DATA PROCESSING
AUTOMATED BY GENERAL ELECTRIC

Progress Is Our Most Important Product

GENERAL ELECTRIC

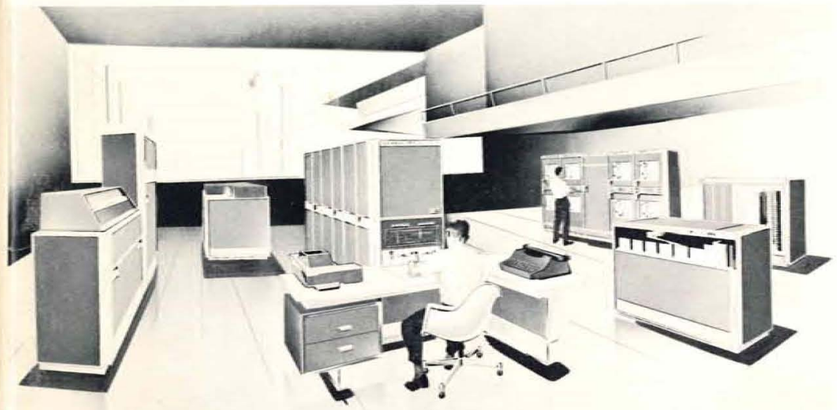
COMPUTER DEPARTMENT, PHOENIX, ARIZONA

CPB-123 (5.5M10-60)

GRAD

CPB-123

GE 225



GENERAL COMPILER MANUAL

GENERAL ELECTRIC

GE 225

ACKNOWLEDGEMENT

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

Air Material Command, U. S. Air Force
Bureau of Standards, Department of Commerce
Datamatic Division, Minneapolis-Honeywell Corporation
David Taylor Model Basin, Bureau of Ships, U. S. Navy
ElectroData Division, Burroughs Corporation
International Business Machines Corporation
Radio Corporation of America
Remington-Rand Division of Sperry-Rand, Inc.
Sylvania Electric Products, Inc.

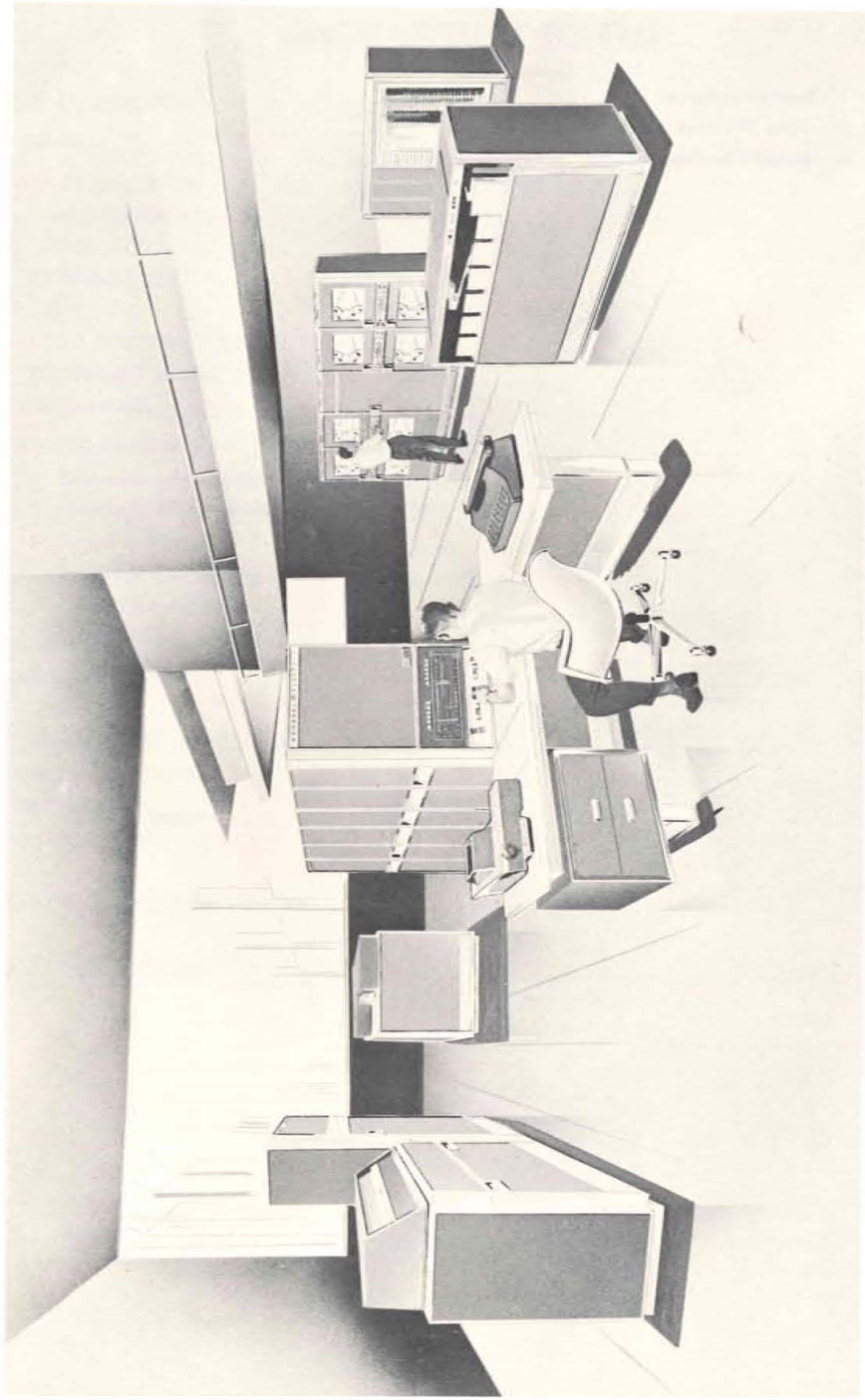
"The committee was joined at a later date by the General Electric Computer Department. The initial specifications for the COBOL language are the result of contributions made by all of the above-mentioned organizations and no warranty expressed or implied, as to the accuracy and functioning of the programming system and language is made by any contributor or by the committee and no responsibility is assumed by any contributor or by the committee in connection therewith.

225
DATA PROCESSOR

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS



GE 225 Information Processing System

I INTRODUCTION

The most recent answer to the problem of programming ease and costs is the concept of automatic coding and pseudo-languages. To date, many automatic coding systems have been developed. Several of these have met general acceptance among certain groups of users; others have only gained recognition at particular installations. The more successful of these systems have pseudo-languages which resemble English sentences or use abbreviated English words for stating the solution of the application to be processed by the computer. A special computer program, commonly called a compiler, is provided to convert the language into computer instructions.

Such a compiler has been provided for the GE 225. Ideas from both ALGOL* and COBOL* are incorporated. Our General Compiler is an end-product of the investigations of both of these committees. It is not a language in the sense that ALGOL and COBOL are languages. Instead, it is more a concept in the design and implementation of a compiler. The language presently available to our General Compiler is based on COBOL, since it satisfies the needs of a broad spectrum of business data processing applications. To accommodate the demands of scientific users, the ability for stating complex equations, Boolean expressions, and floating point arithmetic was also incorporated into the language format of COBOL. There are also many other things that have been taken from ALGOL and other sources that are available optionally. Therefore, the present version of the 225 General Compiler is capable of accepting programs written in one or two, or in a combination of the two language forms.

The GE 225 General Compiler preliminary specifications outlined in this manual are mainly definitions of allowable operations and imply and require computer knowledge if they are to be used effectively. Therefore, the manual presented here should not be used as a TEXTBOOK, but rather as REFERENCE MATERIAL to be used to augment already realized skills. Because COBOL is a dynamic language, changes are to be expected. The General Compiler language that is described here is an approach to building compilers that facilitates making such changes. With this facility we will be able, at all times, to keep our General Compiler language current. Any changes which we make will be reflected in future manuals or in supporting material.

A program written in a language other than machine language is termed a "source" program. There are four divisions of a source program written in the General Compiler language:

1. Procedure Division - consisting of an ordered set of sentences specifying the steps the computer is to follow in solving a problem.

2. Data Division - defining the arrangements and characteristics of the data to be processed.

3. Environment Division - containing information about the configurations of devices needed by the source program.

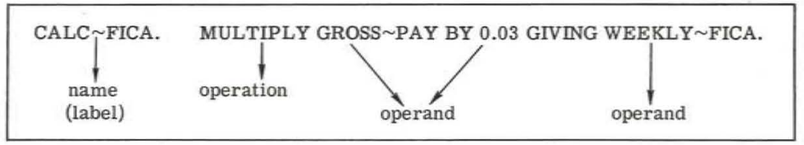
4. Identification Division - providing the label and other information about the source program.

Each division is a separate level of program preparation that can be altered without affecting the others. This allows ease of programming and facilitates program conversion to those General Electric computers having the General Compiler automatic coding system.

* COBOL - Common Business Oriented Language
ALGOL - Algorithmic Language

II SENTENCES: BASIC ELEMENTS

The procedure portion of a source program is an ordered set of sentences specifying the steps the computer is to follow in solving a problem. (See Figure 1.) Basically, a sentence is made up of a name, an operation (usually expressed by a verb), and one or more operands.



Here CALC~FICA is the name (or label), Multiply is the operation, and Gross~Pay, 0.03, and Weekly~FICA are the operands. These basic elements may be diagnosed more fully as follows.

A. Sentence Names

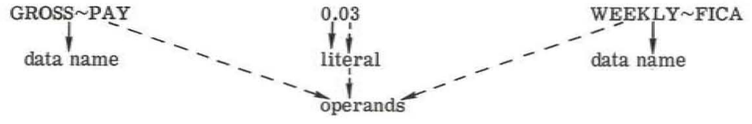
The name (CALC~FICA in our example), may describe the purpose of the sentence or it may be a number indicating a particular sequence, and needs to be given only when a sentence is referred to by another sentence. Names should not exceed 12 characters and the words of the source language vocabulary should not be used as sentence names.

B. Operation

A word (usually a verb) indicating the function of a sentence is the operation. In our example Multiply tells the compiler to create the machine instructions necessary to multiply the gross pay amount by 0.03 and store the product in the weekly F.I.C.A. field.

C. Operand

An operand is the quantity which is being operated on according to the function of the sentence. An operand may be a data name, a literal, or a sentence name. Therefore, in our example:



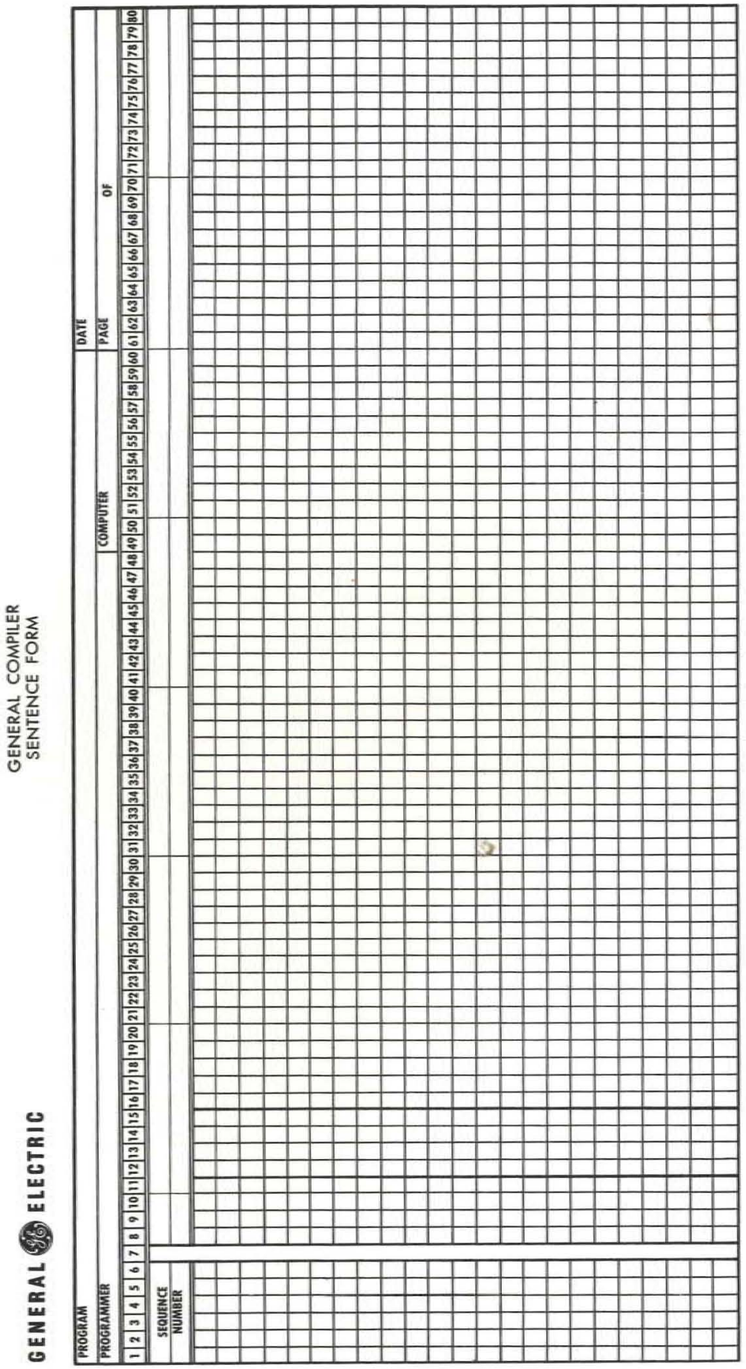


Figure 1 Sentence Form

D. Data Names

Names representing data (files, records, fields, elements, variables, constants, arrays of numbers, etc.) are arbitrarily assigned by the programmer. They are formed from:

Alphabetics	A, B, C, ..., Z
Integers	0, 1, 2, ..., 9
Tilde	~

Data names should not exceed 12 characters and should contain at least one alphabetic. Words of the source language vocabulary should not be used nor should a data name begin or end with a tilde.

E. Subscripts

Subscripts are used to identify elements in a list or in an array of values. For example, a list of values X_1, X_2, X_3, ...X_{10} may be given the name X. If the values are stored consecutively, each value can be referenced by a subscript ranging from 1 through 10. In this case the list would be described as:

X(I)

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, the value X_1 is referenced: if I = 7, X_7 is referenced.

An array may be multi-dimensional: that is, it may have more than one subscript. Also, subscripts may be written as arithmetic expressions containing other subscripted arrays. For example:

```
ABC (R + L)
K(A - B * C, L (I, J), X)
RATE (T + L, D - 4)
```

If an expression is used as a subscript, the integral part of the result is only used to obtain the position of the element in an array.

F. Conditional Fields

In some applications it may be desirable to assign a name to each value of a field. For example:

Field Value	Meaning	Conditional Name
1	white	WHITE
2	pink	PINK
3	blue	BLUE
4	turquoise	TURQUOISE

The conditional (representing a condition or value) name stands for a particular value and may be used as an operand to mean the value. Conditional names are principle operands of logical expressions:

IF PINK, GO TO 123.

F. Conditional Fields

tells the computer to test the color code field for a 2. If a 2 is present, control passes to sentence 123. If a 2 is not present, control passes to the sentence following the IF sentence.

G. Literals

A literal is a quantity itself rather than a name given to a quantity and may be either numeric or alphanumeric. Numeric literals may be written as an

Integer	230
Decimal Integer	230.1
Power of ten	2.301E2

The letter "E" in the power of ten notation delimits the power of the exponent. Numeric literals may be enclosed in quotation marks (see MOVE verb).

Alphanumeric literals must be enclosed by quotation marks (") e.g., "A". Further, an alphanumeric literal may contain any character in the set of the computer, e.g., "3A".

H. Figurative Constants

Certain literals are called figurative constants. These have been assigned fixed names as follows:

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

For a figurative constant, the compiler produces a string of identical characters whose length is dependent upon context. For example, if a QTY~ON~HAND field was described as five digits long, the statement "IF QTY~ON~HAND EQUALS ZERO" would compare the contents of the QTY~ON~HAND field against 00000.

I. Qualifiers

Every name in a source program should be unique either because no other name has the identical spelling or because the name exists within a hierarchy of names such that the name can be made unique by mentioning one or more names higher 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. A file name is the highest level qualifier available for a data name. Three basic rules should be used for qualification:

1. A qualifier should exist outside (above) the name it is qualifying.
2. A name may not appear at two levels in a hierarchy so that it would appear to qualify itself.
3. If a data name 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.

Take, for example, two records named MASTER and NEW~MASTER each containing CURRENT~DAT and a TRANSACT~DAT field. If each of these fields contains

I. Qualifiers

three elements, MONTH, DAY, and YEAR, we can refer to the current month in the NEW~MASTER record as:

MONTH OF CURRENT~DAT OF NEW~MASTER

and we may refer to the day of the transaction in the master record as:

DAY OF TRANSACT~DAT OF MASTER

J. Arithmetic Expressions

An arithmetic expression is a sequence of variables, numbers, and mathematical functions connected by symbols which represent the arithmetic operations add, subtract, multiply, divide, and exponentiation. We may write an expression as:

FED~TAX = (GROSS~PAY - (NUM~DEP * 13.00)) *0.18

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

Arithmetic expressions are evaluated from left to right and indicated operations are performed in the following order:

Operation	Symbol
Functions and Exponentiation	**
Multiplication and Division	* /
Addition and Subtraction	+ -

Parentheses are used to establish a precedence of evaluation. When they are used the evaluation is from the innermost to the outermost set of parentheses.

K. Relational Expressions

Any expressed or implied combination of two field names, element names, literals or arithmetic expressions connected by any of the following relations is called a relational expression.

Relation	Abbreviation
Exceeds	GR
Greater than	GR
Not greater than	NGR
Less than	LS
Not less than	NLS
Equal to	EQ
Equals	EQ
Not equal to	NEQ
Unequal to	NEQ

Relational expressions are evaluated from left to right.

K. Relational Expressions

EXAMPLES:

1. IF PART~NUMBER OF MSTR~INVNTRY EQUALS PART~NUMBER OF TRANSACTIONS (two field names).
2. IF WEEKLY~FICA OF MASTR~PAYROL + ANNUAL~FICA OF MASTR~PAYROL EXCEEDS 144.00 (arithmetic expression and literal).

L. Logical Expressions

A logical expression is any combination of conditional names and relational expressions connected by the logical AND and OR (inclusive). Logical expressions are evaluated from left to right with the logical AND having precedence over the logical OR. Parentheses are used to establish precedence.

EXAMPLES:

1. IF PART~NUMBER OF MSTR~INVNTRY EQUALS PART~NUMBER OF TRANSACTIONS AND TRANSACT~COD IS GREATER THAN 1.
2. IF GRADUATE OR EXPERIENCED.

III SECTIONS

Sections provide a way to group (under a single name) an ordered set of sentences having a common function and needing to be executed from more than one place in a program. The programmer may partition a program into sections as he chooses. However, he must designate sections as follows:

section~name SECTION.
[INPUT variables field~name$_{I1}$, field~name$_{I2}$ .....] .
[OUTPUT variables field~name$_{O1}$, field~name$_{O2}$ .....] .
BEGIN.
(One or more sentences)
END section~name SECTION.

The "Section name" identifies a section and provides a reference to it. BEGIN-END separates the body of a section from its heading. The "head" portion defines those fields of the body which must be assigned values before the section can be executed. END is the common exit point for those sentences appearing in the body. As such it may be given a sentence name. This way of using a section allows us to program a generalized routine using "dummy" parameters which take on actual values when the section is executed at object program running. The following format of the PERFORM sentence is used to do this:

PERFORM section~name [USING field~name$_{I1}$, field~name$_{I2}$ .....]
[GIVING field~name$_{O1}$, field~name$_{O2}$ .....] .

When the PERFORM is executed, field~name$_{I1}$ of the using clause is assigned to field~name$_{I1}$ of the INPUT list in the section head. Field~name$_{I2}$ of the PERFORM is assigned to field~name$_{I1}$ of the section head, etc. Then the body of the section is performed. If a GIVING clause is specified with the PERFORM and an OUTPUT list appears with the section, the output variables of the section head are assigned to those listed with the GIVING clause. The assignment is one-to-one with field~name$_{O1}$ of the section head going to field~name$_{O1}$ of the GIVING clause, and so on. When this is completed, the sentence following the PERFORM is executed.

There is another way in which sections can be used. Some applications may warrant copying a given set of sentences and while doing so require that certain names be replaced by other names used elsewhere in the program. PERFORM is used in this case also. The section is written as before. But, since value assignment is not required, no listing is necessary within the section head. The section now takes the form:

section~name SECTION.
BEGIN.
(One or more sentences)
END section name SECTION.

To modify the names within the body, the PERFORM is stated:

PERFORM section~name REPLACING name$_1$ with name$_2$, name$_3$ with name$_4$.....

The section body is copied or inserted in place of the PERFORM. This occurs during compilation. While it is being copied, old names are replaced with new names. Name$_1$, name$_3$, etc. may be any name within the section's body. Caution should be exercised since the replacement takes place with every mention of these names.

Sometimes we may wish to divide a run or program into sections and compile and test each section separately. Then, when the sections are individually checked out, piece them together to form a continuous run. The rules given here permit just this. When initiating compilation, the compiler can be directed to compile a section and assign all coding addresses relative to zero instead of the usual absolute address. Now the object coding for the section is in "relocatable" form and can only be put in absolute form when loaded prior to object running. A special load routine is used to do this. When a section is tested as an entity; i.e., without a PERFORM executing it, the END section name acts as a STOP sentence.

Once the relocatable sections are tested, we may piece them into any compatible program by using a PERFORM sentence. The section format is used again to indicate that part of a run or program exists in relocatable form. This time it hasn't any body since the sentences of the body were previously compiled. Its format now looks like

section~name SECTION.
[INPUT variables field~name~1, field~name~2, ...]
[OUTPUT variables field~name~1, field~name~2, ...]
BEGIN.
COPY RELOCATABLE.
END section~name SECTION.

Only the PERFORM -- USING -- GIVING may execute a relocatable section since the PERFORM -- REPLACING serves as a modify-copy and can take place only during the actual compilation of the section.

A NOTE statement may also appear within a section's head (not shown in formats). When used, it serves only to document the function of the section and in no way influences the compilation or object program which the compiler creates.

IV PROCEDURE DIVISION

This section describes the format and specifications for each type of procedure sentence. Certain conventions are applied in using procedure sentences to formulate a problem. Minimum general conventions are:

A. Words in source vocabulary should not be used as data names.

B. Words in a sentence should be separated by at least one space. The space separator is optional if the words are bound by +, -, *, /, #, (), ", =, and ,.

C. Each sentence should end with a period (.). The decimal point is illegal when used as follows: X=A+B - (Q*C + Y * 12.)*4, since it would be interpreted as ending the sentence.

D. Subscripts should be enclosed in parentheses and may be written apart or as part of the array name.

E. If two arithmetic expressions are written side by side they should be separated by a comma. For example, when we write X (A+B, C#2) or A*3.14, SIN ALPHA. In all other cases, the comma may be used as a separator in conjunction with, or instead of, the space.

To facilitate the presentation and explanation of the sentences and their formats, the following editorial conventions are used:

A. Key Words -- where shown in the sentence formats, underlined upper case words are required to complete the meaning of the sentences. These words must be correctly spelled.

B. Noise Words -- upper case words without underlining are shown in sentence formats to improve the readability of the language. These words may or need not be used as desired. If they are used they must be correctly spelled.

C. Operands -- lower case words indicate types of operands supplied by the user.

D. Choices -- entries enclosed in braces show available programmer choices. One entry should be selected from those within a set of braces.

E. Options -- entries enclosed in square brackets indicate options which the programmer may include or omit. In some cases options have been separated into individual numbered formats.

ADD

FUNCTION: - - TO ADD TWO QUANTITIES AND STORE THE SUM IN EITHER THE LAST NAMED FIELD OR ELEMENT, OR THE SPECIFIED FIELD OR ELEMENT.

FORMAT:

ADD { literal~1 / field~name~1 / element~name~1 } , { literal~2 / field~name~2 / element~name~2 }
[GIVING { field~name~3 / element~name~3 }] [ROUNDED] [IF SIZE ERROR GO TO sentence~name~1] .

ADD

CONVENTIONS:

A. If the "GIVING" option is not present, then the last-named field or element receives the result. The result field or element should not be a literal.

B. Eleven (11) decimal digits is the maximum size of a fixed point operand.

C. Only a numeric literal may be used. If a sign (+ or -) is included, it should appear as the most significant character of the literal. If a decimal point appears in the literal, it is not included in the stored representation, but is used only to align the literal with associated operands.

D. The result will be stored according to the description of the result as specified in the Record Description. The operands and the result may have different formats. Decimal position alignment for operands is automatically supplied according to the type of operation. The decimal position of the result will be aligned before it is stored in the result field.

E. If the ROUNDED option is specified, the result is rounded before placing it in the result field or element.

F. When the ROUNDED option is not used, the number of decimal places in the calculated result will be truncated (if necessary) according to the size of the result field or element.

G. A SIZE ERROR condition arises whenever the number of integral places in the computed result exceeds that which may be stored in the result field. This causes truncation of the most significant digits of the result field before they are stored. This action occurs whether or not the SIZE ERROR option is used.

H. The SIZE ERROR option may only be used in conjunction with the arithmetics. This option causes compilation of additional object coding.

I. Arithmetic is performed either in the fixed point or in the floating point mode. However, if one of the quantities is defined as floating point, the entire operation is performed in the floating point mode. In this case those quantities defined as fixed point are converted to floating point. The mode of the result is always that of the result quantity regardless of the mode of arithmetic.

EXAMPLES:

A. ADD 1, DAY OF DATE OF INPUT GIVING CURRENT~DAY.

B. ADD WEEKLY~FICA OF MASTR~PAYROL, ANNUAL~FICA OF MASTR~PAYROL.

C. ADD ON~HAND~QTY OF MSTR~INVNTRY, ON~ORDR~QTY OF ORDER~FILE GIVING TOTL~INVNTRY OF MSTR~INVNTRY, IF SIZE ERROR GO TO ERROR~RTN~1.

ALTER

FUNCTION: - - TO MODIFY A PREDETERMINED SEQUENCE OF OPERATIONS.

FORMAT:

ALTER sentence~name~1 TO PROCEED TO sentence~name~2,
[sentence~name~3 TO PROCEED TO sentence~name~4 . . .] .

CONVENTIONS:

A. "sentence~name~1", "sentence~name~3", ... are names of GO sentences as defined under Option 1 of the GO verb.

EXAMPLES:

A. ALTER SWITCH~1 TO PROCEED TO ERROR~RTN~2

B. ALTER SWITCH~4 TO PROCEED TO COMPUTE~FICA, SWITCH~9 TO PROCEED TO DEDUCT~RTNS.

ASSIGNMENT

FUNCTION: - - TO EVALUATE AN ARITHMETIC EXPRESSION AND ASSIGN THE RESULT OF THE EVALUATION TO A SPECIFIED FIELD.

FORMAT:

Field~name~1 [ROUNDED] = { field~name~2 / arithmetic expression / literal~1 } .

CONVENTIONS:

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

B. Arithmetic Operators

Operation	Symbol
Add	+
Subtract	- (also used for negation)
Multiply	*
Divide	/
Exponentiation	**

C. Mathematic Functions

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

ASSIGNMENT

D. Expressions are evaluated from left to right according to the following operation priority:

Exponentiation and Functions
Multiplication and Division
Addition and Subtraction

Parentheses may be used to establish a precedence. If they are used, the evaluation is performed from the innermost to the outermost pair.

E. The mode of evaluating an expression is determined from the data description of the variables. If one of the variables is defined as floating point, the entire expression is evaluated in the floating point mode. Those variables defined as fixed point are converted to floating point. The results of both fixed and floating point evaluations are stored according to the mode of the result variable.

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

G. If ROUNDED is specified, the result of the evaluation is rounded before it is placed in the result variable.

H. No more than 50 operations and/or function symbols may appear in a single expression.

CLOSE

FUNCTION: - - TO TERMINATE THE PROCESSING OF BOTH INPUT AND OUTPUT REELS AND FILES, WITH OPTIONAL REWIND AND/OR LOCK.

FORMAT:

CLOSE file~name~1 [WITH { NO LOCK / NO REWIND }] [, file~name~2 ...] .

CONVENTIONS:

A. A "CLOSE file~name" should be executed once and only once for a given file unless the file has been reopened. It will initiate the final closing conventions for the specified file and release its data area.

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

C. If the NO REWIND option is used on a tape file, the tape will remain positioned for reading or writing another file.

D. If neither NO LOCK nor NO REWIND is specified, the tape will be rewound and locked to prevent the tape from being read or written upon.

EXAMPLES:

A. CLOSE MASTR~PAYROL, BOND~FILE WITH NO LOCK,
SUMMARY WITH NO REWIND.

DIVIDE

FUNCTION: - - TO DIVIDE ONE NUMBER INTO ANOTHER AND STORE THE RESULT IN THE LAST NAMED FIELD OR ELEMENT OR THE SPECIFIED FIELD OR ELEMENT.

FORMAT:

DIVIDE { literal~1 / field~name~1 / element~name~1 } INTO { literal~2 / field~name~2 / element~name~2 }
[GIVING { field~name~3 / element~name~3 }] [ROUNDED]
[IF SIZE ERROR GO TO sentence~name~1] .

CONVENTIONS:

A. All conventions specified under the ADD verb apply to the DIVIDE verb.

EXAMPLES:

A. DIVIDE NO~OF~TESTS INTO TOTAL~SCORE GIVING
AVERAGE ROUNDED.

ENTER

FUNCTION: - - TO INDICATE THE INCLUSION OF A SUBROUTINE WRITTEN IN THE GENERAL ASSEMBLY PROGRAM LANGUAGE.

NOTE

The ENTER verb is not yet fully defined. As soon as specifications are complete, they will be provided.

EXCHANGE

FUNCTION: - - TO TRANSPOSE THE CONTENTS OF TWO FIELDS.

FORMAT:

EXCHANGE field~name~1, field~name~2.

CONVENTIONS:

A. The data descriptions of field~name~1 and field~name~2 must be identical.

B. Field~name~1 and/or field~name~2 may be subscripted.

GO

FUNCTION: - - TO DEPART FROM THE NORMAL SEQUENCE OF PROCEDURES.

FORMAT:

Option 1:

GO TO [sentence~name].

Option 2:

GO TO sentence~name~1, sentence~name~2 [, sentence~name~3 . . .]

DEPENDING ON { field~name / element~name } .

CONVENTIONS:

A. In Option 1, if a GO sentence is to be ALTERED, it should be named. The name of the GO sentence is referred to by the ALTER verb in order to modify the sequence of the program. If "sentence~name" is omitted, the compiler will insert an error stop in the object program. Therefore, the GO sentence should be referenced by an ALTER sentence before the first execution of the GO sentence.

B. In Option 2, the field~name or element~name should have a positive integral value. The branch will be to the 1st, 2nd. . . , nth "sentence~name" as the value of the field or element is 1, 2, . . . , n. If the value is zero, or exceeds n (i.e., the number of sentences named) the next sentence in normal sequence will be executed.

EXAMPLES:

A. SWITCH 1. GO.

B. GO TO COMPUTE~FICA.

C. GO TO SHIPMENT, RECEIPT, CHANGE, ADDITION, DELETE DEPENDING ON TRANSACT~COD.

FUNCTION: - - TO TRANSFER 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).

FORMAT:

Option 1: (conditional names):

IF conditional~name, GO TO sentence~name~1.

Option 2 (relational expressions):

IF { field~name~1 / element~name~1 / literal~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 / literal~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 / literal~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 } [{ AND / OR } { conditional~name~3 , / relational~expression~3 } . . . . [{ AND / OR } { conditional~name~20 / relational~expression~20 }]

, GO TO sentence~name~1.

Option 4 (tests):

IF { field~name~1 / element~name~1 / arithmetic~expression~1 } IS [NOT] { POSITIVE / NEGATIVE / ZERO }, GO TO sentence~name~1.

CONVENTIONS:

A. The abbreviations for relations listed on page 7 may be used instead of the English words shown in the above formats.

B. 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 neither POSITIVE nor NEGATIVE.

C. The mode of an expression is determined from the data it operates upon. If one of the quantities is floating point, fixed point quantities are converted to floating point and the evaluation is performed in the floating point mode.

D. For additional details, see "Conditional Fields" (page 5), "Arithmetic Expressions" (page 9), "Relational Expressions" (page 7), and "Logical Expressions" (page 8).

EXAMPLES:

A. Option 1

1. IF MALE, GO TO 789.

B. Option 2

1. 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.

2. IF WEEKLY~FICA OF MASTR~PAYROL ± ANNUAL~FICA OF MASTR~PAYROL EXCEEDS 144.00 GO TO COMP~WK~FICA.

3. 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.

C. 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.

D. Option 4

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

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

MOVE

FUNCTION: - - TO TRANSFER A LITERAL OR THE CONTENTS OF AN ELEMENT, FIELD, GROUP, OR RECORD TO ONE OR MORE OTHER ELEMENTS, FIELDS, GROUPS, OR RECORDS.

FORMAT:

MOVE { literal / element~name~1 / field~name~1 / group~name~1 / record~name~1 } TO { element~name~2 / field~name~2 / group~name~2 / record~name~2 } [{ element~name~3 / field~name~3 / group~name~3 / record~name~3 } ...] .

CONVENTIONS:

A. A numeric literal not enclosed in quotation marks, or a numeric element or field being moved will be aligned in accordance with the decimal point of the destination element or field with truncation or zero fill on either end as required.

B. A non-numeric element or field being moved will be left-justified with space fill on the right if the destination element or field is larger than the source data. The compiler will give a warning if the destination element or field is smaller than the source data. If the warning is ignored, the non-numeric literal, element, or field being moved will be left-justified and truncated on the right as necessary to fit the destination.

C. If a numeric or non-numeric literal is enclosed in quotation marks (") and consists of a single character, or if the literal is a figurative constant, the entire element, field, group, or record is filled with the character specified.

D. If a numeric or non-numeric literal is enclosed in quotation marks (") and consists of more than one character, the specified characters are placed repeatedly in the element, field, group, or record starting at the left (most significant digit) until the entire element, field, group, or record is full. Then the literal will be truncated if necessary.

E. The compiler will only provide for the movement of 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 movements to output.

EXAMPLES:

A. MOVE PART~NUMBER OF MSTR~INVNTRY TO PREV~PART~NO.

B. MOVE ADDRESS OF CHANGE~FILE TO ADDRESS OF MASTR~PAYROL.

C. MOVE MASTR~RECORD OF MSTR~INVNTRY TO HOLD~RECORD.

D. MOVE "0" TO SUBTOTL~AREA.

E. MOVE SPACES TO HEADER~AREA.

F. MOVE "Z" TO PART~NUMBER OF MSTR~INVNTRY.

G. MOVE 9 TO ERROR~CODE.

H. MOVE "ZY" TO PART~NUMBER OF MSTR~INVNTRY.

I. MOVE 001 TO COUNTER.

MULTIPLY

FUNCTION: - - TO MULTIPLY TWO QUANTITIES TOGETHER AND STORE THE RESULT IN THE LAST NAMED FIELD OR ELEMENT OR THE SPECIFIED FIELD OR ELEMENT.

FORMAT:

MULTIPLY { literal~1 / field~name~1 / element~name~1 } **BY** { literal~2 / field~name~2 / element~name~2 }

[GIVING { field~name~3 / element~name~3 }] [ROUNDED]

[, IF SIZE ERROR GO TO sentence~name~1].

MULTIPLY

CONVENTIONS:

A. All conventions specified under the ADD verb apply to the MULTIPLY verb.

EXAMPLES:

A. MULTIPLY HOURS~WORKED OF TIME~CARD BY RATE OF MASTR~PAYROL GIVING GROSS~PAY OF MASTR~PAYROL, IF SIZE ERROR GO TO ERROR~RTN~2.

B. MULTIPLY GROSS~PAY OF MASTR~PAYROL BY 0.03 GIVING WEEKLY~FICA OF MASTR~PAYROL.

NOTE

FUNCTION: - - TO ALLOW THE PROGRAMMER TO WRITE EXPLANATORY MATERIAL IN HIS PROGRAM WHICH WILL BE PRODUCED ON THE LISTING BUT NOT COMPILED.

FORMAT:

NOTE...

CONVENTIONS:

A. Any sentence may follow the word NOTE if the rules for sentence structure are followed.

B. The NOTE sentence should not be named.

EXAMPLES:

A. NOTE THIS SENTENCE IS NOT NAMED BECAUSE NO REFERENCE IS MADE TO IT.

B. NOTE THIS SENTENCE IS USED FOR CLARITY.

C. NOTE IN NO WAY DOES THIS SENTENCE STATE COMPILER OR OBJECT PROGRAM ACTION.

OPEN

FUNCTION: - - TO INITIATE THE PROCESSING OF BOTH INPUT AND OUTPUT FILES. PERFORMS CHECKING OR WRITING OF LABELS, AND OTHER INPUT-OUTPUT FUNCTIONS.

FORMAT:

OPEN [INPUT file~name~1 [, file~name~2...]] [OUTPUT file~name~3 [file~name~4...]]

CONVENTIONS:

A. OPEN should be applied to all files and should be executed before the first READ or WRITE of a file.

OPEN

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

C. OPEN does not obtain or release the first data record. A READ or WRITE should be executed to obtain or release the first data record.

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

E. 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 (i.e., the file is not present) the file will not be OPENED. A print-out indicating the absence of the file will occur, and an end-of-file signal will be sent to the input-output control system of the object program. Thus, when the first READ for this file is met, the end-of-file path for this sentence will be taken.

PERFORM

FUNCTION: - - THE PERFORM EXECUTES A SECTION. UPON COMPLETING THE FUNCTION OF THE SECTION, CONTROL REVERTS TO THE SENTENCE FOLLOWING THE PERFORM.

FORMAT:

PERFORM section~name [{ [USING field~name$_{I1}$, field~name$_{I2}$, ... [GIVING field~name$_{O1}$, field~name$_{O2}$, ...]] / [REPLACING name$_1$ WITH name$_2$, name$_3$ WITH name$_4$, ...] }].

CONVENTIONS:

A. When the USING-GIVING option is used, field~name$_{I1}$, field~name$_{I2}$ . . . and field~name$_{O1}$, field~name$_{O2}$, . . . are considered to be assignment variables.

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.

B. PERFORM . . . REPLACING invokes a copy of the section body in place of the PERFORM sentence. This copy takes place during compilation. As this is being done, the names appearing in the section (name$_1$, name$_3$, . . .) are replaced with new names (name$_2$, name$_4$, . . .).

Option 1

FUNCTION:- -TO ALLOW A LIMITED AMOUNT OF INPUT FROM AVAILABLE DEVICES.

FORMAT:

READ field~name~1 [, field~name~2 . . . field~name~20] FROM hardware~name.

Option 2

FUNCTION:—TO MAKE AVAILABLE FOR PROCESSING THE NEXT LOGICAL RECORD FROM AN INPUT FILE AND TO TRANSFER CONTROL TO THE SPECIFIED SENTENCE WHEN THE END OF THE FILE IS REACHED.

FORMAT:

READ file~name RECORD [, IF END OF FILE GO TO sentence~name~1].

Option 3

FUNCTION:—TO ADVANCE AN INPUT FILE, OR TO COPY RECORDS FROM AN INPUT FILE ONTO AN OUTPUT FILE UNTIL THE SPECIFIED CONDITION IS SATISFIED. THEN THE CURRENT LOGICAL INPUT RECORD IS MADE AVAILABLE FOR PROCESSING. TO TRANSFER CONTROL TO THE SPECIFIED SENTENCE WHEN THE END OF THE INPUT FILE IS REACHED WITHOUT SATISFACTION OF THE SPECIFIED CONDITION.

FORMAT:

READ file~name~1[COPYING ON file~name~2] UNTIL { field~name~1 / element~name~1 / literal~1 }

{ EQUALS / IS EQUAL TO / IS GREATER THAN OR EQUAL TO } { field~name~2 / element~name~2 / literal~2 }

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

CONVENTIONS:

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

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

C. Any number of READ sentences may be stated for the same file. At least one END clause should 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.

D. 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. End of file procedures will not be performed.

E. When a file consists of more than one type of data record, these records automatically share the same memory area. Each type may be of a different size, but every record of the same type should be of the same size. When a READ sentence is executed, the next logical record is made available without regard to its type. Only the data in the current record is accessible. The programmer should employ an IF sentence to determine the type of data record after it has been READ. (See CONTROL~KEY in DATA DIVISION.)

F. In Option 1, processing stops when the sentence is executed and continues after the last value is entered.

G. In Option 3, the abbreviation EQ may be used for equality and the abbreviation GREQ may be used for GREATER THAN OR EQUAL TO.

EXAMPLES:

A. Option 1

1. READ CURRENT~DATE FROM CONSOLE KEYBOARD.
2. READ PARAMETER~1, PARAMETER~2 FROM CONSOLE KEYBOARD.

B. Option 2

1. READ TIME~CARD RECORD.
2. READ MASTR~PAYROL, IF END GO TO FINAL~STOP.

C. Option 3

1. READ TRANSACTIONS UNTIL TRANSACT~COD EQUALS 3.
2. READ MSTR~INVNTRY COPYING ON UPD~MSTR~INV UNTIL STOCK~NUMBER OF MSTR~INVNTRY IS GREATER THAN OR EQUAL TO STOCK~NUMBER OF TRANSACTIONS, IF END GO TO CLOSEOUT.

STOP

FUNCTION: - - TO HALT THE COMPUTER EITHER PERMANENTLY OR TEMPORARILY.

FORMAT:

STOP [RUN] [literal].

CONVENTIONS:

A. If the word RUN is used, the literal will be typed out (if one is specified) and the standard end-of-run procedure will be executed.

B. If the word RUN is not used, the literal will be typed out (if one is specified) and the computer will be stopped in a loop. Operating instructions will be provided to resume processing at the next sentence.

EXAMPLES:

A. STOP 9999.

B. STOP RUN "P01".

SUBTRACT

FUNCTION: - - TO SUBTRACT ONE QUANTITY FROM ANOTHER AND STORE THE RESULT IN THE LAST NAMED FIELD OR ELEMENT OR THE SPECIFIED FIELD OR ELEMENT.

FORMAT:

SUBTRACT { literal / field~name~1 / element~name~1 } FROM { literal~2 / field~name~2 / element~name~2 }

[GIVING { field~name~3 / element~name~3 }] [ROUNDED]

[IF SIZE ERROR GO TO sentence~name~1].

CONVENTIONS:

A. All notes specified under the ADD verb apply to the SUBTRACT verb.

EXAMPLES:

A. SUBTRACT UNION~DUES OF MASTR~PAYROL FROM ADJUSTED~ PAY OF MASTR~PAYROL.

B. SUBTRACT RECEIPTS OF TRANSACTIONS FROM ON~ORDR~QTY OF ORDER~FILE GIVING ADJ~ORDR~QTY, IF SIZE ERROR GO TO ZERO~RTN.

VARY

FUNCTION: - - TO INITIATE AND CONTROL THE REPEATED EXECUTION OF THE SENTENCES IT PRECEDES.

FORMAT:

Sentence~name~1. VARY field~name~1, FROM { field~name~2 / expression~1 / literal~1 } BY

{ field~name~3 / expression~2 / literal~2 } UNTIL expression~3.

: "A set of one or more sentences"
EXIT sentence~name~1.

CONVENTIONS:

A. The sentence is undefined if expression~1 and/or expression~2 are other than arithmetic expressions. If expression~3 is arithmetic, the sentence is also undefined.

B. When the VARY is first executed, the FROM parameter, obtained from evaluating expression~1, is assigned to field~name~1, the control variable. Expression~3 is then evaluated. If the evaluation results in a true truth-value, the sentence following the EXIT is executed.

If a false value is obtained, the sentences following the VARY are performed. Upon reaching an EXIT that has the same sentence~name as the VARY, the BY-parameter, obtained from evaluating expression~2, is added to field ~name~1. Expression~3 is evaluated again. From here on the sentences following the VARY are executed for successive false values of expression~3. The "loop" is said to be satisfied when expression~3 results in a true evaluation and control reverts to the sentences after the EXIT.

C. EXIT may have a sentence~name of its own. Caution should be exercised in transferring control to it, since it causes the BY-parameter to be added to field~name~1.

D. A transfer of control from outside the VARY range into the range is undefined.

E. Any number of VARY-EXIT sentences may be imbedded within the sentences of a VARY-EXIT range.

EXAMPLES:

A.
```
MOVE~TABLE~A. VARY I FROM 1 BY 1 UNTIL I GR 10.
    K = 11 - I.
    MOVE TABLE~A (I) TO TABLE~B (K).
    EXIT MOVE~TABLE~A.
```

B.
```
LOOP~1. VARY P FROM ABS (X-1) BY 1 UNTIL P GR 20.
            D (P) = O
LOOP~2. VARY Q FROM 1 BY 1 UNTIL Q GR 10.
            D (P) = D (P) + A (Q) * X (Q, P).
            EXIT LOOP~2.
            IF ABS D (P) GR L, GO TO SET~L.
EXIT~L1. EXIT LOOP~1
SWITCH~ALPHA. GO TO CALC~STRESS, PRT~1, ERR~5
DEPENDING ON ALPHA.
SET~L. L = D (P)
        GO TO EXIT~L1.
```

WRITE

Option 1

FUNCTION: - - TO DISPLAY A LIMITED AMOUNT OF INFORMATION ON AVAILABLE DEVICES.

FORMAT:

WRITE { field~name~1 / element~name~1 / literal~1 } [, { field~name~2 / element~name~2 / literal~2 } . . . { field~name~10 / element~name~10 / literal~10 }]

ON hardware~name.

Option 2:

FUNCTION: - - TO RELEASE A LOGICAL RECORD TO AN OUTPUT FILE.

FORMAT:

WRITE record~name RECORD.

CONVENTIONS:

A. In option 1, literals may be used to identify the contents of the data-fields or elements displayed or to give meaning to the display.

B. In option 2:

1. An OPEN sentence should be executed before the first WRITE is given for a particular file.

2. After the WRITE is executed, the data in the record is no longer accessible to the programmer.

3. If a record is to be printed, it will be edited as specified on the Data Division form.

4. The data description of the output records must contain a list of all data intended for output. When the WRITE is executed, only the data listed in the record description are moved to the output file. The move is automatic and implied with each execution of the WRITE.

5. The actual writing on tapes and the punching of cards, etc., is controlled by implied routines which are generated by the compiler.

EXAMPLES:

A. Option 1:

1. WRITE "UNMATCHED" CLOCK~NUMBER OF TIME~CARD ON TYPEWRITER.

   NOTE THIS SENTENCE WILL CAUSE THE LITERAL UNMATCHED TO BE TYPED OUT FOLLOWED BY THE CONTENTS OF THE CLOCK~NUMBER FIELD.

B. Option 2:

1. WRITE MSTR~INVNTRY RECORD.
2. WRITE HEADER OF MSTR~INVNTRY.
3. WRITE SUBTOTL~LINE OF SUMARY~REPRT.

V DATA DIVISION

A. GENERAL

1. Overall Approach

Data to be processed falls into three categories:

a. That which is contained in files and enters or leaves the internal memory of a computer from specified areas.

b. That which is developed internally and placed in intermediate or working storage.

c. Constants which are defined by the user. Figurative constants and literals used in procedure statements are not listed in the DATA DIVISION. Tables may fall into any of the above categories. In defining file information, the approach taken is to distinguish between the physical aspects of the file (i.e., File Description) and the conceptual characteristics of the data contained in the file. By physical aspects we mean the mode in which the file is recorded, the grouping of logical records within the physical limitations of the file-media, the means by which the file can be identified, etc. By conceptual characteristics we mean the explicit definition of each logical entity within the file itself.

The contents of a file are divided into logical records for purposes of processing. A logical record is any consecutive set of related information. For example, in an Inventory Transaction File, a logical record could be a single transaction, or all consecutive transactions pertaining to the same stock item. Several logical records may occupy a block (i.e., physical record). A logical record may not extend across physical records. These may be different types of logical records in the same file, e.g., a master record and a detail record. Each type may be of a different size, but every record of the same type should be of the same size.

The concept of a logical record is not restricted to file data but is carried over into the definition of working storages and constants. Thus, working storages and constants may be grouped into logical entities and defined by a Record Description.

File Descriptions and Record Descriptions may be stored on a library tape. The Data Division format described in these specifications is that in which descriptions would be stored in the library. The library format will be the first Data Division format which the compiler will accept since it requires less compilation time than a free format. At a later date, a service routine will be provided to convert free format data divisions to the fixed library format. In the meantime, the user should write his Data Divisions in the fixed library format on the Data Division Form (Fig. 2).

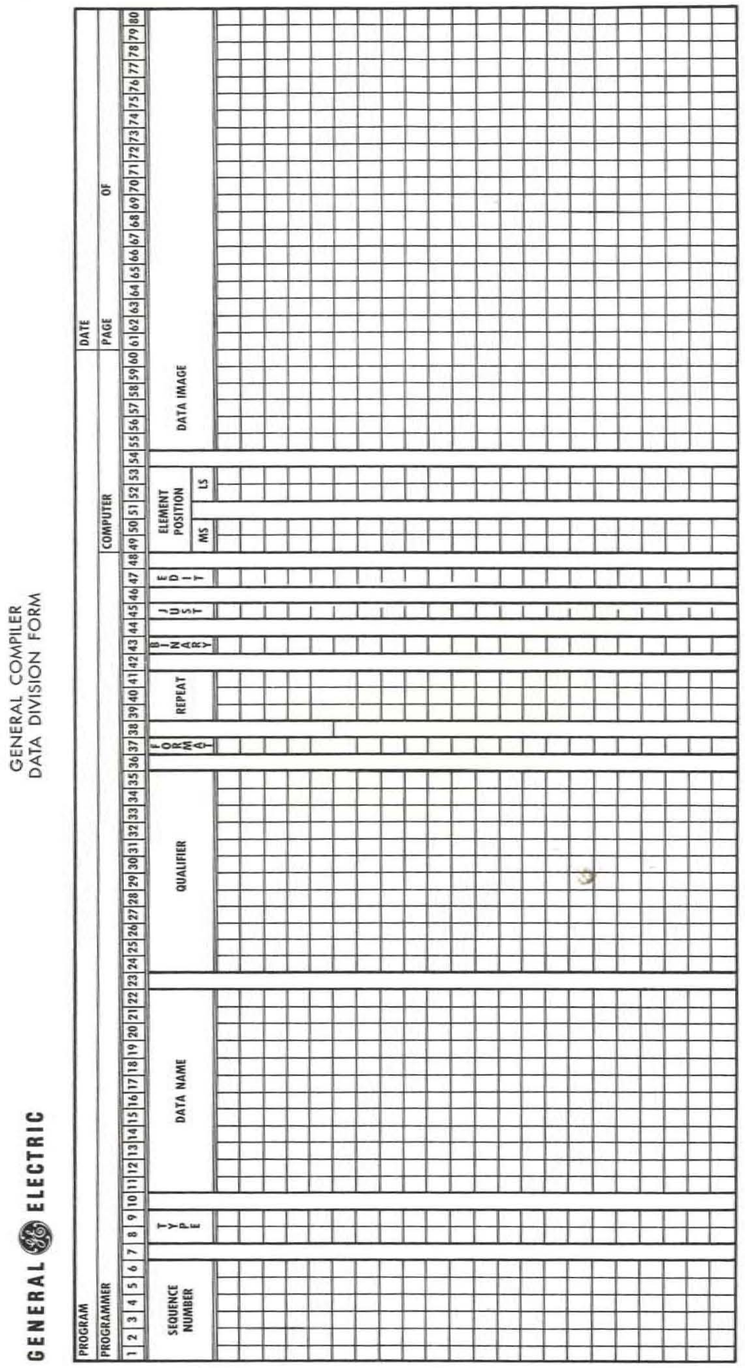


Figure 2 Data Division Form

2. Organization

The DATA DIVISION is subdivided according to types of data. It consists of a FILE Section, a WORKING-STORAGE Section, and a CONSTANT Section.

a. FILE Section

Both of the DATA DIVISION elements (descriptions of files and descriptions of records) appear in the FILE Section. The section contains File Descriptions and Record Descriptions for both label and data records. These two kinds of records are defined in the same manner. However, since the input-output system of the object program must perform special operations on label records, fixed names have been assigned to those label fields on which specified operations must be performed.

b. WORKING-STORAGE and CONSTANTS

These sections consist solely of Record Descriptions and unrelated field or element (subfield) descriptions.

3. Structure

All entries are written on the Data Division Form (Fig. 2). This form is an image of an 80-column punch-card on which a six-digit sequence number is assigned in Cols. 1 through 6. An optional sequence check on these numbers will be provided. The compiler will give a warning when it detects a number out of sequence.

The DATA DIVISION begins with the header:

DATA DIVISION

Each of the three sections begins with its appropriate title, followed by the word SECTION and a period. For example:

FILE, WORKING-STORAGE, OR CONSTANT.

When a section is not required, its name need not appear. These headers start in Col. 8 of the Data Division form.

B. FILE DESCRIPTION

A File Description entry consists of a type indicator, a file name, and a series of independent clauses which define the physical characteristics of the file. The mnemonic type indicator FD is used to identify the start of an entry. Individual clause formats are arranged in alphabetic order; clauses in the "Complete Entry" are shown in the recommended order.

FILE DESCRIPTION

Complete Entry

FUNCTION: - - TO FURNISH INFORMATION CONCERNING THE PHYSICAL STRUCTURE OF A GIVEN FILE.

FILE DESCRIPTION

FORMAT

Option 1:

FD file~name~1 COPY file~name~2.

Option 2:

FD file~name~1, [RECORDING MODE IS BINARY]
[, BLOCK CONTAINS integer~1 WORDS]
[, LABEL RECORDS { ARE / IS } { NONSTANDARD / OMITTED }]
[, CONTROL~KEY IS field~name~1 [OF record~name~1, field~name~2
OF record~name~2 . . .]]
[, SEQUENCED ON field~name~3 [, field~name~4]] .

CONVENTIONS:

A. In Option 1 the entire File Section (File Description and Record Description) for file~name~2 is copied from the library and file~name~2 is changed to file~name~1.

B. The type indicator FD identifies the beginning of the file description entry. It is written under TYPE in Cols. 8 and 9 on the DATA DIVISION Form. The file~name is written under DATA NAME in Cols. 11 through 22.

C. The clauses are written across the form after the file~name. If more than one line is required, the sentence may be continued on the next line, starting in Col. 11 or in any column following 11. It is recommended that sentence continuations be indented to start in Col. 15. Splitting a word over two lines is indicated by a tilde (~) in Col. 7 of the next line. If the programmer prefers not to split a word, he may start the word on the next line as any number of blank columns may be left at the end of a line.

EXAMPLES:

A. FD MASTR~PAYROL, BLOCK CONTAINS 300 WORDS, LABEL RECORDS ARE NONSTANDARD, SEQUENCED ON BADGE~NUMBER.

B. FD MSTR~INVNTRY, RECORDING MODE IS BINARY, BLOCK CONTAINS 1000 WORDS, LABEL RECORDS ARE OMITTED, SEQUENCED ON STOCK~NUMBER.

BLOCK SIZE

FUNCTION: - - TO SPECIFY THE SIZE OF THE PHYSICAL RECORD (I. E., BLOCK).

FORMAT:

[, BLOCK CONTAINS integer~1 WORDS]

BLOCK SIZE

CONVENTIONS:

A. This clause is required except when a physical record contains one and only one complete logical record.

CONTROL~KEY

FUNCTION: - - TO IDENTIFY A FIELD WHICH CONTAINS A VALUE CORRESPONDING TO A TYPE OF DATA RECORD.

FORMAT:

[, CONTROL~KEY is field~name~1 [OF record~name~1,
field~name~2 OF record~name~2 . . .]]

CONVENTIONS:

A. This clause is required only when there is more than one type of data record in a file. It is necessary because the READ verb simply makes the next record in a file available. An IF sentence should be used to test the control field so that the programmer may determine the type of record read. In addition, the object program may take certain implied actions depending on the type of record. It can only do so if it can identify the type of record read.

B. It is recommended that the specified field appear in the same position relative to the beginning of each data record in the file and that it have the same name and description except that it would have a different value for each type of record. The user may, at his own risk, use different field names and locations for each type of record.

EXAMPLES:

A. CONTROL~KEY IS TRANSACT~COD OF TRANSACTION, IDENTIFICATION OF MASTER.

B. CONTROL~KEY IS RECORD~TYPE.

COPY

FUNCTION: - - TO OBTAIN A FILE SECTION FROM THE LIBRARY.

FORMAT: COPY file~name~2.

CONVENTIONS:

A. COPY is used when the library contains the entire File Section.

B. During compilation, the COPY clause is replaced by the File Description and Record Description(s) filed in the library under file~name~2. Thus, the file~name which precedes the COPY clause will replace the file~name appearing within the library.

COPY

EXAMPLE:

A. FD UPDAT~PAYROL COPY MASTR~PAYROL.

LABEL RECORDS

FUNCTION: - - TO INDICATE OMMISSION OF LABELS OR USE OF NON-STANDARD LABELS.

FORMAT:

$$\left[, \underline{\text{LABEL}}\ \text{RECORDS}\ \left\{\begin{matrix}\text{ARE}\\ \text{IS}\end{matrix}\right\}\ \left\{\begin{matrix}\underline{\text{NONSTANDARD}}\\ \underline{\text{OMITTED}}\end{matrix}\right\}\right]$$

CONVENTIONS:

A. Absence of this clause implies that the file contains standard label records.

B. The following four types of label records may appear on the tapes associated with a file. Since the type of label is significant, the fixed record names shown in capital letters have been assigned:

1. BGN~TAP~LABL appears at the beginning of each tape, precedes all other information, and contains information about the tape.

2. BGN~FIL~LABL appears only once preceding the first data record in the file but following the beginning tape label if one is present. This label contains information about the file.

3. END~TAP~LABL immediately follows the last valid data or label record on the tape. The end of the tape label must appear before the physical end of the tape is encountered. When both end of file and end of tape labels are being employed on the same tape, the end of tape label will follow the end of file label. On a MULTIPLE~FILE~TAPE, when all four types of labels are being employed, the end of tape label follows the last end of file label. All other end of file labels are followed by the next beginning file label.

4. END~FIL~LABL appears only once immediately following the last data record on the last reel of a file, and may contain information about the file.

C. Label records are described in the same manner as data records (see RECORD DESCRIPTION). However, if the label records are standard, only variable information (such as the value of a label field) need be supplied.

RECORDING MODE

FUNCTION: - - TO SPECIFY THE ORGANIZATION OR TYPE OF DATA AS IT EXISTS ON THE EXTERNAL MEDIA.

RECORDING MODE

FORMAT:

[RECORDING MODE IS BINARY]

CONVENTIONS:

A. Absence of this clause implies that the file is recorded in decimal.

SEQUENCED

FUNCTION: - - TO INDICATE THE KEYS ON WHICH THE DATA RECORDS ARE SEQUENCED.

FORMAT:

[. SEQUENCED ON field~name~3 [,field~name~4 . . .]]

CONVENTIONS:

A. "field~name~3" represents the major key, "field~name~4" represents the next lower key, etc.

B. This clause does not imply an automatic sequence check at object time.

C. RECORD DESCRIPTION

1. Elements of a Record

A set of related units of data is called a record. A field is the basic unit of data most frequently processed as a specific entity. Some fields may be subdivided into smaller units called elements (sub-fields). Consecutive fields within a record may be grouped under one name. For example, a record in a customer file might include the following:

CUSTMR~RECRD	record name
CUSTMR~NAME	group name
FIRST~NAME	field name
LAST~NAME	field name
CUSTMR~ADRES	group name
STREET	field name
NUMBER	element name
CITY	field name
ZONE	field name
STATE	field name

2. Elements of a Record Description

A record description consists of a set of entries. Entries for each unit of data are made in fixed columns on the Data Division form. One line on the form (one punch card) is required for each unit of data unless more than one qualifier is required. Then one line is required for each qualifier. Units of data are described consecutively as they appear in the record. Each position in the record must be accounted for. A fixed data name, FILL, is used to describe unused positions.

For a unit of data the following information may be entered in the columns indicated:

Type (columns 8 and 9).
Data name (columns 11-22).
Qualifiers (columns 24-35).
Format (column 37). This entry is intended primarily for scientific data. Specific entries in this column have not yet been defined.
Repetitions (columns 39-41).
Binary (column 43).
Justification (column 45).
Editing (column 47).
Element position (columns 49-53).
Data image (columns 55-80). At times, actual values may be written in these columns. When they are, the actual values are always enclosed by quotation marks (").

An entire Record Description may be copied from the library by making the following entries on the Data Division form:

The letter R in column 9.
The record name in columns 11-22.
The word COPY in columns 24-27.
The name of the library record description in columns 29-40.
The word OF in columns 42-43.
The name of the file under which the library record description appears in columns 45-56.

When this procedure is followed, the record description will be copied from the library and the library record name will be replaced by the name in columns 11-22.

3. Specific Entries By Type of Data Unit

a. Input Records

(1) For an input record the letter R is entered in column 9 under type and the record name in columns 11-22. Descriptions of all units of data within the record provide a complete picture of the record.

(2) For a group of fields, the letter G is entered in column 9 under type and the group name in columns 11-22. The descriptions of all units of data within the group provide a complete picture of the group. NOTE: if a group of fields is named, then another group name is required to define the end of the group. This holds true unless the first named group is at the end of a record.

(3) For a field in an input record, the following entries may be made:

(a) The letter F in column 9 under type.

(b) The field name in columns 11-22.

(c) Entries in the format column are possible but not yet defined.

(d) The total number of fields is entered in columns 39-41 under Repet if the field is repeated consecutively. If the number of fields is variable, the letter V is entered in column 39 and the maximum number in columns 40 and 41. A unique character or set of characters, not exceeding field size, must terminate a variable list of fields. The terminating character or set of characters should be enclosed in quotation marks (") and entered after the data image (see paragraph (f) below).

(e) Any character entered in column 43 under Binary indicates that the data in the field is so represented. If no character appears in column 43, the implication is that the data is represented in decimal.

(f) The detailed structure of the field is shown in columns 55-80 under Data Image by entering any combination of the following characters:

Character	Definition
A	Position contains an alphabetic character.
X	Position contains an alphanumeric character.
9	Position contains a numeric character.
B	Position is always blank.
O	Position always contains a zero.
V	Represents an assumed decimal point.
S	Position contains a plus sign if the field or exponent (power of ten) notation is positive, or a minus sign if negative.
I	Position contains a number with a positive or negative overpunch.
R	Position contains a number with a negative overpunch if negative, or no overpunch if positive.
-	Position contains a minus sign if the quantity is negative, or, if positive the most significant digit of the quantity.
T	Position contains a minus sign if the quantity is negative, or a blank (space) if positive.

Any number enclosed in parentheses specifies that the class of data represented by the character preceding the left parenthesis exists in the data that number of times, e.g. A(7)X(5) and AAAAAAAXXXXX both indicate the presence of seven alphabetic characters followed by five alphanumeric characters.

(4) For an element, the letter E is entered in column 9 under type and the element name in columns 11-22. The position of the element in the field is given in columns 49-53. The most significant position is entered in columns 49 and 50 and the least significant position in columns 52 and 53. No other entries are required for an element. Descriptions of elements are listed immediately after the descriptions of the fields in which they are contained.

(5) For condition names, the letter C is entered in column 9 under type and the condition name in columns 11-22. The actual value representing the condition is written between quotation marks (") in columns 55-80 under Data Image. No other entries are required. Condition descriptions are listed immediately after the description of the field for which they are conditions.

(6) For field literals (named fields with fixed values, e.g., a CONTROL ~KEY field), the letters FL are entered in columns 8 and 9 and the field name in columns 11-22. The actual value of the field is written between quotation marks (") in columns 55-80 under Data Image. No other entries are required.

(7) For unused positions in a record, the only description required is the Data Image and the word FILL in columns 11-14.

b. Output Records

(1) Entries for an output record are the same as those for an input record except that a qualifier (identifying the source of the record) may be given in columns 24-35. In other words, if an entire output record is obtained unchanged from an input file, the name of the input file is given as the qualifier. It is not necessary to specify movement of such a record to an output area since this is done automatically when a WRITE is given. In such a case it is not necessary to describe the contents of a record.

(2) Entries for an output group of fields are the same as those for an input group of fields except that one or two qualifiers (identifying the source record and/or file) may be given. If two qualifiers are necessary to identify the source uniquely, the first qualifier is written in columns 24-35 and the second qualifier is written in the same columns on the next line with a tilde (~) in column 7. If the fields within the group are not to be modified in any way for output, it is not necessary to describe these fields.

(3) Entries for an output field are the same as those for an input field with the following exceptions:

(a) One, two, or three qualifiers may be given in columns 24-35 of successive lines. The second and third qualifiers would each carry a tilde (~) in column 7. Here, qualification would identify the source group, record, and/or file. Enough qualification should be given for unique identification. If the field is to be moved to output unchanged, no description other than type, name and qualification need be given.

(b) If the field is to be modified in any way, the following additional entries may be given:

1. The letter L in column 45 indicates left justification of the field, R -- right justification.

2. The letter Z in column 47 indicates that leading zeroes and commas are to be suppressed; a dollar sign ($) means that this symbol is to be floated to the first significant character of the field, suppressing leading zeroes and commas; an asterisk (*) specifies fill (check protection) to the first significant digit of the field, suppressing leading commas.

3. In addition to those characters shown for an input field, the following may be used in the Data Image of an output field.

Character	Definition
E	Position always contains the letter E signifying that a signed exponent will follow. This description is used only when the field is in floating point and its print form is a power of ten notation.
$	Position may contain a $ sign.
.	Position may contain an actual decimal point.
,	Position may contain a comma.

Any printable character (other than those with assigned meanings) appearing in the Data Image of a field to be printed will be printed in the position in which it appears in the Image.

(4) Elements, conditions, field literals, and FILL are not described in output records. However, a description of FILL can be made by describing a literal (see next paragraph 5).

(5) For literals (actual values without names, e.g., headings and titles), the letter L is entered in column 9 under type, and the actual value, enclosed in quotation marks (") is entered in columns 55-80 under Data Image.

D. WORKING STORAGE

The Working Storage Section starts with the header, WORKING~STORAGE SECTION written on the Data Division form starting in column 8. Working Storage is described in the same manner as input data except that unrelated fields and groups of fields need not be grouped into records. The order of entries is as follows:

WORKING~STORAGE SECTION

F E F	field~name element~name . . . field~name	1. Fields and elements
G F E G	group~name field~name element~name . . . group~name	2. Groups of fields, fields, and elements.
R G F E R	record~name group~name field~name element~name . . . record~name	3. Records, groups of fields, fields, and elements.

Conditions and field literals may be included in Working Storage.

E. CONSTANTS

The CONSTANT Section starts with a header, CONSTANT SECTION written on the Data Division Form starting in column 8. Constants are described in the same manner as input data except that actual values are always given instead of data images. Unrelated constants and groups of constants need not be grouped into records. Conditions may not be described. Order of entries is the same as that shown for the Working Storage Section.

VI ENVIRONMENT DIVISION

The basic approach to the ENVIRONMENT DIVISION is to centralize those aspects of the total data processing problem which are dependent upon the physical characteristics of a specific computer. It provides a link between the logical concepts of data and records and the physical aspects of the files on which they are stored.

The ENVIRONMENT DIVISION has been divided into three sentences, each consisting of several clauses:

A. The OBJECT~COMPUTER sentence defines the computer on which the program produced by the General Compiler is to be run.

B. The I~O~CONTROL sentence defines non-standard procedures, rerun, and multiple file tapes.

C. The FILE~CONTROL sentence names and associates the files with the external media.

OBJECT~COMPUTER

FUNCTION: - - TO DESCRIBE THE COMPUTER UPON WHICH THE OBJECT PROGRAM IS TO BE RUN.

FORMAT:

OBJECT~COMPUTER, 225
[, MEMORY SIZE integer~1 MODULE(S)]
[, [integer~2] hardware~name~1 [, [integer~3] hardware~name~2]]
[, ASSIGN OBJECT~PROGRAM TO input~unit~1 [integer~4]
[ON PLUG integer~5]] .

CONVENTIONS:

A. The OBJECT~COMPUTER sentence is not required. It is intended for documentation purposes. If the memory size is given, the object program will run within the specified number of modules. If no memory size is stated, one module will be assumed. The compiler does not use any other information in this sentence, other than to copy the entire sentence onto the edited listing.

B. A warning will be given during compilation if the memory size specified by the user is less than the minimum size required for the program to run.

OBJECT~COMPUTER

C. The compiler will assign the object program to magnetic tape unit zero on plug zero unless specified otherwise at compilation time. Operating instructions will be provided so that other input units may be specified for the reading of the program instructions.

D. The following standard hardware names and/or abbreviations should be used.

Name	Abbreviation
CARD PUNCH(ES)	CP
CARD READER(S)	CR
HIGH SPEED PRINTER(S)	HSP
MAGNETIC TAPE(S)	MT
MASS RANDOM ACCESS FILE(S)	MRAF
PAPER TAPE PUNCH(ES)	PTP
PAPER TAPE READER(S)	PTR
PLUG(S)	PL

E. Magnetic tape numbers and plug numbers may be written ZERO through SEVEN or 0 thru 7. When a plug number is not specified, zero will be assumed.

EXAMPLES:

A. OBJECT~COMPUTER. 225, MEMORY SIZE 2 MODULES, 2 PLUGS, 8 MAGNETIC TAPES, 1 HIGH SPEED PRINTER, ASSIGN OBJECT~PROGRAM TO MAGNETIC TAPE ONE ON PLUG ZERO.

B. OBJECT~COMPUTER. 225, 2 PL 8 MT 1 HSP OBJECT~ PROGRAM MT 1.

FILE~CONTROL

FUNCTION: - - TO NAME EACH FILE AND ALLOW PARTICULAR HARDWARE ASSIGNMENTS.

FORMAT:

FILE~CONTROL. SELECT [OPTIONAL] file~name~1
[RENAMING file~name~2], ASSIGN TO hardware~name~1
integer~1 [ON PLUG integer~2] [, hardware~name~2 . . . .]
[FOR MULTIPLE REEL] [, SELECT . . .].

CONVENTIONS:

A. The key word SELECT will identify the beginning of the information for each "file~name~1".

B. The name of each selected file (e.g., "file~name~1") must be unique within a program.

FILE~CONTROL

C. The key word OPTIONAL is required for input files which will not necessarily be present each time the object program is to be run.

D. The RENAMING option must be included if more than one file uses the same FILE SECTION (both File and Record Descriptions) and that FILE SECTION is included with the source program. That is, "file~name~1" uses the FILE DESCRIPTION written for file~name~2, e.g., when a file is to be processed both as an input and output in the same program. RENAMING "file~name~1" or "file~name~2" implies the sharing of a single FILE SECTION and does not allow these files to be referenced interchangeably in the program.

E. All files used in the program must be assigned to an input or output unit ("hardware~name"). (See Conventions D and E under OBJECT~COMPUTER.)

F. The MULTIPLE REEL option should be included when a magnetic tape file may exceed one reel.

G. The same unit should be assigned to all files existing on the same reel (see MULTIPLE FILE option in I~O~CONTROL sentence).

EXAMPLES:

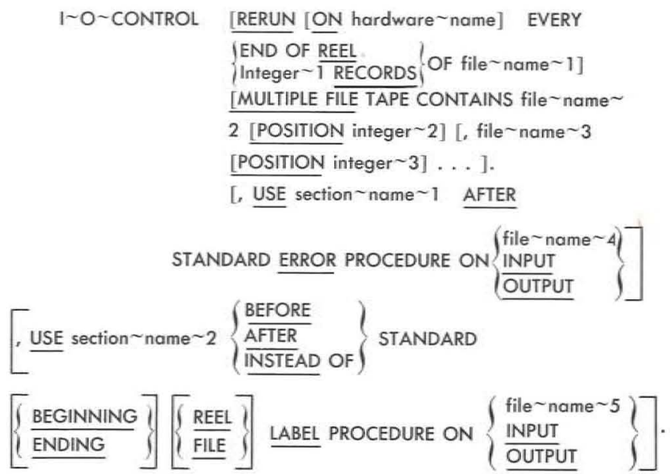
A. FILE~CONTROL. SELECT MASTR~PAYROL, ASSIGN TO TAPE 1, TAPE 2, TAPE 3, MULTIPLE REEL, SELECT TIME~CARDS, ASSIGN TO TAPE 4, SELECT NEW~MST~ PYRL, RENAMING MASTR~PAYROL, ASSIGN TO TAPE 1 ON PLUG 1, TAPE 2 ON PLUG 1, TAPE 3 ON PLUG 1, MULTIPLE REEL, SELECT PAYROL~RGSTR, ASSIGN TO TAPE 4 ON PLUG 1, TAPE 5 ON PLUG 1, MULTIPLE REEL, SELECT UNION~DUES, ASSIGN TO TAPE 6 ON PLUG 1, SELECT BOND~REGISTR, ASSIGN TO TAPE 7 ON PLUG 1.

B. FILE~CONTROL. SELECT MASTR~PAYROL ASSIGN MT 1 MT 2 MULTIPLE SELECT TIME~CARDS ASSIGN MT 3 SELECT NEW~MST~PYRL RENAMING MASTR~PAYROL ASSIGN MT 4 MT 5 MULTIPLE SELECT PAYROL~RGSTR ASSIGN MT 6 MT 7 MULTIPLE SELECT UNION~DUES ASSIGN MT 1 PL 1 SELECT BOND~REGISTR ASSIGN MT 2 PL 1.

I~O~CONTROL

FUNCTION: - - TO SPECIFY NONSTANDARD PROCEDURES, THE POINTS AT WHICH RERUN IS TO BE ESTABLISHED, AND THE LOCATION OF FILES ON A MULTIPLE FILE REEL.

FORMAT:



CONVENTIONS:

A. This sentence is required only when one of the above clauses is desired.

B. If RERUN is specified, it is necessary to indicate when a rerun point is to be established and where the memory dump is to be written.

1. Memory dumps are written in the following ways:

   a. The memory dump is written at the end of each reel of an output file.

   b. The memory dump is written on a separate rerun tape ("hardware ~name").

2. Rerun points may be established by the following conditions:

   a. When the end of REEL option is used for a particular output file and we wish to write the memory dump on the same file. In this case "hardware~name" is not required. For example, RERUN EVERY END OF REEL OF UPD~INVNTRY.

   b. When the end of REEL option is used for an input or output file and we wish to write the memory dump on a separate rerun tape. Here, "hardware~name" should be specified.

   c. When a number of records ("integer~1") of an input or output file have been processed. In this case, "hardware~name" should be specified.

C. 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 need not be given. If any file in the sequence is not listed, the position relative to the beginning of the reel should be given.

D. In the USE clause, section~name~2 will be executed as specified:

1. BEFORE, AFTER, or INSTEAD of the standard input label check.
2. BEFORE a standard output label is created.
3. AFTER a standard output label is created but before it is written.
4. INSTEAD of creating and writing a standard output label.
5. If BEGINNING or ENDING are not included, section~name~2 will be executed for both beginning and ending labels.
6. If REEL or FILE are not included, section~name~2 will be executed for both REEL and FILE labels.

EXAMPLES:

A. I~O~CONTROL. RERUN ON TAPE 7 ON PLUG 1 EVERY END OF REEL OF MASTR~POLICY, MULTIPLE FILE TAPE CONTAINS RATE~TABLE POSITION 6, STATE~CODES POSITION 9, USE BYPASS AFTER ERROR ON INPUT, USE PATTERN~CHEK AFTER LABEL PROCEDURE ON TRANSACTIONS.

B. I~O~CONTROL. RERUN ON TAPE 0 EVERY 1000 RECORDS OF MASTR~POLICY, USE LABEL~RTN INSTEAD OF STANDARD ENDING REEL LABEL PROCEDURE ON TRANSACTIONS.

VII IDENTIFICATION DIVISION

The Identification Division allows the programmer to label and describe the source program. Although this division is not required, if used the compiler will copy the information given onto the edited listing. If a PROGRAM~ID is supplied, the compiler will use it according to standard programming conventions. The following illustration shows the type of information which would usually be included in the Identification Division.

```
IDENTIFICATION DIVISION.
PROGRAM~ID. P 13.
AUTHOR. JOHN DOE.
INSTALLATION. GENERAL ELECTRIC.
DATE~WRITTEN. AUGUST 29, 1960.
DATE~COMPILED. AUGUST 30, 1960.
LAST~CHANGE. AUGUST 31, 1960.
SECURITY. UNCLASSIFIED.
REMARKS. GROSS TO NET RUN.
```

VIII SENTENCE FORM

Fig. 3 shows the Sentence Form on which the Identification, Environment and Procedure Divisions are written. This form is an image of an 80-column punch card. Entries are made as follows:

A. A sequence number may be assigned in Cols. 1-6 for each line (card). An optional sequence check on these numbers will be provided. The compiler will give a warning when it finds a number out of sequence.

B. All division, section and sentence names are written starting in Col. 8. These names are followed by a period and at least one space.

C. Each sentence must start on a new line. However, the first sentence following a name may start on the same line as the name.

D. Although unnamed sentences may start in Col. 8, it is recommended that they be indented four spaces and start in Col. 12.

E. If a sentence exceeds one line, it may be continued on the next line starting in Col. 8. However, it is recommended that continuations be indented eight spaces and start in Col. 16.

F. If a word is split at the end of a line, a tilde (~) is placed in Col. 7 of the second line before continuing with the word. If you do not want to split words across lines, the first line may be space filled through Col. 80 and the word started on the second line.

GENERAL ELECTRIC

GENERAL COMPILER
SENTENCE FORM

PROGRAM | DATE
PROGRAMMER | COMPUTER | PAGE OF

SEQUENCE NUMBER		
000100		IDENTIFICATION DIVISION.
000200		PROGRAM~ID. P13.
000300		DATE~COMPILED. AUGUST 30, 1960.
000400		LAST~CHANGE. AUGUST 31, 1960.
000500		REMARKS. GROSS TO NET RUN.
000100		ENVIRONMENT DIVISION.
000200		OBJECT~COMPUTER 225, MEMORY 2 MODULES,
000300		3 PLUGS, 16 MAGNETIC TAPES, ...
000500		FILE~CONTROL. SELECT MASTR~PAYROL, ...
000600		I~O~CONTROL. RERUN. . . .
000100		PROCEDURE DIVISION.
000200		OPEN INPUT MASTR~PAYROL, TIME~CARDS,
000300		OUTPUT PAYROL~RGSTR, UNION~DUE
000400	~	S, BOND~REGISTR.
001000		MAIN~CHAIN. READ TIME~CARDS, IF END OF
001001		FILE GO TO END~RTN~3.
002000		MATCH~BADGES. READ MASTR~PAYROL COPYIN
002001	~	G ON NEW~MST~PYRL UNTIL BADGE~
002002	~	NUMBER OF MASTR~PAYROL IS GREA
002003	~	TER THAN OR EQUAL TO BADGE~NUM
002004	~	BER OF TIME~CARDS, IF END OF
002005		FILE GO END~RTN~1.

Figure 3 Sample Sentence Form

GECOM
differences

names only 12 char long

More elaborate subscripting (with nested) - also Algol

Can use floating pt literals

Ordinal nos as fig constants

Abbreviated & diff names for relational operators

Begin - End Section (ala algol)
handling generalized closed procedures
specify - input var, output var -
pull a table by a perform
can also copy a closed procedure in place.

only two operand add -

Assign instead of compute for arith exp and built in functions -

no define oper

Enter works to call subroutines - GAsy &
no Examine
Exchange - Move, Switch
IF no compound conditional statements
; , AND ALSO, AND, OR
Exit - not used as needed
Include - not available
MOVE - no move corresponding, ... as FILLING, TO
OPEN - no REVERSED operation
PERFORM - no EXACTLY n TIMES option -
- no VARYING function
- no UNTIL option
- has a REPLACING option = handles copying in place