIPLIER.
RESULTS: 1234V56 234V5

2. MOVE PT~NUMBER TO PART~NUMBER.
RESULTS: AA1673BBCC AA1673BBCC

MOVE SAVE~AREA TO PT~NUMBER.
RESULTS: AA1673 AA1673△△△

3. MOVE PART~NUMBER TO SAVE~AREA.
RESULTS: AA1673BBCC AA1673

MOVE "12345678" TO SAVE~AREA.
RESULTS: 123456

4. MOVE 123456 TO SAVE~AREA.
RESULTS: 123456

MOVE TOTAL TO SAVE~AREA.
RESULTS: 007340V 007340

5. MOVE SAVE~AREA TO TOTAL.
RESULTS: 653000 653000V

MOVE "1235" TO MULTIPLIER.
RESULTS: 123V5

6. MOVE "Z" TO PART~NUMBER.
RESULTS: ZZZZZZZZZZ

7. MOVE "XY" TO PART~NUMBER.
RESULTS: XYXYXYXYXY

MOVE "XYZ" TO PART~NUMBER.
RESULTS: XYZXYZXYZX

8. MOVE ZEROS TO MULTIPLIER.
RESULTS: 000V0

9. MOVE SPACES TO PART~NUMBER.
RESULTS: △△△△△△△△△△

10. If a constant is moved to a record or group, the receiving area cannot contain more than fifty (50) fields.

11. Fields may be subscripted.

12. Movement of numerics, literals, figurative constants, or fields to groups or records is treated as separate MOVE sentences--one for each field present in the group or record. Therefore, the group or record fields must satisfy the above rules for moving numerics, literals, figurative constants, or fields to fields.

13. The compiler provides for the movement of only one record or group of fields to other records or groups of fields of the same size and format. Any other movement can be accomplished by moving elements and/or fields, and/or the implied movement under the WRITE verb. Note that it is unnecessary to specify data movement to output.

14. A numeric field may be moved to a Working Storage or input field with editing. If an edited field is moved to a numeric field, all editing characters are ignored except that the sign and decimal scale of the source field influence the conversion process. If a field containing editing characters is moved to a field containing editing characters, it is treated as a BCD alphanumeric move.

15. Arrays may be referenced by using the array field name without a subscript. In this case, the array is treated as a string of fields and may be used as a source and/or destination. If the source is an array, the destination must be an array. The compiler provides for moving a literal, a numeric, a figurative constant, or a field to an array. The array is filled under the conventions governing literals, numerics, figurative constants, or field to field movement. In moving an array to an array, the source array is moved to the destination array under the conventions governing field-to-field movement.

16. See Chapter 9 - Data Manipulation for a detailed discussion of movement of repeated groups.

EXAMPLES

The following fields are assumed to have the indicated images in the examples below:

FIELD NAME	DATA IMAGE
TOTAL	999999V
FACTOR	9999V99
PART~NUMBER	A(10)
SAVE~AREA	A(6)
MULTIPLIER	999V9
PT~NUMBER	A(10)

1. MOVE 1000 TO FACTOR.
   RESULTS: 1000V00

   MOVE 300 TO FACTOR.
   RESULTS: 0300V00

   MOVE 1.235 TO FACTOR.
   RESULTS: 0001V23

MOVE MULTIPLIER TO FACTOR.
RESULTS: 024V5 0024V50

MOVE FACTOR TO MULTIPLIER.
RESULTS: 1234V56 234V5

2. MOVE PT~NUMBER TO PART~NUMBER.
RESULTS: AA1673BBCC AA1673BBCC

MOVE SAVE~AREA TO PT~NUMBER.
RESULTS: AA1673 AA1673△△△△

3. MOVE PART~NUMBER TO SAVE~AREA.
RESULTS: AA1673BBCC AA1673

MOVE "12345678" TO SAVE~AREA.
RESULTS: 123456

4. MOVE 123456 TO SAVE~AREA.
RESULTS: 123456

MOVE TOTAL TO SAVE~AREA.
RESULTS: 007340V 007340

5. MOVE SAVE~AREA TO TOTAL.
RESULTS: 653000 653000V

MOVE "1235" TO MULTIPLIER.
RESULTS: 123V5

6. MOVE "Z" TO PART~NUMBER.
RESULTS: ZZZZZZZZZZ

7. MOVE "XY" TO PART~NUMBER.
RESULTS: XYXYXYXYXY

MOVE "XYZ" TO PART~NUMBER.
RESULTS: XYZXYZXYZX

8. MOVE ZEROS TO MULTIPLIER.
RESULTS: 000V0

9. MOVE SPACES TO PART~NUMBER.
RESULTS: △△△△△△△△△△

FUNCTION

The MULTIPLY verb multiplies two quantities and stores the result in the last-named field or the specified field.

SENTENCE FORMAT

```
MULTIPLY  { numeric-1    }          BY  { numeric-2    }
          { field-name-1 }              { field-name-2 }

          [GIVING field-name-3]          [ROUNDED]

          [IF SIZE ERROR GO TO sentence-name-1] .
```

CONVENTIONS

1. If the GIVING option is not present, the last-named field receives the result.
2. Decimal points do not appear in stored fields, and are used only to properly align data before execution of an arithmetic operation.
3. Only a numeric may be used. If a sign (+ or -) is included, it must appear as the most significant character of the numeric.
4. The ROUNDED option may be used to round off the result before it is stored in the receiving field. If the receiving field is in floating point mode or if the operands are all integers, then rounding is ignored.
5. The SIZE ERROR option may be used to truncate the most significant digits of a number.
6. The result of multiplying two numbers, each 11 digits long, may not result in a number 22 digits long. Only 11 digits are significant and only 11 digits may be saved.

EXAMPLES

1. MULTIPLY 0.18 BY PAY GIVING TAX, IF SIZE ERROR GO TO ERROR~SENT.
2. MULTIPLY A OF FILE~1 BY B OF FILE~2 GIVING C OF FILE~3.
3. MULTIPLY 3.1416 BY R1. (Note the contents of R1 is multiplied by 3.1416 and the result is stored in R1.)

NOTE

FUNCTION

The NOTE verb allows the programmer to write explanatory material in his program which will be produced on the listing but not compiled.

SENTENCE FORMAT

NOTE ... .

CONVENTIONS

1. Any sentence may follow the word NOTE if the rules for sentence structure are followed.
2. The NOTE sentence must not be named.
3. The effect of a NOTE sentence can also be achieved by punching the letter C in Column 7 of a Procedure Division card. The contents of such a card will be printed in the Edited List, but no corresponding coding is produced. This option may not be used between a card on which a sentence begins and a continuation card on which it ends. Furthermore, it may not be used on cards within the bounds of ENTER GAP ... END statements. However, it may be used anywhere in a TABSOL table.

EXAMPLES

1. NOTE THIS SENTENCE IS NOT NAMED BECAUSE REFERENCE IS NOT MADE TO IT.
2. NOTE THIS SENTENCE IS USED FOR CLARITY.
3. NOTE THAT THIS SENTENCE DOES NOT STATE COMPILER OR OBJECT PROGRAM ACTION.

FUNCTION

The OPEN verb initiates the processing of both input and output files and performs checking or writing of labels and other input/output functions.

SENTENCE FORMAT

Option 1:

```
OPEN INPUT file-name-1        [WITH NO REWIND]
     [, file-name-2           [WITH NO REWIND]  ...]
      OUTPUT file-name-3      [WITH NO REWIND]
     [, file-name-4           [WITH NO REWIND]  ...] .
```

Option 2:

```
OPEN { INPUT  } file-name-1   [WITH NO REWIND]
     { OUTPUT }
     [, file-name-2           [WITH NO REWIND]  ...] .
```

Option 3:

```
OPEN ALL [ { INPUT  } ]  FILES  [WITH NO REWIND] .
         [ { OUTPUT } ]
```

CONVENTIONS

1. All input and output files except JOURNAL~TAPE files (See Environment Division, DSU~CONTROL sentence) must be referred to in an OPEN sentence which should be executed before the first READ or WRITE sentence of a file. If a JOURNAL~TAPE file is not opened, no label is written even though a label description may be given for the JOURNAL~TAPE file in the Data Division. If not opened, the JOURNAL~TAPE file remains in position; it is not rewound before recording is done.

2. A second OPEN sentence of a file cannot be executed before the execution of a CLOSE sentence of the file.

3. The OPEN sentence does not obtain or release the first data record. A READ or WRITE sentence must be executed to obtain or release the first data record. For DSU files, a READY sentence must be executed prior to the READ or WRITE sentences to obtain or release the first data record.

4. When checking or writing the first label, the user's beginning label procedure will be executed if specified by the USE clause. (See Chapter 8, Environment Division.)

5. If an input file has been designated as optional in the Environment Division, the object program will cause an interrogation for the presence or absence of this file. If the reply to the interrogation is negative (that is, the file is not present) the file will not be opened. A printout indicating the absence of the file occurs, and an end-of-file signal is sent to the input/output control system of the object program. Thus, when the first READ sentence for this file is met, the end-of-file path for this sentence will be taken. Files assigned to DSU's may not be designated as optional.

6. OPEN rewinds a tape file unless the NO REWIND option is stated.

7. Only one file of a multiple file tape may be in OPEN status at any given time. The ALL option should not be used if a multiple file tape exists in the program.

8. More than one DSU file assigned to the same area (See Environment Division, I~O~CONTROL sentence) may be open at the same time. It is the programmer's responsibility to know which file occupies the area at any given time. A DSU file described as buffered cannot share the same area with an unbuffered file if they are open at the same time. When DSU files are sharing the same area and are open at the same time, they must be assigned to the same plug.

9. The OPEN sentence does not advance a printer file to top-of-page.

EXAMPLES

1. OPEN INPUT MASTR~PAYROL WITH NO REWIND, EMP~FILE, TAX~FILE OUTPUT UP~DT~PAYROL, NEW~TAX~FILE.

2. OPEN INPUT FILE~A, FILE~B, FILE~C OUTPUT FILE~1, FILE~2, FILE~3.

3. OPEN ALL INPUT FILES.

4. OPEN ALL FILES.

FUNCTION

The PERFORM verb executes a section or a segment or a set of procedures within a segment. Upon completion of the specified procedures, control reverts to the sentence following the PERFORM sentence.

SENTENCE FORMAT

Option 1:

PERFORM section-name SECTION

$$\left[\text{USING} \left\{\begin{matrix}\text{constant-1}\\ \text{field-name-1}_i\end{matrix}\right\} \left[, \left\{\begin{matrix}\text{constant-2}\\ \text{field-name-2}_i\end{matrix}\right\} , \ldots\right]\right.$$

$$\left.\left[\text{GIVING field-name-1}_o\text{, field-name-2}_o\text{, } \ldots\right]\right].$$

Option 2:

PERFORM $\left\{\begin{matrix}\text{procedure-name}\\ \text{segment-name}\end{matrix}\right\}$ SEGMENT.

CONVENTIONS

1. When the USING-GIVING option is used, field-name-1_i, field-name-2_i, ... and field-name-1_o, field-name-2_o, ... are considered to be assignment variables. Fields may be subscripted.

   To have meaning, the section head must have a corresponding set of defined input and output variables. The assignment takes place in accordance with the MOVE specifications.

   The section to be executed must appear before the PERFORM...USING...GIVING sentence in the Procedure Division.

2. In Option 2, the USING and GIVING statements may not be used. The segment-name is the PROGRAM-ID of the segment to be executed. Procedure-name is the name of a set of procedures within a segment to be performed from outside the segment.

3. An overlay segment must be loaded before any portion of it can be performed.

4. See Sections, Segments, and Overlay Segmentation, starting on Page 79 of this manual.

FUNCTION

The READ verb has the following options:

Option 1 allows a limited amount of input data to be entered from the control console switches.

Option 2 makes available for processing the next logical record or group from an input file and transfers control to the specified sentence when the end-of-file is reached.

Option 3 advances an input tape file, (or copies records or groups from an input tape file onto an output tape file), until the specified condition is satisfied. Then the current logical input record or group is made available for processing and control is transferred to the next sentence. In addition, it transfers control to the specified sentence when the end of the input tape file is reached without satisfaction of the specified condition.

Option 4 makes available for processing the next logical record or group from a DSU input buffer and transfers control to the specified sentence when the buffer is empty.

Option 5 scans a buffer of blocked DSU input records until the specified condition is satisfied. Then the current logical input record or group is made available for processing and control is transferred to the next sentence; steps (COPY) output records as input records are being passed over when blocked DSU input and output files are sharing the same area; transfers control to the specified sentence when the end of the buffer is reached without satisfaction of the specified condition.

SENTENCE FORMAT

Option 1:

READ field-name-1 [, field-name-2, . . . , field-name-19]
FROM CONTROL SWITCHES.

Option 2:

READ file-name [{RECORD / GROUP}] [, IF END OF FILE GO TO sentence-name-1] .

Option 3:

READ file-name-1 [COPYING ON file-name-2]

UNTIL { field-name-1 / element-name-1 / constant-1 } { IS [NOT] GREATER THAN / IS [NOT] LESS THAN / IS [NOT] EQUAL TO / IS UNEQUAL TO / EQUALS / EXCEEDS }

{ field-name-2 / element-name-2 / constant-1 } [, IF END OF FILE GO TO sentence-name-1] .

Option 4:

READ file-name-1 [{ RECORD / GROUP }] [, IF END OF BLOCK GO TO sentence-name-1] .

Option 5:

READ file-name-1 [COPYING ON file-name-2] UNTIL { field-name-1 / element-name-1 / constant-1 }

{ IS [NOT] GREATER THAN / IS [NOT] LESS THAN / IS [NOT] EQUAL TO / IS UNEQUAL TO / EQUALS / EXCEEDS } { field-name-2 / element-name-2 / constant-1 } [, IF END OF BLOCK GO TO sentence-name-1] .

CONVENTIONS

1. An OPEN sentence must be executed before the first READ sentence is given for the particular file. A file must not be reopened unless it has previously been closed.

2. The filling of the input area, tape movement, flow of cards, etc., is controlled entirely by routines generated by the compiler.

3. After recognition of the end-of-reel of a tape file, the READ sentence performs the following operations:

   - The standard ending-reel label procedure and the user's ending-reel label procedure (if specified by the USE clause in the INPUT-OUTPUT CONTROL sentence).
   - A tape unit swap if more than one tape unit is assigned to the file. If one unit is assigned to a multiple reel file, the object program will type out a request to mount the next reel.
   - The standard beginning reel label procedure and the user's beginning reel label procedure (if specified by the USE clause in the INPUT-OUTPUT CONTROL sentence).
   - Makes the next logical record available.

4. Any number of READ sentences may be stated for the same file. At least one END clause must be specified for each input file. If more than one END clause is given for a particular file, it is not required that each END clause go to the same sentence. If the END clauses for a particular file go to different sentences and the END clause is not stated in every READ sentence for that file, the compiler will give a warning. If all END clauses for a particular file go to the same sentence, the END clause need be stated only once, preferably in the first READ sentence executed for that file.

5. If an optional file is not present in a given running of the object program, the END clause will be executed on the first READ sentence.

6. When a file consists of more than one type of data record or group (if such groups are referenced by READ sentences), each type may be of a different size, but every record or group of the same type must be of the same size. When a READ sentence is executed, the next logical record or group is made available without regard to its type. Only the data in the current record or group is accessible.

   The programmer should employ an IF sentence to determine the type of data record or group after it has been read. (See Control-Key in Chapter 5, Data Division.)

7. In Option 1, field-name-1 through field-name-19 must be the names of switches that must be described in the Data Division. Field-name-1 corresponds to switch 1, field-name-2 to switch 2, field-name-3 to switch 3, etc. The down position represents the true state (ON), and the normal position represents the false state (OFF). Note that switch 0 on the Control Console is retained for use in conventional error procedures. If it is desired to read only switch N, it is necessary to read all lower order switches preceding switch N; that is, READ field-name-1, field-name-2, ..., field-name-N FROM CONTROL SWITCHES.

8. When Option 1 of the READ verb is used, it should be preceded by a WRITE (Option 1) sentence in which a literal is typed out. The user must provide the operator with instructions for switch settings when the appropriate literal is typed out. When a READ verb, Option 1, is executed, the object program enters a "read control switch loop" to allow the operator to set the appropriate switches as explained in the operating instructions mentioned above for the previously typed literal. Toggling switch zero then causes the switches to be read and processing to resume at the next sentence.

   If switch 19 is used under Option 1, all switches (1-19) must be described in the Data Division. If any other switch is used, all switches through the switch used must be described.

9. In Option 3, the abbreviations for relations may be used instead of the English words shown in the sentence format. For faster object programs, Option 3 is recommended where the frequency of satisfaction of the specified condition is low.

10. In Option 5, field-name-2, element-name-2, or constant-1 must be in the same mode (bin. 1's or BCD) and have the same size as field-name-1, element-name-1, or constant-1. If file-name-1 contains more than one type of record, then field-name-1 or element-name-1 must be in the same position relative to the beginning of each record. The Procedure Division may not contain more than four READ UNTIL sentences per file. Note that only one constant may be used in an UNTIL clause. One of the fields or elements named must be contained in the file read. Both fields or elements named should not be in the file read.

11. Options 3 and 5 may be used with multirecord files.

12. Options 4 and 5 may be used for DSU fields only.

13. Conventions 5, 9, 10, and 11 above apply to DSU files as well as other files. Conventions 1 through 3, 7 and 8 above do not apply to DSU files.

14. For consequential DSU files, a READY verb must be given to obtain a physical record. READ verbs are used to make available for processing the next logical record or group within a physical record.

15. In Options 4 and 5, the END OF BLOCK clause may be specified only for DSU files designated as blocked. The normal end of block sentinel (all 1 bits in the first word following the last valid data word) for partial blocks is used. The END OF BLOCK option should be used when the number of records in a block are unknown and the programmer needs an indication to issue another READY sentence for the file.

16. In Option 5, the END OF BLOCK clause is required for DSU files not described by the sequential clause in the Environment Division. If the relation is not satisfied before the end of block is reached, transfer is made to sentence-name-1.

17. In Options 4 and 5, when the END OF BLOCK clause is used and different sentence names are used in the END OF BLOCK clauses for the same DSU file, every READ (or READ...UNTIL) sentence for that file must have an END OF BLOCK clause.

15. In Option 5, when blocked DSU input and output files are sharing the same area, the COPYING option must be exercised to step the output record area to input records are being passed over. A subsequent write of the output record will update the corresponding input record that satisfied the READ...UNTIL sentence.

19. Use of Option 5 for DSU files described as sequential in the Environment Division causes records to be interrogated until the relation specified is satisfied. Additional record blocks are brought into memory from the DSU as required; and field mentioned in the USING clause of the READY sentence for the file is updated. See READY sentence, Convention 7.

The COPYING option may not be specified for sequential DSU files.

EXAMPLES

Option 1:

1. READ VALUE~1, VALUE~2 FROM CONTROL SWITCHES.

Option 2:

1. READ TIME~CARD RECORD.

2. READ MASTR~PAYROL, IF END GO TO FINAL~STOP.

Option 3:

1. READ IN~MASTER COPYING ON OUT~MASTER UNTIL MASTER~STOCK EQUALS TRAN~STOCK, IF END OF FILE GO TO CLOSEOUT.

Option 4:

1. READ MASTER RECORD, IF END OF BLOCK GO TO GET~NEXT.

Option 5:

1. Assume INFILE and OUTFILE are blocked and share the same buffer.
   READY INFILE FOR READING.
   READY OUTFILE.
   READY INFILE COPYING ON OUTFILE UNTIL IN~STOCK EQUALS TRAN STOCK
   IF END OF BLOCK GO TO TRAN ERROR.

   Procedures to update INFILE RECORD-
   WRITE AND RELEASE OUTFILE REC.

FUNCTION

The READY verb initiates or tables a seek on a disc storage unit in preparation for a physical read or write.

SENTENCE FORMAT

READY [{ INPUT / OUTPUT }] file-name-1 [FOR { READING / WRITING }]

[USING field-name-1 FOR ADDRESSING] .

CONVENTIONS

1. The { INPUT / OUTPUT } option must be used when file-name-1 is both an input and an output file name.

2. When the READING option is used, physical data movements to the input buffer begins immediately after the seek is completed. The next READ request for file-name-1 encountered in the object program is considered as a demand read, that is, processing is delayed until the data is in memory. No data is available to the program until the demand READ sentence is given.

3. If the READING option is not specified for an input file, the seek is performed, but no data enters memory until a READ sentence is issued for the file. Thus the READ entry is a demand read; processing is delayed until the data is in memory. The programmer must ensure that no demand read or no RELEASE sentence for the disc storage unit is given for any other file until the demand read for file-name-1 is given.

4. When the WRITING option is used on nonsequential files, the seek is initiated, but no recording is done until a RELEASE sentence is given for file-name-1. The programmer must insure that no demand read or no RELEASE sentence (for the disc storage unit involved) is given for any other file before the RELEASE sentence is given for file-name-1. On output files described as sequential, the seek is always issued just before a buffer is written onto the DSU. The optional word WRITING is ignored.

5. If the WRITING option is not specified for a nonsequential output file, the seek is not initiated until a RELEASE or WRITE/RELEASE sentence is given for the file. Recording then starts immediately after the seek is completed.

6. A USING clause must be specified in at least one READY sentence for each file assigned to disc storage. If different field names are employed in USING clauses for the same file, every READY sentence for that file must have a USING clause. The contents of the USING field at object time must contain the absolute DSU address desired. The USING field-name clause may be subscripted.

7. Files described as sequential in the Environment Division must use the READY sentence to initialize or origin the seek address. The DSU address is increased by the number of frames in the record block to develop the next DSU address to be used. Each time a new seek address is developed, the new address replaces the previous address contained in the field named in the USING clause. If an illegal address is developed (frames 96-127), a decimal 32 is added to the address to make it legal. No further test of the developed address is made.

   READY sentences may be issued to establish new origins for sequential files, but the file must first be closed and then re-opened before issuing the READY sentence.

8. Every READY sentence for a file-name must be followed by a READ sentence for the file named if input, or a WRITE sentence for a record of the file named if output, before another READY sentence for the same file is issued.

EXAMPLES

1. READY OUT~FILE USING FLD~A FOR ADDRESSING.
   WRITE BLANK~REC.
   READY OUT~FILE FOR WRITING.
   WRITE AND RELEASE OUT~FILE~REC.

   Note that the first WRITE verb is used to blank the output buffer. The second READY verb initializes to the beginning of the buffer and starts the seek. The second WRITE verb fills the buffer and releases it for recording on the DSU.

2. READY MASTR~IN FOR READING USING ADDR~FLD FOR ADDRESSING.

FUNCTION

The RELEASE verb starts the physical recording of output data onto the DSU.

Note: Blocked sequential process files are not automatically released. If used, record count should be kept by user and a **WRITE RELEASE** sentence issued when block is full.

SENTENCE FORMAT

RELEASE file-name.

CONVENTIONS

1. Only output files can be released.
2. On SEQUENTIAL files, the **RELEASE** causes the buffer to be emptied onto the DSU. However, the RELEASE is not required since the buffer is automatically dumped when full.
3. If a SEQUENTIAL output file is sharing the same area with a non-SEQUENTIAL blocked input file, the output file must be **RELEASED** when the input file reaches end of block.

EXAMPLE

1. RELEASE INVENTORY.

STOP

FUNCTION

The STOP verb halts the object program either permanently or temporarily.

SENTENCE FORMAT

```
STOP [ { RUN                          } ]   [literal-1] .
     [ { ON SWITCH integer-1          } ]
```

CONVENTIONS

1. The STOP verb may be used as the last logical command to stop an object program, as a means of typing error stop indicators, or to allow the operator to halt the object program under console switch control.

2. If the STOP verb is used without the RUN, SWITCH, or literal options, the object program will enter a "read control switch loop." The operator may toggle switch zero to resume processing at the next sentence. Prior to this type of STOP sentence, the user should provide a message (by means of the WRITE verb, Option 1) containing some explanation for the STOP and any necessary operating instruction.

3. When the SWITCH option is used, the object program will enter a "read control switch loop" whenever the STOP is encountered and the switch integer-1 is down. The operator may raise switch integer-1 to resume processing at the next sentence. The integer-1 corresponds to the control switches and may be any of the numbers 0-19 inclusive.

   If the switch specified is not down when the STOP-generated coding is encountered in the object program, processing continues with the coding produced for the next sentence.

   If the literal option is used with the SWITCH option, the literal is typed regardless of the condition of the switch specified.

4. If the STOP verb is used with the literal option only, the literal will be typed out, and the object program will then enter a "read control switch loop." The operator may toggle switch zero to resume processing at the next sentence. The literal must be enclosed in quotation marks. The literal should be composed of typewriter characters and/or typewriter control symbols only. The user should prepare a list of such typeouts showing the appropriate literals with their corresponding explanations and operating instructions.

5. The RUN option may be specified only once in any given segment.

   When the RUN option is used, the word END will be typed out followed by the literal, if one is specified. It is suggested that the PROGRAM-ID shown in the Identification Division be used as the literal. In any case, the literal must be enclosed in quotation marks. The literal should be composed of typewriter characters and/or typewriter control symbols only.

If the object program has been assigned to the card reader or the DSU in the OBJECT~COMPUTER sentence, the next program is assumed to be in the card reader. In this case, after the typeout occurs, the object program enters a "read control switch loop." If the operator toggles switch zero, the program will attempt to read a binary card into absolute zero and branch to zero. The binary card read is assumed to be the loader for the next program.

6. When the RUN option is used and the object program has not been assigned to an input hardware name or has been assigned to a magnetic tape unit in the OBJECT~COMPUTER sentence, the next program is assumed to be on tape. The tape unit and plug number of the next program are assumed to be the same as those from which the current program was loaded. The loader for the object program passes the tape unit and plug number used to the main segment for use in its run completion. If the program was not loaded from tape (being tested from cards), its run completion routine recognizes this condition and enters a halt loop instead of searching for a sequential run locator to load the next program.

7. The run completion calling sequence produced for the STOP RUN option takes two forms:

   a. NO NEXT~PROGRAM clause specified in the Identification Division:

Coding Produced			Comments
	SPB	*RC 1	Run completion routine is *RC
	ALF	RUN	
	ALF	XXX	X represents the name of this program
	ALF	XXX	(from PROGRAM~ID)
	ALF	XXX	
	SEL	0	The plug number from which the main segment was loaded is stored in bits 10-13 at object time.
	Z00		The tape unit code from which the main segment was loaded is stored in bits S-4 at object time. This location becomes zero if the main segment was loaded from cards.
FR2	DEC	-1	No NEXT~PROGRAM
	ALF	000	
	ALF	000	
	ALF	000	
	SEL	0	
	Z00		
	DEC	-2	

b. NEXT~PROGRAM clause is specified in Identification Division:

	Coding Produced			Comments
	SPB	*RC	1	Run completion routine is *RC
	ALF	RUN		
	ALF	XXX		X represents the name of this
	ALF	XXX		program (from PROGRAM ID)
	ALF	XXX		
	SEL	0		The plug number from which the main segment was loaded is stored in bits 10-13 at object time.
	Z00			The tape unit code from which the main segment was loaded is stored in bits S-4 at object time. This location becomes zero if the main segment was loaded from cards.
FR2	ALF	RUN		
	ALF	YYY		Y represents the name of the NEXT-
	ALF	YYY		PROGRAM from the Identification
	ALF	YYY		Division
	SEL	0		
	Z00			
	DEC	-2		

The compiler automatically generates a field named NEXT~RUN with an assumed description of X(12). The programmer is free to use this field name to move a literal RUNYYYYYYYYY into the run completion calling sequence, where Y represents the name of the desired run. Thus, the programmer is able to choose the next run to be executed on the basis of his input data. The name NEXT~RUN is given the symbolic name FR2 at compile time. Presently, the next run in sequence on tape can be obtained by moving the literal NEX$\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta$ to the NEXT-RUN field.

EXAMPLES

1. STOP.
2. STOP "999".
3. STOP RUN.
4. STOP RUN "PAYROLL 13".
5. STOP ON SWITCH 7.
6. STOP ON SWITCH 6 "BREAKPOINT".

FUNCTION

The SUBTRACT verb subtracts one quantity from another and stores the result in the last-named field or the specified field.

SENTENCE FORMAT

```
SUBTRACT  { numeric-1    }          FROM  { numeric-2    }
          { field-name-1 }                { field-name-2 }

          [GIVING  field-name-3]          [ROUNDED]

          [IF SIZE ERROR GO TO sentence-name-1] .
```

CONVENTIONS

1. If the GIVING option is not present, the last-named field receives the result.
2. A numeric constant may not be used to receive the result.
3. Decimal points do not appear in stored fields, and are used only to properly align data before execution of an arithmetic operation.
4. Only a numeric may be used. If a + or - is included, it must appear as the most significant character of the numeric. If the receiving field is in floating point mode or if the operands are all integers, then rounding is ignored.
5. The ROUNDED option may be used to round off the result before it is stored in the receiving field.
6. The SIZE ERROR option may be used to truncate the most significant digits of a number.

EXAMPLES

1. SUBTRACT UNION-DUES OF MASTR~PAYROL FROM ADJUSTED-PAY OF MASTR~PAYROL.
2. SUBTRACT RECEIPTS OF TRANSAC~FILE FROM ON~ORDER~QTY OF ORDER~FILE GIVING ADJ-ORDR~QTY, IF SIZE ERROR GO TO ZERO~RTN.
3. SUBTRACT A FROM B. (Note the quantity in A is subtracted from the quantity in B, and the result is stored in B.)

VARY

FUNCTION

The VARY verb initiates and controls the repeated execution of the sentences it precedes.

SENTENCE FORMAT

Sentence-name-1. VARY field-name-1,

FROM { field-name-2 / arithmetic-expression-1 / numeric-1 }

BY { field-name-3 / arithmetic-expression-2 / numeric-2 }

UNTIL field-name-1 { relational expression / logical expression } [OR { ... }]

(A set of one or more sentences)

EXIT sentence-name-1.

CONVENTIONS

1. When the VARY sentence is first executed:
   - The FROM parameter is assigned to field-name-1 (the control variable).
   - The expression in the UNTIL phrase is evaluated.
   - If the evaluation results in a true truth-value, the sentence following the EXIT sentence is executed.
   - If a false value is obtained, the sentences following the VARY sentence are performed.
2. When the EXIT sentence that has the same sentence name (as an operand) as the VARY sentence name has been reached:
   - The BY parameter is added to field-name-1.
   - The expression in the UNTIL phrase is evaluated again.
   - The true or false value of truth-value in the UNTIL phrase causes the sentence following the EXIT sentence, or the sentences following the VARY sentence (respectively), to be executed as indicated in 1 above.

3. The EXIT sentence may be named. Caution should be exercised in transferring control to it, since it causes the BY parameter to be added to field-name-1. (Execution of the VARY sentence proceeds as under 2 above.)

4. A transfer of control from outside the VARY range into the range is undefined.

5. Any number of VARY-EXIT sentences may be imbedded within the sentences of a VARY-EXIT range.

6. Fields may be subscripted.

7. Standard rules for arithmetic expressions apply to the FROM and BY parameters and standard rules for relational and logical expressions apply to the UNTIL parameter. (See Chapter 3.)

EXAMPLES

1. SENT-5. VARY J FROM 1 BY 1 UNTIL J IS GREATER THAN 3.
   LIST (J) = LIST (J) * 16.2.
   EXIT SENT-5.

   Explanation:

   LIST has been defined in the Data Division as a field, and is the first location of a set of numbers (in this case a table of 3 numbers). The programmer desires to multiply each of the 3 numbers by 16.2 and store each result back into the table. Below is the sequence logic which occurs during the execution of the VARY sentence to accomplish the above hypothetical example:

   a. The first execution of the VARY sentence:

      1) J is assigned the value 1.
      2) J is immediately tested to determine if it is greater than 3.
      3) Since J is less than 3, control is transferred to the next sentence in sequence.
      4) LIST (J) (meaning the first value of the table since J equals 1) is multiplied by 16.2, and the result is stored back into the first location of the table.
      5) Upon reaching the EXIT sentence, J is increased by 1 and now equals 2.
      6) J is tested again for a greater than 3 value.
      7) Since J is not greater than 3, control is transferred to the VARY sentence.

   b. Second execution of VARY sentence:

      1) LIST (J) (meaning the second value of the table since J equals 2) is multiplied by 16.2, and the result is stored back into the second location of the table.
      2) Control again reaches the EXIT sentence where J is again increased by 1 making J equal to 3.
      3) J is again tested to see if J is greater than 3.
      4) Since J is not greater than 3, control is transferred to the VARY sentence.

c. Third execution of VARY sentence:

1) LIST (J) (the third value of the table) is multiplied by 16.2 and the result is stored back into the third position of the table.
2) J is increased by 1 making J equal to 4.
3) J is tested for a greater than 3 value.
4) Since J is now greater than 3, control is transferred to the sentence following the EXIT sentence.

2. SENT~14. VARY I FROM 1 BY 1 UNTIL I GR 5.
SENT~15. VARY J FROM 1 BY 1 UNTIL J GR 7.
C(I, J) = A(I, J) + B(I, J).
EXIT SENT~15.
EXIT SENT~14.

Explanation:

The above example is the basic coding required to add two matrices together. Each matrix occupies 5 rows and 7 columns. Each value of matrix A will be added to the corresponding value of matrix B. The result of the addition will be stored in the corresponding position of matrix C.

It is interesting to note that SENT-15 and its corresponding EXIT are executed 7 times to every 1 time for SENT-14 and its corresponding EXIT.

FUNCTION

The WRITE verb has the following options:

Option 1 displays a limited amount of information on the typewriter.

Option 2 releases a logical record or *group to an output file.

Option 3 releases a logical record or *group to an output printer file and slews the printer paper.

Option 4 releases a logical record or group to a DSU output buffer, and optionally releases the entire buffer for recording onto the DSU.

SENTENCE FORMAT

Option 1:

```
WRITE { field-name-1    } [ , { field-name-2    } ... { field-name-n    } ]
      { element-name-1  }     { element-name-2  }     { element-name-n  }
      { literal-1       }     { literal-2       }     { literal-n       }

    ON TYPEWRITER.
```

Option 2:

```
WRITE { record-name } [ { RECORD } ] .
      { *group-name }   { GROUP  }
```

Option 3:

```
WRITE { record-name } [ { RECORD } ]   ADVANCING [ { integer LINES       } ] .
      { *group-name }   { GROUP  }                 { field-name-LINES   }
                                                   { TO TOP OF PAGE     }
```

Option 4:

```
WRITE [AND RELEASE]   { record-name } [ { RECORD } ] .
                      { *group-name }   { GROUP  }
```

CONVENTIONS

In Option 1:

1. Literals may be used to identify the contents of the data fields or elements displayed in order to give meaning to the display or to identify subsequent "read control switch loops" caused by STOP sentences or READ (Option 1) sentences. (See explanations under STOP and READ verbs.)

2. The object program assumes that the typewriter carriage has previously been returned to the left margin. Normally, all of the data specified in the sentence is typed 84 characters to a line. The carriage is automatically returned after the 84th character on a line and/or after the last character of the specified data has been typed.

   There is no keypunch character available for a carriage return punch. Each installation must choose some set of characters such as c/r which will signal a 12-7-8 punch to the keypunch operator. These characters may then be inserted as the first character of a literal to be displayed on the typewriter when a new line is to be started. For example:

55	56	57 80
"	c/r	START NEW LINE".

3. Additional typewriter action symbols may be included in literals so that the user may edit his own typeouts. Literals should be composed of typewriter characters and/or typewriter control symbols only, excluding the quotation mark character. The quotation mark identifies the beginning and ending of a literal and, therefore, cannot be part of a literal.

In Option 2:

1. An OPEN sentence must be executed before the first WRITE is given for a particular file.

2. If a record or *group is to be printed, it will be edited as specified on the Data Division form.

3. The data description of the output records or *groups must contain a list of all data intended for output. When a WRITE is executed, only the data listed in the record or group description are moved to the output file. The move is automatic and implied with each execution of the WRITE.

4. The actual writing on tapes, punching of cards, etc., is controlled by routines which are generated by the compiler.

5. After recognition of the end-of-reel of a tape file, the WRITE performs the following operations:

   - The standard ending-reel label procedure and the user's ending-reel label procedure (if specified by the USE clause in the INPUT-OUTPUT CONTROL sentence).

   - A tape handler swap if more than one handler is assigned to the file. If one handler is assigned to a multiple reel file, the object program will wait for a blank reel to be mounted.

- The standard beginning reel label procedure and the user's beginning reel label procedure (if specified by the USE clause in the INPUT-OUTPUT CONTROL sentence).

- Writes the record or *group on the new reel.

In Option 3:

1. See Conventions 1, 3, 4, 5, 7, and 8 for the ADVANCE verb and Conventions 1 through 4 for the WRITE (Option 2) sentence.

2. The printer is slewed the specified number of lines after the WRITE sentence. The amount of the advance is added to the LINE~COUNT entry for the specified file. Note that ADVANCING 1 LINE is redundant since the WRITE (Option 2) sentence would do the same thing. An ADVANCING 0 LINES is legitimate, and causes no slewing after printing.

In Option 4:

1. An OPEN sentence and a READY sentence must be executed before the first WRITE sentence is given for a particular DSU file.

2. A WRITE sentence without the RELEASE option is used to build an output record in the output buffer. In this case a RELEASE or WRITE RELEASE sentence must be given before recording is done on nonsequential files.

3. A WRITE sentence with the RELEASE option is used to build an output record in the output buffer and then start the physical recording on the DSU. This option is ordinarily used for writing records of nonblocked files, and for writing the last record of a block in blocked files. See the conventions for the RELEASE verb.

4. The object program will stop if a WRITE sentence is given for a nonsequential file and there is not room for the record in the buffer. The programmer must insure that the RELEASE option is given at the proper time.

EXAMPLES

Option 1:

1. WRITE "UNMATCHED△" CLOCK~NUMBER OF TIME~CARD ON TYPEWRITER.

   Note that this sentence will cause the literal UNMATCHED to be typed out, followed by the contents of the CLOCK~NUMBER field. For example, if the CLOCK~NUMBER field contained the value 727313, the following line would be printed on the typewriter:

   UNMATCHED 727313.

Option 2:

1. WRITE RECORD~1 OF FILE~6.
2. WRITE INVENTORY~1 RECORD.
3. WRITE XYZ GROUP.
4. WRITE PAY~GROUP OF PAY~REC OF PAY~FILE GROUP.

Option 3:

1. WRITE SHIPMENT RECORD ADVANCING 5 LINES.
2. WRITE SHIPMENT ADVANCING 5.
3. WRITE SHIPMENT ADVANCING COUNTER~1 LINES.
4. WRITE SHIPMENT ADVANCING TOP.
5. WRITE SHIPMENT RECORD ADVANCING TO TOP OF PAGE.

Option 4:

1. WRITE INVENTORY~1 RECORD.
2. WRITE AND RELEASE INVENTORY~1 RECORD.

7. ENVIRONMENT DIVISION

PURPOSE

The Environment Division centralizes those aspects of the total data processing problem which are dependent upon the physical characteristics of the GE-200 Series computer. It provides a linkage between the data described in the Data Division and the peripheral hardware devices on which the data is stored.

ORGANIZATION

The Environment Division consists of four sentences, and its presence is indicated by the words:

ENVIRONMENT DIVISION.

These words followed by a period are written on the GECOM Sentence Form beginning in Column 8. No other information may be written after the period.

ENVIRONMENT SENTENCES

The 5 sentences comprising the Environment Division are:

```
OBJECT~COMPUTER.△225 MEMORY SIZE . . .

I~O~CONTROL. RERUN ON . . .

FILE~CONTROL. SELECT . . .

DSU~CONTROL. SELECT . . .

COMPUTATION~MODE. USE . . .
```

Each of these sentences is written on the GECOM Sentence Form and may begin in Column 8 or may be indented any number of spaces to the right of Column 8. All indentation columns must be blank.

The remainder of this section explains the function of these sentences and the conventions which must be followed to complete the Environment Division portion of a GECOM source program.

OBJECT~COMPUTER

FUNCTION

The OBJECT-COMPUTER sentence describes the computer system upon which the object program is to be run.

SENTENCE FORMAT

```
OBJECT~COMPUTER { 215 }  [ MEMORY SIZE integer-1 MODULE [S] ]
                { 225 }
                { 235 }

  [ [integer-2] hardware-name-1 [integer-3] hardware-name-2 ....]

  [ASSIGN [OVERLAY~SEGMENTED] OBJECT~PROGRAM TO input-hardware-name

  [integer-4 [ON PLUG integer-5]]]

PLACE [MAIN] SEGMENT IN { UPPER } MEMORY
                        { LOWER }

  [RELOCATE BY integer-6 WORDS IN LOWER MEMORY]

  [RELOCATE BY integer-7 WORDS IN UPPER MEMORY]

  [API OPEN IS xxxxx [API CLOSE IS xxxxx]]

  [MIC ENTRANCE IS xxxxx]

  [BEGIN COMMON~STORAGE AT xxxxx]

  [BEGIN *COMMON~STORAGE AT xxxxx] .
```

CONVENTIONS

1. An OBJECT~COMPUTER sentence is required in the source program. Within the OBJECT~COMPUTER sentence, a PLACE clause is required. IF PLACE IN UPPER clause is specified, a MEMORY SIZE 4 MODULES option must be specified.

2. When giving the memory size, one module of memory is 4096 20-bit words. The entry for integer~1 may be the numerals 1, 2, or 4. An error message will occur during compilation if the specified memory size is less than the size needed to run the compiled object program.

3. The words "hardware-name-n..." are used to name hardware devices of the object computer. When applicable, integer-n is used to give the quantity of a particular hardware device. "Hardware-name-" may be any of the following standard names or their abbreviations.

AUTOMATIC PRIORITY INTERRUPT	API
CARD PUNCH	CP
CARD READER	CR
DISC STORAGE UNIT(S)	DSU(S)
FLOATING POINT HARDWARE	FLPT
HIGH SPEED { PRINTER / PRINTERS }	HSP
MAGNETIC { TAPE / TAPES }	MT
MOVE	MOVE
PLUG	PL

4. If the MOVE option is listed as a hardware name, the compiler will select appropriate object program subroutines which employ the optional MOVE command during word moves.

5. The ASSIGN clause is intended primarily for MAIN segments. Its usage determines which run-completion subroutine is included in the object program at the STOP RUN entry and also the output medium on which the object program is produced. The BRIDGE II compatible run-completion subroutine (*RC) is produced whenever the complete object program is to be run from magnetic tape (options 1, 3, and 4 below); otherwise, the run-completion subroutine (RLC or RL*) is produced. RLC and RL* assume that the next program is in the card reader. RL* is produced when API is present.

Option	ASSIGN to:	OVERLAY~ SEGMENTED	OUTPUT Medium	Run-Completion Subroutine
1	No ASSIGN	#	Cards	*RC
2	CR	#	Cards	RLC or RL*
3	MT	No	MT	*RC
4	MT	Yes	Cards	*RC
5	DSU	No	Cards	RLC or RL*
6	DSU	Yes	Cards	RLC or RL*

- illegal combination

Option 3 may be used only when the total object program contains no independently compiled segments. Even if the total object program contains no independently compiled segments, the user may prefer to use Option 1. In this case, the object program can be checked out from cards and then put onto magnetic tape via the BRIDGE II Operating Service System, CD225J1.001.

If the total object program contains segments and is eventually to be run from magnetic tape, Option 1 or 4 should be used. After each segment deck is checked out, all segments can be consolidated onto magnetic tape via BRIDGE II.

If the total object program is to be run from disc storage, Option 5 or 6 should be used. After all decks are checked out, the total object program can be put onto magnetic tape via BRIDGE II. The BRIDGE-to-DSU Absolute Translator (BRAT) programming routine, CD225E2.005R, is then used to place the program onto DSU.

The OVERLAY~SEGMENTED option should be specified in the MAIN segment ASSIGN clause whenever the total object program contains overlay segments. Note that OVERLAY~SEGMENTED object programs cannot be assigned to the card reader. They must be run from magnetic tape or disc storage.

If a segment other than a MAIN segment is being compiled, the ASSIGN clause need not be given. If it is given, it serves as documentation only.

6. Magnetic tape and plug numbers may be given using the words zero, one, . . . , seven or the numerals 0, 1, . . . , 7.

7. The PLACE. . . IN UPPER MEMORY option causes the compiler to assign the body of object coding to the upper 8k words of a 16k memory. This does not include common constants, file tables, COMMON~STORAGE subroutines, etc. The PLACE. . . LOWER option causes the compiler to assign the entire segment (except *COMMON~STORAGE) to lower 8k memory.

   MAIN must be specified for a MAIN segment of for a total program produced in one compilation (no independently compiled segments). If the MAIN option is not specified, the segment is assumed to be other than the MAIN segment. (See Procedure Division, Segments.)

8. To specify that an object program may be interrupted, the user must indicate the presence of Automatic Priority Interrupt (API) in the list of hardware names. Inclusion of the API hardware name or its abbreviation causes the compilation of an object program which can be interrupted by a priority program. It does not cause compilation of a priority program. API must be specified whenever DSU files are assigned. If any segments of a complete object program specify API in their OBJECT~COMPUTER sentence, all of the segments should list API as a hardware name.

9. An interruptable object program turns API off before any typing and turns API on after completing any typing. The object program input/output routines turn API off whenever functions must be executed which cannot be interrupted and turn API on after such functions are completed.

10. The RELOCATE clauses cause the object program (main control load or overlay segments) to be displaced by the specified number of words from the normal position in lower memory and by the specified number of words from the normal position in upper memory. The UPPER option may be used only when the PLACE clause specifies upper memory. See Procedure Division, Overlay Segmentation.

    An alternative method of displacing the object program does not require the RELOCATE clauses. The user may manually punch the lower and upper relocation amounts into the type 1 card required by the Multicapability Modular Loader II (MCML II Loader), CD225B1.006R.

11. When API is specified for a MAIN segment, the object program card read areas become 00400_8 and 00600_8. The object program or priority program card punch areas are 01000_8 and 01200_8. The MCML II loader, depending on the modules used, can load with a relocation constant as low as 01116_8 for object programs with API. If buffered card punching is required at object time, either in the total object program or in a priority program, the main segment's relocation constant must be adjusted accordingly. The RELOCATE option can be used to do this. Alternatively, the user may wish to punch the relocation amount into the type 1 card required by MCML II.

    Example: Normal loading starts at 01116_8. Card punch buffering is required.

    RELOCATE BY 00092 WORDS IN LOWER MEMORY

    This causes the lower relocation constant to become 01252_8, leaving enough room for the second punch buffer at 01200_8.

12. In the RELOCATE clauses, integer-6 and integer-7 must be 5-digit decimal numbers, zero filled on the left if necessary.

13. The following conventions apply if API is included in the list of hardware names and the MAIN option is specified in the PLACE clause:

- If neither API OPEN nor API CLOSE clauses are given, the compiler provides the Automatic Program Interrupt (API) Executive, CD225J4.000R, as an object program subroutine and sets up the API open and close linkage via the loader so that the object program will transfer control to the API Executive to initialize the API function at the start of the program and to perform the close function at at the STOP RUN sentence.

- If the API OPEN clause is given without an API CLOSE clause, the compiler will not include the API Executive among the object program subroutines, but the object program will transfer to the specified absolute decimal address to initialize the API function at the start of the program. However, no API close function is performed at the STOP RUN sentence.

- If both API OPEN and API CLOSE clauses are given, the compiler will not include the API Executive among the object program subroutines but the object program will transfer to the absolute decimal address specified for the OPEN clause to initialize the API function at the start of the program and to the absolute decimal address specified for the CLOSE clause to perform the close function at the STOP RUN sentence.

- Whenever the API OPEN entrance is given, it is assumed that prior to the loading of the object program the API Executive will have been loaded into its fixed memory area as part of a priority control program.

  Linkage subroutines AP2 and/or MI2 are required in the object program whenever the API OPEN clause is given. To obtain the variable AP2 and/or MI2 subroutines, the user should put console switch 1 down for the compilation whenever the object program is punched on cards.

- The API CLOSE clause may be given only if the API OPEN clause is given. The API CLOSE clause need not be given, however, if the object program is to be one in a string of main programs which are loaded while a priority program is operating. In this case, only the last main program of the string would have the API CLOSE clause.

14. If DSU files are to be processed, the SIOS (MIO), CD225E8.000, will be supplied by the compiler as an object program subroutines unless an MIO ENTRANCE clause is given. The MIO ENTRANCE clause must never be given unless the API OPEN entrance is also given. When the MIO ENTRANCE clause is given, the compiler will not include SIOS (MIO) in the object program subroutines. Instead, the object program will transfer control to the entrance to call SIOS (MIO). SIOS (MIO) is assumed to have been loaded into its fixed memory area as part of a priority control program.

    Note: The address to be given should be an absolute, even decimal number. The actual entrance to MIO will be that number plus one.

15. The BEGIN COMMON~STORAGE clause causes the compiler to assign the designated lower 8k memory location as the upper limit of the COMMON~STORAGE area.

16. The BEGIN *COMMON~STORAGE clause causes assignment of the designated upper 8k memory location as the upper limit of the *COMMON~STORAGE area.

17. In the clauses where xxxxx appears, xxxxx represents a 5-digit decimal address, zero filled on the left if necessary. For *COMMON~STORAGE, if this address is 08191 or less, it is assumed to be relative to upper 8k memory.

18. An error message will occur during compilation if no location is specified in a BEGIN *COMMON~STORAGE or BEGIN COMMON~STORAGE clause.

19. When a MAIN segment has been assigned to disc storage, BRAT assigns a directory table to the object program. The table contains the DSU addresses of any overlay segments required at object time. The table is placed below *Common~Storage if one has been specified. If there is no *Common~Storage, the table is placed in lower memory. Thus, the user may wish to assign a *Common~Storage address (even though it is not required by the object program) to force the directory table into upper memory.

EXAMPLES

1. OBJECT~COMPUTER. 225, MEMORY SIZE 2 MODULES, 8 MAGNETIC TAPES, 1 HIGH SPEED PRINTER, ASSIGN OBJECT~PROGRAM TO MAGNETIC TAPE FIVE ON PLUG ONE.

2. OBJECT~COMPUTER. 225 8 MT 1 HSP ASSIGN OBJECT~PROGRAM MT 1 PL 1.

3. OBJECT~COMPUTER. 225 6 MT, 1 CR, 1 CP, 1 HIGH SPEED PRINTER, ASSIGN OBJECT~PROGRAM TO MT TWO ON PL 1.

4. OBJECT~COMPUTER. 225 MEMORY 4, HSP CR CP 6 MT FLPT ASSIGN OBJECT~PROGRAM CR, PLACE SEGMENT IN UPPER MEMORY, BEGIN *COMMON~STORAGE AT 16000.

5. OBJECT~COMPUTER. 225 MEMORY 4, HSP, CR, CP, 6 MT DSUS, FLPT, API, ASSIGN OVERLAY~SEGMENTED OBJECT~PROGRAM TO DSU 3 PLUG 0, PLACE MAIN IN UPPER, RELOCATE BY 00092 LOWER, API OPEN IS 08020, API CLOSE IS 08009, BEGIN COMMON~STORAGE AT 08000.

6. OBJECT~COMPUTER. 225 MEMORY 4, HSP, CP, API, CR, MOVE, 6 MT, FLPT, 1 DSU, PLACE MAIN IN UPPER MEMORY RELOCATE BY 02000 LOWER, RELOCATE BY 04172 UPPER MEMORY, API OPEN IS 07116, API CLOSE IS 07127, MIO ENTRANCE IS 06924, BEGIN COMMON~STORAGE AT 06900, BEGIN *COMMON~STORAGE AT 16300.

FUNCTION

The I-O-CONTROL sentence is used to indicate nonstandard label checking, rerun information, those tapes which contain more than one file, and assignment of more than one DSU file to the same buffer area.

SENTENCE FORMAT

```
I-O-CONTROL. [ RERUN [ON output-hardware-name integer-1 ON PLUG integer-2]
                     EVERY { END OF REEL
                           { END OF REEL OF file-name-1
                           { integer-3 RECORDS OF file-name-1 } ]

[MULTIPLE FILE { INPUT  } TAPE CONTAINS  file-name-4
               { OUTPUT }
        [POSITION integer-4] [file-name-5
        [POSITION integer-5]    ...]]    [MULTIPLE FILE ...]

[SAME AREA { INPUT  } file-name-2 AND { INPUT  } file-name-3
           { OUTPUT }                 { OUTPUT }
      [ { INPUT  } ...]]    [SAME ...]
        { OUTPUT }

[USE section-name-1 AFTER STANDARD ERROR PROCEDURE ON { file-name-6 }
                                                      { INPUT      }
                                                      { DSU OUTPUT } ]

USE section-name-2 AFTER STANDARD

[{ BEGINNING } ] [{ REEL } ] LABEL PROCEDURE ON file-name-7].
 { ENDING    }   { FILE }
```

CONVENTIONS

1. This sentence is optional and is required only when one of the above clauses is needed for completing a source program.

2. If the RERUN option is specified, it is necessary to indicate the rerun point and where the rerun memory dump is to be written.

   Memory dumps are written either at the end of each reel of an output file or on a separate rerun tape.

   If the memory dump is to be on a tape other than the output file tape, the tape number must be specified as the hardware-name after the rerun is indicated:

   RERUN [ON output-hardware-name integer-1 ON PLUG integer-2]

   Rerun points may be established at:

   - Every end of reel of file-name-1, where file-name-1 is a particular output file, and the memory dump is to be placed at the end of each reel after the end-of-tape reflector. In this case hardware-name is not required. For example, RERUN EVERY END OF REEL OF UPD~INVNTRY.

   - Every end of reel of file-name-1; where file-name-1 is an input or output file and the memory dump is to be placed on a separate rerun tape. In this case, the hardware-name must be specified.

   - A number of records (integer-3) of an input or output file have been processed. In this case hardware-name should be specified. Integer-3 records is the physical record (block) count and not the logical record count.

   - Every end of reel; this pertains to all output files in the source program. Hardware-name must be specified.

     RERUN EVERY integer-3 RECORDS and RERUN EVERY END OF REEL clauses must not both be stated for the same file.

3. The MULTIPLE FILE option is required when more than one file shares the same physical reel of tape. Regardless of the number of files on a single reel, only those files which are used in the object program should be specified. If all file names have been listed in consecutive order, the POSITION option need not be given. However, if there are any intervening files on the tape not referenced by the program, then position must be given for each file referenced. All labels must be either present or omitted on a multiple file tape. If labels are present, the positions for output multiple file tapes are placed into the labels of the individual files. There can be any number of multiple file input or output tapes. However, all files listed for each tape must be contained on a single reel. Optional files are not permitted on multiple file tapes. Note that not more than one file on a multiple file tape can be processed at the same time, that is, a file must be closed before another file on the same tape can be opened.

4. There are two types of USE clauses for files not assigned to the DSUS. The first, USE ... AFTER... ERROR procedure, applies to input files. If a tape error occurs during reading, a standard error procedure attempts to successfully perform the read five times. When it is impossible to successfully read the block, the data from the first logical record is unpacked and made available to the programmer for use in the AFTER section. He may type the key fields or any message. A transfer of control is made to the AFTER.... ERROR section for each logical record in the block so that all key fields in the bad block may be recorded. The next block is then read and processing continues.

   The second USE clause, USE... AFTER... LABEL, allows the programmer to enter a section in which he may access the contents of an input label record, or calculate values for an output label. When this clause is used on input, the contents of the label record are unpacked and available for use by the AFTER Section. In the USE clause, section-name-2 is executed as follows:

   - After the standard input label check.
   - After a standard output label is created but before it is written.
   - When both beginning and ending labels are being checked, do not specify BEGINNING and ENDING clauses.

5. The USE... AFTER... ERROR clause may be specified for files assigned to DSUS. After an error is detected (see SIOS (MIO), CD225E8.000), control is transferred to the section named so that the program can be brought to an orderly halt. In the case of an input file error, a return from the section causes the program to accept the data in the input buffer; in the case of an output file, a return from the section allows the program to continue. Under no circumstances should the section attempt to use the DSU.

6. USE clauses may not be specified for a Journal Tape file.

7. The SAME AREA option may be used to assign more than one DSU file on the same buffer area. More than one of the files assigned to the same area may be open at the same time. It is the programmer's responsibility to know which file occupies the area at any given time. When an input file is being read, updated, and written as an output file from the same area, the description of the output file should be the same, field by field, as the description of the shared input file. A file described as buffered cannot share the same area with an unbuffered file if they are open at the same time. When DSU files are sharing the same area and are open at the same time, they must be assigned to the same plug.

EXAMPLES

1. I-O-CONTROL. RERUN ON TAPE 7 ON PLUG 1 EVERY END OF REEL OF MASTER FILE.

2. I-O-CONTROL. RERUN ON TAPE 0 ON PLUG 1 EVERY 1000 RECORDS OF MASTER-PLCY, USE LABEL-RTN AFTER STANDARD ENDING REEL LABEL PROCEDURE ON TRANS-FILE.

3. I-O-CONTROL. MULTIPLE FILE INPUT TAPE CONTAINS EMP-FILE POSITION 6, PAYROL-FILE POSITION 8, TAX-FILE POSITION 11.

4. I~O~CONTROL. SAME AREA INPUT STK~MASTER OUTPUT OUT~STOCK, SAME AREA INPUT TRAN~FILE, INPUT INQUIRY, AND INPUT SCRATCH~PAD.

5. I~O~CONTROL. SAME AREA INPUT STK~MASTER AND OUTPUT OUT~STOCK, INPUT FIN~FILE OUTPUT FINAN, SAME AREA OUTPUT WORKING~FILE, OUTPUT RECAP.

FUNCTION

The FILE~CONTROL sentence identifies the input/output files and provides for their assignment to specific input/output hardware units.

SENTENCE FORMAT

```
FILE~CONTROL.  SELECT  [OPTIONAL]  file-name-1  ASSIGN TO
        hardware-name-1  integer-1  [ON PLUG integer-2] [FOR OFF LINE PRINT]
        [hardware-name-2 ...]  [BUFFER]  [FOR MULTIPLE REEL]  [SELECT ...].
```

CONVENTIONS

1. The FILE~CONTROL sentence is optional if the source program does not process input/output.

2. The word OPTIONAL is required for input files which are not necessarily present each time the object program is run. Optional files may not be contained on multiple file tapes, output tapes, or assigned to the card reader or DSUS.

3. All input and output files used in the program must be assigned to an input or output unit (hardware-name). If plugs are not specified, plug 1 is assumed for magnetic tapes and plug 6 for printer.

4. The MULTIPLE REEL option must be included when a magnetic tape file exceeds one reel of tape. All reels of a multiple reel file must be mounted on tape units associated with the same tape controller. A single magnetic tape unit is also permissible for a multiple reel file. All multiple reel input files must contain the standard magnetic tape labels.

5. The same tape unit must be assigned to all files existing on the same reel. It is not necessary to mention every file-name on a multifile reel. Only those file-names used and their relative position (in ascending order) on the reel are needed.

6. Buffering is supplied for those files for which the BUFFER option is stated. It is recommended that short files such as error files and tables not be buffered, making more efficient use of storage in the object program.

7. The OFF LINE phrase must be specified for report tapes to be created by the Report Writer for deferred printing. This applies whether the off-line printer or Peripheral Package (PIP), CD225E1.009, for on-line printing is employed. In this case, file-name-1 is the name specified in the Report File Definition entry in the Report Section. (See Report Writer Section.)

EXAMPLES

1. FILE~CONTROL. SELECT MASTR~PAYROL, ASSIGN TO TAPE 1 PL 1, MULTIPLE REEL, SELECT TIME~CARDS, ASSIGN TO MT 4 PL 1, SELECT NEW~MST~PR, ASSIGN TO MT 1 ON PLUG ONE, TAPE 2 PL 1, MT 3 ON PLUG 1 BUFFER FOR MULTIPLE REEL, SELECT PAYROL~RGSTR, ASSIGN TO TAPE 4 ON PLUG ONE TAPE FIVE ON PLUG ONE BUFFER MULTIPLE, SELECT UNION~DUES ASSIGN TO MT 6 PL 1, SELECT BOND~REGISTR, ASSIGN TO MT 7 PL 1 BUFFER.

2. FILE~CONTROL. SELECT MASTR~PAYROL ASSIGN MT 1 MT 2 MULTIPLE SELECT TIME~CARDS ASSIGN MT 3 SELECT NEW~MST~PAYR ASSIGN MT 4 MT 5 MULTIPLE SELECT PAYROL~RGSTR ASSIGN MT 6 MT 7 MULTIPLE SELECT UNION~DUES ASSIGN MT 1 PL 2 SELECT BOND~REGISTR ASSIGN MT 2 PL 2.

FUNCTION

The DSU~CONTROL sentence identifies the input/output files assigned to the DSUS and describes their usage.

SENTENCE FORMAT

```
DSU~CONTROL. [SELECT JOURNAL~TAPE ASSIGN TO MT integer~1 [ON PLUG integer~2]]
    SELECT [SEQUENTIAL] [BLOCKED] file~name~1
    ASSIGN TO [JT AND] DSU integer~3 [ON PLUG integer~4] [DSU ....]
    [READ AFTER WRITE] [USE field~name~1 FOR UNIT NUMBER] [RESERVE ALTERNATE AREA]
    [SELECT ....]
```

CONVENTIONS

1. The DSU~CONTROL sentence follows the FILE~CONTROL sentence.

2. The SELECT JOURNAL~TAPE option is used to assign the Journal Tape to a magnetic tape unit. (See Appendix J.)

3. The JOURNAL TAPE (JT) option may be exercised by output file~name assignments to indicate that each time a record or block of the file is recorded on the DSUS, the same information plus two words is to be written on the Journal Tape. The first additional word contains the disc storage unit code in bits 5-7 and the plug number in bits 11-13. The second additional word contains the DSU address where the recording was done. (See Tape-to-DSU routine (TAPER), CD225E8.003.)

4. The recording mode of the Journal Tape is always binary. Label information should be given in the Data Division under the fixed file name JOURNAL~TAPE.

5. The Journal Tape may be opened or closed as any other tape file. However, unlike other files, it does not have to be opened or closed. (Beginning-tape label will not be written on the Journal Tape and it will not rewind if it is not opened.)

6. A file may be assigned to more than one disc storage unit, but all of the disc storage units used by the file must be on the same plug. DSU files should be assigned to plug 0 or 1. If no plug is specified, plug 0 is assumed.

7. The USE field~name option should be included only when a file is assigned to more than one disc storage unit. The field named must be a Working-Storage or Common-Stroage field of description 9 (n) where n is less than or equal to 5. The contents of the field at object time determines which disc storage unit is being addressed. The field must contain a 0, 1, 2, or 3. The programmer must ensure that the field contains the proper unit number at object time.

8. Output files may use the READ AFTER WRITE option. When used, the information is parity checked with a special read after it has been written on the DSU. This requires an additional disc revolution for the read back.

9. The SEQUENTIAL option is specified when it is desired to process the file in a sequential manner starting at some DSU address. Only one READY sentence needs to be issued to assign the initial DSU address. Thereafter, when additional input blocks are needed from the DSU or an output buffer is full, a new DSU address is developed by the object program. The number of frames required for the block is added to the DSU address to develop a new DSU address. If the new address is illegal (frames 96-127) a decimal 32 added to the address to make it legal.

10. The RESERVE option may be used only with files described by the SEQUENTIAL option, and allows the assignment of a second buffer area.

11. If the records of a file are blocked, the BLOCKED option must be specified.

EXAMPLES

1. DSU~CONTROL. SELECT JOURNAL~TAPE ASSIGN TO MT 6 PLUG 1, SELECT MASTER ASSIGN TO JT AND DSU 0 PLUG 0, SELECT SCRATCH~PAD ASSIGN TO DSU 0 PLUG 0, SELECT TRAILER ASSIGN TO JT AND DSU 1 PLUG 0.

2. DSU~CONTROL. SELECT SEQUENTIAL, BLOCKED SEARCH~DSUS ASSIGN TO DSU 0 PLUG 0, DSU 1 PLUG 0, DSU 2 PLUG 0, DSU 3 PLUG 0, USE CURRENT FOR UNIT NUMBER, RESERVE ALTERNATE AREA.

3. DSU~CONTROL. SELECT STOCK~FILE ASSIGN TO DSU 3 PL 0, DSU 2 PL 0, USE STOCKUN FOR UNIT NUMBER, SELECT OUT~STOCK ASSIGN TO DSU 3 PL 0, DSU 2 PL 0, READ AFTER WRITE, USE STOCKUN FOR UNIT NUMBER.

FUNCTION

The COMPUTATION~MODE sentence is used to indicate that data is to be in floating point format and that calculations are to be performed using floating point arithmetic.

SENTENCE FORMAT

```
COMPUTATION~MODE.  USE  { HARDWARE   }  { FLOATING POINT }
                       { PROGRAMMED }  { FLPT          }  .
```

CONVENTIONS

1. This sentence is optional.
2. Hardware floating point calculations are done in the Auxiliary Arithmetic Unit.
3. All data is stored in floating point format except that which is described as integer, fixed point, and true-false in the Integer, Working~Storage, Common~Storage, and True~False Sections of the Data Division.

8. IDENTIFICATION DIVISION

PURPOSE

The Identification Division enables the programmer to label the source program as well as the outputs of a compilation.

ORGANIZATION

The Identification Division may consists of one or more sentences. The division is indicated on the sentence form by the following heading which starts in Column 8:

IDENTIFICATION DIVISION.

The heading is followed by a period but no other information is entered on the same line.

The sentences which may be entered in the division are:

```
PROGRAM~ID.        ...
NEXT~PROGRAM.      ...
AUTHOR.            ...
DATE~COMPILED.     ...
INSTALLATION.      ...
SECURITY.          ...
REMARKS.           ...
```

Each sentence is written on the GECOM sentence form beginning in Column 8 or indented any number of columns. Each of the above sentence names is followed by a period and at least one space before the sentence itself.

CONVENTIONS

1. The PROGRAM~ID sentence is required and may consist of nine or less BCD typewriter characters as the name of the program. A space (blank), comma (,), or period(.) is interpreted as the end of the name.

   The compiler inserts the run name left justified into the header card of the object program. If the *RC run-completion subroutine (see STOP verb) is required, the run name is placed into the *RC calling sequence. The run name is also inserted in the heading on each page of the Edited List.

2. The NEXT~PROGRAM sentence is optional and may consist of nine or less BCD typewriter characters representing the name of the next run to be executed at object time. The name must be terminated by a period. If the *RC run-completion subroutine is required, the next run name is inserted into the *RC calling sequence.

3. The AUTHOR sentence is optional. The sentence itself may contain 30 or less BCD characters followed by a period.

   If given, the author's name will appear in the heading on each page of the Edited List.

4. The DATE COMPILED sentence is optional. The sentence may consist of 30 or less characters followed by a period.

   If given, the date of compilation will appear in the heading on each page of the Edited List.

5. The above four sentences may be written in any order. They may be followed by any other sentences which the compiler simply reproduces on the Edited List.

6. The INSTALLATION, SECURITY, and REMARKS sentences are all optional, and if used, may contain any information the programmer requires to be reproduced on the Edited List.

EXAMPLE

```
IDENTIFICATION DIVISION.
PROGRAM~ID. PAYROLL 13.
NEXT~PROGRAM. PAYROLL 14.
AUTHOR. GEORGE GECOM.
DATE~COMPILED. APRIL 25, 1961.
INSTALLATION. GENERAL ELECTRIC.
SECURITY. CLASSIFIED.
REMARKS. GROSS TO NET RUN.
```

9. DATA MANIPULATION

OBJECT PROGRAM DATA STORAGE AND MANIPULATION

Data Storage-General

The following explanations and descriptions are presented to assist the GECOM user in understanding his object program and improving its efficiency. The compiler uses the field data descriptions to determine storage mode. The data images directly affect the coding produced for object data manipulation. Carefully written data descriptions are necessary to enable the compiler to produce an accurate and efficient object program.

Numeric Fields

A numeric field has a value in relation to other numeric fields or zero. The data image of a numeric field represents the maximum range of its value. A numeric field must be described using only the symbols 9, +, -, T, I, K, R, ., V, P, E, $, comma(,), Z, and *.

The data image of a numeric field is used by the compiler to determine the range and decimal point alignment when used in conjunction with other numeric fields. A field should be described as numeric only when the compiler must know its range and decimal point alignment.

Numeric constants appearing in the Constant Section are stored as numeric fields.

Alphanumeric (or Alphabetic) Fields and Elements

An alphanumeric field or element consists of any mixture of characters comprising the computer's character set. This type of field or element should be thought of as a string of characters. It has no "value" as associated with a numeric field. An alphanumeric field must be described using only the symbols A and/or X.

Literal constants appearing in the Constant Section are stored as alphanumeric fields.

Procedure Division Numeric Constants

Numeric constants may appear in procedure sentences in conjunction with numeric fields. The compiler produces a numeric field from the numeric constant as written. The numeric constant is stored in the mode indicated in the data description of the numeric field. Numeric constants must be written with the same care as numeric fields are described, since the compiler must, in effect, create a data image for the numeric constant.

The only sentence in which a numeric constant may appear without an accompanying numeric field is in an ADVANCE sentence.

Procedure Division Literal Constants

Literal constants may appear in procedure sentences in conjunction with alphanumeric (or alphabetic) fields or elements. An alphanumeric field is produced by the compiler from the literal constant as written. Literal constants always appear as strings of characters and should be thought of as alphanumeric fields. Literal constants may appear without an accompanying alphanumeric field or element in the STOP and WRITE, Option 1, sentences.

Figurative Constants

Figurative constants may be used in procedure sentences to imply strings of characters.

Figurative constants may be used in conjunction with either alphanumeric or numeric fields. If used with an alphanumeric field or element, a string of characters as represented by the figurative constant is created as an alpahnumeric field. If used with a numeric field, the figurative constant is stored according to the data image of the numeric field (for example, "ones" used with a numeric field data image of 999V999 yields 111V111 as a numeric constant). The created numeric constant is stored in the same mode as the numeric field.

Figurative constants should be used only as a convenience for implying strings of characters.

Process Storage

The compiler analyzes the Procedure Division to decide when an input field or element should be stored outside of the input record for more efficient Procedure Division manipulation. This storage area (outside of input, Working Storage, and output areas) is called Process Storage since it contains the fields and elements being processed by the Procedure Division. Process Storage is a compiler generated extension of Working Storage.

Any input field or element referenced by a procedure sentence other than MOVE, EXCHANGE, or WRITE, Option 1, sentences is placed in Process Storage in a format compatible with data manipulation techniques and the computer instruction repertoire.

All numeric fields placed in Process Storage are stored in either fixed or floating point binary mode, depending on the COMPUTATION MODE sentence in the Environment Division and the True-False and Integer declarative sections. If the computation mode is floating point, all numerics other than integers and true-false variables are stored in floating point binary mode. If the COMPUTATION MODE sentence is absent from the Environment Division, all numerics are stored in fixed point binary mode. Integers and true-false variables are always stored in fixed point binary mode at a binary scale of 19.

All alphanumeric fields placed in Process Storage are stored in BCD mode, left justified with no significance as to fill on the right. Alphanumeric fields in Process Storage are always unpacked.

Array fields appearing in Process Storage are stored consecutively. Numeric arrays are stored under the same conventions that determine numeric field storage. Alphanumeric arrays are stored unpacked in an integral number of words.

Working Storage

Fields and arrays named in Working Storage are stored under the same conventions as Process Storage fields and arrays.

The compiler analyzes the Procedure Division to determine if a Working Storage field is referenced. All fields described under the Working Storage Section are assigned memory space if referenced in any procedure sentence. If a Working Storage field is never referenced, it is never assigned a storage area.

Elements of Alphanumeric Fields

An element when used in procedure sentences, other than the EXCHANGE, MOVE, and WRITE, Option 1, sentences is stored separate from its parent field as an alphanumeric field. When the element is a destination under the MOVE sentence, the compiler provides for updating the parent field. Also, the compiler provides for updating any elements that overlap the original element via character position. When the parent field is a destination under MOVE or EXCHANGE sentences, the compiler provides for updating all elements of the parent field.

Object Program Action in Executing a READ Sentence

The following actions take place when the object program executes the coding corresponding to a READ, Option 2, sentence:

1. The next logical record (or *group) from the specified file is made available to the Procedure Division.

2. All fields and elements that have been assigned to Process Storage by the compiler are moved from the input record to their assigned locations in Process Storage. An input array is moved to consecutive storage locations.

3. BCD numeric input fields being moved to Process Storage are converted to fixed or floating point binary mode depending upon the mode assigned by the compiler.

4. Alphanumeric fields are moved and unpacked to integral word storage. When necessary, elements are unpacked from their parent fields to separate storage. After the READ sentence is executed, any fields or elements from the previous input record of the same type have been destroyed. The user should "save" input fields (if required for later use) by moving them to Working Storage before executing the READ sentence for the next record.

Object Program Action in Executing a WRITE Sentence

The following actions take place when the object program executes the coding corresponding to a WRITE, Option 2, sentence:

1. The output record (or *group) is assembled from the first described field to the last described field.

2. The WRITE sentence acts as a gather move from input records, Process Storage, and Working Storage depending upon output qualifiers and the storage assignments for the output fields.

3. When necessary, numeric fields are converted from fixed or floating point binary to BCD numeric with editing if specified in the output field data image.

4. Alphanumeric fields are moved to the output record or group with truncation or space fill to conform to the output field data images.

5. If no data image appears in the output record description for the field, the field is moved without alteration to the output record or group.

The WRITE sentence does not destroy the fields described under the output record or group. These fields or elements are still available at their sources (not in the output record or *group) for Procedure Division manipulation.

Where fields named in output are in an input record, the compiler determines the most efficient type of move to make (if the field has an associated Process Storage area).

1. If the output field is alphanumeric and has been assigned a Process Storage area, it is always moved from Process Storage.

2. If the output field is numeric and binary, it is always moved from Process Storage.

3. If the output field is numeric and BCD, it is moved from the input area unless the output field has ever been a receiving field, in which case it is moved from Process Storage.

Dating

If date symbols are not used the object program expects to find the six BCD date characters in locations 1076_8 and 1077_8.

BINARY SCALING

The compiler stores all data that is numerically described into two GE-200 Series words. Integers and true-false variables are stored with a fixed binary scale of 19, with the second GE-200 Series word zero filled.

Fixed point numbers, which are represented as decimal numbers on external media, are read into a GECOM program and converted, if necessary, to binary numbers. At compilation time, the data image of the decimal number is examined to determine if the position of the decimal point permits a binary point of 19. The number is stored in two GE-200 Series words, if possible, with a binary point of 19. This is standard GECOM scaling.

If the number of digits to the left or right of the decimal point (or assumed decimal point) in the data image is too large (exceeds 5) to allow a binary scale of 19, the scale as shown in Figure 11 is assigned.

	DATA DESCRIPTION OF DECIMAL NUMBER	GECOM BINARY POINT
Number of Integer Digits	Number of Fraction Digits	
0 to 5	0 to 5	19
1 to 5	0	19
0	1 to 5	19
0	11	1
0 to 1	10	5
0 to 2	9	8
0 to 3	8	11
0 to 4	7	15
0 to 5	6	18
6	1 to 5	21
7	1 to 4	25
8	1 to 3	28
9	1 to 2	31
10	1	35
6 to 11	0	38

Figure 11. Binary Scale Assignment

EXAMPLE:

If the data image is:	2 words of memory are:	and binary scale is:
999V99	word1: 999 word2: 99	19
99999V99	word1: 99999 word2: 99	19
99999999V9	word1: 99999 word2: 9999	28

Use of Scaling Factor

The GECOM scaling factor, S, is used in conjunction with fixed point data images which exist on external media as nonstandard binary numbers. The number is nonstandard if:

- It is only one word length, or
- It is two word lengths, but its scaling does not follow the above table.

The binary scaling factor for nonstandard input data must be known and described in the Data Division:

- A 1 or 2 in the format column to indicate the nonstandard data are one or two word lengths, and
- An S and the binary scale of the nonstandard data must follow the data image: 999V99S28

The data may be written from a GECOM program by indicating the binary scale in the same manner. For instance, 99999V99 would ordinarily have a binary scale of 19. If the data is to go to tape with a nonstandard scale, for instance 15, it must be indicated in the data image column as 99999V99S15.

The S can also be used in conjunction with M on input files to force a given internal scale, for example, M99K9S38 forces a scale of 38 instead of 19 in process storage. This technique may be used on card or tape files.

Use of 1 or 2 in the Format Column

When the 1 is used, the binary scale will not exceed 19.

The scaling factor and the 1 and 2 in the format column may now appear in output files in addition to input files and Working Storage.

Regardless of the binary scale of the data on the external media, the data are stored internally in two GE-200 Series words with the binary scale as indicated in the above table. The scaling factor, S, is not intended as a tool for manipulating the internal binary scale. Figure 12 gives examples of external and internal storage.

Table 9. External and Internal Storage

Format Column	Data Image	Storage	
		External	Internal
1	99V99S5	S1 5^ 19	S 19 S20 38
1	9V9S4	S1 4^ 19	S 19 S20 38
1	999V99S16	S1 16^ 19	S1 19 S20 38
2	999V99S28	S1 19 S20 28^ 38	S1 19 S20 38
2	999999V9S24	S1 19 S20 24^ 38	S1 19 S20 21^ 38

Figure 12. External and Internal Storage

Integer Arithmetic

Though the scaling factor may not be used to indicate internal scaling, it is possible to force a binary point of 38 when necessary for accuracy in decimal representation of a binary numeric.

The loss of accuracy in decimal places is due to conversion of decimal fractions to binary fractions in binary computers. In this conversion, representations of decimal fractions become endless binary numbers. Because this binary fraction is limited by the binary scale assigned to the field, it must be truncated. Therefore, for accuracy, the data must be described as integer and the fractional part must appear to the left of the decimal point. For example, instead of an image of 9(9)V99, an image of 9(11)V could be used. The 9(11)V image forces a binary point of 38 to be carried internally resulting in all arithmetic operations being carried out in integer arithmetic.

If the field is to be converted to BCD at output time, it must be divided by 100 and described in Working Storage with a decimal point. The field (qualified, if necessary) may then be named in the output files.

When using integer arithmetic, alignment of (understood) decimal points is the responsibility of the programmer. Therefore, when carrying all numeric fields as integers, carry the maximum necessary understood decimal places in all data. For example, a payroll program dealing primarily in dollars and cents may have constants requiring three decimal places (as 1.5% = .015). It is advisable under such circumstances to carry all data with three understood decimal places; for example, an amount field 25.02 should be set equal to the integer 25020.

USING K IN DATA DESCRIPTIONS

As a convenience for users doing computations using integer arithmetic, conventions have been extended to allow the insertion of actual decimal points in output fields that were described in input, Working Storage, Common Storage, *Common Storage, or Constant Sections as integers. This extension pertains only to numeric fields.

Input, Working Storage, Common Storage, *Common Storage, or Constant Section numeric fields may be described with a K in place of an actual or assumed decimal point. The K is not a character of the field, but merely designates decimal point placement with respect to output. Input, Working Storage, Common Storage, *Common Storage, or Constant Section fields described with a K are interpreted as integers for all internal manipulations.

Fields which are described with a K in input, Working Storage, Common Storage, *Common Storage, or Constant Sections, if named in output, may be described in output with an actual decimal point. The combination of K in input, Working Storage, Common Storage, *Common Storage, or Constant Sections and an actual decimal point in output causes the insertion of a decimal point in output data without scaling.

EXAMPLES:

	BINARY	DATA IMAGE	ACTUAL INPUT AND OUTPUT
INPUT-FIELD		999K99	12345
OUTPUT-FIELD		+ 999.99	+ 123.45
INPUT-FIELD		99999K9	123456
OUTPUT-FIELD		99999.9 +	12345.6 +
INPUT-FIELD	B	999K9S19	0000173 0000000 (Octal)
OUTPUT-FIELD		ZZ9.9	12.3
INPUT-FIELD		99K9	123
OUTPUT-FIELD		999	123

K Conventions

- Output fields must be designated as BCD for the decimal point insertion to have meaning.
- Input fields described with K must be integers, that is, without a decimal point.
- K may be used as a means of documentation in problems using integer computations. Constant Section, Working Storage, Common Storage, and *Common Storage field data images may contain K even though the fields are not named in output.
- All other editing features are available for output fields where the decimal point is being inserted from a K position.
- A source field, with K in its description, and a receiving field with V in its description results in all zeros being added to the right of the V and the entire sending field (which is an integer) to the left of the V. Conversely, the fractional part of a source V field is lost if the receiving field is a K field. It is recommended that K and V not be used at different places to describe the same data. If data are integers, the V is not needed and adds confusion.

EXAMPLE: All are input or Working Storage fields.

DATA NAME	IMAGE	ACTUAL VALUE
TOT~1	9(5)K9	123456
TOT~2	9(4)V99	
FACTOR	99V999	12345
FACT	999K999	

SENT~1. MOVE TOT~1 TO TOT~2.

The actual value result is: 345600.

SENT~2. MOVE FACTOR TO FACT.

The actual value result is: 000012.

When using integer arithmetic, with or without K in the data image, the user maintains the decimal point alignment, or scaling. In integer arithmetic the decimal point is assumed at the right of the number. The following examples should be studied before using K in data images.

EXAMPLES:

DATA NAME	IMAGE	ACTUAL VALUE
INPUT ITEM~A	999K999	123456
INPUT ITEM~B	9K999	1234
WORKING~STORAGE TOTAL	999K999	

```
SENT~1.   ADD ITEM~A    TO ITEM~B    GIVING TOTAL .
             123456          1234          124690
```

In this case alignment causes no problem, but if the following images were used, the result 124690 might not be desired.

DATA NAME	IMAGE	ACTUAL VALUE
ITEM~C	99K9999	123456
ITEM~D	999K9	1234
TOTAL~1	999K999	

```
SENT~2.   ADD ITEM~C AND ITEM~ D GIVING TOTAL~1.
```

The execution of SENT~2 still gives the integer 124690 as the total.

In arithmetic operations the scaling of ITEM~C, ITEM~D and TOTAL~1 should be aligned by multiplication.

Multiplication

If ITEM~A and ITEM~B above are multiplied, the receiving field should have six 9's to the right of K, for the field remains an integer regardless of the position of K.

If the following scaling (as denoted by K) were desired, the product should be divided by 100.

MULTIPLIER	99K99
MULTIPLICAND	999K99
PRODUCT	9(5)K99
PRODUCT~WS	9(5)K9999

```
SENT~3.   MULTIPLY MULTIPLICAND BY MULTIPLIER GIVING PRODUCT~WS.

          DIVIDE 100 INTO PRODUCT~WS GIVING PRODUCT.
```

Division

The principles of integer arithmetic apply to division, for example:

SENT~4. DIVIDE Y·INTO X GIVING Z.

where,

	IMAGE	ACTUAL VALUE
X	99K99	333
Y	9K99	222
Z	9K9	

The result will be equal to 1 since an integer is specified as the receiving field. The actual result is 1.5 which may be obtained by:

SENT~4. MULTIPLY 10 BY X.
DIVIDE Y INTO X GIVING Z.

Summary

The K is used in input, Working Storage, Common Storage, *Common Storage, or Constant Sections to aid the programmer who is using integer arithmetic. The use of K accomplishes two things:

1. It is a visual aid in the program listing for keeping track of assumed decimal points.
2. When a field with K is moved to output via the implied move and output is described with a "." (actual decimal point), the actual decimal point is inserted in the K position.
3. A field with K may be moved only to an edited field in output via an implied move.

REPEATED GROUPS

Data described in a repeated group (see Language Structure, Arrays) may have identical or different data descriptions. Nonrepeated groups may not be described after repeated groups for a given file in the Data Division.

For greater program efficiency it is recommended that:

1. Homogeneous data in output files be described as arrays.
2. Repeated groups only be used to describe nonhomogeneous data in input files.

3. Repeated groups not be used in output files. However, if a repeated group is used in output, the group name must appear in the output record description. For example:

Correct Method	Incorrect Method
OUTPUT FILES	OUTPUT FILES
FD OUTPUT1	FD OUTPUT1
R OUTREC	R RECORD1
F AA	INPUT FILES
F BB	FD INPUT1
G GROUP1	R RECORD1
INPUT FILES	F AA
FD INPUT1	F BB
R RECORD1	G GROUP1
F AA	F CC
F BB	F DD
G GROUP1	
F CC	
F DD	

For purposes of discussion and examples, consider the following sample Data Division:

GENERAL ELECTRIC

COMPUTER DEPARTMENT PHOENIX ARIZONA

GENERAL COMPILER DATA DIVISION FORM

PROGRAM | DATE

PROGRAMMER | COMPUTER | PAGE OF

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	SYNC	REPEAT	JUST	BLANK	ELEMENT POSITION #S	ELEMENT POSITION S.S	DATA IMAGE
		OUTPUT FILES.								
	FD	FILE~2.								
	R	RECORD~								
	F	FIELD~D								
	F	FIELD~E								
	F	FIELD~F								
		INPUT FILES.								
	FD	FILE~1.								
	R	RECORD~								
	G	GROUP~A			3					
	F	FIELD~A								X(4)
	F	FIELD~B								9(3)
	F	FIELD~C								X(7)
		WORKING~STORAGE	SECTION.							
	F	FIELD~D								X(4)
	F	FIELD~E								9(3)
	F	FIELD~F								X(7)
	F	FIELD~B~WS			3					9(3)
	G	GROUP~A~WS			3					
	F	FIELD~A~WS								X(4)
	F	FIELD~B~WS								9(3)
	F	FIELD~C~WS								X(7)

CA14 (1 62)

To facilitate data manipulation by the Procedure Division, the compiler allocates memory to the repeated group entries as shown below:

TYPE	DATA NAME	REPEAT	DATA IMAGE
(PROCESS STORAGE)			
F	FIELD~A'	3	X(4)
F	FIELD~B'	3	9(3)
F	FIELD~C'	3	X(7)
(WORKING~STORAGE)			
F	FIELD~A~WS'	3	X(4)
F	FIELD~B~WS'	3	9(3)
F	FIELD~C~WS'	3	X(7)

The user need not consider the memory allocation when referencing items listed in the repeated group. However, the repeated group may be thought of as a two-dimensional array where each single group is a row, and each field described as a column. Pictorially, the above input repeated group could be represented as:

GROUP~A(1):	FIELD~A	FIELD~B	FIELD~C
GROUP~A(2):	FIELD~A	FIELD~B	FIELD~C
GROUP~A(3):	FIELD~A	FIELD~B	FIELD~C

To clarify the repeated group technique, the following examples are offered:

(In each example, the second GECOM sentence, or set of GECOM sentences, is intended to illustrate how the compiler interprets the user sentence for implementation.)

1. To move the entire repeated group to another repeated group of the same size, use the group names with no subscripting:

   ```
   GROUP~TO GROUP:   MOVE GROUP~A TO GROUP~A~WS.
   ```

   is interpreted as:

   ```
   MOVE FIELD~A'  TO  FIELD~A~WS'
   MOVE FIELD~B'  TO  FIELD~B~WS'
   MOVE FIELD~C'  TO  FIELD~C~WS'
   ```

2. To move a column of the group to a one-dimensional array, give the field name and the qualifying group name with no subscripts:

   ```
   COLUMN~TO~ARRAY.  MOVE FIELD~B OF GROUP~A TO FIELD~BP~WS.
   ```

   is interpreted as:

   ```
   MOVE FIELD~B'  TO  FIELD~BP~WS.
   ```

3. To move a single field of the group to a field, use the field name and the qualifying group name with subscripting:

```
FIELD~TO~FIELD.  MOVE FIELD~A OF GROUP~A(2) TO FIELD~D.
```

is interpreted as:

```
MOVE FIELD~A'(2) TO FIELD~D.
```

4. To output the entire repeated group, the group name only is entered in the output description without listing the fields under the group name:

TYPE	DATA NAME
OUTPUT	
R	RECORD~A
G	GROUP~A

The above entry would output the entire repeated group as it appeared in the input file record.

If the field names were also listed, the group would appear in the output record twice; first, as it appeared on input and, next, as the three columns.

5. To output a column from the group, the field name and the qualifying group name is listed in output.

TYPE	DATA NAME	QUALIFIER
OUTPUT		
R	RECORD~A	
F	FIELD~A	GROUP~A

The above entry would output FIELD~A of GROUP~A(1), FIELD~A of GROUP~A(2) and FIELD~A of GROUP~A(3) when RECORD~A is written.

6. To output a row, the individual fields must be moved to Working Storage and those Working-Storage fields are listed in output.

The necessary procedure sentences to output a row would be:

```
MOVE FIELD~A OF GROUP~A(1)  TO  FIELD~D.
MOVE FIELD~B OF GROUP~A(1)  TO  FIELD~E.
MOVE FIELD~C OF GROUP~A(1)  TO  FIELD~F.
WRITE RECORD~2.
```

7. If the fields listed in the repeated group have different data descriptions, certain conventions must be followed.

Repeated groups may contain alphanumeric data (image A or X) and numeric data (image 9) either in BCD or binary. However, any alphanumeric field that precedes a numeric field recorded in binary must be a multiple of 3 characters or must be unpacked.

Only type F and FL fields may be used in a group which is repeated.

EXAMPLE:

All fields are packed, none of the numeric fields are carried as binary.

GENERAL ELECTRIC — GENERAL COMPILER DATA DIVISION FORM

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

PROGRAM
PROGRAMMER — COMPUTER — DATE — PAGE — OF

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	FORMAT	REPEAT	BINARY	JUSTIFY	ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
	G	G R O U P ~ 1		P	0 0 5					
	F	A A								X (
	F	B B								9 (V
	F	C C								A (
	F	D D								9 9
	F	E E								X

All fields are unpacked, all numeric fields are carried as binary.

GENERAL ELECTRIC — GENERAL COMPILER DATA DIVISION FORM

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

PROGRAM
PROGRAMMER — COMPUTER — DATE — PAGE — OF

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	FORMAT	REPEAT	BINARY	JUSTIFY	ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
	G	G R O U P ~ 2		U	0 0 5	B				
	F	A A								X (2)
	F	B B								9 (4) V 9
	F	C C								A (2)
	F	D D								9 9 V 9
	F	E E								X

All fields are packed, all numeric fields are carried as binary. Note the fill used on the alpha-numeric.

GENERAL ELECTRIC — GENERAL COMPILER DATA DIVISION FORM

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

PROGRAM
PROGRAMMER — COMPUTER — DATE — PAGE — OF

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	FORMAT	REPEAT	BINARY	JUSTIFY	ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
	G	G R O U P ~ 3		P	0 0 5	B				
	F	A A								X (2) B
	F	B B								9 (4) V 9
	F	C C								A (2) B
	F	D D								9 9 V 9
	F	E E								X B B

10. USING GECOM TO OBTAIN EFFICIENT OBJECT PROGRAMS

For any compiler to produce efficient programs, there are certain rules and techniques to follow which are independent of the compiler itself.

The user should first analyze the problem from an input/output point of view, looking for the most efficient way to lay out his input and output with respect to data movement. In doing this, the following rules should be considered to reduce execution time and the number of instructions generated.

1. An incoming decimal number must be converted to binary before it can be operated on. Therefore, all fields which will be used arithmetically must be described as 9-type fields in the Data Division. Also, these fields should be kept in binary from run to run to eliminate repeated conversions until they are required as a decimal field in output. Repeated conversions only add unnecessarily to each object-program's execution time. Fields not used arithmetically should be described as alphanumeric.

2. Fields which are not referenced or operated on in the Procedure Division but are moved from input to output are defined as throughput fields. Throughput fields should be combined into strings of characters. A string may not exceed 83 characters in length. Each string must then be described as an alphanumeric field. Arranging the referenced fields separately from the unreferenced fields provides stringing capabilities. Strings of characters (up to 83 each) result in fewer instructions being generated and faster execution time than the same number of characters in shorter fields.

3. Fields of high activity should be maintained as unpacked from run to run. Unpacking may be accomplished either by the compiler at object time or by the arrangement of output data. Input data is assumed to be packed unless stated as unpacked in the format column.

   Fill may be used to force the fields to form an integral number of GE-200 Series words. Character manipulation of fields causes more instructions to be generated and executed as opposed to word manipulation.

4. Advantage should be taken of an output record that is a direct reflection of an input record. The input record name should be used as the output record name with the corresponding input file name as its qualifier. This is sufficient description for the output record.

   When only the record name is used in output, the entire input record is moved to output rather than a field by field move to the output area. The record movement is made by words without regard to field positioning, resulting in fewer instructions and faster execution.

5. Files assigned to the card reader and/or the card punch should be buffered. Memory space is allocated to provide for buffering of these files whether or not used. Files assigned to the high-speed printer and magnetic tape units should also be buffered if

memory space permits it. Also, blocking of logical records should be used where memory space is available. Through blocking and/or buffering of files, faster execution time will result from fewer actual reads or writes per file. A balance must be struck by the user between execution time and memory space.

6. Fields used arithmetically should be kept at the same decimal or binary scale. Constants should be compatible scale even at the cost of additional constants. Fields which have the same scale require fewer instructions to perform the arithmetic functions which result in faster execution.

The primary contribution to efficiency in the Procedure Division is through the use of fields as they have been described in the Data Division and through judicious of the various verb options.

1. Use the ADD, SUBTRACT, MULTIPLY and DIVIDE verbs when performing arithmetic operations. A more extensive analysis is made of the data descriptions for these verbs as opposed to the assignment sentence. Use of the ASSIGNMENT verb may be an easier method of writing expressions but is to be used for complex arithmetic expressions involving operations that cannot be done with the ADD, SUBTRACT, MULTIPLY, and DIVIDE verbs.

2. The MOVE verb should be used in preference to the ASSIGNMENT verb to move one field to another. Fewer instructions result from using the MOVE verb.

   Example: MOVE A TO B. instead of B = A.

3. Under the MOVE verb, multiple receiving (TO) fields should be used rather than many MOVE sentences.

   Example:

```
MOVE A TO B, C, D.
     instead of
MOVE A TO B.
MOVE A TO C.
MOVE A TO D.
```

   The compiler automatically provides for the initializing of elements when they are affected by a MOVE sentence. The user should not initialize these elements individually.

   Example:

```
DATA DIVISION

F   A                       A(10)
E   B               0103
E   C               0207
    .
    .
    .
MOVE SPACES TO A. is sufficient.

It is not necessary to:

MOVE SPACES TO A, B, C.
```

4. Option 4 of the IF verb should be used to test the value (in relation to zero, positive, negative) of a single numeric field.

   Example:

```
IF A IS ZERO GO TO SENT~1
        instead of
IF A EQ 0 GO TO SENT~1.
```

   Using Option 4 will eliminate instructions and their execution.

5. The "GO TO sentence-name$_1$, sentence-name$_2$ ... DEPENDING ON field-name" sentence should be used for a series of sequential tests rather than multiple IF statements or a series of IF statements. The field does not have to start with the numeric value of one. The arithmetic verbs ADD and SUBTRACT may be used to adjust the field prior to using the GO TO ... DEPENDING ON option.

   Example:

```
A series of values begins with 51 and ends with 56.
SUBTRACT FIFTY FROM A GIVING B.
GO TO SENT~1, SENT~2, SENT~3, SENT~4, SENT~5, SENT~6,
        DEPENDING ON B.
```

   Fewer instructions and faster execution time results in using the GO TO ... DEPENDING ON option rather than a series of IF statements which must be executed sequentially.

6. Sections should be used to conserve memory for sets of common sentences that appear in two or more areas of the program. The use of section results in a saving of memory space.

7. Program loops that have the same duration and are executed in a logical chain should be controlled by one VARY statement. Fewer instructions are generated to control a single loop as opposed to many loops resulting in faster execution time. Also, memory storage is saved by one VARY statement.

8. Numeric fields referenced in a single ASSIGNMENT, ADD, SUBTRACT, MULTIPLY, DIVIDE, or IF sentence should have like integer and fraction portions in their data images. This also includes any numeric constants used in the statement. Data images for constants may be manipulated by leading or trailing zeros. Also, Working-Storage fields may be manipulated by increasing size descriptions.

   Example:

```
ADD     A to B GIVING C.
    999V99   +999.99   999.99-
ADD     A to 001.00 GIVING C.
    999V99             999.99-
IF      AS EQ BX GO TO SENT~1.
    999999V9     $999999.9+
IF      AX2 EQ 0000010.0 GO TO SENT~2.
    Z999999V9
```

   Note: Different editing and sign features are not significant--only 9-type characters.

9. Two and three dimensional subscripts, if used repeatedly in the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements, should be computed by the user and stored in an integer field. The computed subscript can then be used as a single subscript.

Example:

```
ARRAY SECTION.
   A(3, 5, 7)
   B(10, 10).
        .
        .
        .
SS1 =    ((I-1)*5+(J)+(K-1)*3*5).
SS2 =    ((I-1)*10+(J)).
ADD      A(SS1) TO B(SS2).
IF       A(SS1) EQ B(SS2) GO TO SENT~1.
     instead of
ADD      A(I, J, K) TO B(5, J).
IF       A(I, J, K) EQ B(5, J) GO TO SENT~1.
```

The general formula for computing two and three dimensional subscripts is as follows:

ARRAY SECTION.

$A(D_1, D_2, D_3)$. $B(D_4, D_5)$.

A(I, J, K)

A subscript = $((I-1)*D_3+(J)+(K-1)*D_1*D_2)$.

B(I, J)

B subscript = $(I-1)*D_5+(J)$.

Since the subscript is computed only once, fewer instructions are generated and a faster execution time results.

The following subroutines (which are described in the GECOM Operations Manual) have been modified with respect to object program efficiency:

1. The subroutine, ZUA, which performs word moves, also handles fields with one or two characters preceding and/or following the full words to be moved.

2. The subroutine, ZIP, moves a source field to a destination field of the same starting character position and the same number of characters whenever ZUA cannot be used because there is not a full word to move.

3. Because these two subroutines improve execution time, the use of the subroutine ZAM should be avoided whenever possible. ZAM is slower because it must handle cases where the destination field is larger than the source, and/or where starting character positions are not the same.

Use of the two subroutines, ZUA and ZIP, should influence file design. With these improvements, the object program is able to take advantage of data layout.

11. TABSOL

INTRODUCTION

This section describes TABSOL (TABular Systems Language), a structuring technique in which the step-by-step decision logic of a problem is described in table form. Tables provide a graphic representation of decision procedures showing alternatives and exceptions explicitly. Relationships between variables are shown clearly, and the sequence of conditions and actions is depicted unambiguously.

GECOM/TABSOL

With GECOM, tables can be written in an easily readable language which can be keypunched directly for computer input, GECOM conventions (see Language Structure, Chapter 3 of this manual) apply to TABSOL except as specifically noted in this section. The rules for formation and usage of words, data names, constants, subscripts, true-false variables, arithmetic expressions, condition names, and qualifiers are the same. A knowledge of GECOM is required to use TABSOL.

DECISION TABLE FORMAT

The format for a decision table is an array of blocks divided into four quadrants by a pair of double lines. The vertical double line separates the decisions or conditions on the left from the actions on the right. The horizontal double line isolates variables from associated operands which appear in the blocks and rows below. This format is illustrated in Figure 13.

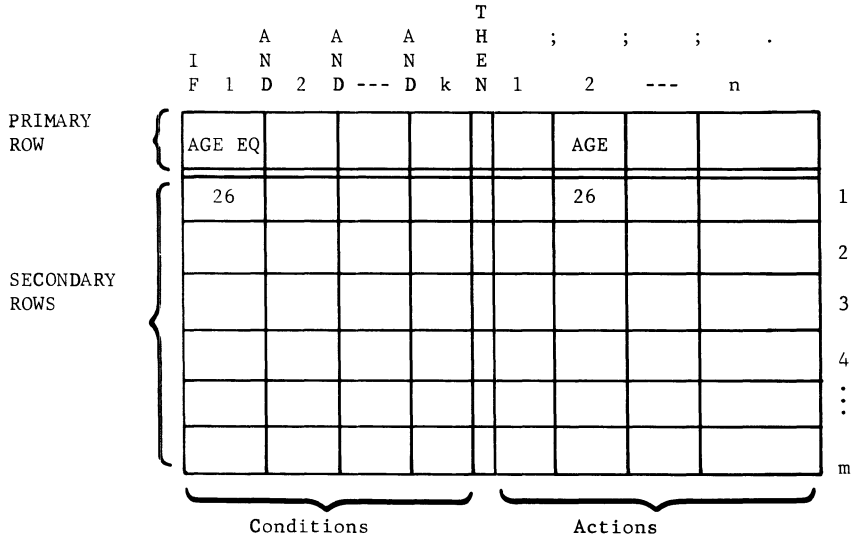


Figure 13. Decision Table Format

A condition is a relation between a variable appearing in a primary block and an operand appearing in a corresponding secondary block. For example, AGE may be written in primary block 1 and EQ 26 may be written in secondary block 1. In this way a condition is stated and the question "if age equals 26" is asked.

An action is a statement of what is to be done. By writing AGE in a primary action block and 26 in its associated secondary block, it is stated that "the value of '26' is to be assigned to AGE."

The vertical lines in the table may be interpreted as follows. The leftmost line may be thought of as representing the word IF. Those lines to the left of the vertical double line may be taken to mean AND; the vertical double line itself the word THEN. Since actions are sequential entities, the lines separating them may be interpreted as semicolons and the rightmost line, which actually terminates the actions, as a period. With this in mind, each secondary row becomes an English sentence. For example, each row now reads:

> IF condition-1 is satisfied AND condition-2 is satisfied AND ... AND condition-k is satisfied THEN perform action-1; action-2; .... action n.

If any condition within a row is not satisfied, the next row is evaluated and so on until all the rows are depleted. When this happens the table is said to have "no solution." The table is considered "solved" when all the conditions of a row are satisfied and their associated actions performed. Figure 16 on page illustrates a sample Decision Table.

TABLE ENTRIES

Formation of Conditions

By definition, a condition is a relation between a primary block entry and some corresponding secondary block entry. A condition, like a relational expression, may be either true or false. From this definition, a condition may be either a relational expression, a logical expression, or a true-false variable since these are the only elements that yield a truth-value.

The formats on following pages show how these expressions may be split between primary and secondary blocks to form conditions. In these examples, the word operand stands for either a variable (data name or subscripted data name) a constant (literal, numeric, figurative, or named constant), or an arithmetic expression. The word relation signifies one of the relational operators (see next paragraph). Since arithmetic expressions may be operands of relational expressions and relational expressions operands of logical expressions, it necessarily follows that arithmetic expressions may appear in logical expressions.

Relational Operators

The only permissible relational operators in a decision table are listed below. Only the symbol for the relation is allowable.

Symbol	Relation
EQ	Equal to
GR	Greater than
LS	Less than
NEQ	Not equal to
NGR	Not greater than
NLS	Not less than

Condition Formats

Format	Example
Operand-1 Relation Operand-2	LEVEL EQ 10
Operand-1 Relation Operand-2	EXPERIENCE GR 4
Operand-1 Relation Operand-2 OR Operand-3	TOTAL (I) NLS PT(1) OR PT(2) OR PT(3)
Operand-1 Relation-1 Operand$_2$ OR Relation-2 Operand-3 ...	(X+Y)**3 GR P+1 OR LS Q(I)
No Entry Condition-name	 PROGRAMMER

Format	Example
NOT Condition-name	NOT FEMALE
No Entry NOT Condition-name	 NOT FEMALE
No Entry True-False Variable	 REQ~1
NOT True-False Variable	NOT END OF FILE OF INVENTORY
No Entry Logical Expression	 PROGRAMMER OR ANALYST
NOT Logical Expression	NOT X GR Y OR X LS (Z+1)

Condition Column Rules

Figure 14 illustrates some of the rules for condition columns.

N = Prohibited				Y = Permissible	
Primary Row Entry / Secondary Row Entry	Operand	Operand-Relational Operator	Operand-Logical Operator-Operand	Blank	"Not"
Operand	N	Y	N	Y	Y
Operand-Relational Operator	N	N	N	N	N
Operand-Logical Operator-Operand	N	Y	N	Y	Y
Blank	N	N	N	N	N
"Not" Operand	N	N	N	Y	Y
Relational Operator-Operand	Y	N	N	N	N
Operand-Relational Operator-Operand	N	N	N	Y	Y

Figure 14. Rules for Condition Columns

The above table is not all inclusive. It shows only the basic combinations. However, from this basic table, the more complex combinations may be determined.

Certain combinations in the above table are permissible for only certain types of operands. For example, a blank column in the primary row and an operand in the secondary row column would be permissible only if the Operand-2 is a condition name or true-false variable.

Formation of Actions

Actions are statements of the things to be done when all the conditions of a row are satisfied. The scope of an action may be one of three kinds: implied assignment, procedural, or input/output.

Value Assignment is an implied function between associated primary and secondary block entries. Placing a data name in a primary block and some number in a secondary block, for example, I and 1, causes the compiler to produce coding to assign the number to the data name. In our example, 1 is assigned to the subscript I. Other examples of value assignment are given on the following page. In these formats the word variable implies either a data name or a subscripted data name and the word constant either a literal, numeric, figurative, or named constant.

Value Assignment Action Formats

Format	Example
Variable	I
Constant	1
Constant	"COPPER"
Variable	MATERIAL
Variable	ALPHA (I,J,K)
Arithmetic Expression	SIN THETA + (X/P) **2
Arithmetic Expression	PI*R**2
Variable	AREA~1
VARIABLE (Destination FIELD)	FIELD~2
VARIABLE (Source FIELD)	FIELD~1
True-False Variable	SWITCH~7
Truth-Value 1 or 0	1
Truth-Value 1 or 0	0
True-False Variable	BETA~REQ

Procedural Actions provide the means for interrupting the normal execution sequence of a table. Any of the following compiler verbs may be used for this purpose.

GO
PERFORM
STOP

The GO verb stipulates an unconditional transfer to a specified part of the table or program. Its destination may be a sentence name or a table name. Note that table names must be unique. Secondary rows are numbered consecutively starting at 1.

GO Verb Action Formats

Format	Example
GO TO Sentence~Name	GO TO TYPE~OUT
GO TO TABLE table~name TABLE	GO TO TABLE 23

The PERFORM verb specifies a transfer to some destination, the execution of a section outside of a table, the execution of a closed table, or the execution of a set of sentences in the heading of a table at the specified destination, and then a return to the action block following the PERFORM verb. The sentences or tables acted upon are by definition a "closed procedure" -- that is, they have a single entrance point and a defined exit point. Conventions for writing closed procedures are given on subsequent pages.

PERFORM Verb Action Formats

Format	Example
PERFORM Sentence~Name	PERFORM GROSS~PAY
PERFORM section~name SECTION	PERFORM FICA SECTION
PERFORM TABLE table~name TABLE	PERFORM ERROR TABLE

The STOP verb may be used as an action. It may be placed in either a primary or secondary block. When it is used, no other action may appear with it in the same action column.

STOP Verb Action Formats

Format	Example
STOP [Literal]	STOP "999"
[Literal] STOP	 STOP

Input-Output Actions are represented by verbs that control the flow of data to and from the computer. They cause reading and writing of data and validation of the tape labels of data files assigned to peripheral input/output devices. When data files are referred to from an action block, they must be defined according to the Environment and Data Division requirements listed in Chapters 7 and 5 of this manual.

Input-Output Action Formats

Format	Example
READ File~Name	READ MASTER~FILE

Format	Example
OPEN { INPUT / OUTPUT } File~Name	OPEN INPUT MASTER~FILE
CLOSE File~Name	CLOSE MASTER~FILE
File~Name CLOSE / OPEN / READ { INPUT / OUTPUT }	MASTER~FILE READ
WRITE { record~name / *group~name }	WRITE DETAIL~LINE
{ record~name / *group~name } WRITE	TRANSACTION WRITE

Action Column Rules

Figure 15 illustrates some of the rules for action columns.

N = Prohibited		Y = Permissible	
Primary Row Entry / Secondary Row Entry	Operand	Verb	Blank
Operand	Y	Y	N
Verb	Y	N	N
Operand-Verb	N	N	N
Logical or Relational Operator	N	N	N
Repeat or Skip Operator	Y	Y	N

Figure 15. Rules for Action Columns

Verbs in Action Columns

Only the GECOM verbs indicated in the list below may be used within the action columns of a TABSOL Table. These permissible verbs are restricted to the formats and options shown on the preceding pages and in the list below. However, GECOM sentences, with all of the extensive verb options, may be written within a table, outside the action columns. This is explained further on page 187. The verb (in GECOM sentences) is then executed from the confines of the action columns by means of the PERFORM verb. The NOTE verb may never be used in a table.

The list below shows those verbs which have been implemented for use in action columns. The user should consult Chapter 6 of this manual for additional information on the functions provided by and the rules for use of the following verb options.

1. CLOSE file name

   See page 98, FUNCTION and CONVENTIONS 1, 5, 7, and 8.

2. GO TO { sentence-name / TABLE table-name TABLE }

   This format is functionally equivalent to Option 1 of the GO verb, (See page 106.) Note, however, that it us not permissible to use the GO verb in an action column without a destination.

3. OPEN { INPUT / OUTPUT } file-name

   See page 117, FUNCTION, and CONVENTIONS 1 through 4 and page 118, CONVENTIONS 5 through 8. It should be noted that a tape file cannot be reopened if it has been closed from an action column because it will be considered locked.

4. PERFORM { sentence-name / section-name SECTION / TABLE table-name TABLE }

5. READ file-name

   See page 120, FUNCTION and Option 2, page 122, CONVENTIONS 1, 2, 3, and 6.

   Note that the IF END-OF-FILE clause is prohibited in an action block but that an end-of-file test may be executed in a condition block (see IF, Option 1, page 108). Although reads and end-of-file tests can be executed from table rows, it is recommended that these functions be included in sentences outside of the table rows and executed therefrom by means of the PERFORM verb. Use of these functions in table rows can cause incorrect end-of-file processing under many conditions which are dependent upon the order of the source statements, the sequence of execution, the logic of the program or even the nature of the data. For additional information on READ and END-OF-FILE clauses, see CONVENTIONS 4 and 5 on page 122.

6. STOP [literal]

   See page 128, FUNCTION, and CONVENTIONS 1, 2, and 4.

7. WRITE { record-name / *group-name }

   See page 135, FUNCTION, Option 2; page 136, CONVENTIONS 1 through 5.

Logical Expressions

The object coding produced for logical expressions is optimum in the sense that only the minimum number of variables are tested, assuming that the expressions are evaluated from left to right. This is not to say that redundant variables are eliminated by the compiler, but rather that an entire expression need not always be evaluated to determine whether it is true or false.

Example:

In the expression, A AND ((B or C) AND D), if A is false, the entire expression is false and only A will be tested.

A NOT in a primary block pertains to the entire entry in a corresponding secondary block.

Example:

NOT
A OR B

is the same as--

NOT (A OR B)

A NOT in TABSOL may be used only to negate Boolean variables. Any other use of NOT is treated as an error.

Example:

A NOT GR B is prohibited.
This must be written as A NGR B.

The effect of NOT in a logical expression is determined according to DeMorgan's Theorem.

Example:

NOT (A OR B) equals NOT A AND NOT B

The validity of this may be demonstrated by the following truth table:

A	B	A or B	NOT (A or B)	NOT A and NOT B
0	0	0	1	1
0	1	1	0	0
1	0	1	0	0
1	1	1	0	0

The relational operator in a negated relational expression is reversed.

Example:

NOT (A GR B) is changed to (A NGR B)

A NOT is prohibited within a relational expression.

Example:

(A GR NOT B) is prohibited; however, if A and B are both Boolean variables, this may be written as (A AND NOT B).

NOT (A GR B) is permitted because the relational expression always has only two values no matter what the values of A and B may be.

The NOT operator is prohibited in an action block.

The Skip and Repeat Operators

The skip operator makes it possible to show that a condition or action is not to take part in the evaluation of a row. This is done by placing a hyphen (~) in the condition or action block.

The repeat operator is a shorthand method of indicating that a condition or action in the block above is repeated. This is shown by entering a ditto or quotation mark (") in the block below the one that is to be repeated.

TABLES IN PROGRAMS

Thus far these specifications have been concerned primarily with table entries within condition and action columns. GECOM source language sentences may be used to support the conditions and actions of tables. A source program may be written without tables, with tables only, or with tables and sections and or sentences.

There are two types of tables:

1. Open tables which are executed in line. Open tables may be interspersed among source language sentences in the Procedure Division. Open tables may contain sentences following the table heading.

2. Closed tables which may be executed only by the PERFORM verb. Closed tables may be interspersed among sections at the beginning of the Procedure Division, but must precede the main body of the Procedure Division; that is, all closed tables and/or sections must appear immediately after the Procedure Division heading. Closed tables may contain sentences in three places:

   a. Between the table header and the word BEGIN.
   b. Between the word BEGIN and the first row of the table.
   c. Between the last row of the table and the words END TABLE.

   Note that when closed tables and/or sections are used, object program execution begins at the first sentence following the closed tables or sections.

OPEN TABLE FORMAT

{table-name TABLE / TABLE table-name} [OPEN] [integer-1 CONDITIONS] integer-2 ACTIONS

integer-3 ROWS.

[Sentences which may be performed only from the confines of the table]

[BEGIN.]

DECISION	TABLE

CLOSED TABLE FORMAT

{table-name TABLE / TABLE table-name} CLOSED integer-1 CONDITIONS integer-2 ACTIONS

integer-3 ROWS.

[Sentences which may be performed only from the confines of the table]

BEGIN.

[Sentences which are executed in line prior to executing the decision table]

DECISION	TABLE

[Sentences which are executed in line following execution of the decision table]

END TABLE table-name.

END table-name TABLE.

END TABLE.

Table Conventions

1. Tables should be written on the General Compiler Sentence Form. The Conventions shown in Chapter 4 apply to tables except as shown in the preceding formats and noted below.

2. A table without actions is prohibited. A table without conditions is permissible. Thus the number of conditions (integer-1) may be zero or omitted. Otherwise, conditions, actions, and rows are numbered sequentially starting at 1. The primary row is not counted in the row count. Row 1 is the first secondary row.

3. If the words CONDITIONS, ACTIONS, and ROWS are omitted from the table heading, the size may be stated with or without parentheses in any of the following formats:

```
 integer~1  integer~2  integer~3.
(integer~1  integer~2  integer~3).
(integer~1) (integer~2) (integer~3).
            integer~2  integer~3.
           (integer~2  integer~3).
           (integer~2) (integer~3).
```

Note that if only two numbers are given, they are interpreted as the number of actions and rows in that order.

4. It is most important that the number of conditions, actions, and rows, as specified by the programmer, be correct. If these numbers do not agree with the number of blocks as delimited by vertical lines, the compiler will mismatch the secondary and primary blocks.

5. A period should always follow the table size, the word BEGIN, and the table name at the end of a table. However, the period should not be used after the table name or the word TABLE in the heading of a table.

6. A maximum of 76 table columns is allowable in a table.

7. If the table contains less than 20 columns, a maximum of 1000 rows is permissible; if it contains more than 19 columns, a maximum of 250 rows is permissible.

TABLE SIZE

TOTAL NUMBER OF COLUMNS	MAXIMUM NUMBER OF ROWS
NGR 19	1000
GR 19	250

8. There is no limit on the number of tables in a program except as imposed by other considerations such as memory size, symbol table limits, etc.

9. If all the conditions in a row are satisfied and there is not a GO verb in the action columns of that row, the next lower row is evaluated.

10. Nested closed tables are permitted, that is, a closed table may be performed from another table.

11. A closed table has only one entrance and one exit, and, therefore, a GO statement to a destination outside of the table is prohibited.

12. Sentences written between the table header and the word BEGIN are treated as sections. A sentence name signals the start of a section. Another sentence name signifies the end of the section and the beginning of the next section.

    For example:

    Table~name TABLE CLOSED 5 CONDITIONS, 5 ACTIONS, 5 ROWS.

    Ex~1. WSR39198=R49595+R49595.
    Move WSA1 to WSA2.

    Ex~2. Exchange WSA2 WSA2X
    Exchange WSA9 WSA9X
    Exchange WSA12 WSA12X.

    BEGIN.

The above procedures produce two sections. The first section contains two sentences beginning with EX~1. The second section contains three sentences beginning with EX~2.

Block Conventions

1. All columns must be bound by the vertical table line (12-4-8 punch). The omission of a vertical line (12-4-8 punch) not only causes an error where the omission occurred, but may also cause all succeeding entries to be mismatched.

2. Conditions should be separated from actions by a double vertical line (two 12-4-8 punches), but a single vertical line will suffice.

3. The size of each block may vary from column to column and row to row.

4. A table column may not be split across cards. The maximum size of a block is therefore 71 characters since the first card column in which a table column may start is column 8 and each table column is bounded by vertical lines.

5. When the repeat and skip operators are used they should be the only characters in a block, other than spaces. The skip and repeat characters may appear in any position within the block.

External Control of Tables

Tables may be executed by source language sentences placed outside of the tables through use of the following verb formats:

```
GO TO {table-name-1 TABLE}  [, {table-name-2 TABLE}
      {TABLE table-name-1}     {TABLE table-name-2}

   [, {table-name-3 TABLE}  ... {table-name-n TABLE}]
      {TABLE table-name-3}      {TABLE table-name-n}

   DEPENDING ON {field-name                    }] .
                {RECORD of file-name           }

PERFORM {table-name TABLE} .
        {TABLE table-name}
```

Conventions:

1. Control may be transferred to an open table by means of the GO verb. Open tables may not be performed; they may be executed in line.

2. Closed tables may only be executed by means of the PERFORM verb, and are performed from another table.

3. For additional information on GO verb conventions, see page 106, CONVENTIONS 2 and 3.

Figure 16 illustrating a sample decision table follows.

GENERAL ELECTRIC

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER SENTENCE FORM

PROGRAM: Sample Decision Table | DATE

PROGRAMMER | COMPUTER | PAGE

SEQUENCE NUMBER							
5	PROCEDURE DIVISION.						
10		OPEN INPUT MASTER~FILE.					
15	GET~RECORD. READ MASTER~FILE RECORD IF	END FILE	GO TO END~RUN.				
20		IF FEMALE GO TO GET~RECORD.					
25		EXPERIENCE = CURRENT~YR - YR~EMPLOYED + PREV~EXP.					
30	TABLE EXAMPLE 3 CONDITIONS 2 ACTIONS 5 ROWS.						
35	LEVEL EQ	EXPERIENCE		TITLE EQ		I	GO TO
40		6	EQ 2	#PROGRAMMER#		1	TYPE~OUT
45		7	EQ 3	#PROGRAMMER# OR #	ANALYST#	2	"
50		8	GR 3	#ANALYST#		3	"
55		9	GR 4	#ANALYST# OR #SR~	ANALYST#	4	"
60		10	GR 4	#SR~ANALYST#		5	"
65		GO TO GET~RECORD.					
70	TYPE~OUT. WRITE DEPARTMENT NAME TITLE	LEVEL EXPERIENCE ON TYPEWRITER.					
75		TOTAL(1) = TOTAL(I) + 1.					
80		GO TO GET~RECORD.					
85	END~RUN. CLOSE MASTER~FILE.						
90		WRITE TOTAL(1) TOTAL(2) TOTAL(3) TOTAL(4) TOTAL(5) ON TYPEWRITER.					
95		STOP "END RUN".					

Figure 16. Sample Decision Table.

12. REPORT WRITER

THE REPORT WRITER IN GECOM

The Report Writer is an extension of GECOM which simplifies the programming of routines to produce reports. It provides readily understandable program documentation in business report oriented language. The source language consists of report descriptions in the Report Section of the Data Division, and Report Writer verbs in the Procedure Division. The formats of the reports are described completely in the Report Section and are not mentioned in the File Section of the source program. Assignment of a report may be made in the Environment Division to the on-line printer or to a magnetic tape file containing one or more reports for selective deferred printing.

Report Writer programs are executed in the Procedure Division by using the Report Writer verbs, GENERATE and TERMINATE. These verbs perform standard reporting procedures which have been tailored by the Report Writer to the individual specifications supplied by the programmer in the Report Section.

Throughout this writeup references are made to figures found at the end of the section. These examples illustrate the various portions of the Report Writer.

The creation of a report or reports may be the primary or secondary purpose of the program. Master-file updating, calculation, and build-up of data arrays are common functions performed along with or prior to reporting. Usually the source program prepares a Process Storage of detail report line source data by reading of successive logical input records. At each record, control is passed to the Report Writer generated procedure so that detail printing can be performed in addition to any other reporting functions necessary at that execution. In brief, the Report Writer performs the following functions:

1. Prints report headings once at the beginning of the report.
2. Prints report footings once at the end of the report.
3. Maintains page control by line count and/or skips to a new page at specified line printings.
4. Maintains line spacing on the page.
5. Prints page headings at the top of each report page.
6. Prints page footings at the bottom of each report page.
7. Numbers pages.
8. Issues detail or body lines of the report.

9. Accumulates detail field values conditionally or unconditionally to one or more levels of total.

10. Counts detail lines and/or detail conditions to one or more levels of total.

11. Detects control breaks at one or more levels so as to:

    - Control the tabulation procedure.
    - Issue logical control totals.
    - Issue logical control headings.

12. Edit data fields for reporting (that is, suppress leading zeros, insert decimal points, dollar signs).

All conventions outside of the Report Section in the Data Division and the Report Writer verbs in the Procedure Division apply to the source program as specified throughout the GECOM II Reference Manual.

METHOD OF REPORT DESCRIPTION

The General Compiler Report Description Form (, see Figure 17) is provided for all entry types in the Report Section. There are two parts of any report description, the layout and the definition.

The first entry for a single report is the Report Layout header which gives the report identification and the keyword, LAYOUT, to distinguish this portion of report description. Under LAYOUT, all unique line structures are laid out by giving a pictorial image of the report as it will appear on the final printed page. Ahead of each line structure of up to 120 print positions, the line identification and any preprint or postprint slewing requirements are entered. The entire entry is called a Line Image entry. No other types of entries are made in the LAYOUT portion of the report description. Figure 17 shows a sample layout.

The DEFINITION portion, or second half of the report description, starts out with a Report Definition header, corresponding in form to the Report Layout header. Under DEFINITION, four main sections are allowed: Line Definitions, Line Control, Page Control, and Line Sections. These sections may appear in any order convenient to the programmer. Usually the programmer will want the Line Definitions Section to be first or second to aid visual association of Line Image entries and Line Definition entries. See Figure 20, line 01090. The other three sections, Line Control, Line Sections and Page Control, are optional. Line Control is necessary to specify Report Writer control of logical control totals and headings. Line Sections specify sections of the Procedure Division to be performed at line preparation time. Page Control specifies any page overflow testing and fixed line numbers which function in page slewing. All sections must be headed by their appropriate header entry giving the section name. See Figures 17, 20, 23, and 26 at the end of this Chapter.

The key feature of the GECOM Report Writer method of description is that the layout of line images is separate from line definition. A Line Definition entry is associated with a Line Image entry by line code. Data names are associated with data images by listing the data names, along with the literal identifier L for any constants on the line, in the order of data image appearance on the line image above. This technique allows the programmer to describe the report without recording detailed columnar print position numbers under the report definition portion.

The Report Writer allows the programmer to give one name to each unique detail field accumulation, using the form, ACC OF detail-field-name. A variation on this name is ACC OF detail-field-name FOR condition-name, used to specify conditional accumulation. With this specification, the detail field is accumulated only on true status of the condition. The same accumulation name must be used on the Line Definition entries at all levels of total lines whenever that accumulation is to be printed. At total printing time, the correct level of accumulation is obtained by the Report Writer, and may be referenced by the user.

One or more successive report descriptions in the Report Section must be headed by a Reports File Definition (RFD) entry when reports are being sent to a magnetic tape file for deferred printing.

The Report Section must be the first section in the Data Division and is identified by the words REPORT∆SECTION. starting in Column 8. No more than seven reports may be described per source program.

LINE DESCRIPTION

The following types of lines may be described to the Report Writer. (Detailed conventions are covered under Line Image Entry.)

RH	Report headings
RF	Report footings
PH	Page headings
PF	Page footings
D	Detail
T	Total
H	Heading
S { D T H }	A series of lines

A report line description requires a Line Image entry showing the literal values and/or the structural characteristics of the data fields. A line which is all literal, such as a page heading, is described completely by a Line Image entry. To associate data and definition names with their images on the Line Image entry, a Line Definition entry is made under the LINE DEFINITIONS entry. This entry identifies all components of the line in order. The programmer must list the Line Definition entries in the same order as the Line Image entries. Approximately 30 line types may be defined per report by the programmer. No more than 99 line types may be defined per report file on magnetic tape.

If report lines of the same type have the same structure with only minor differences in content, it may be advisable to define a single line rather than to define separate line images and line definitions for each case. A common situation is several levels of totals containing the same accumulated fields, with level 1 showing the literal "UNIT", level 2 the literal "SUBSECTION", etc. The field images can be made large enough to accommodate the highest level of total, so that one line image suffices for all levels. The literal values "UNIT", etc., can be assigned to a field titled ORG-NAME, for example, in a section of the Procedure Division (see Line Section entry) executed at line preparation time. On detail lines, a similar example is a file processing problem where a separate line image can be written for each type of input record reject. The lines are identical in content except for the literal which spells out reject reason. This again may be handled by one line image and associated definition to achieve a saving in object program space.

A series of lines functions as a single "folded" line, and can be defined for D, T, or H line types by means of a series header ahead of the individual Line Image entries.

LINE SPACING ON THE PAGE

The normal and most efficient method of achieving line spacing on the page is to associate paper slew requirements with individual line types. The pre and post positions in the Line Image entry are used to indicate a slew of a fixed relative number of lines before and/or after a particular line issue. The integers "1" through "9" and the letter "E", meaning eject to a new page, are allowed. Wherever possible, a postslew should be used in preference to a preslew to gain object efficiency. This allows the print and slew to be performed with one printer command.

In special circumstances a slew to a fixed line number from the top of the page can be specified. An alphabetic character is adopted as a preslew symbol and a fixed line number is assigned to this symbol by means of the Line Number entry under PAGE CONTROL heading.

PAGE OVERFLOW TESTING

Page overflow testing is an optional reporting feature. Standard tabulated reports usually require a statement of page overflow conditions in addition to any line slewing parameters because of the unknown quantity of details making up a logical control group and the high probability of exceeding a page with any one group. The page overflow specification by the user allows the Report Writer to control the report format in a desirable manner. At control total printing, the total lines are not printed on the current page unless space is available for all lines at the level of break detected. Similarly, control headings are not started unless there is space for at least one detail after the headings at that level. Detail lines overflow whenever the current issue cannot be fitted on the page.

The page overflow entry under PAGE CONTROL heading may take two forms: LINES/PAGE NN or LAST-DETAIL NN, where NN is a constant value representing a line number relative to the top of the page. Only one entry is allowed. The Report Writer uses this line number in conjunction with other report specifications to calculate the overflow line numbers to operate at object time.

TABULATION LOGIC

Tabulation is the process of accumulating certain detail field values until a control break is reached. At control break time the appropriate level of accumulated values may be issued on a total line or series which is defined to operate at that level under LINE CONTROL.

A CONTROL BREAK hierarchy statement is made by means of a CONTROL BREAKS entry immediately following the LINE CONTROL header. This statement lists the logical control fields in order of lowest to highest level of break. A break (change in value) for any of the fields listed automatically defines that level of control break and any lower levels. Control break fields usually represent file sequence key fields (for example, organization code, date)

The fields to be accumulated and the counts to be made at the detail level are defined by special Accumulation and Count Names in Line Definition entries. Fields for accumulation may be input or calculated values.

Whenever a GENERATE sentence is executed by the programmer, the detail values are accumulated to the lowest level of total accumulation. At GENERATE detail-name entry, the accumulation takes place immediately after line printing of the named detail. At execution of the GENERATE report-name entry, which means no detail lines are defined, the accumulation occurs at this same juncture in the object program with the standard detail line preparation and printing omitted.

At detection of the first level of control break, any total lines at that level are printed. Then first level accumulated values are "rolled forward" (added to the second level and reset to zero for the next build-up). This process is extended to all levels of control break defined by the programmer. The printing requirements of the accumulated values on the various levels of total lines has no influence on the tabulation procedure. In this way users may reference accumulated values at total time even if the value is not to be printed at that level of total.

REPORT WRITER LINE CONTROL

Line control is implied by line code for the fixed headings and footings: RH, RF, PH, PF. Line control must be specified by a Line Control entry for all control heading (H), and control total (T) lines. A series of control headings or totals may be controlled only at the series level. Only the body lines of the report (D or SD) are controlled by GENERATE detail-name sentences. However, at every GENERATE detail-name sentence, the programmer must expect that any or all Report Writer controlled lines may appear ahead of the detail, if appropriate conditions are satisfied. Page overflow forces page heading and/or page footing lines, and control breaks forces control total and or control heading lines.

A GENERATE report-name sentence must be employed when no detail line types are defined on the report. All of the automatic reporting functions available in the GENERATE detail-name mode of operation are also available in the GENERATE report-name mode. Printing occurs only at total, heading, or footing time.

The last control break level defined must consist of the key word FINAL when final levels of accumulations are required for the report. See Figure 23, line 01190. A final control break is determined by end-of-report, or execution of a TERMINATE report-name sentence. Standard business reports usually contain control break totals of one or more levels, in addition to a single detail type. Figures 17 through 28 illustrate the use of control break totals.

EXECUTION OF USER PROCEDURES AT LINE TIME

Standard reports often require some additional calculation or field value assignment within the reporting procedure. The Line Section entry provides a method of executing a section of the Procedure Division in the line preparation procedure. At detail time, the user may prefer to carry out these procedures prior to report program execution, before the GENERATE detail-name sentence execution. At all other line types, which are controlled by the Report Writer, he must use the Line Section entry.

One Line Section entry is allowed for any type of line, including the fixed types of headings and footings. The section is always executed immediately before line printing (including preslew). Tabulation procedures occur after line printing on detail lines as well as control totals; this allows detail accumulation values to be calculated in a Line Section entry at detail time. At control total time, Accumulation and Count Names can be referenced for calculation (that is, crossfooting) by the same name given to the field on the control total Line Definition entry. Control break level may also be interrogated by means of a control break condition name.

A logical control group is a group of detail lines plus any control headings and control totals. User reference to control break data names during the span of these lines yields the value for that control group, as a result of appropriate handling of control break data names by the Report Writer.

DATA DIVISION--REPORT SECTION

The **REPORT SECTION** of Data Division consists of several entries which are discussed on the following pages.

REPORT FILE
DEFINITION ENTRY

FUNCTION

The Report File Definition entry (RFD) specifies a magnetic tape file of stacked reports for deferred printing.

FORMAT

RFD in columns 8-11.

File-name in columns 13-24.

[BLOCK [{ SIZE IS / CONTAINS }] integer-1 WORDS] . in columns 28, etc.

CONVENTIONS

1. The RFD entry must immediately precede report descriptions for the file named. One to seven reports may be defined per file, provided the total number of reports in the Report Section does not exceed seven.

2. The file-name must be used for all references to that file in the Environment Division. (See Chapter 7, File-Control.)

3. Reports going directly to the printer must not have an RFD entry.

4. Each report belonging to the file must have a report format code supplied under the format column in the Report Layout header. This becomes the report select number at printing time.

5. A reports file description is terminated only by another RFD entry or the end of the Report Section.

6. The tape files are formatted according to the conventions for the GE-200 Series Off-Line Printer (see GE-200 Series High-Speed Off-Line/On-Line Printer Reference Manual (CPB-1075)) and the generalized GE-200 Series peripheral-to-peripheral program, Peripheral Package (PIP), CD225E1.009 designed to accept the same format for tape-to-printer with or without Automatic Priority Interrupt hardware.

7. If the BLOCK clause is omitted, a standard of 341 words (maximum acceptable for off-line printing) is assumed.

8. The RFD entry is also required when a single report is assigned to a tape file. In this case the file-name in the RFD entry and the report name in the Report Description (RD) entry must be different.

FUNCTION

The Report Layout header (RD) indicates the start of a report layout, assigns a name to a report, and assigns a report format code for use in deferred printing of a reports file magnetic tape.

FORMAT

Report code, RD [integer] in columns 8-11.

Report-name in columns 13-24.

[Report-format-code] in column 27.

LAYOUT. starting in column 28.

CONVENTIONS

1. RD initiates a report description, analogous to the FD for a file description in the File Section. The integer position may contain a number 1-9 for user documentation only.

2. A report name of up to twelve characters must be entered under data-name on the Report Description form. The programmer must use this report name when report name is called for in Procedure Division and Environment Division references to the report.

3. Report format code is valid only when the report is assigned to magnetic tape for deferred printing (see Report File Definition Entry, page). The integers 1-7 are allowed in this position for report identification and selection at printing time.

LINE IMAGE ENTRY

FUNCTION

The Line Image entry displays the structure of a line and certain key information about a line as a whole.

FORMAT

Line identification:

Line or line series code in columns 8-11.
Line or series name in columns 13-24.

Page slewing:

Preprint slew in column 25.

[literal]

Postprint slew in column 26.

[literal]

Line image in columns 28, etc. (See Report Description Form Conventions, page .)
1 to 120 print positions of any combination of:

Data image entries
Literal image entries
Space fill (no entry made)

CONVENTIONS

1. Line Identification

Line code is required for every Line Image entry and supplies the necessary line identification to be used in referencing the line under the Report Definition portion. This code must be unique within a single report.

Line identifications consist of functional symbols suffixed by integers (or alphabetics) to achieve uniqueness. The Report Writer allows one to four positions for this designation under line code. The symbols which are valid in the leading positions of the line code to designate line function or type are as follows:

RH, RF — Report headings and footings, respectively. These are printed once, at the beginning or the ending of the entire report, in the order of their appearance in the source program. A common use of report headings is a cover page for the report to inform operating personnel about report identification and certain instructions for preparing and distributing the report.

PH, PF — Page headings and page footings, respectively, follow the same principles as RH and RF, with printing at the beginning and ending of each page in the report. Multiple page headings are common to most reports. (See Figures 17 through 28.)

D — Detail line--a body line of the report. The Report Writer procedure which prepares and writes a detail line is controlled by means of the GENERATE detail-name sentence (see Report Writer verbs). Figures 17, 20 and 26 show typical detail line types.

T — A logical control total line. This line normally contains accumulated values of tabulated detail fields and is issued automatically by the Report Writer as control breaks occur. Total lines may be defined with or without printing of any detail lines of the values being accumulated. A good total line practice is to allow the second character of total line type to indicate the total control-break level number. Thus, T1 is a level one total, T2 a level two total, etc. A Line Control entry under Report Definition defines the total level of the line; thus, the use of a level number in line code is a documentation aid only. Whenever a control break occurs, total lines are printed in order of low-to-high control-break level. The number of lines printed reflects the current level of control break.

H — A logical control heading line. This line functions similarly to control totals but heads rather than follows the detail lines associated with it. Again, a Line Control entry specifies the level of the line. In the case of control headings, the lines are printed in high-to-low order of control break, or the reverse of control totals.

S — Series of lines. Any series of D, T, or H lines may be described by a series header. This header identifies a number of consecutive lines which function essentially as an entity; that is, all lines in the series are to be printed consecutively whenever the first one is printed. The series line code begins with an S and is followed by one to three characters which are duplicated in the leading positions of every line code within the series. Thus, a series can contain only one line type. The Line Image entries for a particular series must follow the header for that series. A series of associated lines at a total control break level is shown in Figure 23. A series header is meaningless for any of the fixed lines, RH, RF, PH, PF, and is recognized as an error.

Line name may be written under data name on the form and serves as user documentation on all line types except detail. If the detail-name (line or series) is written, it must be used in the GENERATE detail-name sentences in the Procedure Division.

2. Pre- and Postslew

Pre- and postprint line spacing, to occur in conjunction with the printing of a given line, is specified in columns 25 and 26 respectively. The literal value can only be one character in length. An integer (1-9) is entered to show a fixed number of lines of spacing to appear on the report before and/or after the printed line every time it is printed. The normal case can be described with a pre or post entry alone, but in certain situations it is advantageous to be able to specify both. For object program efficiency, a post entry alone is recommended.

The letter E may be entered in the pre and post columns to specify ejection to a new page to whatever has been set as top-of-page by a channel punch in channel 8 of the printer paper tape loop. This slew resets the line counter to zero.

E need not be entered in post of the last page footing or pre of the first page heading. this slew is automatic at page overflow or page skip (E) time. If entered in either of these positions, the designation will be ignored by the Report Writer.

E is invalid within a series except as an initial preslew or a final postslew on the individual Line Image entries.

Any alphabetic character other than E defines a pre- or postslew to a fixed line number relative to the top of the page. These symbols should be used only where the desired line spacing cannot be achieved by slewing a fixed number of lines relative to the current line being printed. At object time, a slew to a fixed line-number less than the current line-count results in a page skip, with printing of PF, PH lines, followed by a slew to that line number. The fixed line number associated with each alphabetic symbol is specified in the Line Number entry (see Page Control Entries, page).

3. Line Images

The line image or physical structure of the line as it will appear on the printed page is entered beginning in column 28. Each unique line is described with respect to its data image, literal image and space fill content.

Data Images

A data image defines the format of a single report field. In addition to the standard data format symbols A, 9, X, field data can be displayed with a wide range of report editing features. The most common editing features pertain to reporting of numeric quantities, where decimal points, commas, zero character suppressions, floating dollar signs, and check protection symbols are desired. Figure 29 presents a concise summary of the allowable data format symbols. The Report Writer adheres to the conventions for output format symbols in GECOM with extensions to include the options of floating leading sign, complete zero suppression, and CR, DB credit and debit symbols.

The parenthesis allowed in the data image of a File Section entry as shorthand to show a repeated symbol is not permissible on the Report Form. Each data character position on the report must contain its format symbol in the appropriate column on the line image.

To accommodate the Report Writer group-indicate function, the symbol G has also been added to the valid data image symbols. It is only allowed in the Report Section. It is placed in the leading character of any data image contained on a detail line. It means that the value for that field is to appear only on the first detail line printed for consecutive lines that contain duplicate values for that field. Group indication occurs on the first line of consecutive detail lines starting on a new page or after a total control break. Usually file sequence key fields (which are also control break fields) are in this category. It is assumed that the data symbol replaced by the G is the same as the symbol immediately after the G. If the field is length 1, an alphanumeric (X) is assumed as the data format symbol. See Figure 20.

Literal Images

Literals are shown exactly as is, with no surrounding quotation marks. A literal image cannot contain a blank. Blanks (space fill) separate literal and field images on the Line Image entry and by definition are not counted as part of the literal or field image.

A series header cannot have a pre, post, or a line image portion in the Line Image entry.

On a single report, the Report Writer allows no more than thirty unique line structures, or this number of Line Image entries.

REPORT DEFINITION HEADER

FUNCTION

The Report Definition header (RD) indicates the start of the Report Definition entries.

FORMAT

RD [integer] in columns 8-11.

[Report-name] in columns 13-24.

DEFINITION. in columns 28, etc.

CONVENTIONS

1. The portion of a report description headed by this entry must follow the report layout portion identified by the same RD (integer) code in the Report Layout header.

2. It is not necessary to repeat the report name in columns 13-24 of this entry.

FUNCTION

The Report Definition entries are:

Line Definition entry
Line Control entries
Line Section entry
Page Control entries

FORMAT (All options selected)

LINE DEFINITIONS. starting in column 8.
(Line Definition entries)

LINE CONTROL. starting in column 8.
(Line Control entries)

LINE SECTIONS. starting in column 8.
(Line Section entries)

PAGE CONTROL. starting in column 8.
(Page Control entries)

CONVENTIONS

1. The report definition must contain at a minimum a LINE DEFINITIONS section unless the report is composed entirely of all-literal lines. Within this section one Line Definition entry is required for each Line Image entry that is not entirely literal. See Line Definition entry, page .

2. The LINE CONTROL, LINE SECTIONS, and PAGE CONTROL sections are all optional. Their functions are described on the following pages by entry name.

3. Each section must be headed by a header entry (as shown under FORMAT) naming the section. See Figures 17, 20, 23, and 26.

4. Only one section of each type may be contained under a single report definition.

5. The sections may appear in any order under Report Definition.

LINE DEFINITION ENTRY

FUNCTION

The Line Definition entry associates data names with the images appearing on the corresponding Line Image entry and specifies Accumulation and Count fields.

FORMAT

Line-code in columns 8-11.

Line definition in columns 28, etc:

Image-1-name [, image-2-name, image-3-name, ---]

CONVENTIONS

1. The Line Definitions must be preceded by the LINE DEFINITIONS. header.

2. Line code must match a line code for a line defined in a Line Image entry of the report layout portion.

3. A Line Definition entry may use up to two full lines on the Report Description form. If more than one line is used, the line code must not be repeated on the second line. The second line is started in column 28 or after. A word may not be split over two lines on the form. (See Report Description Form Conventions, page .)

4. A Line Definition entry consists of a list of names in order of corresponding image (data or literal) appearance in the Line Image entry.

5. The Line Definition entries must appear under the LINE DEFINITION header in order of corresponding Line Image entries under Layout header.

6. Lines composed entirely of literals and space fill, as is common in headings, are not described under definition, and are skipped by the programmer in his order of Line Definition entries.

7. The image names within a definition list can be separated by one or more spaces, a comma, or a comma and one or more spaces. Each list must be terminated by a period.

8. As defined under Line Image entry, an image is a data image or a literal image. These two types of images along with blanks, or space fill, make up a line image. The two classes of image names are as follows:

   Literal identifier
   Data image-name

9. A literal identifier by definition is the letter L. A series of N consecutive separated literal images can be identified by L(N) or LΔ(N) or by separate literal identifiers, L, L, L.

The L(N) or LΔ(N) causes the series of literals and the encompassed spaces to be created as one literal string by the Report Writer. This notation is recommended for object efficiency whenever the consecutive literals are not widely separated.

10. Adjacent images on a report line occur occasionally, when it is necessary to lay out data and literals so that one image appears for the combination. Adjacent data images are shown by slash separators between the data image names in place of the normal spaces and/or comma in Line Definition. (See Figure 20.) In this case, the data image under the Layout header is for user documentation only and the Report Writer obtains the individual data descriptions from the input file, Working-Storage, or Constant Sections. Literal portions of adjacent images must be specified as field literals defined outside of the Report Section.

11. A data image name can be the name of a field, element, or group defined elsewhere in the Data Division or an Accumulation or Count field-name. The latter fields are set up and maintained automatically by the Report Writer as described in the section titled Accumulation and Count Names.

12. The Line Definition entry for a control total or heading line may contain a control-break-data-name for any level. At total time, the data-name value for the previous details is moved by the Report Writer to the field named as control break for that level. The move is made prior to execution of a line section. After total printing, the value is set to the next detail value; or the one which forced the total. This value may be appropriately printed on any control headings to follow, by using the same data-name on the Line Definition entry as used at total line definition.

13. In keeping with the GECOM conventions for output data-names, subscripts are not allowed on data-names in the Line Definition entry.

FUNCTION

The Line Control entries specify the control break hierarchy and associated control line printing.

FORMAT

Control Breaks Entry:

CONTROL~BREAK[S] ON control-break-data-name$_1$, [control-break-data-name-2, ----, control-break-data-name-n [FINAL]] .

starting in column 13 or after.

Line Control Entry:

Line-code in columns 8-11.

{Control-break-data-name / ALL} starting in column 13 or after.

CONVENTIONS

1. Line Control entries must follow the LINE CONTROL. header under Report Definition.

2. Only one Control Breaks entry is allowed. It must immediately follow the LINE CONTROL. header.

3. The Control Breaks entry defines any control breaks to be detected by the object program. One (to fifteen, at the most) control-break-data-names (element or field) may be listed. These are usually file sequence key data names, such as organization, pay number, name, etc. Qualifiers are written when needed for uniqueness. (See Figures 17, 20, and 23.)

4. When more than one control break level exists, the data names must be listed in order of minor to major level. The order of listing establishes the hierarchy of control breaks. By definition, a control break at any level (other than lowest) assumes all lower level breaks automatically without testing.

5. The Control Breaks entry supplies the Report Writer with the number of levels of accumulations (see Accumulation and Count Names) to be set up and maintained automatically. When final or report totals are required in addition to those for control-break-data-names, the key word FINAL must be written as the highest level of control-break-data-name. FINAL may be the only level given.

6. A Line Control entry associates a control-break data-name with a line (or series), thus defining the level of control break that determines issue of the line or series. The line or series code must match one supplied in a Line Image entry. The control-break data-name must match one of the names given in the Control Breaks entry. (See Convention 13.)

7. A Line Control entry is valid only for an H or T line or line series code. Line Control is implied by definition for the RH, RF, PH, PF line codes and cannot be stated. D lines are controlled by the programmer.

8. Lines within a series cannot have a Line Control entry. The line control must be given at the series level.

9. Only one heading line or series and one total line or series can function at each control break level.

10. A heading type (H or SH) cannot be associated with the FINAL control break name. A total (T or ST) may have the FINAL name, and is printed after all lower levels of totals which are forced automatically at end of report (FINAL condition true).

11. The same or different control break levels can be associated with T and H lines. At a control break of level 3, where all three levels are associated with T and H lines, the lines appear on the report as follows:

```
        (detail lines)
T       (level 1)
T       (level 2)
T       (level 3)
H       (level 3)   (start of new logical group)
H       (level 2)
H       (level 1)
        (detail lines)
```

12. In many cases one T line (or H) can be defined to operate at all control break levels, at a saving in object program space. This is true when the same or nearly the same line format applies at all levels. (See Figures 20 and 23.) In this case, the ALL option may be used to avoid writing multiple Line Control entries.

13. The issue of lines at control breaks is independent of the overall control break procedure of detection, roll forward, and resetting of specified accumulation and count field values. A control break can be defined and have no line issue associated with it; or, it may have an H line and no T line, etc.

LINE SECTION ENTRY

FUNCTION

The Line Section entry identifies a section of the Procedure Division to be performed at line printing time.

FORMAT

1. Line code in columns 8-11.
2. Section-name starting in column 28 or after.

CONVENTIONS

1. Section-name identifies a section of the source program in the Procedure Division. Line code establishes the line time at which the section will be performed under control of the Report Writer.
2. All entries must be placed under the LINE SECTIONS header. No ordering requirements are imposed.
3. A Line Section entry cannot be given for a series code.
4. No more than one entry of this type may be made for each line defined by a Line Image entry.
5. The section is performed immediately prior to line formation and printing. Accumulation of detail values is performed after detail line printing; hence, values to be accumulated can be calculated in the section.
6. The same section name may be given for two or more lines.
7. See Accumulation and Count Names on page to obtain the conventions for use of these special names in a section of the Procedure Division.

FUNCTION

The Accumulation and Count Names specify page numbering of a report and detail accumulation and counting requirements of the report.

FORMAT

Page numbering:

PAGE~COUNT

Detail Accumulation and Counting:

```
{ DETAIL~COUNT
  COUNT FOR {condition-name | true-false data-name}
  {ACC | ACCUMULATION} OF field-name [FOR {condition-name | true-false data-name}] }
```

CONVENTIONS

1. The Accumulation and Count Names of the formats above operate as data image names on the Line Definition entry. In addition to naming the corresponding data image on the Line Image entry, each type of name implies an accumulation or count action to be performed by the Report Writer.

2. The PAGE~COUNT clause requests that the pages of the report be consecutively numbered and that the current page number appear in the associated image position on each page of the report.

3. The DETAIL~COUNT clause specifies automatic counting of detail lines. The DETAIL~COUNT field is incremented by 1 every time a detail line or series is printed.

4. To obtain an automatic accumulation of detail field values into one or more total levels, one of the above forms of accumulation definition names in a Line Definition entry is used. Figures 17, 20, and 23 show examples of the use of this name type.

5. The same detail accumulation or counting field name (formats above) may appear on any or all total line definitions, where separate lines are defined for separate levels of control break. One appearance of a given name suffices to specify the desired action at detail time.

6. Knowledge of the control break hierarchy of levels gained from the Control Breaks entry under the LINE CONTROL header determines the exact manner of "roll forward" of counts and accumulated values. At each occurrence of a control break the current accumulated amount for that level is rolled forward or accumulated to the next higher level, and then reset to zero. It is not necessary that a total line be printed at every level of accumulation. At each detail reporting cycle (line or series), the current values of the designated fields are accumulated to the lowest level of accumulation. At control break time, appropriate roll forward and reset occurs with or without associated total line printing.

7. The FOR--- phrase is used to specify conditional accumulation or counting. The accumulation or count is performed at detail time only at true status of the condition or data name. Condition names and true-false data-names are defined in the File or Working-Storage Section to suit the needs of the programmer. Where a condition name is not appropriate, any complexity of logical expression can be examined in a Procedure Division sentence and a true-false data-name set for interrogation by the Report Writer. Once a detail value is accumulated (or a count incremented by 1) to the lowest level, that value automatically carries forward to all higher levels of accumulation or count. The condition name or true-false data-name must not have a qualifier.

8. The control-break level associated with the total line or line series is the implied level of accumulation to be printed for all of the accumulation and counts contained on the Line Definition entry. The current level of accumulation is made available for printing (and user reference) at object time by the Report Writer object program.

9. At detail time, a series of lines operates in exactly the same manner as an individual detail line. The values for accumulation and count fields are added once per series to the lowest level of accumulation. The field names do not need to appear in the detail line or series Line Definition entry. The detail field values to be accumulated may be calculated in a section of the Procedure Division at line printing time. See Line Section Entry, page .

10. Accumulation and Count Names may be referenced in a section of the Procedure Division executed at line preparation time. (See Line Section Entry, page .) The names shown under the formats above may be used as field names for this purpose. The following conventions apply to the accumulated values obtained:

    - At total line time, the value referenced is the current accumulated value for the level of line about to be printed.
    - At detail, heading, or footing time, the value referenced is the accumulated value printed on the last previous total time.

11. Field name may be qualified in accordance with the General Compiler language conventions. The qualifier data names (group, record, and/or file name) are written as "OF data-name" after the original field name and before any additional clauses. A given source field should always be written with the same qualifiers. Report name is not needed as a final qualifier in the Report Section Line Definition entries for that report. However, if the same Accumulation or Count Name is defined on more than one report, the report name must be used as a qualifier in all Procedure Division references to that field.

12. The programmer must not enter Accumulation and Count Names into the Working-Storage Section. Internal Process Storage is set up by the Report Writer.

13. The internal storage mode for Report Writer Accumulations is determined by the source data description of the item to be accumulated. The programmer may describe an Input (or Working-Storage) field as integer, fixed point, or floating point. Integer arithmetic retains decimal accuracy which may be lost in fixed point calculations as a result of the binary mode of computation. For this reason the programmer is urged to define fields for accumulation as integers with the K symbol to show implied decimal scaling rather than the V symbol. If the scaling of the input field in Process Storage will not also accommodate the accumulation, another scaling can be set by the programmer with the M data image convention on the input field. The leading character of the image must be M and the trailing character S followed by the scale desired. For example, an input field of size 9(4) can be assigned 38 rather than 19 scaling internally by describing the field as M9(4)S38, or M999K9S38. Count fields are given an internal scaling of 19 by the Report Writer. The decimal scaling shown by the K symbol is assumed at report format time for decimal alignment purposes when accumulation fields are printed. Count fields in Process Storage are always set up by the Report Writer as integer with a binary scale of 38.

14. At present, floating point numbers cannot be described on the Report Description form for printing other than in fixed form.

15. Approximately 30 unique Accumulation Names may be defined per report.

16. Accumulation and Count names may be included in the output files section. The qualifier area is used for the field names and any additional qualifiers or FOR names. The key words ACC, COUNT, etc. will appear under data name. The current level of accumulation or count will be moved to output.

CONTROL BREAK
CONDITION NAMES

FUNCTION

Control Break Condition Names allow interrogation of control break status in Procedure Division sentences.

FORMAT

CONTROL~BREAK [integer]

CONVENTIONS

1. True-false data-names are set up automatically by the Report Writer for each level of break defined under LINE CONTROL. header. These fields are set to true at detection of a break and remain set to true until completion of the printing of any total control break lines at that level.

2. These true-false data-names may be interrogated but not altered by the programmer. A common use is in a section of the Procedure Division for a control line operating at more than one level of break. Knowledge of the level may influence the assignment of certain field values to be printed. See Figures 22 and 25.

3. If more than one control-break level (other than FINAL) is defined, an array of true-false fields is set up by the Report Writer. The integer is the subscript value necessary to interrogate the correct level of break. Subscript values 1, 2, --- are level numbers assigned to control-break data-name-1, control-break data-name-2 --- respectively (See Control Breaks Entry).

4. The key word FINAL is used as a condition name to interrogate final reporting condition.

5. Unless control break fields are used in calculations by the programmer outside of the Report Section, they should be described as alphanumeric (X) to avoid unnecessary conversion to internal binary form. The Report Writer sets up an auxiliary field for each control-break data-name to hold previous data-name values for comparison purposes.

The two Page Control entries, Page Overflow and Line Number follow.

FUNCTION--PAGE OVERFLOW ENTRY

The Page Overflow entry specifies requirements for page overflow testing at line printing time.

FORMAT

{LINES/PAGE / LAST~DETAIL} [ARE / IS] integer-1 } starting in column 13 or after.

CONVENTIONS

1. The Page Overflow entry must be made under the PAGE CONTROL. header.

2. Page overflow causes the page break procedure of page footing print (if any), slew to top of next page, and page heading print, before printing of the intended line or lines. The page overflow test consists of a comparison of current line count on the page with an overflow line number calculated at report program generation time. Depending on the Page Overflow entry type and the lines defined for the report, an overflow line number may be calculated for use with each given line or series of type H, D, or T.

3. Either LINES/PAGE or LAST~DETAIL entry but not both, may be specified. The integer-1 value may not exceed 99.

4. If neither LINES/PAGE or LAST~DETAIL entry is specified, the Report Writer assumes that no page overflow testing is required in the object program in this case all page control and line spacing is accomplished by the pre- and postslew parameters in the Line Image entries.

5. A fixed line number (see Line Number entry), or E under the PRE column obviates any page overflow testing for that line type or series when a Page Overflow entry has been made.

6. LAST~DETAIL entry means that page overflow is tested only on detail (D) and any control heading (H) line types. LINES/PAGE entry means that page overflow is tested on these types as well as on the only other variable line type, control totals (T).

7. LAST~DETAIL is the normal parameter for specifying page overflow where both detail and total lines exist. The programmer specifies the LAST~DETAIL line number relative to the top of page (line number 0) that may contain a detail line type. This line number should allow room on the bottom of the page for all possible levels of totals and any page footings. To obtain a last detail line number for page overflow testing, the LAST~DETAIL line number is adjusted by the number of lines required for printing the detail line or series.

8. Page overflow line number for control headings (H) is calculated so that overflow occurs whenever the current level of heading lines plus at least one detail issue will not fit on the page.

9. The LINES/PAGE entry specifies the line number counting from the top of page on which the last line of printing on the page may occur. To obtain a last line number for page overflow test, the specified line number is first adjusted by the number of lines (including pre- and postslewing) required by any page footings, and second by the number of lines required by the current issue. On detail and control headings the conventions given in notes 4 and 5 apply to this latter adjustment. On control totals, the calculation is made so that overflow occurs whenever the number of lines needed for all total lines at the current level of control break will not fit on the page.

10. The object program does not test for page overflow within a series of lines, as the method of calculation of page overflow line number has protected a series from being broken across a page.

FUNCTION--LINE NUMBER ENTRY

The Line Number entry assigns fixed line numbers to report lines where the desired line spacing cannot be achieved by relative line spacing from other print lines.

FORMAT

LINE~NUMBER[S] { IS / ARE } literal-1 (integer-1) [, literal-2 (integer-2)]

---- [literal-n (integer-n)] . starting in column 13

CONVENTIONS

1. This entry must be made under the PAGE CONTROL. header.

2. The LINE~NUMBER[S] parameter entries assign fixed line numbers, relative to the top of the page, to one or more report lines of a given report. A common use of this parameter is to force the control-break total lines to a line number past the LAST~DETAIL line number. In this way the start of a new page with every new logical control group may be achieved.

3. The literal-1, literal-2, etc., values must be alphabetic characters shown in the PRE columns of Line Image entries. The integer-1, integer-2, etc., values must not exceed 99.

4. For purposes of object program efficiency, the LINE~NUMBER[S] parameter should not be used where the desired results can be obtained by slewing a fixed number of lines (or to top of page) before and/or after the printing of a given line. With the exception of the case outlined in convention 2 above and occasionally in the use of preprinted forms, spacing and page control can be maintained easily without use of this parameter.

5. The top of the page is defined by the channel punch in channel 8 of the printer paper tape loop.

6. The line number values integer-1, integer-2, etc., cannot exceed the LINES/PAGE value, if given.

7. A fixed line number can only be assigned to a pre-slew of an individual line or the initial line of a series, including RH, PH, PF, RF, lines.

8. When a fixed line number is assigned, the line or series will not be given a page overflow test.

9. At object time, a slew to a fixed line number less than the current line count results in page overflow followed by a slew to the fixed line number on the page.

PROCEDURE DIVISION--REPORT WRITER VERBS

The Report Writer uses two verbs in the Procedure Division, **GENERATE** and **TERMINATE**, which are discussed on the following pages.

FUNCTION

The GENERATE verb releases data for printing according to specifications given for report preparation.

FORMAT

```
GENERATE { detail-name   LINE[S]  [OF report-name REPORT] }
         { report-name            REPORT                 } .
```

CONVENTIONS

1. GENERATE detail-name causes execution of the Report Writer procedure which controls all report lines and the subprocedure for the detail named. After execution, control is returned to the next source sentence.

2. The line or series identified by detail-name must be described in the Report Section.

3. Detail-name is the name shown under the DATA NAME entry in the Line Image entry unless these positions were left blank. If blank, the designation under the LINE CODE entry is used. On a series of lines these are taken from the series header.

4. Any number of GENERATE sentences may appear in the Procedure Division.

5. The first GENERATE sentence executed for a report causes appropriate initialization of the report; this includes an initial slew to top-of-page. The programmer must open the report file in the Procedure Division, before the first GENERATE sentence.

6. At GENERATE detail-name, more than one type of line may be printed, depending on the outcome of evaluation of explicit and implied line control conditions. The intent of the GENERATE sentence is to produce a single body (detail) line or line series from current process values. However, if page break and/or control break conditions are satisfied, any or all of the line types, T, PF, PH, H may be printed before the current detail information going to the D or SD lines is printed. All Report Writer functions requested in the Report Section may be performed at any one GENERATE verb execution.

7. If multiple reports are defined, and the same detail name is given to details on separate reports, the OF report-name phrase must be included in the GENERATE sentence to identify the correct report. If it is missing, the first detail of that name in the Report Section will be referenced. The phrase may be written for documentation purposes even when not required for uniqueness.

8. GENERATE report-name phrase is valid only where no details have been defined for the report. All functions except detail printing are performed at each execution of the GENERATE verb. Lines are issued only at the occurrence of page break or control break conditions.

9. Once a report is terminated by a TERMINATE verb, a GENERATE sentence cannot be executed for that report.

TERMINATE

FUNCTION

The TERMINATE verb concludes preparation of a specified report.

FORMAT

TERMINATE report-name **REPORT.**

CONVENTIONS

1. A **TERMINATE** sentence causes execution of report closing functions. Return is to the source sentence immediately following the **TERMINATE** statement.

2. Only one **TERMINATE** sentence can be executed for each report.

3. Entrance to the final reporting procedure commonly is associated with an end-of-file detection on read-in.

4. A **TERMINATE** verb cannot be executed prior to a **GENERATE** verb on the same report.

5. A **TERMINATE** report-name phrase causes the **FINAL** condition to be set to true.

6. After the **TERMINATE** report-name phrase, no further **GENERATE** sentences can be executed for that report.

7. The programmer must close the report file in the Procedure Division after termination of the one or more reports assigned to that file.

8. If a report consists of only final report totals, a single **GENERATE** report-name sentence followed by a single **TERMINATE** report-name sentence may be executed to achieve the desired results.

ENVIRONMENT DIVISION--REPORT SECTION

(Refer to File-Control, page 149.)

The following phrase in the FILE-CONTROL sentence is used as described below:

FORMAT

[FOR OFF LINE PRINT]

CONVENTIONS

1. The OFF LINE phrase must be specified for report tapes to be created by the Report Writer for deferred printing. This applies whether the off-line printer or the generalized peripheral program (PIP) for on-line printing is employed. When printing is deferred file-name-1 is the name specified in the Report File Definition entry in the Report Section.

REPORT DESCRIPTION FORM CONVENTIONS

Programming Conventions

The Report Description form reserves columns 1-6 for sequence number, in conformance with other General Compiler forms. The programmer may request a sequence check for the source program as a whole. (See Chapter 4.)

The programmer may prefer to enter the Data Division header on the Report Description form rather than the Data Division form since the Report Section immediately follows this header in the source program input to the Compiler.

The following conventions apply to entries on the Report Description form

- Columns 7 and 12 must always be left blank by the programmer.
- Entries written in columns 8-11 (LINE CODE) that do not require the allotted number of columns may be entered in any consecutive subset of these column numbers.
- The RFD entry must contain RFD under LINE CODE and the file name in columns 13-24. The file name may start anywhere in these columns. The optional BLOCK CONTAINS clause must begin in column 28 or after and must be terminated by a period before the last column of the form.
- Report Layout portion:

  1. The word LAYOUT in the Report Layout header must start in column 28 or after and must be terminated before column 80 of the form.

  2. On Line Image entries, an entry in columns 13-24 (DATA NAME) may be contained in any consecutive subset of these column numbers. With the exception of detail line types (D or SD), these columns are ignored by the Report Writer.

- Report Definition portion:

  1. The word DEFINITION in the Report Definition header must start in column 28 or after and must terminate before column 80 of the form.

  2. The main section headers, LINEΔDEFINITIONS, LINEΔCONTROL, LINE\SECTIONS, PAGEΔCONTROL, must start in column 8.

  3. In Line Definition entries, columns 13-27 are ignored by the Report Writer. An entry may exceed one line on the form. The second line is started in column 28 or after and must be blank in columns 8-11 (LINE CODE). A word may not be split across these two lines. No other entries under DEFINITION may exceed one line on the form.

  4. The Control Breaks entry must be the first entry under LINEΔCONTROL. The entry must start in column 13 or after.

5. Except for line code, when required under columns 8-11, all other specifications in the definition portion must start in column 13 or after.

6. All entries should be terminated by a period.

- The Report Section is terminated by any of the following section headers starting in column 8:

```
ARRAY△SECTION.
TRUE~FALSE△SECTION.
INTEGER△SECTION.
FILE△SECTION.
```

Keypunching Conventions

Each line entry on the Report Description form is keypunched according to the following conventions:

1. Columns 1 through 80, terminating with print position 53, are keypunched in columns 1 through 80 of card 1.

2. A repeat of columns 1 through 6 and columns 8-11 is keypunched in these columns of card 2.

3. A hyphen (-) is keypunched in column 7 of card 2.

4. Columns 81 through 158, labeled 12 through 78 on the form and starting in print position 54, are keypunched in columns 12-78 of card 2.

5. If no information is contained past column 80 of card 1, or print position 53, card 2 of the line need not be keypunched.

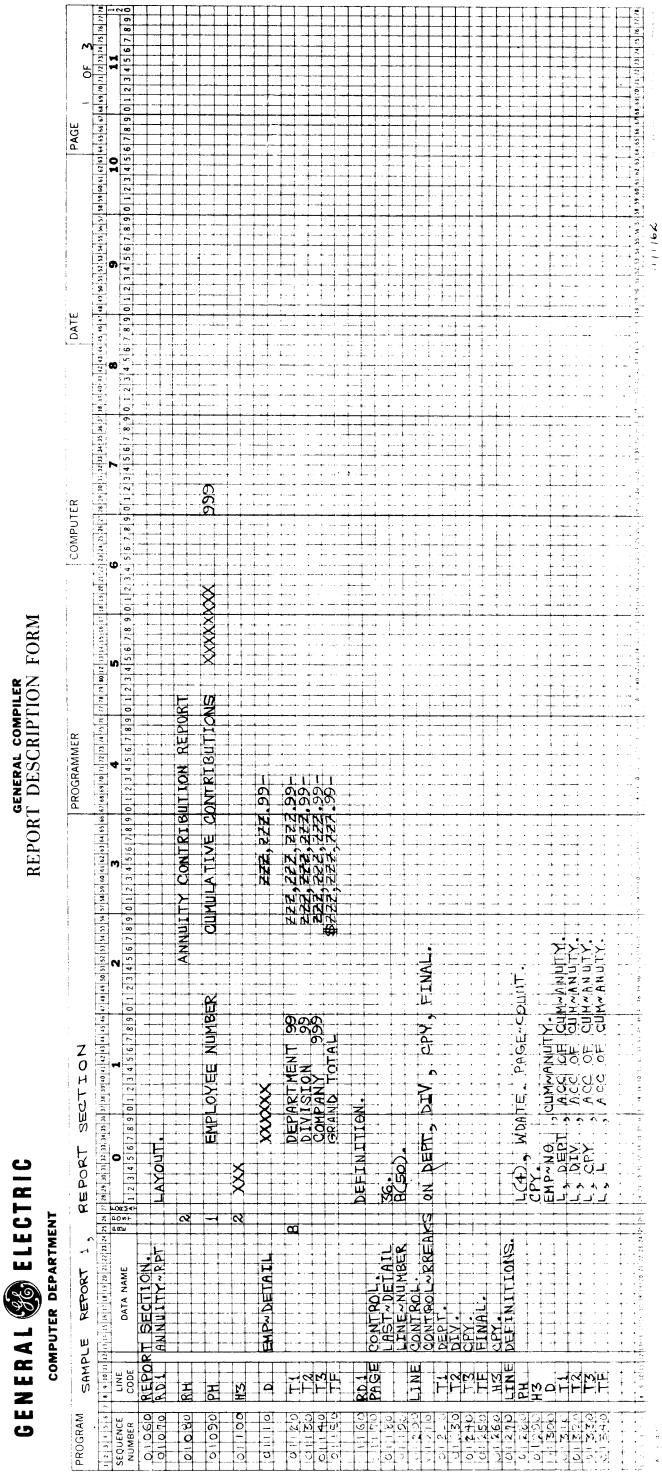


Figure 17. Sample Report 1, Report Section

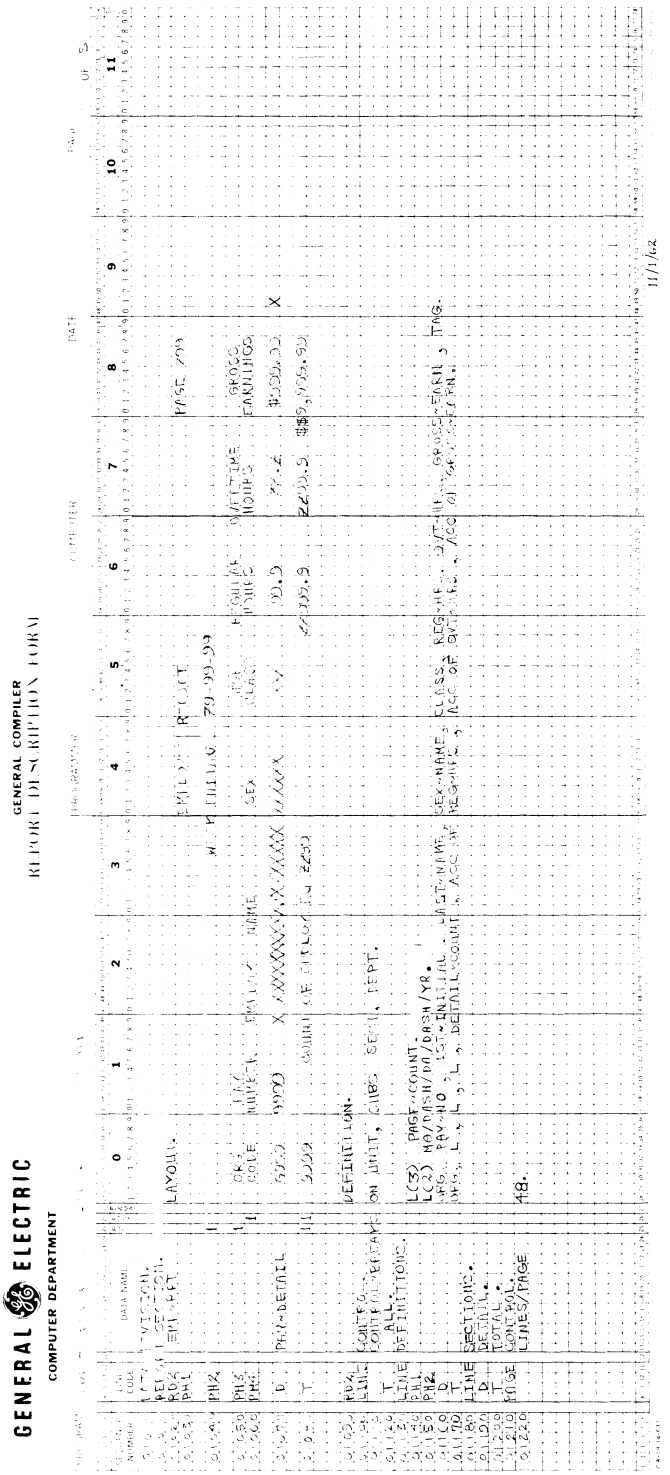


Figure 20. Sample Report 2, Report Section

GENERAL ELECTRIC

COMPUTER DEPARTMENT PHOENIX, ARIZONA

GENERAL COMPILER DATA DIVISION FORM

PROGRAM SAMPLE REPORT 2, DATA DIVISION — COMPUTER — DATE — PAGE 2 OF 5

PROGRAMMER

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	CODE	REPEAT	BLANK WHEN ZERO	JUSTIFIED	ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
01230		FILE SECTION.								
01240		INPUT FILES.								
01250	FD	EMP~MASTER, BLOCK 300 WORDS, RECORDING BINARY, CONTROL~KEY RECORD~TYPE.								
01260		LABEL RECORDS OMITTED.								
01270	R	EMP~RECORD								
01280	FL	RECORD~TYPE								1
01290	F	ORG								XXXX
01300	E	DEPT						01	01	
01310	E	SECT						02	02	
01320	E	SUBS						03	03	
01330	E	UNIT						04	04	
01340	F	PAY~NO								XXXX
01350	F	PAY~STATUS								AA
01360	F	1ST~INITIAL								A
01370	F	LAST~NAME								A(18)
01380	F	SEX								X
01390	C	MALE								1
01400	C	FEMALE								2
01410	F	CLASS								XXX
01420	F	WDATE								X(6)
01430	E	MO						01	02	
01440	E	DA						03	04	

CA14-1-62

11/1/62

Figure 21. Sample Report 2, Data Division

GENERAL ELECTRIC GENERAL COMPILER DATA DIVISION FORM

PROGRAM: SAMPLE REPORT 2, DATA DIVISION (CONT'D)

01450	F	YR		05	06	
01460	F	REG~HRS	x			99V9
01470	F	OVT~HRS	x			99V9
01480	F	DEDUCT~CODE				XX
01490	F	WKLY~WITHOLD				999V99
01500	F	WKLY~FICA				999V99
	ETC.					
01610	WORKING~STORAGE SECTION.					
01620	F	GROSS~EARN				999999V99
01630	F	SEX~NAME				X(6)
01640	F	TAG				X
01650	CONSTANT SECTION.					
	FL	DASH				"-"

CA 14 (1-62) 11/1/62

Figure 21 (cont.)

GENERAL ELECTRIC

COMPUTER DEPARTMENT PHOENIX ARIZONA

GENERAL COMPILER SENTENCE FORM

PROGRAM SAMPLE REPORT 2, PROCEDURE DIVISION DATE

PROGRAMMER COMPUTER PAGE 4 OF 4

SEQUENCE NUMBER	
01660	PROCEDURE DIVISION.
01670	DETAIL SECTION.
01680	BEGIN.
01690	GROSS~EARN = RATE*(REG~HRS + 1.5*OVT~HRS).
01700	MOVE SPACE TO TAG.
01710	IF GROSS~EARN NGR 100.00 GO TO NEXT.
01720	MOVE "*" TO TAG.
01730	NEXT. IF MALE, GO TO M~DO.
01740	MOVE "FEMALE" TO SEX~NAME.
01750	GO TO OUT.
01760	M~DO. MOVE "MALE" TO SEX~NAME.
01770	OUT. END DETAIL SECTION.
01780	TOTAL SECTION.
01790	BEGIN.
01800	IF CONTROL~BREAK(1), GO TO OUT.
01810	IF CONTROL~BREAK(2), GO TO RESET~1.
01820	IF CONTROL~BREAK(3), GO TO RESET~2.
01830	MOVE 0 TO SECT.
01840	RESET~2. MOVE 0 TO SUBS.
01850	RESET~1. MOVE 0 TO UNIT.
01860	OUT. END TOTAL SECTION.

11/1/62

Figure 22. Sample Report 2, Procedure Division

GENERAL ELECTRIC

COMPUTER DEPARTMENT PHOENIX, ARIZONA

GENERAL COMPILER SENTENCE FORM

PROGRAM SAMPLE REPORT 2, PROCEDURE DIVISION

PROGRAMMER COMPUTER DATE

PAGE 5 OF 5

SEQUENCE NUMBER		
01870		OPEN ALL FILES.
01880	A1.	READ EMP~MASTER IF END GO TO A2.
01890		GENERATE PAY~DETAIL OF EMP~RPT.
01900		GO TO A1.
01910	A2.	TERMINATE EMP~RPT.
01920		CLOSE EMP~MASTER, EMP~RPT.
01930		STOP RUN.
01940	END.	PROGRAM.

CA 13 (10-61)

11/1/62

Figure 22(cont.)

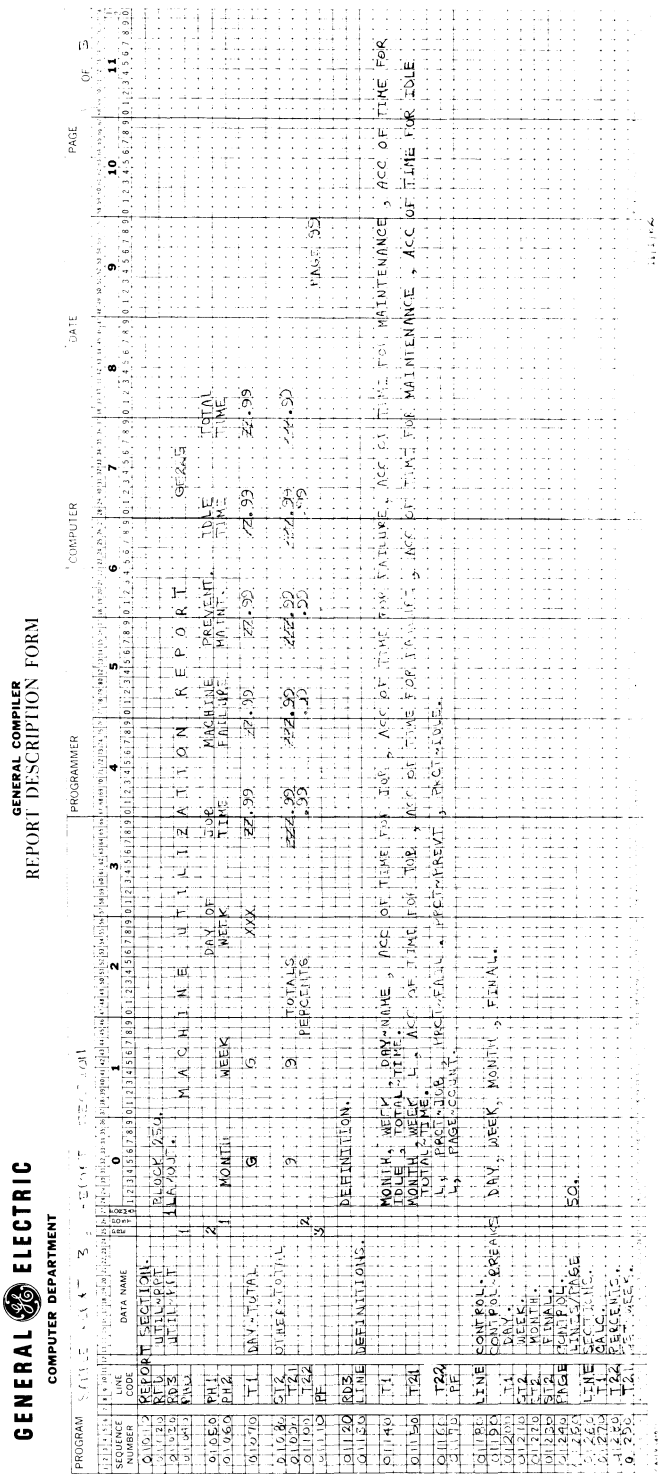


Figure 23. Sample Report 3, Report Section

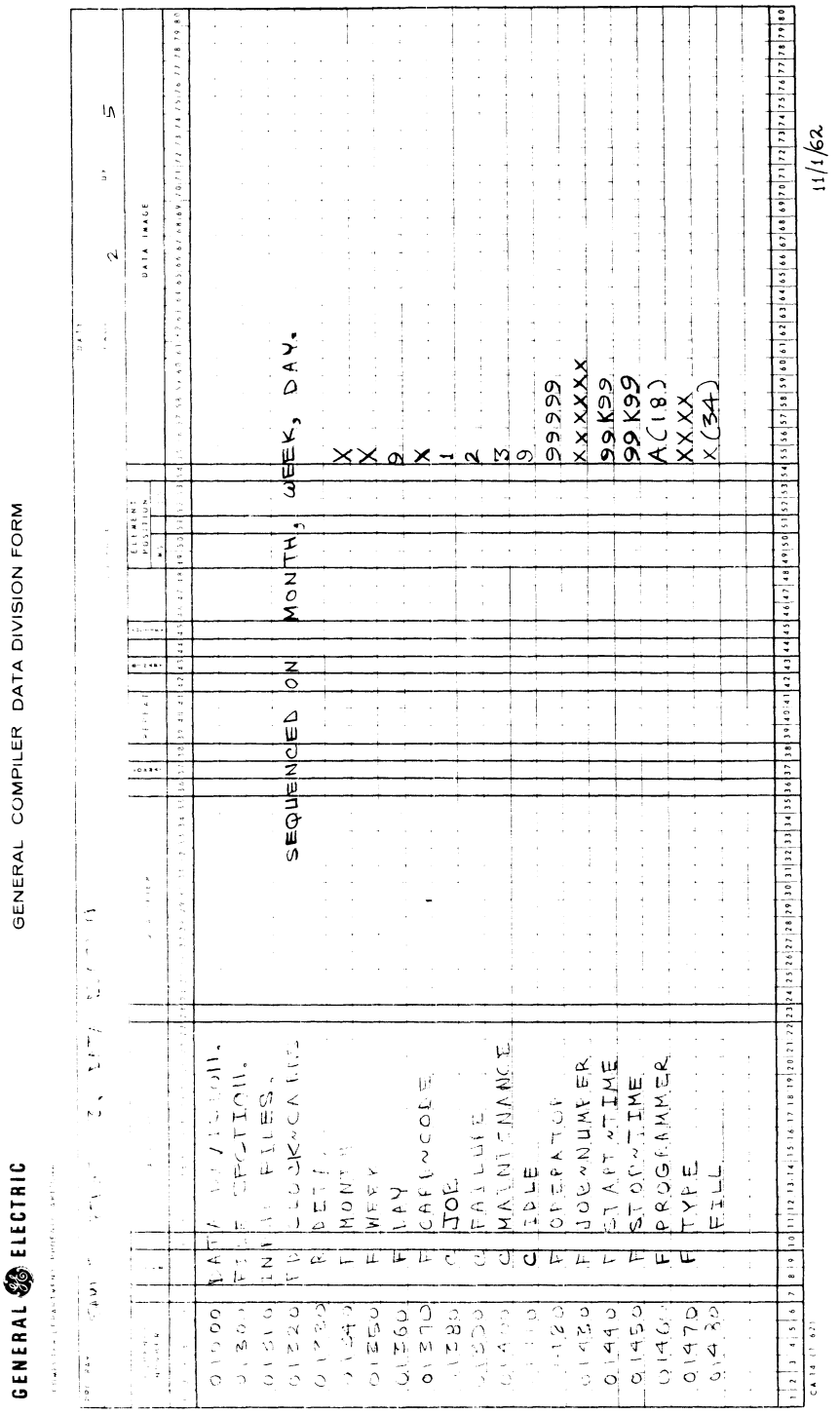


Figure 24. Sample Report 3, Data Division

GENERAL ELECTRIC

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER DATA DIVISION FORM

PROGRAM SAMPLE REPORT 3, DATA DIVISION (CONT'D) COMPUTER DATE

PROGRAMMER PAGE 3 OF 5

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	FORMAT	REPEAT	BLANK	JUSTIFY	ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
01490		WORKING~STORAGE SECTION.								
01500	F	TIME								99K99
01510	F	PRCT~JOB								K999
01520	F	PRCT~FAIL								K999
01530	F	PRCT~PREVT								K999
01540	F	PRCT~IDLE								K999
01550	F	TOTAL~TIME								99K99
01560	F	DAY~NAME								XXX
01570	CON	STANT SECTION.								
01580	FL	NAME			007					"MON"
01590										"TUE"
01600										"WED"
01610										"THU"
01611										"FRI"
01612										"SAT"
01613										"SUN"

CA14 (1 62)

11/1/62

Figure 24(cont.)

GENERAL ELECTRIC

GENERAL COMPILER SENTENCE FORM

PROCEDURE

COMPUTER

DATE

PAGE

SEQUENCE NUMBER		
0 1 6 2 0	PROCEDURE	DIVISION.
0 1 6 3 0	CROSSFT~TIME	SECTION.
0 1 6 4 0	BEGIN.	
0 1 6 5 0		TOTAL~TIME = ACC OF TIME FOR JOB + ACC OF TIME FOR FAILURE +
0 1 6 6 0		ACC OF TIME FOR MAINTENANCE + ACC OF TIME FOR IDLE.
0 1 6 7 0	END	CROSSFT~TIME SECTION.
0 1 6 8 0	PERCENTS	SECTION.
0 1 6 9 0	BEGIN.	
0 1 7 0 0		PERFORM CROSSFT~TIME.
0 1 7 1 0		PRCT~JOB = ACC OF TIME FOR JOB / TOTAL~TIME.
0 1 7 2 0		PRCT~FAIL = ACC OF TIME FOR FAILURE / TOTAL~TIME.
0 1 7 3 0		PRCT~PREVT = ACC OF TIME FOR MAINTENANCE / TOTAL~TIME.
0 1 7 4 0		PRCT~IDLE = ACC OF TIME FOR IDLE / TOTAL~TIME.
0 1 7 5 0	END	PERCENTS SECTION.
0 1 7 6 0	CALC	SECTION.
0 1 7 7 0	BEGIN.	
0 1 7 8 0		MOVE NAME(DAY) TO DAY~NAME.
0 1 7 9 0		PERFORM CROSSFT~TIME SECTION.
0 1 7 9 1	END	CALC SECTION.
0 1 7 9 5	SET~WEEK	SECTION.
0 1 7 9 6	BEGIN.	
0 1 8 0 0		IF CONTROL~BREAK(2), GO TO SET~1.

Figure 25. Sample Report 3, Procedure Division

GENERAL ELECTRIC

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER SENTENCE FORM

PROGRAM: SAMPLE REPORT 3, PROCEDURE DIVISION | DATE

PROGRAMMER | COMPUTER | PAGE

SEQUENCE NUMBER		
010810		IF CONTROL~BREAK(3), GO TO SET~2.
010820		MOVE SPACE TO MONTH.
010830	SET~2	MOVE SPACE TO WEEK.
010840	SET~1.	MOVE SPACE TO DAY~NAME.
010850	END. SET~WEEK SECTION.	
010860		OPEN ALL FILES.
010870	READ~CARD.	READ CLOCK~CARDS, IF END OF FILE GO TO END~RTN.
010880		TIME = STOP~TIME - START~TIME.
010890		IF TIME IS POSITIVE, GO TO NEXT.
010900		TIME = TIME + 12.00.
010910	NEXT.	GENERATE UTIL RPT.
010920		GO TO START.
010930	END~RTN.	TERMINATE UTIL RPT.
010940		CLOSE UTIL RPT.
010950		STOP RUN.
010960	END.	PROGRAM.

Figure 25(cont.)

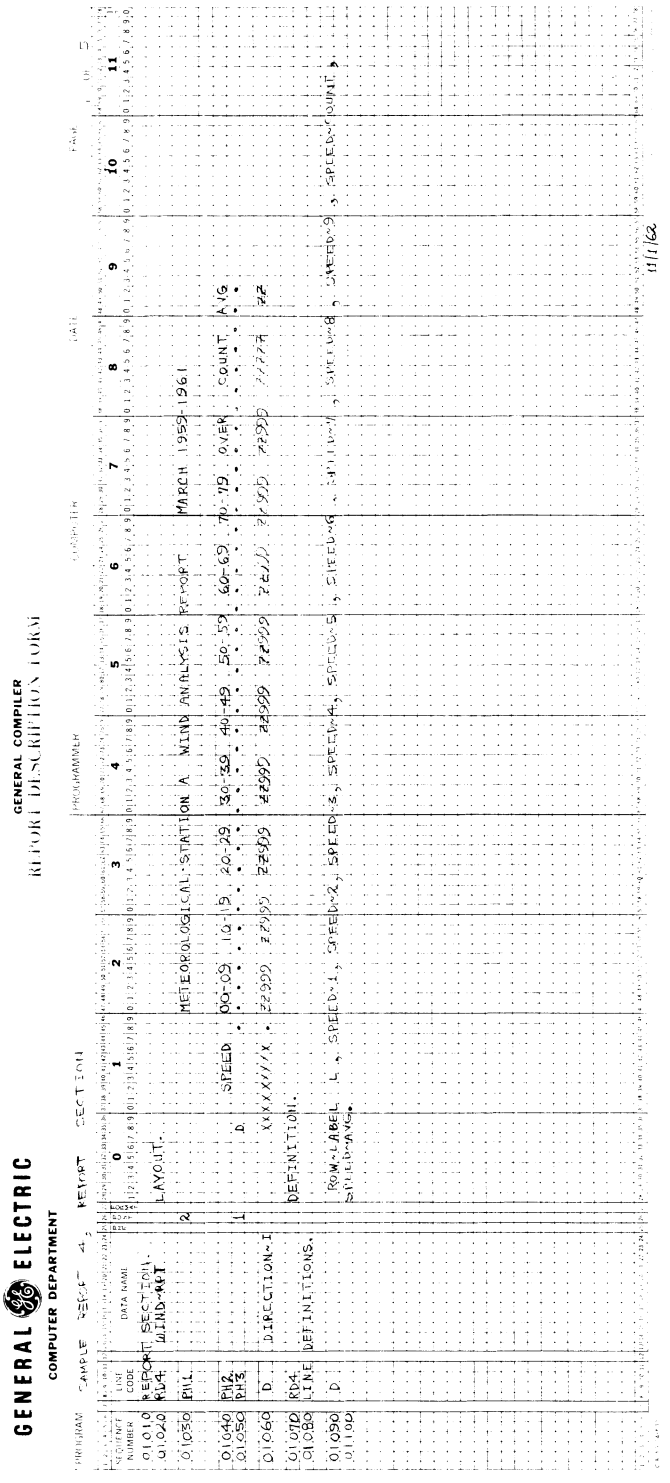


Figure 26. Sample Report 4, Report Section

GENERAL ELECTRIC

COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER DATA DIVISION FORM

PROGRAM SAMPLE REPORT 4, DATA DIVISION COMPUTER DATE

PROGRAMMER PAGE 2 OF 5

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	FORMAT	REPEAT	BLANK	JUSTIFY	ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
01000	DATA	DIVISION.								
01110	ARRAY	SECTION.	WIND(11,12).							
01120	FILE	SECTION.								
01130	INPUT	FILES.								
01140	FD	WNDA~QTR2.								
01150	R	WIND~DATA								
01160	F	SUBSTATION								XX
01170	F	YEAR								99
01180	F	MONTH								9
01190	C	MARCH								3
01200	C	APRIL								4
01210	C	MAY								5
01220	F	TIME								9999
01230	F	HEIGHT								99
01240	F	DIRECTION								99
01250	F	SPEED								99
01260	F	TECHNICIAN								A(18)
01270	F	DAY								99
01280		FILL								X(45)
01290	WORKING STORAGE	SECTION.								
01300	F	WIND			132					9999
01310	F	I								9

CA14 (1 62)

11/1/62

Figure 27. Sample Report 4, Data Division

GENERAL ELECTRIC

COMPUTER DEPARTMENT PHOENIX ARIZONA

GENERAL COMPILER DATA DIVISION FORM

PROGRAM [illegible] DATA DIVISION CONT'D
PROGRAMMER
COMPUTER
DATE
PAGE 2 OF 5

SEQUENCE NUMBER	TYPE	DATA NAME	QUALIFIER	FORMAT	REPEAT	BINARY	JUSTIFY	ELEMENT POSITION MS	ELEMENT POSITION LS	DATA IMAGE
01320	F	J								9
01330	F	SPEED~CLASS								99999
01340	F	SPEED~1								99999
01350	F	SPEED~2								99999
01360	F	SPEED~3								99999
01370	F	SPEED~4								99999
01380	F	SPEED~5								99999
01390	F	SPEED~6								99999
01400	F	SPEED~7								99999
01410	F	SPEED~8								99999
01420	F	SPEED~9								99999
01430	F	SPEED~COUNT								99999
01440	F	SPEED~AVG								99999
01450	F	ROW~LABEL								X(9)
01460	CONSTANT SECTION.									
01470	FL	LABEL			10					"IΔΔΔWΔΔΔ1"
01471										"RΔΔΔNWΔΔ2"
01472										"EΔΔΔNΔΔΔ3"
01473										"CΔΔΔNEΔΔ4"
01474										"TΔΔΔEΔΔΔ5"
01475										"IΔΔΔSEΔΔ6"
01476										"OΔΔΔSΔΔΔ7"
01477										"NΔΔΔSWΔΔ8"
01478										"ΔΔΔΔCOUNT"
01479										"ΔΔΔΔΔΔAVG"

CA 14 (1 62)

11/1/62

Figure 27 (cont.)

GENERAL ELECTRIC
COMPUTER DEPARTMENT PHOENIX, ARIZONA

GENERAL COMPILER SENTENCE FORM

PROGRAM SAMPLE REPORT 4, PROCEDURE DIVISION DATE
PROGRAMMER COMPUTER PAGE 3

SEQUENCE NUMBER		
01480	PROCEDURE DIVISION.	
01490		OPEN ALL FILES.
01500		MOVE ZEROES TO WIND.
01510	READ~FILE.	READ WIND~QTR2, IF END OF FILE GO TO CALC~DO.
01520		IF NOT MARCH, GO TO READ~FILE.
01530		I=DIRECTION.
01540		SPEED~CLASS=09.
01550	CLSFY~SPEED.	VARY J FROM 1 BY 1 UNTIL J EQ 9.
01560		IF SPEED NGR SPEED~CLASS, GO TO WIND~DO.
01570		SPEED~CLASS = SPEED~CLASS + 10.
01580	EXIT CLSFY~SPEED.	
01590	WIND~DO.	WIND(I,J) = WIND(I,J) + 1.
01600		WIND(I,10) = WIND(I,10) + 1.
01610		WIND(I,12) = WIND(I,12) + DIRECTION.
01620		WIND(9,J) = WIND(9,J) + 1.
01630		WIND(11,J) = WIND(11,J) + SPEED.
01640		GO TO READ~FILE.
01650	CALC~DO.	WIND(9,10) = WIND(9,1) + WIND(9,2) + WIND(9,3) + WIND(9,4) + WIND(9,5) +
01660		WIND(9,6) + WIND(9,7) + WIND(9,8) + WIND(9,9).
01670	SPD~AVGS.	VARY I FROM 1 BY 1 UNTIL I GR 8.
01680		WIND(I,11) ROUNDED = WIND(I,12)/WIND(I,10).
01690	EXIT SPD~AVGS.	

11/1/62

Figure 28. Sample Report 4, Procedure Division

GENERAL ELECTRIC

GENERAL COMPILER SENTENCE FORM

```
01700  DIR~AVGS. VARY I FROM 1 BY 1 UNTIL I GR 9.
01710      WIND(10,J) ROUNDED = WIND(11,J)/WIND(9,J).
01720  EXIT DIR~AVGS.
01730  LINE~DO. VARY I FROM 1 BY 1 UNTIL I GR 10.
01740      MOVE WIND(I,1) TO SPEED~1.
01750      MOVE WIND(I,2) TO SPEED~2.
01760      MOVE WIND(I,3) TO SPEED~3.
01770      MOVE WIND(I,4) TO SPEED~4.
01780      MOVE WIND(I,5) TO SPEED~5.
01790      MOVE WIND(I,6) TO SPEED~6.
01800      MOVE WIND(I,7) TO SPEED~7.
01810      MOVE WIND(I,8) TO SPEED~8.
01820      MOVE WIND(I,9) TO SPEED~9.
01830      MOVE WIND(I,10) TO SPEED~COUNT.
01840      MOVE WIND(I,11) TO SPEED~AVG.
01850      MOVE LABEL(I) TO ROW~LABEL.
01860      GENERATE DIRECTION~I LINE.
01870  EXIT LINE~DO.
01880      TERMINATE WIND~RPT.
01890      CLOSE WNDA~QTR2, WIND~RPT.
01900      STOP RUN.
01910  END PROGRAM.
```

11/1/62

Figure 28(cont.)

LEADING POSITION	INTERMEDIATE POSITION	TRAILING POSITION	ZERO SUPPRESSION	FLOATING	SYMBOL	MEANING
X	X	X			A	Alphabetic (A--Z, blank)
X	X	X			X	Alphabetic, numeric or any of computer set
X	X	X			9	Numeric (0--9)
X	X				.	Decimal point
	X				,	Comma
X			X		Z	Zero suppression
X			X	X	$	Dollar sign
X		X	X	X	*	Check protection
X		X	X	X	+	Plus sign if positive
						Minus sign if negative
X		X	X	X	-	Blank if positive
						Minus sign if negative
		X			CR	Blank if positive
						CR if negative
		X			DB	Blank if positive
						DB if negative
X					G	Group indicate (1 only)
X		X			I	12 row overpunch if positive
						11 row overpunch if negative
X		X			R	No overpunch if positive
						11 row overpunch if negative

* Complete zero suppression requires the Z symbol in all numeric positions, both left and right of the decimal point.

Figure 29. Report Data Image Symbols

APPENDIX A
COMPILER VOCABULARY

The vocabulary of the General Compiler consists of the following words. These words always must be spelled correctly and cannot be used as data or procedure names. Key letters after each word (IEDPR) denote the division in which that word is most commonly used. (I=Identification, E=Environment, D=Data, P=Procedure, and R=Report writer.

ABS			P		FOUR(S)		E	P		PRINT		E			
ACC				R	FROM			P		PRINTER(S)		E			
ACCUMULATOR				R						PRIORITY		E			
ACTIONS			P		GAP			P		PROCEDURE		E		P	
ADD			P		GENERATE			P		PROCEED				P	
ADDRESSING			P		GIVING			P		PROCESS			D		
ADVANCE			P		GO			P		PROGRAM	I	E		P	
ADVANCING			P		GR			P		PROGRAM-ID	I				
AFTER		E			GREATER			P		PROGRAMMED		E			
ALL				R	GROUP			D	P	PTP		E			
ALTER			P							PTR		E			
ALTERNATE		E			HARDWARE		E			PUNCH		E			
AND			P		HIGH		E								
API		E			HSP		E			READ				P	
ARE			D							READING				P	
AREA		E			IDENTIFICATION	I		D		READER		E			
ARRAY			D		IF				P	READY				P	
ASSIGN		E			IN		E			RECORD(S)		E	D	P	
AT			P		INPUT		E	D	P	RECORDING			D		
ATAN			P		INTEGER			D		REEL		E	D		
AUTHOR	I				INTERRUPT		E			RELEASE				P	
AUTOMATIC		E			INTO				P	RELOCATABLE				P	
					I-O-CONTROL		E			RELOCATE		E			
BEGIN		E		P	IS			D	P	REPLACING					
BEGINNING		E								REPORT					R
BGN-FIL-LABL			D		JOURNAL TAPE		E	D		RERUN		E			
BGN-TAP-LABL			D		JT		E	D		RESERVE		E			
BINARY			D							REWIND				P	
BLOCK		E	D	P	L			D	R	ROUNDED				P	
BLOCKED		E			LABEL		E	D		ROW(S)				P	
BUFFER		E			LAST-DETAIL				R	RUN				P	
BY				P	LAYOUT				R						
					LESS				P	SAME		E			
CARD		E			LINE(S)				P	SECTION			D	P	R
CHAIN				P	LINE-COUNT				P	SEE					
CHAINING				P	LINE-NUMBER(S)				R	SEGMENT		E		P	
CLOSE(D)				P	LINES/PAGE				R	SELECT		E			
COBOL-MODE	I				LN				P	SENTINEL			D		
COMMON-STORAGE		E	D		LOAD				P	SEQUENCED			D		
*COMMON-STORAGE		E	D		LOCK				P	SEQUENTIAL		E			
COMPUTATION-MODE		E			LOG				P	SEVEN(S)		E		P	
CONDITIONS				P	LOWER		E			SIN				P	
CONSOLE				P	LS				P	SIX(ES)		E		P	
CONSTANT			D							SIZE		E	D	P	
CONTAINS		E	D		MAGNETIC		E			SPACE(S)				P	
CONTROL				P	MAIN		E			SPEED		E			
CONTROL-BREAK(S)				R	MEMORY		E			SQRT				P	
CONTROL-KEY			D		MIO		E			STANDARD		E			
COPY				P	MODE			D		STOP				P	
COPYING				P	MODULE(S)		E			STORAGE			D		
COS				P	MOVE				P	SUBTRACT				P	
COUNT				R	MT		E			SWITCH(ES)				P	
CP		E			MULTIPLE		E								
CR		E			MULTIPLY				P	TABLE				P	
CS			D	P						TAPE(S)		E			
					NEGATIVE				P	TERMINATE			D		
DATA			D		NEQ				P	THAN				P	
DATE-COMPILED	I				NEXT-PROGRAM	I				THREE(S)		E		P	
DEFINITION(S)				R	NGR				P	TO		E		P	
DEPENDING				P	NINE(S)		E		P	TOP				P	
DETAIL-COUNT				R	NLS				P	TRUE-FALSE			D		
DISC		E			NO				P	TWO(S)		E		P	
DIVIDE				P	NONE					TYPEWRITER				P	
DIVISION	I	E	D	P	NOT				P						
DSU-CONTROL		E			NOTE				P	UNEQUAL				P	
DSU(S)		E			NO-SET			D		UNIT		E			
					NUMBER		E			UNTIL				P	
EIGHT(S)		E		P						UPPER		E			
END		E	D	P	OBJECT-COMPUTER		E			USE		E			
ENDING		E			OBJECT-PROGRAM		E			USING				P	
END-FIL-LABL			D		OF		E		P R						
END-TAP-LABL			D		OFF		E			VARY				P	
ENTER				P	OMITTED			D							
ENTRANCE		E			ON		E	D	P	WITH				P	
ENVIRONMENT		E			ONE(S)		E		P	WORDS			D		
EQ				P	OPEN				P	WORKING-STORAGE			D	P	
EQUAL(S)				P	OPTIONAL		E			WRITE				P	
ERROR		E		P	OR				P	WRITING				P	
EVERY		E			OUTPUT		E	D	P	WS			D	P	
EXCEEDS				P	OVERLAY-SEGMENTATION		E								
EXCHANGE				P						ZERO(S)		E		P	
EXIT				P	PAGE				P	ZEROES		E		P	
EXP				P	PAGE-COUNT				R	ZERO-SET			D		
					PAPER		E								
FILE(S)		E	D	P	PERFORM				P	18-BIT			D		
FILE-CONTROL		E			PL		E								
FINAL				R	PLACE		E			215		E			
FIVE(S)		E		P	PLUG(S)		E			225		E			
FLOATING		E			POINT		E			235		E			
FLPT		E			POSITION		E								
FOR	I			R	POSITIVE				P						

APPENDIX B
ORDER OF SOURCE PROGRAM

IDENTIFICATION DIVISION	Mandatory
PROGRAM~ID.	Mandatory
NEXT~PROGRAM.	Optional
AUTHOR.	Optional
DATE~COMPILED.	Optional
INSTALLATION.	Optional
SECURITY.	Optional
REMARKS.	Optional
ENVIRONMENT DIVISION	Mandatory
OBJECT~COMPUTER.	Optional
I~O~CONTROL.	Optional
FILE~CONTROL.	Optional
DSU~CONTROL.	Optional
COMPUTATION~MODE.	Optional
DATA DIVISION	Mandatory
REPORT SECTION.	Optional
ARRAY SECTION.	Optional
TRUE~FALSE SECTION.	Optional
INTEGER SECTION.	Optional
FILE SECTION.	Optional
OUTPUT FILES.	Optional
INPUT FILES.	Optional
WORKING~STORAGE SECTION.	Optional
COMMON~STORAGE SECTION.	Optional
*COMMON~STORAGE SECTION.	Optional
CONSTANT SECTION.	Optional
PROCEDURE DIVISION	Mandatory
Sections and closed decision tables (may be intermixed)	Placement mandatory if used
Master program (including open decision tables if used)	Mandatory
END PROGRAM.	Mandatory

APPENDIX C
OBJECT PROGRAM RELOCATABLE DECK FORMATS

All binary instruction decks (segments or subroutines) are preceded by one or more cards called Header cards. These cards contain information for the loader. They indicate the names of all subprograms (segments or subroutines) referenced by the particular routine, all entrances to the routine plus other information required in loading.

HEADER CARD FORMAT

1. The contents of words 0 through 6 are explained in Figure 30.

2. Entrance name blocks are five words in length. The first entrance name block starts in word 7. The first four words of each block contain a left-justified BCD name of twelve or less characters of an entrance to the routine. The fifth word of each block specifies, in binary, the relative position of the entrance. The entrance to the first instruction of a routine is position zero. (An entrance ten locations down from the first instruction would be position ten.) There are as many five word blocks as specified by the count in word 6.

3. Subprogram name blocks follow the entrance name blocks and are four words in length. Each block contains a left-justified BCD name of a required subprogram. There are as many four word blocks as are specified in word 5.

4. Additional Header cards contain the first two words of basic information followed by the continuation of the previous card.

5. Checksums are only thirteen bits long. They are computed first as twenty-bit checksums and then broken down into seven and thirteen-bit groups which are added together, overflow checked, and stored as a thirteen-bit checksum.

INSTRUCTION CARD FORMAT

Instructions are in relocatable binary format. Each group of nine or less words to be loaded is preceded by a control word. Bits 2-3 of the word control the loading of the first word of the group; bits 4-5 control the loading of the second word of the group; and so on, until bits 18-19 control the loading of the ninth word of the group if that many words are required.

Control Bits

The control bits have the following meaning:

00 - relocate the word with respect to the origin assigned to the first instruction by the loader.

01 - relocate the word as for 00 above, and increment the address portion of the word by the origin assigned to the first instruction of the routine.

10 - relocate the word as for 00 above, and develop the address as for 01 above, then complement the address developed.

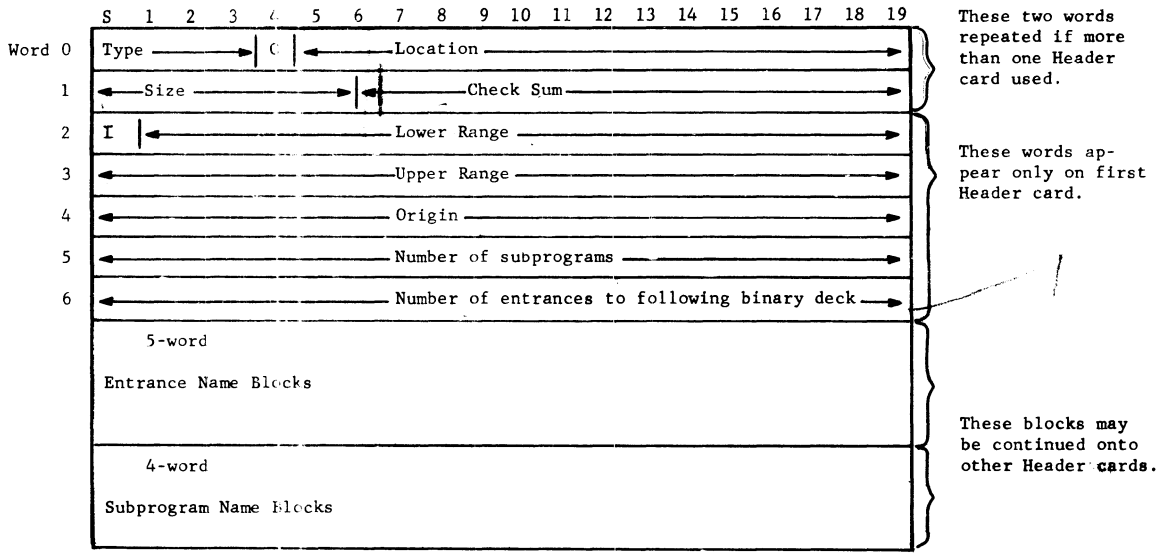
Full Instruction Card

A full Instruction card has the following format:

Word 0	Bits S, 1, 2, 3: Type (0011)
	Bit 4: Same meaning as C in Header card (Figure 30).
	Bits 5-19: Same meaning as Origin in Header card (Figure 30).
Word 1	Bits S-6: Same meaning as Size in Header card (Figure 30).
	Bits 7-19: Checksum.
Word 2	Control word for first nine words to be loaded.
Words 3-11	Nine words to be loaded.
Word 12	Control word for next nine words.
Words 13-21	Nine words to be loaded.
Word 22	Control word for next nine words.
Words 23-31	Nine words to be loaded.
Word 32	Control word for next seven words.
Words 33-39	Seven words to be loaded.

Partial Instruction Card

If an Instruction card is not full, only the control words or bits necessary are used.



Legend

Type: Binary 1000 in bits S, 1, 2, 3.

C: Always 0 when produced by compiler. Can be changed to 1 to override checksum.

Location: Address to begin Vector Table.

Size: Number of words used on this card.

I: 1 means to load up to lower range into upper 8k.
0 means to load up to lower range into lower 8k.

Lower Range: Relocatable origin of first location of input file tables (if present), 8177 if not present.

Upper Range: Relocatable address of last location of constant area (if present), 8191 if not.

Origin: Beginning relocatable address of following binary deck.

Figure 30. Header Card Format

APPENDIX D
OBJECT PROGRAM CONSTANTS

The references to constants used by a GECOM-produced object program and the library routines are equated to absolute memory locations. The constants themselves are included in the object program whenever the subroutines are punched. These constants may be referenced by name in General Assembly Program coding written by the programmer in the source program. The values of the constants are given below:

Name	Constant		Remarks
ZER	DDC	0	Always even location
Z00	EQU	ZER	
Z01	DEC	1	
Z02	DEC	2	
Z03	DEC	3	
Z04	DEC	4	
Z05	DEC	5	
Z06	DEC	6	
Z07	DEC	7	
Z08	DEC	8	
Z09	DEC	9	
Z10	DEC	10	
Z11	DEC	-1	
Z12	DEC	-2	
Z17	DEC	17	
Z18	DEC	18	
Z19	DEC	-18	
Z20	DEC	-19	
Z24	DEC	24	
Z25	DEC	25	
Z26	DEC	20	
Z30	ALF	00.	
Z31	ALF	00,	
Z32	OCT	0000060	
Z33	ALF	00-	
Z34	ALF	00+	
Z37	DEC	37	
Z38	DEC	38	
Z40	OCT	3777717	
Z41	OCT	3606060	
Z42	OCT	2777777	
Z43	OCT	3777760	
Z44	OCT	3776077	
Z45	OCT	3607777	
Z46	OCT	3000077	
Z47	OCT	3770000	
Z48	OCT	3777700	
Z49	OCT	3007777	

Name	Constant		Remarks
Z50	OCT	0000000	Always even location
Z51	OCT	0000013	
Z52	OCT	0006000	Floating powers of 10
	OCT	0000000	
	OCT	0022400	
	OCT	0000000	
	OCT	0037100	
	OCT	0000000	
	OCT	0053720	
	OCT	0000000	
	OCT	0072342	
	OCT	0000000	
	OCT	0107032	
	OCT	1000000	
Z80	OCT	2000000	
Z81	OCT	3700000	
Z82	OCT	0100000	
Z83	OCT	3777001	
Z84	OCT	3577777	
Z85	OCT	0000066	
Z86	OCT	0006600	
Z87	OCT	0077777	
Z88	DDC	1B38	
BLK	ALF		Double Length Blanks
	ALF		
FCA	OCT	0167000	
	OCT	0000000	
FCB	OCT	0164000	
	OCT	0000002	
FCC	OCT	0173000	
	OCT	0000000	
FCD	OCT	0170000	
	OCT	0000001	
FLB	OCT	0006060	
Z*	ALF	00*	
Z$	ALF	00$	

APPENDIX E
OBJECT PROGRAM TYPING SUBROUTINES

The following typing subroutines produced in GECOM object programs are also available to the programmer writing General Assembly Program coding in a GECOM source program, if the General Assembly coding is not a part of a separate relocatable segment.

1. TY1
   CALL - SPB TY1, 3
   PURPOSE: TAB ONCE, SET RED, TYPE

2. TY1 + 2
   CALL - SPB TY1 + 2, 3
   PURPOSE: TAB TWICE, SET RED, TYPE

3. TY1 + 4
   CALL - SPB TY1 + 4, 3
   PURPOSE: TAB ONCE, SET BLACK, TYPE

4. TY1 + 6
   CALL - SPB TY1 + 6, 3
   PURPOSE: TAB TWICE, SET BLACK, TYPE

5. TY1 + 8
   CALL - SPB TY1 + 8, 3
   PURPOSE: SET RED, TYPE

6. TY1 + 10
   CALL - SPB TY1 + 10, 3
   PURPOSE: SET BLACK, TYPE

In all of the TY1-type entries, a 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. A third word is required in the calling sequence and gives the beginning address of the list.
Sign Bit OFF	-	Type the 3 BCD characters in positions 2-19 of this word.
Position 1 Bit ON	-	No carriage return after typing.
Position 1 Bit OFF	-	Carriage return after typing.

Examples:

1.
```
SPB TY1+6, 3
ALF ERR
```
One word, Carriage return.

2.
```
SPB TY1, 3
OCT 2000006
LDA MESS
```
Type 6 words starting at MESS and Carriage Return

3.
```
SPB TY1+6, 3
OCT 1255151
```
One word, no carriage return.

4.
```
SPB TY1+10, 3
OCT 3000007
LDA LIST
```
Type 7 words starting at LIST. No carriage return.

APPENDIX F
INPUT/OUTPUT SYMBOLIC NAME ASSIGNMENT

FILE NUMBERS

Each file is assigned a 2-digit BCD file number starting with 00 and continuing to a possible 99. Numbers are assigned in order. The first output file described in the Data Division is assigned number 00.

The file number appears as the first two characters of every name associated with coding for that file and is referred to in examples as fn.

SPECIAL CHARACTERS

The third character of the name depends on the routine or area in which a name is used as shown in Figure 31.

Character	Related Routine or Area
S	Entries of file table
T	I/O Control
U	Open
V	Close
W	Read, Write Entrance
X	I/O Area 1
Y	I/O Area 2
Z	Unpacking or Packing*
#, @, :, %, $, '	READ AND COPY UNTIL

* Unpacking is the move/convert of input fields mentioned in the procedures. Packing is the move/convert to output buffer.

Figure 31. Third Character for I/O Symbolic Name Assignment

RECORD NUMBERS

Record numbers are assigned to each data record of a file starting with 00 and continuing to a possible 98. Label records are assigned BT (Beginning Tape), BF (Beginning File), ET (End Tape), or EF (End File), if they contain nonstandard portions.

Record numbers (referred to as rn) are used in the names of the packing and unpacking routines.

Examples:

01Z02	If 01 is an input file number, this is the name of the routine that unpacks record number 02 defined for file 01. If 01 is an output file number, this is the name given to the routine that packs record number 02.
03ZET	Name of routine that unpacks or packs added information in the END~TAP~ LABL of file 03.

SYMBOLIC I/O ENTRANCES AND FUNCTIONS PERFORMED

The symbols are described in the order they will appear for that file in the input/output coding portion of the Edited Listing.

Name	Description
fnU	Open File Entrance Functions: a. Set read switch (fnW) to process. b. Set end-of-file indicator to off. c. Test open/close indicator; if file open, type error message. d. Set open/close indicator to open. e. Clear record count to zero. f. If buffered file, initialize read command and end-of-file test. g. If buffered file, prime first buffer. h. Exit to procedure coding.
fnW	Read File Entrance. (This is the open/close switch. It is set to go to TER when closed and set to go to fnW+2 when open.) Functions: a. If not buffered, read a card into input buffer. b. Delay until last card read is in the input buffer. c. Check sychronization word for card read error. d. If read error go to fnTEC. e. Check for end-of-file card. If end-of-file, set indicator in the first word of the file table and transfer control to fnT. f. Update card count. g. If a buffered file: 1. Execute read command to fill next input buffer. 2. Exchange buffers and modify read command, synch. test, and end-of-file test. 3. Set input area pointer fnTCP. h. If no control keys or read until, go to unpacking (fnZ00). i. If control keys or read until, go to fnZCK.

Figure 32. Input Card Files

Name	Description
fnTS3	Card Read Error Check (this name presently only on buffered files.)
fnTS0	End-of-File Test (This name is present only on buffered files.)
fnTS2	Read Command (This name present only on buffered files.)
fnTEC	Card Reader Error Recovery Functions: a. Type error message. b. Wait for operator to reposition the file. c. If not buffered, return to read card again. d. If buffered, read card and go to wait for card reader ready.
fnV	Close File Entrance Functions: a. Set read switch (fnW) to error. b. Test file closed; if closed, type error message. c. Set open/close indicator to close. d. Type record count. e. Return to procedure coding.
fnZrn	Entrance to Input Unpacking for Record Number rn.
TER	Type error message when programmer tries to read file before opening it.
TCT	Type Count a. Type card count on closing. b. Type record count on control-key error.
fnS	First word of file table.
fnTXT	Temporary storage for exit to Procedure coding.
fnTCP	Input Area Pointer.
fnT	End-of-File Transfer Address to Procedure coding.
fnTLL	Length of Record just Read.
fnTMW	End-of-File Image.

Figure 32. (Cont.)

Name	Description
fnU	Open File Entrance Functions: a. Set write switch (fnW) to Process. b. Test open/close indicator; if file open, type error message. c. Set open/close indicator to open. d. Clear record count to zero. e. Exit to procedure coding.
fnW	Write File Entrance (This is the open/close switch. If the file is closed, it is set to go to TER and types an error message. If the file is open, it is set to go to fnW+2.) Functions: a. Set output buffer packing transfer. b. Save exit to procedure coding. c. If buffered: 1. Go to TBP to clear the output area unless no-set option used or if a Process File. 2. Go to packing. 3. Wait for punch to come ready. 4. Execute punch command. 5. Update record count. 6. Exchange output buffers in the file table. 7. Set punch command to next output buffer. 8. Set output area pointer (fnTCP), to next buffer. 9. Return control to procedure coding. d. If not buffered: 1. Wait for punch to come ready if not Process File. 2. Clear the output buffer unless no-set option was used or if a Process File. 3. Go to packing. 4. Issue punch command. 5. Update record count. 6. Wait for punch to come ready if a Process File. 7. Return control to procedure coding.
fnWI	Used as symbol for making modifications when buffered, and entrance to control point from packing, thru fnWE.
fnV	Close Entrance Functions: a. Set write switch to error. b. Test if file closed; if closed, type an error message. c. Set open/close indicator to close. d. Type record count. e. Exit to procedure coding.

Figure 33. Output Card Files

Name	Description
fnWrn	Write Record Entrance Functions: a. Pass record packing address to write file entrance. (After output area has been prepared for packing, control is transfered to fnZrn which packs the record and returns to fnWE.) b. Transfer control to Write File Entrance.
fnZrn	Entrance to output packing for record number n
TER	See Input Card Files.
TBP	Entrance to TBK.
TBK	Store blanks or zeros in output area before packing.
fnS	First word of file table.
fnTXT	Temporary Storage for Exit to Procedure Coding.
fnTCP	Output Area Pointer
fnWE	Entrance to Control Routine from Packing Coding Always branches to fnWT.

Figure 33. (Cont.)

Name	Description
fnU	Open File Entrance Functions: a. Set write switch (fnW). b. Test open/close indicator; if open, type an error message. c. Set open/close indicator to open. d. Clear record count. e. Exit to procedure coding.
fnW	Write File Entrance (This is the open/close switch. If the file is closed, the switch is set to go to TER to type an error message. If the file is open, the switch is set to go to fnW+2.) Functions: a. Set packing transfer b. Save exit to procedure coding.

Figure 34. Printer Files

Name	Description
	c. If any advancing, other than normal one line, execute SPL. d. Clear line counter (PC6) if "top-of-page" option used on this write. e. If buffered: 1. Go to TBP to clear the output area unless no-set option used, or if an advance only, or if a Process File. 2. Go to packing routine unless an advance only. 3. Wait for printer to come ready. 4. Execute print or slew command. 5. Update line count. 6. Exchange buffers. 7. Reset print command. 8. Reset line count indicator. 9. Check for advance complete and if not, go back to finish advancing. 10. Exit to Procedure Coding. f. If not buffered: 1. Wait for printer to come ready if not a Process File. 2. Go to TBP to clear the output area unless no-set option used, or an advance only, or if a Process File. 3. Go to packing unless an advance only. 4. Execute print command or slew command. 5. Update line count. 6. Reset print command. 7. Reset line count indicator. 8. Wait for printer to come ready if a Process File. 9. Check for advancing. Completed; if not go back to finish advancing. 10. Exit to procedure coding.
fnWT	Use as symbol for making modifications to control coding, and entrance to control coding from packing thru fnWE
fnV	Close Entrance Functions: a. Set write switch to error. b. Test if file closed; if closed, type an error message. c. Set open/close indicator to close. d. Type count. e. Exit to procedure coding.
fnWE	Entrance to control routine from packing coding. Always branches to fnWT.
fnADV	Advance Printer Entrance Function: Set write instruction to "advance only" and go to write file entrance.

Figure 34. (Cont.)

Name	Description
fnWrn	Write Record Entrance Functions: a. Pass record packing address to write file entrance. (After output buffer has been prepared for packing and write instruction developed, control is transfered to fnZrn. After packing is finished control is returned to fnWE.) b. Transfer control to write file entrance.
fnZrn	Entrance to output packing for record number n.
SPL	Printer Advancing Setup. Builds printer instructions.
TER	See Input Card Files.
TBP	Entrance to TBK
TBK	Store blanks or zeros to clear an output area before packing.
fnS	First word of file table.
fnTXT	Temporary storage for exit to procedure coding
fnTCP	Output Area Pointer.

Figure 34. (Cont.)

Name	Description
fnU	Performs the open functions. a. If optional file execute OPT. b. Execute OPN.
fnV	Performs the close functions. a. If input execute ICL. b. If output execute OCL.
fnW	Read/Write Entrance (This is a switch which is set by the OPN routine to go to TWR on index 3.) a. If the file is not open, execute TER. b. TWR used this call to obtain the file table address.
fnWrn	Write Record Entrance. Load A and Q registers with record packing address (fnZrn) and record length, transfer control to fnWE.
fnZrn	Entrance for unpacking or packing of record number n.

Figure 35. Tape Files

Name	Description
CF0	Ready for Overlay Load. a. If tape overlay, check last tape instruction and branch to overlay load subroutine. b. If disc storage unit overlay: 1. Go to MI0 to hold all disc storage instructions. 2. Save first word of punch buffer. 3. Execute overlay load subroutine. 4. Restore punch buffer. 5. Return to procedure coding.
TY1	Type Control. a. To type one word or a list of words after performing one of the following: 1. Set black, no tab. 2. Set red, no tab. 3. Set black, tab once. 4. Set red, tab once. 5. Set black, tab twice. 6. Set red, tab twice. b. To return or not return the carriage depending on the call word.
TY3	Return Carriage.
TY2	Type tape and plug number in the form PxTy. a. If output file referenced, tab twice. b. If input file referenced, tab once. c. If an error call, type in red.
TER	Type error message when programmer tries to read or write a file which has not been opened.
UCV	Add one in BCD to a number in the A-register. Used to update reel number and multifile tape position.
THL	Read console switches. Calls on RCS subroutine. Used for operator decisions.
TBK	Store blanks or zeros to clear an output area before packing.
TCT	Type Count. a. Type record on control-key error. b. Type block count on input tape error.
WSP	Tape Swap. a. Initialize first tape assign during open. b. Swap tapes at the end of a reel.

Figure 35. (Cont.)

Name	Description
OPN	Open Tape File. a. Set read/write switch (fnW) to open (SPB TAR, 3). b. Set exit to procedure coding (uses TSI). c. Initialize buffer synchronization. d. Initialize E of indicator. e. Check open/close indicator. f. Type error message if closed. g. Check lock/no-lock indicator. h. Type error message if locked. i. Execute WSP if multiple tapes assigned. j. Initialize tape mark indicator. k. Initialize tape error indicator. l. Initialize reel count to zero. m. Rewind tape if indicated in the call. n. If output execute TBK and transfer control to OLC. o. If input multifile file execute UFP to position tape. p. If a blocked input file set fnTCP to end of block to force a tape read on the first procedure read call. q. If input go to ILC.
ULM	Label Move. Moves an input label from the buffer to the label area (CLA) for label checking. Buffer address is set by WST.
YPC	Tape Alert Halt Check. a. Set plug number if more than one tape plug. b. Loop counting until controller becomes ready or count becomes zero. c. If controller comes ready, return normal (BRU1, 3). d. If count becomes zero: 1. Type E7 error. 2. Go to error return (other functions using this routine takes care of recovery).
WDH	Wait on tape Rewinding.
WDT	Rewind Tape.
WRL	Read or Write of Label Record.
WBR	Backspace Tape.
WRT	Read or Write Tape.
WRO	Read Zero Words (To space 1 block.)
WTM	Write Tape Mark.

Figure 35. (Cont.)

Name	Description
YIN	Entered from one of the above six entries to execute the tape function. a. Execute command. b. Execute YPC. c. Repeat command if YPC indicates an alert halt.
WST	Set tape, plug, read or write, buffer address, and number of words where needed for WDT, WRL, WBR, WRT, WRO, WTM, YIN, and ULM. This routine and the above seven entries are used for tape movement other than normal read/write, such as label read or write, error recovery, multifile positioning, rewinding, and priming a buffered input file.
WXI	Used as exit for the above nine routines.
(P)WT	Error check and recovery for plug P. The file table address for the last tape used for a normal read or write is stored in (P)UB. Before any other tape action can take place, a pass thru this routine is necessary. a. Check (P)UB for zero, if zero exit. b. If non-zero: 1. Execute YPC. 2. Repeat command if YPC indicates an alert halt. 3. Check for any error, if yes go to 1TX to recover. 4. If input, check for end-of-file mark and set end-of-file indicator in the file table. 5. If output, check for end-of-reel and set indicator in the file table.
1TX	Error recovery. If a two tape plug system, error recovery for second plug starts at 2TX. Each of these is a switch. When an error is detected, the switch is set to go to TRY and transfers control to TX9.
TX9	Error Recovery Initialization. a. Set exit to error check (PWT). b. Execute WST. c. Set error counters. d. If two controllers, set plug number in test instructions. e. Execute TRY. f. If error is corrected, check for end-of-reel detected and set end-of-reel indicator in the file table; then exit.
TRY	Error Recovery Instructions. a. Decrease error counter by one.

Figure 35. (Cont.)

Name	Description
	b. If error counter is nonzero: 1. Check for end of reel and set indicator. 2. Backspace the tape (WBR). 3. Reread or rewrite the tape block. 4. Re-enter (P)WT at 1WK (2WK for second controller). The routine will return by the 1TX switch if still in error. c. If error counter zero: 1. Input a) Type error message. b) Wait for operator action. c) If operator sets switch 19 down, reset error counter and go to TRY. d) If operator does not set switch 19, set error indicator in the file table and exit. 2. Output a) Backspace the tape. b) Write two tape marks if non-pucker pocket, one if pucker pocket. c) Check for error on writing tape mark. If any error skip note e4. d) If skip counter non-zero: 1) Backspace the tape. (This should skip 3 inches of tape.) 2) Set error counter. Go to repeat five more times. e) If unrecoverable error on output: 1) Type error message. 2) Wait for operator decision. 3) If operator sets switch 19, it is assumed that another tape has been mounted, and exit is made to repeat write. 4) If operator does not set 19, it is assumed that he wishes to try again on the same tape.
TCR	Write Rerun Dump a. On end of reel: 1. Set plug number if two controllers. 2. Set tape number. 3. If two controllers, wait for both to be ready. 4. Wait for card reader ready. 5. Update rerun count. 6. Type rerun number. 7. Write memory in one record. 8. If it cannot be written correctly in five trys, type NO after rerun number and exit. b. On separate rerun tape: 1. Do 3 thru 6 above. 2. Write rerun label. 3. Write rerun dump. 4. If error, type NO and repeat the write until a good dump is executed; then exit.

Figure 35. (Cont.)

Name	Description
OCL	Output Tape File Close a. Set write switch (fnW) to SPB TER, 2 and save exit to procedure coding (uses TSI). b. Check for file closed. If closed, type an error message. Set close indicator (uses TZQ). c. Check lock indicator of call and set to lock if indicated. d. If two controller system, set error bucket address (PUB) and error check address (PWT) in control routine (TWR). e. If a blocked file go to write last block after setting return in XWT. f. Error check last record. g. Write an end-of-file mark and go to OTC.
OTC	Output Ending Label Create a. Executes TY2. b. If labeled file: 1. Type beginning label information. 2. Execute TBP for label area. 3. Create ending tape or file label. 4. If end-of-file: a) Execute after end-of-file label section, if any (uses TAS). b) Execute end-of-file label packing for non-standard fields, if any (users TAS). 5. If end-of-tape: a) Execute after end-of-tape label section, if any (uses TAS). b) Execute end-of-tape label packing for non-standard fields, if any (uses TAS). 6. Execute OLW (Output Label Write). c. If nonlabel file, execute TY3. d. Go to OTR.
OTR	Output end-of-tape or -file close out. a. If end-of-file: 1. Rewind tape, if indicated. 2. Return to exit. b. If end-of-tape: 1. Execute TCR if rerun indicated. 2. Rewind tape. 3. Execute WSP if multiple tape units assigned to this file. Otherwise, wait for operator to mount blank tape on this handler. 4. Go to OLC.

Figure 35. (Cont.)

Name	Description
OLC	Output beginning label create. a. Update reel number through UCV. b. If labeled file: 1. Execute TBP for label area. 2. Create beginning tape or file label. 3. If a multifile tape: a) Execute after beginning file label section, if any (uses TAS). b) Execute beginning file label packing for nonstandard fields, if any (uses TAS). 4. If other than multifile: a) Execute after beginning tape label section, if any (uses TAS). b) Execute beginning tape label packing for nonstandard fields, if any (uses TAS). 5. Execute OLW. c. If executed from open go to exit. d. If executed from write, return to TWR routine.
OLW	Output Label Write a. Execute LRR. b. If label write error: 1. Type error message. 2. Operator action: a) Skip and try again, or b) Skip label write.
UFP	Multifile input tape position. a. Check open/close multifile indicator. b. If a file on this multifile is open: 1. Type error message. 2. Ask operator if it is desired to proceed. c. Position tape to file indicated by file table.
ICL	Input tape file close. a. Set exit to procedure coding (uses TSI). b. Check open/close indicator for open (uses TZQ). c. If closed: 1. Type error message. 2. Ask operator if it is desired to proceed. d. Set lock/no lock indicator as desired. e. If not a file or multifile tape: 1. Rewind if indicated 2. Go to exit (uses TSF). f. If a file of a multifile tape: 1. Rewind if indicated or position to next file if no rewind. 2. Set multifile open/close indicator to closed. 3. Update multifile position indicator. 4. Go to exit (TSF).

Figure 35. (Cont.)

Name	Description
ILC	Input beginning label check. a. Update reel number in file table. b. Execute TY2. c. If labeled file: 1. Execute LRR. 2. If label read error: a) Type error message. b) Operator action: 1) Repeat read, or 2) Skip label check. 3. If no label read error: a) Execute ULM. b) Type label. c) Check label information against file table information. 1) If label information error: -Type error message. -Type file table information. -Ask operator if it is desired to mount correct tape, or accept label as is and move label information to file table. 2) Execute TY3. 4. Execute beginning label unpacking for non-standard fields, if any (uses TAS). 5. Execute after beginning label section, if any (uses TAS). d. If buffered file, read first block. e. If executed from OPN, go to exit (TFS). f. If executed from TWR, return to execute read thru TSO.
ITC	Input ending tape or file label check. a. If labeled file: 1. Execute LRR. a) If label read error (go to VER). 1) Type error message. 2) Ask operator if it is end-of-tape or end-of-file. b) If no label read error: 1) Execute ULM. 2) Check record count. 3) Check block count. 4) If count error type error message (at T X 2), and ask operator if it is desired to continue. 2. If end of tape: a) Execute TCR if rerun indicated. b) Rewind tape. c) Execute end-of-tape label unpacking for non-standard fields, if any (uses TAS).

Figure 35. (Cont.)

Name	Description
	d) Execute after end-of-tape label section, if any (uses TAS). e) If multiple tape units indicated, execute WSP. f) If one tape unit only, type MT and wait for operator to signal to continue. g) Go to ILC. b. If end-of-file (nonlabeled files always go to end-of-file transfer address.) 1. Execute end-of-file label unpacking for non-standard fields, if any (uses TAS). 2. Execute after end-of-file label section, if any (uses TAS). 3. Set end-of-file indicator in file table. 4. Go to end-of-file transfer address set by procedure coding in word 17 of file table.
LRR	Label read/write a. Execute WST. b. Execute WRL. c. Check for tape error. 1) If no error, return. 2) If error, repeat up to five times. 3) If error persists after five attempts, go to return plus one.
TAS	Label After Section or label pack/unpack execute. a. If execute address equal zero, return. b. If execute address not equal to zero, execute section or pack/unpack then return.
TSI	Set exit a. Set index register one to indicate first location of file table of file beging operated on. b. Set exit in file table.
OPT	Optional file a. Type OPT. b. Type the first word of the file table. c. Set end-of-file indicator in the file table. d. Set end-of-file transfer address into read entrance. e. Ask operator if this file present. 1) If not present a) Type FD. b) Return to main program. 2) If present a) Execute TY3. b) Go to open this file.

Figure 35. (Cont.)

Name	Description
TWR	Input/Output Control In each file table there is a seven word branch table which is used to control the path of the program through this routine once the initial conditions are set. This method is used to eliminate as many tests for file conditions as possible. The initial conditions which are set are as follows: a. Store record length and packing address for an output file. b. Set exit. c. If more than one plug, set error check address and bucket address. The actual read or write is executed in the file table starting at word 21 and control is returned to this routine at word 24. The individual routines as described below are entered from one of the following: a. In line. b. From the branch table. c. From one of the other service routines. d. From one of the other sections of TWR. e. From the file table.
TSO	a. Execute error delay and check on previous READ/WRITE. b. Set file table location in the error check bucket.
TS1	a. Test for end-of-tape. b. Test for tape error on input.
TS2	a. Execute delay and error check. b. Update block count.
TBC	Update block count.
TS3	Change buffer areas.
TSZ	Update record count.
TS4	Change end of block address.
TS5	a. Update rerun counter. b. Execute rerun if counter is equal to rerun count of file table.
TS7	a. Test for end-of-block. b. Update block process address.

Figure 35. (Cont.)

Name	Description
TCC TS8	Update record count a. Test for block overflow, if not go to TSG. b. Test for full block. c. Store end of block sentinel, if not a full block. d. Go to TS0.
TS9	Execute TBK for output area.
TSA	a. Update output block process address. b. Update intermediate record count. c. Update record count. d. If Process File, go to TS8
TEC	Execute input error procedure.
TSG	Transfer control to packing
TDS	Execute delay and error check routine (for Process Files).
TSF	Exit to procedure coding
TSB	End-of-tape test for input and go to ITC or output go to OTC.
TRN	Used by the control key analysis section (from ZCK) to type message if control key error is detected.

Figure 35. (Cont.)

Name	Description
fnZCK	Start of control key analysis. Unpack control key field and compare it to the literals. When a match is found, branch to fnFND.
fnFND	Control key match found. a. If records are of different length, the length of the record found is picked up from fmCLL and stored in fnTLL. b. If no UNTILs on this file, control is transferred to the appropriate record unpacking (which is obtained from fnREC table). c. If any UNTILs,examines contents of SRU. 1) If SRU is zero, go to unpack. 2) If SRU is nonzero, store unpacking transfer at fn$Z and branch to SRU.
fnCLL	Table of record lengths.
fnREC	Table of record numbers.
fnUPA	Table of record umpacking addresses.
fnTAB	Table of addresses for the literals to which the control key field is compared.
fnZ	Location into which the matching record's record is stored.

Figure 36. Control Key Analysis

Name	Description
fn#	Entrance for first read until statement if input file.
fn@	Entrance for second read until statement.
fn*	Entrance for third read until statement.
fn%	Entrance for fourth read until statement.
fnUTL	Created only if more than one until statement on this file a. Initializes read until switch (SRU). b. If copy, initializes the transfer to the copy file. c. Go to the read entrance.
fn/	Unsatisfied return from procedure test of until field. a. If copy, executes copy function by going to entrance at fn#. b. Go to read next record.
fn$	Satisfied return from procedure tests of until field. a. Reset read until switch (SRU) to zero. b. Go to unpacking.
fn%Z	Unpacking transfer switch for read until. Set by control key analysis.
fn#	Entrance for copy function if output file. Save record length and input buffer pointer.
fn#UP	Unpacking of until field for first read until statement.
fn@UP	Unpacking of until field for second read until statement.
fn*UP	Unpacking of until field for third read until statement.
fn%UP	Unpacking of until field for fourth read until statement.
SRU	Read until switch. a. If zero, no READ UNTIL is in process. b. If non-zero, contains branch to until field unpacking.
fn#CP	Temporary storage for input file record length and input area address when there is a copy function on this file.

Figure 37. Read Until or Read Copy Until

Name	Description
RDY	Entrance from procedure coding to DSU I/O for ready statement on input files. a. Execute STX to store exit and setup call to MIO subroutine. b. Check for overriding ready and type error message if yes. c. Get DSU record address from using field and store in file table. d. If more than one DSU used get DSU number from unit field and store in file table. e. Set ready given indicator. f. Set first read or write indicator (used on blocked files). g. If blocked or buffered file, set beginning of buffer pointer. h. Go to open/close switch for next function (word 29 of File Table).
ROT	Entrance to control routine from word 29 of output file table. a. If sequential file go to EXT to return to procedure section. b. If non-sequential file store the indication of whether or not the seek is given on the ready. 1. If seek is indicated by the calling sequence execute RRW to stack the seek in MIO DSU command table. 2. Return to procedure section.
RIT	Entrance to control routine from word 29 of input file table. a. Execute RRW to stack the seek in MIO DSU command table. If calling sequence from procedure section specifies "ready for reading" the read will also be stacked. b. Return to procedure section.
FXT	Return control to procedure section.
STX	Control initialization a. Save exit to procedure section in word 19 of the file table. b. Set index 1 to file table address. c. Setup calling sequence to MIL.
RRW	Execute call to MIO a. Execute call to MIO b. Restore file table address in index 1. c. Check for error return and if so, go to type error message.
RDS	Entrance from procedure section to DSU I/O for ready statements on output files. a. If nonsequential file go to RDY. b. If sequential file check for last operation complete. 1. If completed go to RDY.

Figure 38. DSU Files

Name	Description
RDS (Cont.)	2. If not completed execute TJT if any journal tape and execute demand call to MIO. 3. Go to RDY.
UDA	Update DSU record address for sequential files.
RFD	Entrance to DSU control to execute read or write. a. Execute STX. b. Check for ready given and if not, type error message.
SDM	Reset ready given indicator for non-sequential files. a. If sequential blocked file and read completed go to TEB to start unblocking. b. If sequential blocked file and read not completed go to MIO to complete the read and set the buffer pointer.
SDD	Execute demand of input and set the buffer pointer
TEB	Unblocking for blocked files.
SSW	Set seek started indicator for not blocked buffered sequential file.
SRD	Update DSU address and start seek/read.
TF2	End of block action on input files. a. If nonsequential go to "if end of block" statement. b. If sequential and cross buffer sharing, release the output buffer and write back on the file. c. If this file is assigned to a journal tape go to error check TJT.
TFS	Reset ready indicator on non-sequential files.
RFL	Entrance for release statement to release an output buffer to be written onto the DSU.
TRL	Check if write statement indicated a release also.
DSW	Set up a seek and write call to MIO.
TBX	Set buffer pointer.
TRG	Check if preset of buffer is required. Execute TBK if yes and go to packing routines

Figure 38. (Cont.)

Name	Description
RSG	If 16k program, used to go to packing routine.
TRP TIP TAP	Blocking and end of block test for output file.
TOP	Indicate buffer sharing on blocked sequential output.
TFD TFP	End of block action on sequential output files.
TDM	Used for blocked sequential input with cross buffer sharing.
WSQ	Used to control action of not-buffered not-blocked sequential output files.
WTQ	Used to control action of buffered not-blocked sequential output.
DRW	To write not-blocked sequential output.
TXA	Reset buffer pointer for blocked sequential output.
DFM	Check write complete and update DSU record address for sequential output.
RUT	Used to perform copy function on DSU files.
WCP	Used to perform copy function on DSU files.
TRU	Entrance for until-files to set until-switch on.
/CU	Entrance for unsatisfied UNTIL to get next record.
$CS	Entrance for satisfied UNTIL to set until switch off and go to unpack this record.
$UP	Unpacking address of record on until-statement being examined.
ROP	DSU open file entrance. a. Check open/close indicator. b. Type error message if already open. c. Set open switch in word 29 of the file table. d. Return to procedure coding.

Figure 38. (Cont.)

Name	Description
RCL	DSU close file entrance. a. Check open/close indicator. b. Type error message if already closed. c. Set close switch in word 29 of the file table. d. Check for any action to be completed on records in buffers. e. Return to procedure coding.
RR1	Type RFR for instruction sequence error discovered on ready.
RR2	Type CFS for instruction sequence error discovered on read/write.
RR3	DSU read or write error.
RR4	Type OCE for open/close error.
RR5	Type FNO for file not open message.
WJT	Write journal tape. 1. Check to see if this file is to be written on journal tape, unless all DSU output is to be on journal tape. 2. If there are no other tape files present, coding will be provided to perform delay and error check and writing on the journal tape. Otherwise, these functions will be performed by utilizing the I/O coding already present.
TJT	Test journal tape. 1. Check to see if this file is to be written on journal tape, unless all DSU output is to be on journal tape. 2. Check status of journal tape.
JTX	Restore indexes and exit.
JTW	Coding to perform journal tape write, delay and error check, etc. Not generated if there are other tape files besides journal tape.
JTE	Journal tape error. 1. Attempt to recover from error on journal tape.
WJR	Write journal tape record. 1. Update record count. 2. Rewind journal tape using JTZ 3. Write labels, if present, using JTZ. 4. Write EOF using JTZ.

Figure 38. (Cont.)

Name	Description
JTZ	Used by WJR.
JTL	Journal tape label and type-outs. 1. Error type-out. 2. Record count. 3. Journal tape label.
JTC	Journal tape commands. 1. Rewind. 2. Backspace. 3. Write end-of-file. 4. Write journal tape record file number of DSU file. 5. Write label record.
RET	DSU error table. 1. RFR - Ready followed by ready. 2. CFS - Command followed by same command. 3. MRE - DSU error. 4. OCE - Open/Close error. 5. FNO - File not open.

Figure 38. (Cont.)

APPENDIX G
FILE TABLES

File tables are the communication link between GECOM's I/O service routine and other I/O coding. I/O service routines, which eliminate the necessity of in-line coding, perform the following functions:

Opening and closing input and output files
Checking input and output file labels
Creating and writing output labels
Setting optional file indicators
Rewinding output tapes
Checking for controller errors
Positioning multifile input tapes
Swapping tapes.

File tables are built in lower memory in accordance with the relocation constant for the program. Two types of file tables are created. Tables for input files are listed first, and may be up to 48 words long. Output file tables follow and may be up to 44 words long. When a file table is set up for the first file of a multifile tape, the last two words of the file table are used for information pertinent to multifile tapes. Otherwise, there is no need for these words, and the file table is two words shorter.

Card reader, card punch and printer file tables are created using eight words. These eight words contain the same information as words 0-7 of the tape file tables. Figures 39 thru 43 show the format for input and output file tables.

Each file is given a symbolic name: Three BCD characters consisting of a 2-digit number (represented as fn in the diagrams) followed by S. The first file is 00S, the second 01S, etc. This file name, fnS, appears in word 0, bits 1-19, of every file table. The sign bit is the EOF indicator. (See Figures 39 and 42).

Word 1 of the table indicates the length of the table which is always an even number.

Words 2 and 3 contain read commands (bits S-4) which contain the address of incoming data (bits 5-19). If the file is not buffered, word 2 will always equal word 3. Word 3 always shows the location into which data was last read.

Words 4 and 5 contain indicators only.

Words 6 and 7 contain the block count and record count of the tape (not the file).

Word 8 contains the address of the starting location of the delay and error check service routine for the tape plug to which the file is assigned. This word is only used if the two tape plugs are specified.

Word 9 contains the address of the location used by the delay and error check service routine to obtain the file table starting location of the last file operated on for the tape plug to which the file is assigned. This is only used if two tape plugs are specified.

Word 10 contains a BCD reel or file number.

Bits 2-19, of words 11, 12 and 13, contain the LABEL ID.

fnCD is the address of the creation date. Words 14 and 15 are reserved for the actual date. This date is taken from the first label of the input tape, or from locations 1076_8 and 1077_8 for an output file if dating symbols are not specified.

Word 16 is used only when the file is part of a multifile tape or when more than one tape unit is assigned to this file. If part of a multifile tape, word 16 contains the position (in BCD) of this file on the tape. If more than one tape unit is assigned, bits 0-4 contain the tape code of the starting tape.

fnT, the file number followed by T to indicate control section, at word 17 contains the address to which control reverts when the **EOF** is encountered on input. Word 17 on output is used as an intermediate record count for blocked files.

fnTCP, word 18, contains the address of the record which is currently being processed. It will be assigned an "EQO;" consequently, all buffer areas (except card) will now be absolute rather than relocatable and will be located in Common Storage.

fnTXT, word 19, contains the exit location from the READ, WRITE, OPEN or CLOSE sentence.

fnTLL contains the length of the record being processed if the table is an input file table. In an output file table, this word (20) contains a branch into the I/O control section from the packing section.

Words 21-23 contain the general select instruction for this file followed by the READ or WRITE command which was last executed.

Word 24 contains a branch to the I/O control section which is executed after a READ or WRITE sentence.

Word 25 is used only when more than one tape is assigned to this file. In this case, it contains the tape codes for each tape unit. (No more than four tapes are allowed for each file.)

fnTEB, words 26 and 27, are present only when records are blocked. They contain the address of the end of the block for each input area. These are zero when blocking is not used. Word 27 is zero if record is blocked but not buffered.

Words 28 through 30 in input file tables and word 44 in output file tables are addresses used to execute USE AFTER LABEL sections.

Words 31-37 control branches through the I/O control section.

Word 38 contains the end of block literal, which is $%$ for BCD, 3777777 for binary and 0777777 for special binary. On output files 0401700 is used for off line print files.

Word 39 contains the number of records between rerun points.

Word 40 contains the record counter for rerun.

Words 42-43 show the end-of-tape and end-of-file label unpacking or packing addresses for fields on the label that are referenced by the user.

Word 44 contains the address of the AFTER STANDARD ERROR PROCEDURE section, if any.

Words 45-47 contain multifile tape information. If this is not a file of a multifile tape, word 45 is zero; otherwise, this word gives the address of the location containing the current tape position. Words 46 and 47 are present only on the first file table produced for a given multifile tape. All files described as multifile and assigned to the same tape handler are considered part of the same multifile system.

Word	Symbol	Description
0	fnS	Sign bit = EOF indicator 3 BCD file name (file no. plus S)
1		Length of this file table (number of words used)
2		Read command plus address of input area 1
3		Read command plus address of input area 2 (if not buffered, area 1)
4		Sign bit = open/close indicator;on=closed,off=open
5		Not used
6		Record count
7		Not used

Figure 39. File Table Format for Input Card Files

Word	Symbol	Description
0	fnS	Three BCD file name (file no. plus S)
1		Bit one on = preblank the output area before packing length of this file table (number of words used)
2		Punch command plus address of output area 1
3		Punch command plus address of output area 2 (if not buffered, area 1)
4		Sign bit = open/close indicator;on=closed,off=open
5		Not used
6		Record count
7		Not used

Figure 40. File Table Format for Output Card Files

Word	Symbol	Description
0	fnS	Three BCD file name (file no. plus S)
1		Bit 1 on = preblank the output area before packing length of this file table (number of words used)
2		Second word of print command for area 1
3		Second word of print command for area 2 (if not buffered, same as 2)
4		Sign bit = open/close indicator;on=closed,off=open
5		Not used
6		Record count
7		Advance counter

Figure 41. File Table Format for Printer Files

Word	Symbol	Description
0	fn.S	Sign bit = EOF indicator, 3 BCD file name (file no. plus #S#)
1		Length of this file table (number of words used)
2		Tape command and input area 1
3		Tape command and input area 2 (if not buffered, same as word 2)
4		Sign bit = open/close indicator; on=closed, off=open Bit 19 = lock/no lock indicator; on=not locked, off=locked
5		Sign bit = unrecoverable tape error if bit on Bit 19 = end of tape characters encounterd if bit on
6		Block count
7		Record count
8		Sign bit on = rerun at end of reel Address of delay and error check routine for this plug
9		Sign bit on = this file is buffered Address of delay and error check file table bucket
10		BCD reel number if not multifile BCD file number if multifile
11		Sign bit on = no labels Literal ID word 1 (3 BCD characters)
12		Literal ID word 2 (3 BCD characters)
13		Literal ID word 3 (3 BCD characters)
14	fnCD	Created date word 1
15		Created date word 2
16		Bits 0 thru 4 = tape code of 1st tape assigned to this file. Position of this file in BCD if multifile
17	fnT	End of file transfer address
18	fnTCP	Current record address
19	fnTXT	Exit to procedure coding
20	fnTLL	Length of current record
21		General select command
22		Words 21, 22, and 23 are actual tape command sequence
23	fnTSB	Executed to read the data block into memory
24		Branch to control routine after executing tape command
		Bits 0 thru 4 = first tape code (used for tape swapping)
25		Bits 5 thru 9 = second tape code (used for tape swapping) Bits 10 thru 14 = third tape code (used for tape swapping) Bits 15 thru 19 = fourth tape code (used for tape swapping)
26	fnTEB	End of block address for area 1 (if blocked file).
27		End of block address for area 2 (if not buffered, zero)
28		Address of beginning tape or file label section
29		Address of end-of-tape label section
30		Address of end-of-file label section Sign bit = I/O indicator; on=output file, off=input file

Figure 42. File Table Format for Input Tape Files

Word	Symbol	Description
31 32 33		Branch table used to direct the execution of the control routine for this particular type of input file. Includes words 31 thru 37.
38		End of block literal $%$ for BCD, all bits on for binary bits 2 thru 19 for special binary
39		Number of records between rerun points
40		Rerun record counter
41		Beginning label unpacking address
42		End-of-tape label unpacking address
43		End-of-file label unpacking address
44		Address of # after standard error procedure #section (Sign on = NO "after standard error procedure" section.)
45		Zero if not multifile If part of a multifile tape, this word will contain the address of the multifile position counter.
46		Multifile position counter
47		Sign bit = open/close indicator of multifile tape; 0=open, 1=closed Note--word 46 and 47 will be present only on the first file table produced for a given multifile tape. All files described as multifile and assigned to the same tape unit are considered part of the same multifile system.

Figure 42. (Cont.)

Word	Symbol	Description
0 S		3 BCD file name (file no. plus S)
1		Length of this file table (number of words used)
2		Tape command and output area 1
3		Tape command and output area 2 (If not buffered, same as 2)
4		Sign bit - = open/close indicator; on=closed, off= locked. Bit 19 = lock/no lock indicator; on= not,locked off= locked
5		Bit 19 on = end-of-tape encountered
6		Block count
7		Record count
8		Sign bit on = rerun at end-of-reel Address of delay and error check routine for this plug
9		Sign bit on = this file is buffered Address of delay and error check file table bucket
10		BCD reel number if not multifile BCD file number if multifile
11		Sign bit on = no labels Literal ID word 1 (3 BCD characters)
12		Literal ID word 2 (3 BCD characters)
13		Literal ID word 3 (3 BCD characters)
14	fnCD	Creation date word 1
15		Creation date word 2
16		Bits 0 thru 4 = tape code of first tape assigned to this file. Position of this file if multifile (in BCD)
17		Intermediate record count for buffered blocked files
18	fnTCP	Current record address
19	fnTXT	Exit to procedure coding
20	fnWE	Entrance from packing to tape control routine
21		General select command
22		Words 21, 22, and 23 are the actual tape command sequence
23	fnTS3	Executed to write the data blocks on tape.
24		Branch to control routine after executing tape command
25		Bits 0 thru 4 = first tape code (used for tape swapping) Bits 5 thru 9 = second tape code (used for tape swapping) Bits 10 thru 14 = third tape code (used for tape swapping) Bits 15 thru 19 = fourth tape code (used for tape swapping)
26	fnTEB	End of block address for area 1 (if blocked file)
27		End of block address for area 2 (if not buffered, same as 26)
28		Sign Bit = present/no set indicator; on=present off=no set. Bit 1 on = present output area to blanks. Bit 1 off = present output area to zeros. Address of beginning tape or file label section
29		Address of end-of-tape label section (Sign bit on = Process File.)
30		Address of end-of-file label section Sign bit on = this is an output file table

Figure 43. File Table Format for Output Tape Files

Word	Symbol	
31 32 33		Branch table used to direct the execution of the control routine for this particular output file. Includes words 31 thru 37
38		End of block literal $%$ for BCD, all bits for binary bits 2 thru 19 for special binary, 040 1700 for off-line print
39		Number of records between rerun points
40		Rerun record counter
41		Beginning label packing address
42		End tape label packing address
43		End-of-file label packing address

Figure 43. (Cont.)

Word	Symbol	Description
0	frS	3 BCD file name (file no. plus S).
1		Seek status indicator.
2		Read status indicator.
3		Buffer availability indicator.
4		Error counter.
5		DSU plug number.
6		DSU unit number.
7		Input/output indicator. 0=input
8		Number of frames.
9		Power mode indicator 0=on 1=off
10		Not used on input files, must be zero.
11		DSU record address.
12		Input area 1.
13		Input area 2 (same as 12, if not buffered).
14		Priority indicator (always zero for GECOM).
15		Used by MIO.
16		Sign Bit=ready given indicator (used to check sequence of operations). Bit 19=open/close indicator 0=open 1=close.
17		Bit 18 on=non-buffered sequential file. Bit 19 on=sequential file.
18	frTCP	Current record address.
19	frTXT	Exit to procedure coding.
20	frTLL	Length of current record.
21		Exit of procedure coding if end of block.
22		Address of DSU error section.
23		Word position of chaining address relative to beginning of record, if present.
24		Address of field containing DSU address.
25		Address of field containing unit number.
26		End of input area 2.
27		End of input area 1 (same as 26 if not buffered).
28		Exit to procedure coding if end of chain.
29		Open/close switch.
30	}	Branch table used to direct the execution of the control routine for this particular input file.
31	}	
32	}	
33	}	

Figure 44. File Table Format for Input DSU Files

Word	Symbol	Description
0	fnS	3 BCD file name (file no. plus S).
1		Seek status indicator.
2		Write status indicator.
3		Buffer availability indicator.
4		Error Counter.
5		DSU plug number.
6		DSU unit number.
7		Input/output indicator 1=output.
8		Number of frames.
9		Power mode indicator. 0=on 1=off
10		Read after write indicator.
11		DSU record address.
12		Output area 1.
13		Output area 2 (same as 12 if not buffered)
14		Priority indicator (always zero for GECOM)
15		Used by MIO.
16		Sign bit = ready given indicator (used to check proper sequence of operations) Bit 19 = open/close indicator. 0=open 1=closed
17		Sign on = write on journal tape. Bit 17 on=blocked sequential process file or not block not buffered sequential process file.
18	fnTCP	Current output area address.
19	fnTXT	Exit to procedure coding.
20		End of block sentinel if any needed.
21	fnWE	Entry to DSU control coding from the packing output coding.
22		After error section name if any.
23		Sign bit used for demand indicator.
		Largest word size less 1.
24		Address of field containing DSU record address.
25		Address of field containing unit number if more than one unit.
26		End of area 2 (same as 27 if not buffered).
27		End of area 1.
28		Sign Bit = Set/no set. 1=set 0=no set Bit 1 = preblank/prezero. 1=blanks 0=zeros
29		Open/close switch.
30 31 32		Branch table used to direct the execution of the control routine for this particular output file.
33		Buffer length plus 2 (used to write journal tape if required.)

Figure 45. File Table Format for Output DSU Files

APPENDIX H
OBJECT PROGRAMS FOR 16K MEMORIES

A GE-200 Series 16,384 word memory consists of four modules of 4,096 words each. However, the total memory may be considered as consisting of two banks of 8,192 words each.

To access data or constants stored in upper 8k, GE-200 Series instructions require the use of an index register. To process information in lower 8k, an index register is not required. Branch instructions from and to upper 8k locations require no special provisions, and a string of instructions without intervening branches behaves the same way in upper 8k as in lower 8k. Thus, lower 8k is the more natural part of memory to store data and constants, while upper 8k is better suited to store program instructions. Because arrays of data are processed using an index register, they could also be conveniently stored in upper 8k.

Subroutines may contain their own constants and working storage areas. If they were located in upper 8k, they would require indexing to refer to such information, but this is unnecessary in lower 8k. Therefore, fewer instructions need be stored and executed when subroutines are located in lower 8k.

UTILIZATION OF MEMORY IN GECOM 16K OBJECT PROGRAMS

In accordance with the above, common constants, file tables, Common Storage, Working Storage, and subroutines are always assigned to lower 8k locations in GECOM object programs. However, the compiler allows the programmer to indicate that the body of coding for the segment to be compiled (see "Segments" in Chapter 6 of this manual) and/or certain repeated numeric fields are to be assigned to upper 8k. To provide these options, the following features have been added to the source language:

1. *COMMON STORAGE Section of the Data Division. (See Chapter 5.)

2. BEGIN *COMMON-STORAGE AT xxxxx clause in the Environment Division. (See OBJECT-COMPUTER sentence in Chapter 7.)

3. PLACE SEGMENT IN UPPER MEMORY clause in the Environment Division. (See OBJECT-COMPUTER sentence in Chapter 7.)

The *Common-Storage Section allows for the description of repeated numeric fields which are to be stored in upper 8k. Ordinarily, memory addresses for *Common-Storage fields are assigned in descending order from 8190 of upper 8k. If the BEGIN *COMMON-STORAGE clause is used, the address specified in that clause will be used instead as the upper limit of *Common Storage. If the *Common-Storage Section is used in more than one segment of a program, the fields must be described identically and in the same sequence in each segment. In this case, if the BEGIN *COMMON-STORAGE clause is used in one segment, it must be used in all segments, and the beginning address must be the same in all clauses.

The PLACE SEGMENT clause directs the compiler to assign the body of object coding for the segment to upper 8k. As indicated previously, this does not include common constants, file tables, Common Storage, Working Storage, or subroutines which are always assigned to lower 8k. If the PLACE SEGMENT clause is not used, the compiler assumes that the entire segment (except *Common Storage, if any) is to be assigned to lower 8k.

HARDWARE REQUIREMENTS

In addition to the four modules of memory, a GECOM 16k object program assumes the presence of the normally optional extra index groups. The object program vector linkage assumes complete use of index group 1. In addition, compiled object coding also makes use of these locations, and their contents should not be changed if and when the user enters sections of General Assembly Program coding.

EXAMPLE OF MEMORY ALLOCATION IN A GECOM 16K OBJECT PROGRAM

In the following example it is assumed that the total program consists of a main segment plus five other segments:

1. The main segment is to be placed in upper 8k. It employs a magnetic tape input file and a magnetic tape output file.

2. Segment~1 is to be placed in upper 8k. It employs a card input file and references subroutine~1 and subroutine~2.

3. Segment~2 is to be placed in upper 8k. It employs a card output file.

4. Segment~3 is to be placed in lower 8k.

5. Segment~4 is to be placed in upper 8k. It employs a printer file and references subroutine~3 and subroutine~4.

6. Segment~5 is to be placed in upper 8k.

7. *Common Storage consists of numeric fields A, B, and C each of which is repeated 500 times. Note that 1000 words of memory are required for each repeated field because two words are assigned to each field.

Figure 41 shows the memory allocation for the above described example.

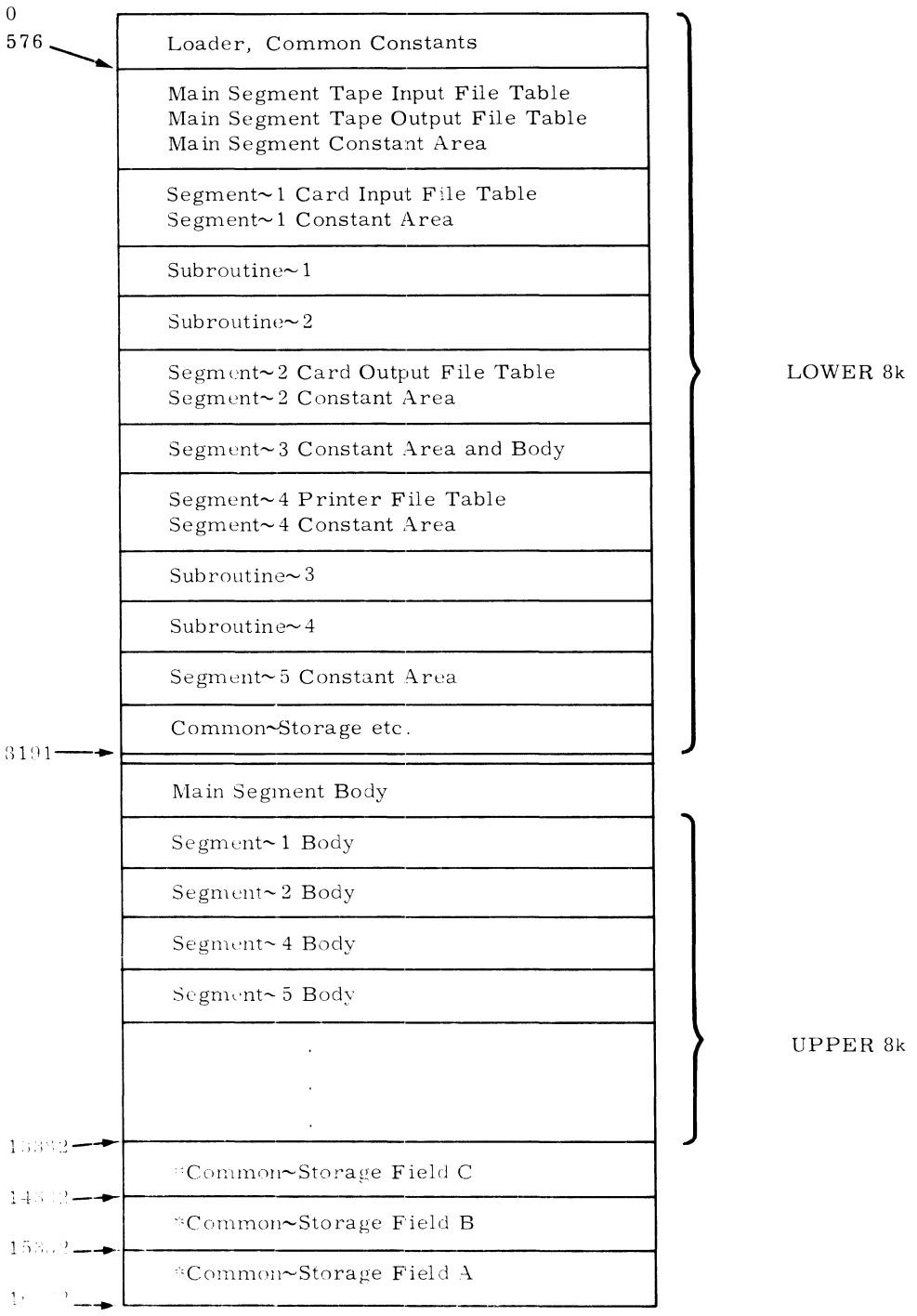


Figure 46. Example of 16k Memory Allocation

APPENDIX I
GECOM RELOCATABLE OBJECT PROGRAMS

INTRODUCTION

This appendix was originally prepared for programmers writing GE-200 Series relocatable routines to be assembled by the General Assembly Program II. Most of the original content was also applicable to GECOM relocatable object programs. The document has been edited to be somewhat more specific to GECOM object programs and especially to reflect the use of the MCML II Loader (CD225B1.006R) and *Common Storage in GECOM object programs.

Thus the following text reads as if the programmer were hand-coding routines for assembly by General Assembly Program II in relocatable format for loading by MCML II. The text refers to an MCML Header Card Writer routine (CD225B6.003R) which the programmer would need to use if hand-coding his routines. However, the reader should keep in mind that GECOM automatically produces General Assembly Program symbolic coding, assembles it into relocatable format, and supplies the necessary header cards to be utilized by MCML II in the loading process.

It is felt that this appendix will aid the GECOM user in obtaining a better understanding of the method by which GECOM object programs are relocated in either or both banks of memory and of the concept of establishing linkage between segments and subroutines, etc.

Relocatable routines are assembled programs, subroutines, etc. that are coded to operate in any assigned area of memory without being reassembled for execution in specifically assigned memory locations. A major advantage of using relocatable routines is that once a commonly used subroutine has been assembled in relocatable form it is available for use in other programs without paying the price of reassembly each time the subroutine is needed in a new program.

Relocatable programs also provide a great deal of flexibility. The user may organize a large program into many small independent subroutines. Each relocatable subroutine may be assembled independently of the main routine. Changes to a relocatable subroutine may be made by correcting symbolic coding and reassembling only the subroutine found to be in error. If the corrections change the size or the storage allocation of this relocatable subroutine, storage assignments in the main program with which the subroutine is used will not be affected. Conversely, independent changes in the size and storage assignments of a relocatable main program do not affect the use of the relocatable subroutines called upon by the main program.

Conflict in the use of common symbolic names in two or more subroutines is eliminated when the subroutines are assembled independently in relocatable form. The assembly operation converts all symbolic addresses to relative machine addresses. In this way there can be no duplicate symbols when two or more relocatable routines are tied together into a common program.

The Multicapability Modular Loader (MCML II) CD225B1.006R, in addition to the MCML Header Card Writer (CD225B6.003R) needed in hand-coding routines are referenced. Other helpful information may be found in Appendix C, Object Program Relocatable Deck Formats, and Appendix H, Object Programs for 16k Memories, of this manual. In addition the reader should refer to the GECOM Operations Manual, Chapter 6, Object Program Operating Instructions.

PREPARING HAND-CODED ROUTINES FOR LOADING WITH MCML II

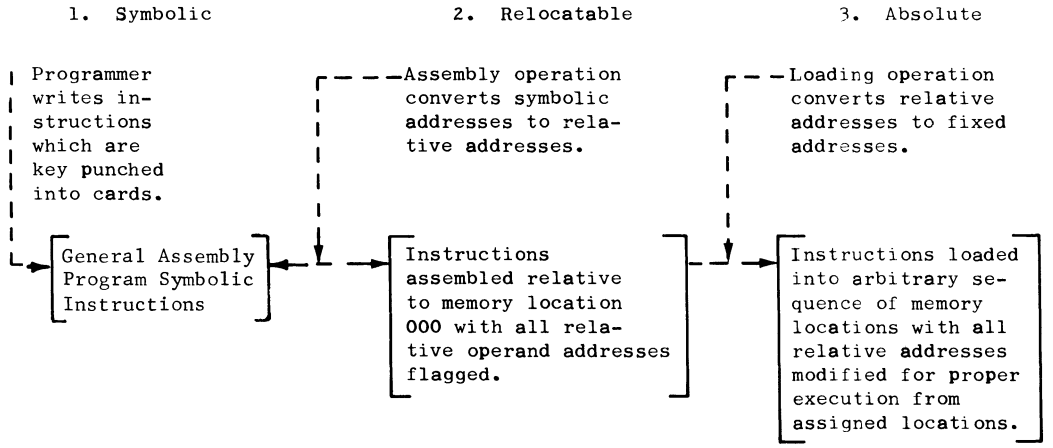
Coded Instructions

Much the same method is used in coding a routine in General Assembly Program symbolic language for assembly in relocatable form as is used in coding for assembly in absolute form. Special rules needed in coding for relocatable form are presented in a later section of this appendix.

The output of instructions for General Assembly Program in relocatable form differs from the output in absolute form. Each relocatable instruction has an associated flag to indicate the nature of the operand address. This address may be one of three types.

1. Constant or absolute address
2. Relative address
3. Two's complement of a relative address

To produce a machine language program stored in the computer, three steps are followed. These steps are illustrated below:



In the illustration below, addresses are modified from symbolic, to relocatable, to absolute in a specific set of GE-200 Series instructions. It is assumed that the loading routine has been instructed to store the relocatable instructions in memory starting at location 200. For ease of reading, operation codes are shown as mnemonics, addresses are shown as decimal numbers, and indexed instructions are indicated by ,2.

1. Symbolic				2. Relocatable					3. Absolute			
	ORG	0		REL			*		ABS			
	LDX	TA	,2	000	LDX	198	,2	Rel.	200	LDX	398	,2
	DLD	ZERO		001	DLD	098		Rel.	201	DLD	298	
LOOP	DAD	0	,2	002	DAD	000	,2	Abs.	202	DAD	000	,2
	INX	2	,2	003	INX	002	,2	Abs.	203	INX	002	,2
	BXL	T+98	,2	004	BXL	(198)	,2	(Rel)	204	BXL	(398)	,2
	BRU	LOOP		005	BRU	002		Rel.	205	BRU	202	
	.			.	.				.	.		
	.			.	.				.	.		
	.			.	.				.	.		
ZERO	DDC	0		098	0000000			Abs.	298	0000000		
T	BSS	98		099	0000000			Abs.	299	0000000		
TA	DEC	T		100					.			
				.					.			
				.					.			
				.					.			
				197					397			
				198	0000100			Rel.	398	0000300		

* Operand Address Flag:

Rel. - Relative Address
Abs. - Absolute Address
(Rel) - Two's complement of relative address. This type of operand address is reserved for BXH and BXL instructions.

Linking Relocatable Routines

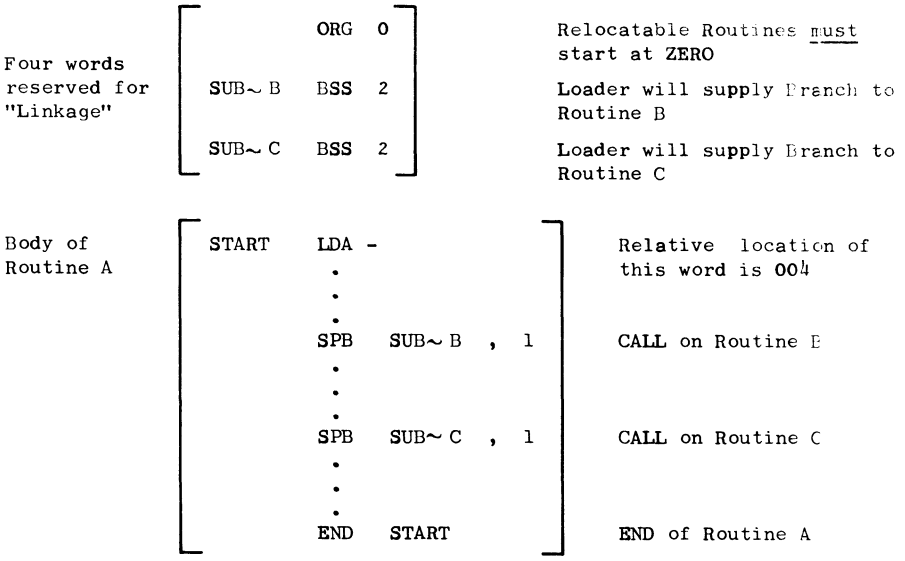
The following example shows how to code a relocatable routine containing references to other independently coded relocatable routines. Assume you are coding Routine A which calls upon Routine B and Routine C. B and C are already coded and assembled in relocatable form. The technique for calling upon other routines from Routine A is to assume there is a list of branch instructions at the head of Routine A, one branch instruction for each subroutine entrance needed by Routine A. Memory space must be provided by symbolic coding in A for this list of branch instructions. Two words must be reserved for each referenced subroutine.

The format for Routine A would be--

Branch Table, "Linkage," or Vector Table

Body of Routine

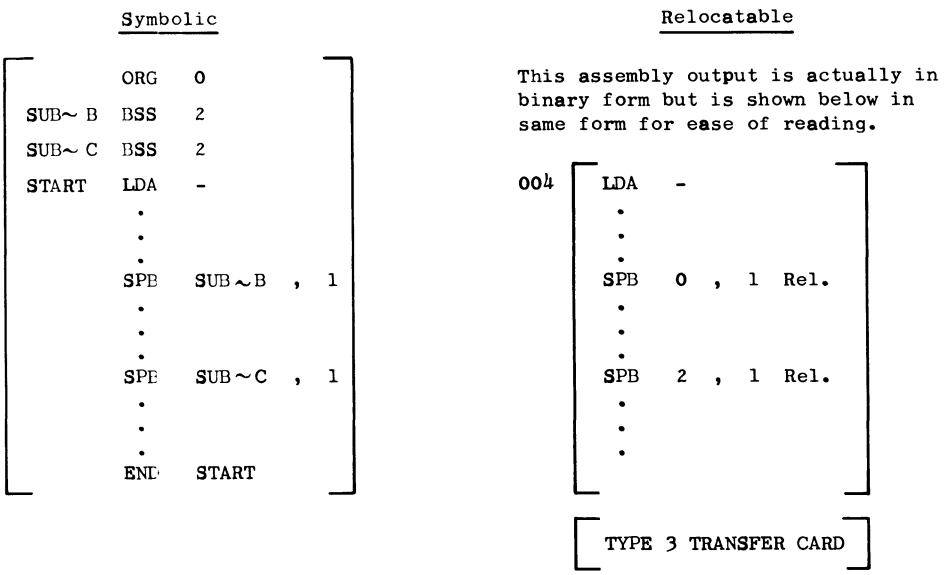
If Routines B and C each have one entrance, then the symbolic coding for Routine A would be written as follows:



When Routine A is assembled in relocatable form the assembled output is punched into binary cards in Standard Binary Format Type 3. (See MCML II, CD225B1.006R.) The first binary card in the Routine A relocatable deck has an origin of 004. The last card of the output deck is a Type-3 transfer card.

A Type-3 transfer card indicates to a Loader Routine that this is the last card to be loaded into memory before transferring to the routine just loaded.

Routine A is translated from symbolic form to relocatable form by the assembly operation as shown in the following example.



Before Routine A is loaded, a header card is prepared for A, giving the Loading Routine information needed to supply branch instructions connecting Routine A to B and C. The Loading Routine assigns absolute memory addresses to the instructions in Routine A, and supplies the missing Linkage between A and Routines B and C. Routines B and C are then loaded with Routine A.

Preparation of Header Cards

After General Assembly Program II assembles a routine in relocatable form, a header card must be prepared to precede each independently assembled relocatable binary deck. Header card information is punched into header description cards. The MCML Header Card Writer Routine, CD225B6.003, then translates the header cards into binary format.

Partial formats for Header description cards are shown below. (For more complete information, see MCML Header Card Writer Routine.)

1. Card One

Columns	Field Description
1-6	Relative location (octal number) of the first instruction word or constant in the body of the relocatable routine. Space for subroutine linkage starts at relative location 000000. Allow two words for each subroutine entrance called by this routine. First word in body of routine follows last location reserved for linkage.

Columns	Field Description
7-12	Number of external subroutine entrances called to by this routine. (Decimal number.)
13-18	Number of entrances to this routine. (Decimal number.)

2. Card Two — One of these cards must be included for each entrance to this routine. If this is the main routine it must have at least one named entrance.

Columns	Field Description
1-12	Alphanumeric name for an entrance to this routine.
13-18	Word number of this entrance (octal) relative to the first program word in this routine. Do not count linkage words. First program word is word number 0.

3. Card Three — One of these cards must be included for each external subroutine entrance called by this routine. These entrance names must be in the same order as assigned in the linkage table at the head of this routine.

Columns	Field Description
1-12	Alphanumeric name of subroutine entrance external to this routine.

Note: The numeric fields (octal or decimal numbers) in all header description cards must be right justified.

Example of Header Card Preparation

The preparation of header cards for the example where Routine A calls upon Routine B and C follows. Routine A is a relocatable main routine which is stored for execution in the lower 8k memory bank. Routine B and Routine C each has a single entrance.

Name of entrance to Routine A: START~RTN~A

Name of entrance to Routine B: OLD~AGE~TAX

Name of entrance to Routine C: INSURANCE~C3

Assume Routines B and C with their own header cards are on file in a library of commonly used relocatable routines. Header description cards used to produce the header card for Routine A follow.

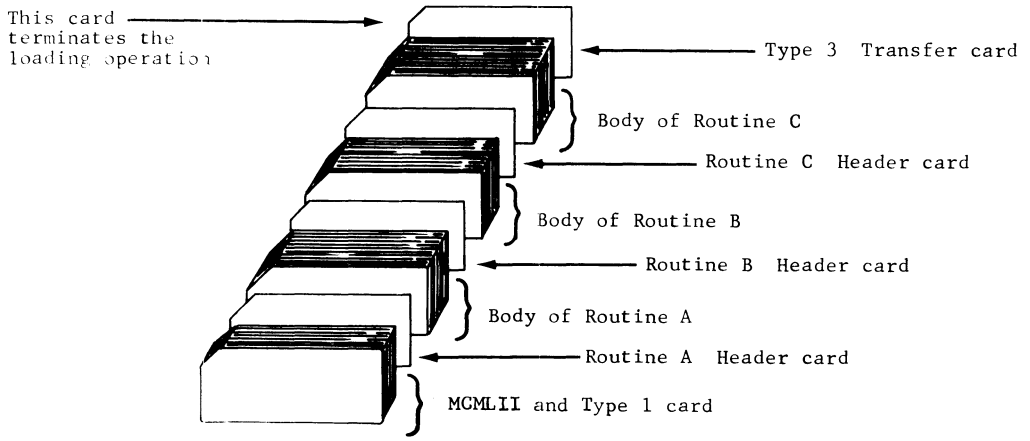
Header Description Cards

	Columns	Field Contents	
Card One:	1-6	000004	First location of Routine A
	7-12	000002	Number of external references
	13-18	000001	Number of entrances to Routine A
Card Two:	1-12	START~RTN~A	Name of Entrance
	13-18	000000	Word number of entrance
1st Card Three:	1-12	OLD~AGE~TAX	Subroutine Entrance Name
2nd Card Three:	1-12	INSURANCE~C3	Subroutine Entrance Name

The above three header description cards used as input to the MCML Header Card Writer Routine (CD225B6.003R) would produce a single binary header card that must be placed as the first card of the Routine A relocatable binary instruction card deck.

Arrangement of Card Decks for Relocatable Loading

Using the example described in the previous sections (a main routine with two relocatable subroutines) the complete sequential arrangement of card decks for relocatable loading would be as follows:

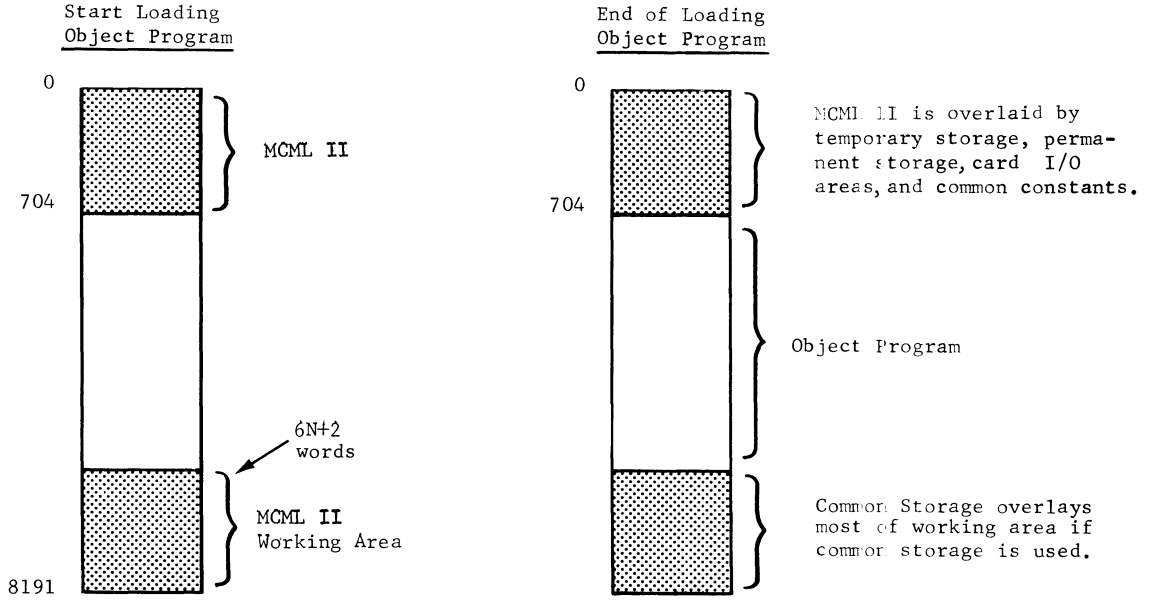


The Type-3 transfer card was produced by General Assembly Program II when Routine A was assembled in relocatable form. It is the last output card from the assembly. The Type-3 card has a 2, 3 punch in column 1. (A Type-5 transfer card was also produced and must be discarded. A Type-5 card has a 1, 3 punch in column 1.)

When subroutines A and B were assembled in relocatable form, the transfer cards produced for these routines (both Types 3 and 5) were removed before the subroutines were filed in the subroutine library. The header cards for Routines A and B were produced and filed with their respective routines in the library.

8K MEMORY ALLOCATION FOR RELOCATABLE LOADING

The header card of the first relocatable routine to be loaded indicates MCML II Working-Storage area high address is location 8191. Working area required for MCML II is 6N+2 words where N is the total number of relocatable routine entrances named in all the header cards to be loaded.



RELOCATING ROUTINES IN UPPER 8K OF A 16K MEMORY

If the body of a relocatable routine is to be stored in the upper 8k bank of a 16k memory, all of the constants used by this routine should be stored in the lower 8k memory bank. Constants stored in the lower 8k bank can be referenced by an instruction stored in the upper bank without indexing that instruction. Relocating the program words of a single routine into two memory banks calls for Dual Relocation.

The routine's header card must contain information necessary for MCML II to load and relocate the body of a routine in the upper 8k memory bank and also to load and relocate the constants used by that routine in the lower 8k memory bank.

The routine itself must be coded in two parts:

1. The first part contains the instructions that are stored in the upper bank.

2. The second part contains all the constants that must be stored for direct addressing in the lower 8k memory bank.

To perform Dual Relocation, MCML II maintains two separate location counters, one for the lower 8k memory bank and one for the upper 8k memory bank. The initial value of the lower 8k counter depends on the MCML II modules used. The initial value of the upper 8k counter is 8192. MCML II automatically increases the proper location counter each time it stores a program word in memory. The value of each location counter is stored as a relocation constant at the start of the loading operation of each relocatable routine. If a location counter happens to be an odd number at the start of a new relocatable routine it is increased by one to make it an even numbered address before being sorted as a current relocation constant.

In the following example a symbolic routine coded for Dual Relocation (body in upper 8k, constants in lower 8k) is shown translated first to relocatable form, then to absolute form. It is assumed at the time this routine is loaded, the upper 8k location counter equals 9000_{10}, and the lower 8k location counter equals 2000_{10}. The header card for this routine has a field indicating that the starting address of the lower 8k constants for this routine is relative address 500_{10}. This field is called the lower limit and is supplied to the MCML Header Card Writer Routine on header description card one. The header card for this routine also indicates that the routine is to be relocated in the upper 8k memory bank. The header card field specifying memory bank assignment is called the Memory Bank Indicator and is supplied to the MCML Header Card Writer Routine on header description card one. (For preparation of header cards refer to MCML Header Card Writer Routine, CD225B6.003R.) The body of the routine in the example is assumed to be less than 500 words including the six words reserved for linkage at the head of the routine.

When any part of a program is relocated into upper memory it is necessary to use two extra instructions in the linkage in each subroutine. MCML II provides this linkage and no provision for these words need be made in the source program. At load time, MCML II will save two cells in each program entrance in front of the program itself. The effect is to relocate the program upward by the extra cells. For instance, a program with one entrance, would be relocated two cells upward. The example that follows these two cells would be cells 9000 and 9001 causing the upper range of this program to relocate to 9002.

Example of Symbolic Routine Coded for Dual Relocation

Symbolic

```
        ORG    0
S1      BSS    2
S2      BSS    2
S3      BSS    2
START   LDA    C1
        STA    C2
        .
        .
        .
        SPB    S3 , 1
        .
        .
        .
        LDX    C3 , 2
        INX     1 , 2
        .
        .
        .
        BRU    START
-------------------------
        ORG    500
C1      ALF    YES
C2      ALF
C3      DEC    0
        END    START
```

Relocatable

```
REL
006   LDA    500          REL
      STA    501          REL
      .
      .
      .
      SPB    004 , 1      REL
      .
      .
      .
      LDX    502 , 2      REL
      INX      1 , 2      ABS
      .
      .
      .
      BRU    006          REL
500          YES          ABS
501          ΔΔΔ          ABS
502          000          ABS
```

Absolute

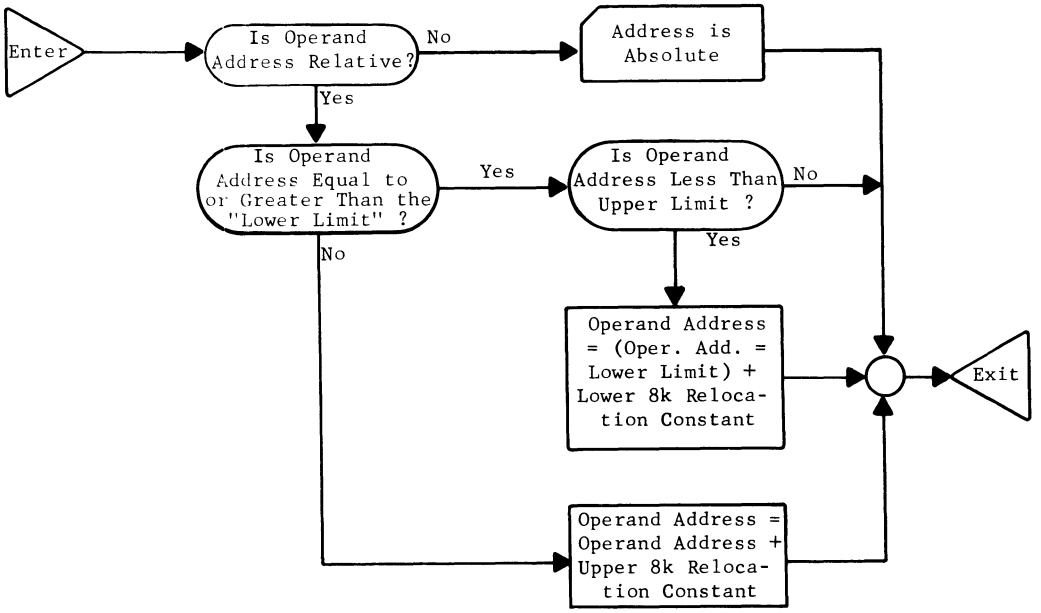
```
ABS
0000
              .
              .
              .
2000         YES              LOWER 8k
2001         ΔΔΔ
2002         000

8192
9008   LDA   2000
9009   STA   2001             UPPER 8k
       .
       .
       SPB   0812 , 1         *
       .
       .
       LDX   2002 , 2
       INX   0001 , 1
       .
       .
       BRU   0814             **
```

* The effective address of this SPB instruction when executed in the upper 8k memory bank is 9004_{10} (8192 + 812).

** The effective address of this nonindexed BRU instruction when executed in upper 8k memory bank is 9006_{10} (8192 + 814).

MCML II compares each relative operand address against the "lower limit" and upper limit relative addresses given in the relocatable routine header card. If a relative operand address is equal to or greater than the "lower limit" that address is changed to an absolute address in lower 8k memory where the constant referred to is to be relocated. If the relative location of a program word is equal to or greater than the "lower limit" (but less than the upper limit) that program word is stored in an absolute location in the lower 8k memory bank. The program is then relocated with respect to the lower 8k relocation constant. (An address greater than or equal to the upper limit is not relocated.) The logic for computing absolute memory addresses for a relocatable routine being stored in upper 8k memory is shown below:



This same logic is used to convert relocatable program word location addresses into the absolute locations in which the program words are stored.

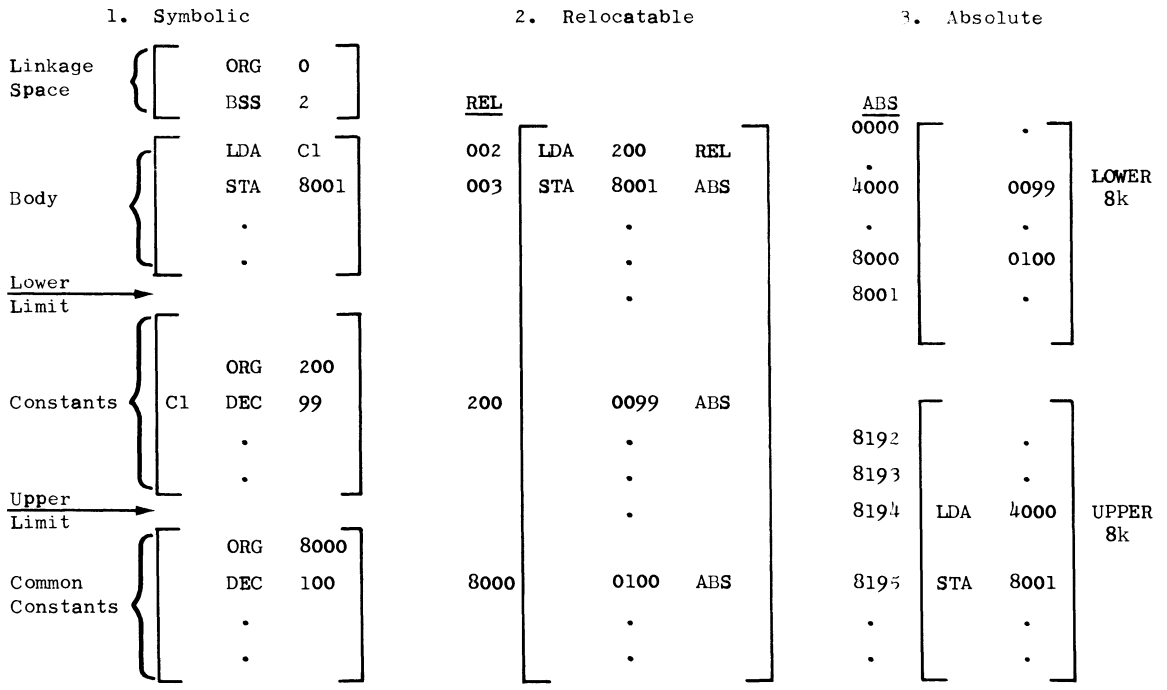
ASSIGNING COMMON STORAGE TO RELOCATABLE ROUTINES

All addresses defined as octal or decimal numbers in a General Assembly Program symbolic coded routine are flagged as absolute addresses. During the loading operation, MCML II does not alter absolute addresses. In this way Routine A can communicate with relocatable Routine B. Commonly agreed upon absolute memory locations are used. These locations are external to both A and B after these routines are loaded. In some programs it may be useful if a set of constants supplied by Routine A can be used directly by other relocatable routines.

Routine A's header card contains the control for loading constants from a relocatable routine into Common Storage. When a program word in a relocatable routine has an origin (storage location) greater than the "upper limit" in that routine's header card, MCML II does not relocate that word but stores it in the location given on the binary instruction card. The Common Storage area must be in the lower 8k memory bank, regardless of whether or not the body of the relocatable routine is stored in the lower 8k bank.

An example of a symbolic routine containing both relocatable constants and constants to be stored in Common Storage follows. The relocatable coding resulting from assembly and the absolute coding produced by loading the routine is governed by the following assumptions:

1. At time of loading MCML II lower 8k location counter = 4000, upper 8k location counter = 8192.
2. Body of routine is less than 100 words.
3. Header card contains:
   Lower Limit = 200
   Upper Limit = 7999
   Memory Bank Indicator = 1 (Upper 8k)



Note that the program constant C1 is loaded into the lower bank absolute location 4000 and the operation address of the upper bank instruction (LDA C1) is changed by the loader to absolute address 4000. The Common Storage constant (DEC 100) is loaded into the absolute address indicated by the ORG 8000 card in the symbolic program since this address is greater than the "upper limit" given on the header card.

LINKAGE ACROSS MEMORY BANKS OF A 16K MEMORY

Four types of linkage between relocatable routines are automatically provided by MCML II.

1. Upper Bank to Lower Bank
2. Lower Bank to Upper Bank
3. Lower Bank to Lower Bank
4. Upper Bank to Upper Bank

The type used depends upon the location of the routines being linked.

The technique for providing linkage requires that only index group zero be used by relocatable routines to make subroutine calls. Also, index group one must not be destroyed by any of the relocatable routines. The linkage logic uses two constants placed by MCML II in index words 1 and 2 of index group 1. Index word 1, group 1 (absolute location 005), contains zeros. Index word 2, group 1 (absolute location 006) contains the value 8192. To branch across memory banks in the generated linkage instructions, the branch instruction is indexed by the appropriate constant.

In creating the linkage between Routine A stored in one memory bank and Routine B stored in the other 8k bank, MCML II generates four instruction words. The first two instructions are placed in the linkage space at the head of Routine A. The second two are generated and stored in the opposite memory bank. These four instructions are--

At the head of Routine A:

1. SXG 1
2. BRU X, 1 (if going from upper to lower) or
   BRU X, 2 (if going from lower to upper)

At location X:

3. SXG 0
4. BRU Y (Y is entrance of Routine B)

In creating the linkage between two routines stored in the same memory bank MCML II generates:

At the head of Routine A:

```
BRU X
BRU X
```

At location X:

```
SXG 0
BRU Y
```

ASSIGNING *COMMON STORAGE

*Common Storage is merely Common Storage placed in the upper 8k memory bank with the following restrictions:

1. Only numeric arrays may be placed into *Common Storage

2. *Common Storage arrays may not be constant arrays. The loader never places values into *Common Storage. The only way to place a value in a *Common Storage field is to make it a receiving field in the object program.

*Common Storage is shown in the GECOM Edited List under Object Listing Storage Reservations, by "Equal Cards" with lower 8k memory assignments. However, when used in the object program, *Common Storage addresses are always indexed with the locations shown in the listing plus 8192.

CONVENTIONS WHEN ASSEMBLING WITH GENERAL ASSEMBLY PROGRAM AND LOADING WITH MCMLII

1. Relocatable Routines coded in General Assembly Program symbolic form are assembled by General Assembly Program II using the relocatable output option.

2. Symbolic relocatable routines must be assembled with all internal addresses relative to memory location zero. (ORG = 0)

3. Two words must be reserved at the head of a relocatable routine for each external subroutine entrance called to from within the routine.

4. The maximum size of a relocatable routine is 8192 words including the space reserved for linkage at the head of the routine.

5. The body of a relocatable routine after loading must be contained within one 8k memory bank. It cannot extend from the lower 8k bank into the upper 8k bank.

6. The MCML II Loader is the routine that performs the loading, relocating, and linking function.

7. The MCML Header Card Writer Routine (CD225B6.003R) is available to prepare required binary header cards for relocatable routines.

8. Dual Relocation of relocatable routines (body in upper 8k, constants in lower 8k) is controlled by the memory bank indicator and the "lower limit" field in the routine's header card.

9. Placing constants from a relocatable routine into Common Storage is controlled by the "upper limit" field in the routine's header card.

10. When two routines are stored in opposite memory banks, four instruction words are generated by MCML II for linkage.

11. Names of relocatable routine entrances may be as large as twelve alphanumeric characters.

12. All relocatable routine entrance names must be unique.

13. Index Group zero must be set in a relocatable routine when a call is made to an independent subroutine.

14. Relocatable routines must not destroy the words in index group one.

15. In order for a General Assembly Program language routine to be translatable into relocatable form, most of the address references within the routine must be symbolic references. If an address is given in the General Assembly Program language routine as an octal or decimal number, that address must be a fixed machine address external to the routine. This address is independent of the memory locations in which the routine itself is to be stored for execution.

16. References from a relocatable routine to Common Storage must be coded in General Assembly Program as octal or decimal addresses.

APPENDIX J
OBJECT PROGRAMS USING DISC STORAGE UNITS (DSU'S)

INTRODUCTION

This appendix pertains to object programs which read data from and/or write data on DSU's. It does not apply to the recording of object program instructions on DSU's or the loading of object program instructions into memory from DSU's.

GECOM produced DSU object programs conform to GET Programming Standards and Conventions. The reader should refer to Input/Output Standards, DSU Files, DSU Error Control and the disc storage glossary contained in the GET Reference Manual.

The user should also be familiar with the SIOS (MIO) package, CD225E8.000, which is employed for DSU input/output functions in GECOM produced object programs.

After reading the remainder of this appendix, the user should consult the following sections of the GECOM Reference Manual for more detailed information on source language functions, formats, and conventions for the compilation of DSU object programs:

1. Data Division, Nonstandard Data.
2. Data Division, File Section, File Description: Recording Mode, Label Records, and NO...SENTINEL clauses.
3. Data Divison, Data Image Input and Output Entries: M.
4. Procedure Divison: CHAIN, CLOSE, OPEN, READ (Options 4 and 5), READY, RELEASE, and WRITE (Option 4) verbs.
5. Environment Division: OBJECT~COMPUTER, I~O~CONTROL, and DSU~CONTROL sentences.

DSU ADDRESSES

In the source program, the desired DSU address is supplied by the user before each seek operation (see Procedure Division, READY verb). The process involved in developing the DSU address depends on the organization of the DSU data files.

DSU FILE ORGANIZATION

Some types of file organization and addressing schemes are discussed below.

1. Records of the same size (same number of frames) can be assigned to the same DSU area. The advantage of this is that one indicator on the first frame of a record can indicate whether the record slot is occupied or is vacant. When it is desired to add a record of N frames, a single test determines if an N frame record slot is vacant. On the other hand, if records of different sizes are assigned to the same storage area, in order to add an N frame record, a test for N consecutive vacant frames would have to be made.

   When records of the same size are assigned to the same DSU area, it is advantageous if the frame number (0-95) of their DSU address is a multiple of the number of frames required for each record. This simplifies the randomizing process to develop the desired address (which must be the address of the first frame of a record). With the rule "frame number of the DSU address must be a multiple of the record length (in frames)" it is easy to develop a beginning-of-record address. Thus, a record requiring three frames would have frame number 0, 3, 6, ..., 90, or 93 in the frame portion of its DSU address. The recommended record lengths are 1, 2, 3, 4, 6, 8, 12, or 16 frames since these all divide evenly into 96, the total number of frames available in one arm position. If a size is selected which does not evenly divide into 96, developed addresses must be checked by the user to avoid "wrap around." Thus, frame number 95 is divisible by 5, but five frames cannot be read or written without wrapping around through frames 0, 1, 2, and 3 of the same arm position.

2. When randomizing techniques are used to determine DSU addresses, the possibility of more than one record randomizing to the same DSU address must be provided for. Records whose keys randomize to a DSU address which is already occupied are assigned to some "overflow areas." One method for linking the records which randomize to the same DSU address is a form of chaining. The record occupying the address to which other records randomize has a chaining word which contains a new DSU address in an overflow area. If the record at the randomized address is not the desired record, the chaining word is interrogated to determine the overflow area address of the next record which randomized to the same address. Many records can be chained in this manner. The last record of a chain always has an end-of-chain indicator.

   Usually word 0 of the first frame of a record is used as the chaining word and indicator word. Bits 2-18 contain the chaining address. Bit 0 is the vacant/occupied indicator. If it is 1 (minus) the record slot is vacant. Note that chaining can be accomplished through a vacant record area. The chaining address is in negated form, however. If bits 0-18 are zero, this record is the end of a chain. If bits 0-18 are all 1 bits, the record is vacant and there is no more chain; that is, the record which pointed to this position was really the last of a chain. Bit 19 is never checked, but it must not contain a 1 bit unless bits 0-18 are also 1 bits.

3. The user may wish to retain some order or sequence to a file which is not in order or sequence on the DSU. Chaining is used to accomplish this; the first record of the file chains or points to the second record, the second record chains to a third, etc.

In this type of chaining it is not always possible to use word 0 of the first frame of a record as the chain word. A record might have several chaining words implying different sequences depending on the file application. The record might belong to more than one file. Note that care must be exercised when a record contains more than one chain word. A record might be deleted from file A (its chain word negated) but still be required for file B; file A must know about file B records to avoid overlaying the record unless it is vacant to all files of which it is a member.

4. Records may be stored and processed sequentially on the DSU's. After an initial positioning, record addresses are developed by adding the frames required for the record to the previous DSU address. The developed address must be checked to avoid the possibility of an illegal address (frames 96-127). On files described as sequential, the GECOM object program checks and adjusts for illegal addresses.

5. Either direct or index table addressing schemes provide an absolute DSU address without intervening address calculations. The DSU address of a desired record might be obtained from a table cross referencing the record keys with DSU addresses. An absolute address might be obtained from another record--a master record providing the address of its trailers, for example.

6. A single frame (64 words) may be able to accommodate several records of different files. A record of file A might occupy words 0-37 of a frame while a record of file B occupies words 38-60. When records of different files are sharing the same frames, only one of the files may be in the open condition. Also, only the file at the "top" (starting in word 0 of the frame) can have blocked records.

   The data descriptions for frame-sharing files are conventional, except that the frame area not occupied by the file being described must be designated as FILL.

   When a frame-sharing file is read, updated, and written out, the input file name and output file name must share the same buffer area. In this way any data belonging to other files "tags along."

JOURNAL TAPES

Output DSU files may be assigned to a Journal Tape. Each time data for one of these files is recorded on the DSU, the same data with two additional words is written onto the Journal Tape. The first added word contains the disc storage unit and plug number for the DSU referenced. The disc storage unit code is in bits 5-7, and the plug number is in bits 11-13. The second additional word contains the DSU address.

The purpose of the Journal Tape is to keep a record of data written on the DSU's. When the Journal Tape option is used all output files assigned to DSU's should also be assigned to the Journal Tape. A possible exception to this is a DSU file which is used as temporary storage for intermediate results.

If some program destroys DSU data, the appropriate Journal Tapes can be processed by TAPER, CD225E8.003, to restore the DSU data. The DSU's must first be reloaded from the last complete DSU dump; then, the Journal Tapes that were created between the dump and the errant program can be processed.

LIST OF ENTRIES TO DSU CODING

Because of the variety of verbs used in referencing files assigned to the DSU--OPEN, READY, READ, WRITE, CHAIN ... UNTIL, READ ... UNTIL, READ ... COPY ... UNTIL, etc.--and the necessity for issuing these commands in the proper sequence for any given file, an ENTER GAP routine may be compiled with a source program to aid in detecting the cause of errors arising from failure to issue these commands in the required order.

The function of the routine is to build a table, LLP, which is a List of Last Procedure addresses where transfers (SPB's) to the DSU coding have been performed. The table is eight locations in length and a pointer, PNT, indicates which value in the table (LLP + PNT) contains the address of the last SPB executed.

Because of the option of assigning the main program to upper or lower memory, two versions of the routine are required. The version for programs with "main assigned to lower" is described in Example 1 which follows. The version for programs with "main assigned to upper" appears in Example 2. The described coding is to be executed as an initialization routine and, therefore, should be placed before any OPEN statements in the source deck.

Example 1

GENERAL ELECTRIC
COMPUTER DEPARTMENT PHOENIX ARIZONA

GENERAL COMPILER SENTENCE FORM

PROGRAM | DATE
PROGRAMMER | COMPUTER | PAGE

SEQUENCE NUMBER

```
      C  ****
      C  ROUTINE - FØR PRØGRAMS WITH MAIN ASSIGNED TØ "LØWER" - TØ BUILD
      C          A LIST ØF ADDRESSES WHERE "SPBS" TØ THE DSU CØDING
      C          ARE PERFØRMED.
      C  ****
      ENTER GAP.
      LDA BRUPAT          SET STX+3 TØ BRANCH TØ "PATCH" EACH TIME
      STA STX+3           STX IS EXECUTED
      BRUP    END~LLP~GAP
PATCH LDA PNT
      ADD Z01             INCREMENT PØINTER
      EXT LLPMSK
      STA PNT             PNT = 0 THRU 7
      LDA 1               ADDRESS ØF SPB
      LDX PNT      1
      STA LLP      1-     PUT ADDRESS IN TABLE
      LDX RX       1
      STA 1        1
      BRU 1        3      RETURN TØ DSU CØDING
BRUPAT BRU 1A CH
LLPMSK ØCT 3777770
PNT   ØCT 0
LLP   BSS 8
      END
      C  ****
      C  CONTINUE PRØCEDURE STATEMENTS.
      C  ****
      END~LLP~GAP
```

Example 2

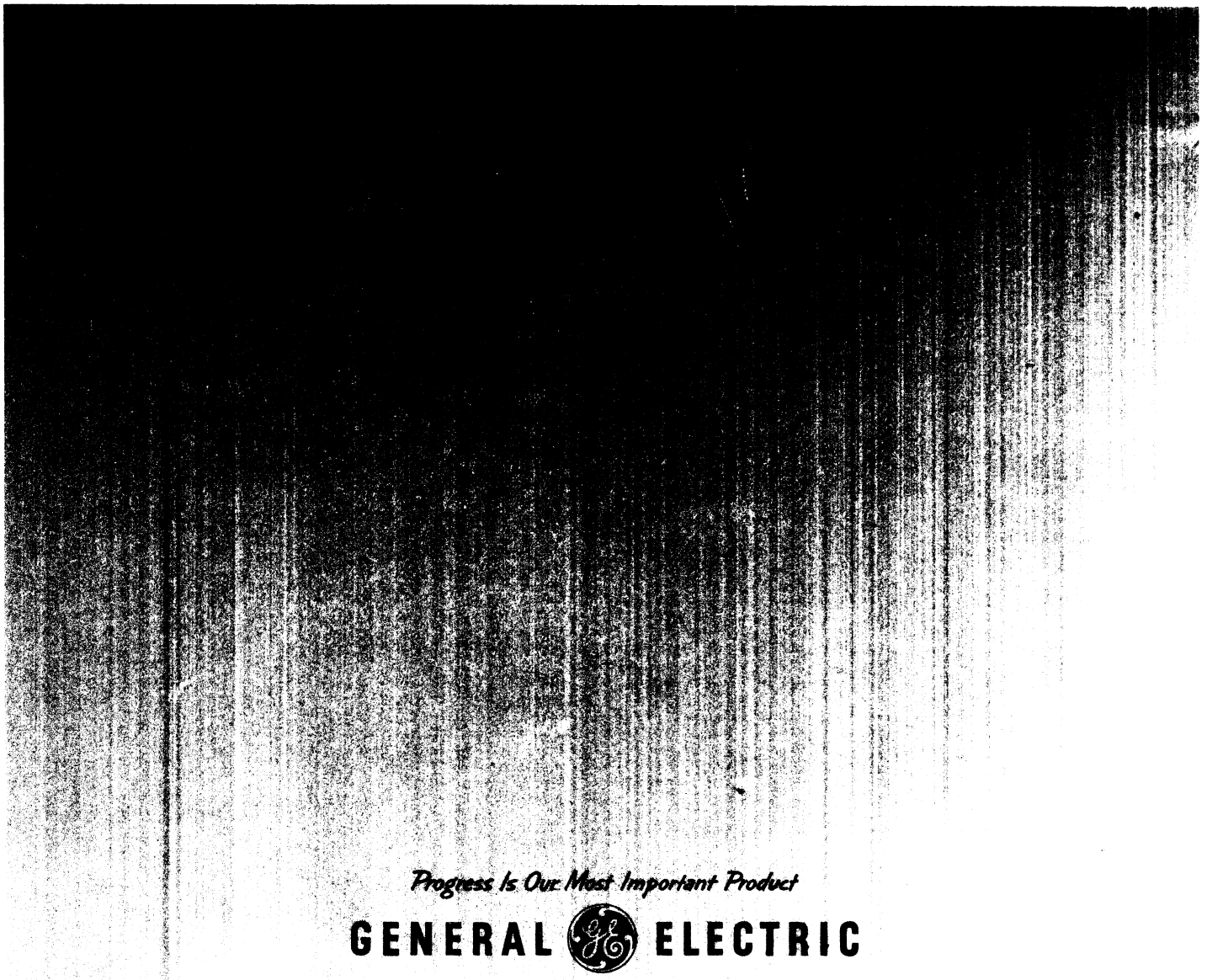
GENERAL ELECTRIC
COMPUTER DEPARTMENT, PHOENIX, ARIZONA

GENERAL COMPILER SENTENCE FORM

PROGRAM | DATE
PROGRAMMER | COMPUTER | PAGE

```
SEQUENCE
NUMBER
      C ** ***
      C ROUTINE - FOR PROGRAMS WITH MAIN ASSIGNED TO "UPPER" - TO BUILD
      C        A LIST OF ADDRESSES WHERE "SPBS" TO THE D-U CODING ARE
      C        PERFORMED.
      C ** ***
        ENTER GAP.
        LDX  6           2          UPPER MEMORY FACTOR (20000)
        LDA  BRUPAT      2          SET STX+4 TO BRANCH TO "PATCH" EACH TIME
        STA  STX+4       2              STX IS EXECUTED
        BRUP       END~LLP~GAP
PATCHS  LDA  PNT         2          XR2 CONTAINS (20000)
        ADD  Z01                    INCREMENT POINTER
        EXT  LLPMSK      2
        STA  PNT         2          PNT = 0 THRU 7
        ADD  6                      ADD UPPER MEMORY FACTOR
        STA  2
        LDA  1                      ADDRESS OF SPB
        STA  LLP         2          PUT ADDRESS IN TABLE
        LDX  RXT         1
        LDX  6           2          RESTORE XR2 TO 20000.
        STA  19          1
        BRU  1           3          RETURN TO DSU CODING
BRUPAT  BRU  PATCHS
LLPMSK  OCT  3777770
PNT     OCT  0
LLP     BSS  8
        END
      C ** ***
      C CONTINUE PROCEDURE STATEMENTS
      C ** ***
        END~LLP~GAP.
```

INDEX



Progress Is Our Most Important Product

GENERAL ELECTRIC

COMPUTER DEPARTMENT • PHOENIX, ARIZONA

LITHO U.S.A.