# RAYTHEON 250 FORTRAN II

PROGRAMMING AND OPERATING MANUAL

RAYTHEON COMPUTER

RAYTHEON

# RAYTHEON 250 FORTRAN II

## PROGRAMMING AND OPERATING MANUAL

Revised July 12, 1965

(Replaces issue of December 22, 1964)

RAYTHEON COMPUTER

2700 SOUTH FAIRVIEW ST., SANTA ANA, CALIFORNIA 92704

RAYTHEON

TABLE OF CONTENTS

# INTRODUCTION

A FORTRAN program appears as a series of statements. On a program form, each line corresponds to a single statement terminated by a carriage return. Statements are normally executed in the order in which they are written (that is to say from top to bottom), however, it is possible to assign five digit numbers to a statement to permit reference to specific statements of a source program. Blanks (spaces) have no significance and are ignored in a compilation.

Raytheon 250 FORTRAN II consists of 23 statements, which may be classified as follows:

1. The arithmetic formula, which allows computation of a result, and whose syntax is described in Chapter III.

2. The 12 control statements, which govern the logic of an object program; they are described in Chapter IV.

3. The 4 input/output statements, whose syntax is peculiar to the Raytheon 250; they are described in Chapter IV.

4. The 2 declarative statements described in Chapter VII.

5. The 4 statements allowing the definition and use of sub-programs. Their usage is described in Chapter V.

6. Format statements which are commonly used by other FORTRAN compilers are not used by 250 FORTRAN.

# CHAPTER I

## CONSTANTS, VARIABLES, SUBSCRIPTS, EXPRESSIONS

1. **CONSTANTS**

Constants may appear in a FORTRAN program and can be of two types:

a. Fixed Point Constants

A positive integer or zero, less than 2097152 and without a decimal point.

Examples: 3732

51

b. Floating Point Constants

A decimal number with an optional decimal point to the left, to the right, or embedded. If there is no decimal point, the point is assumed to the right of the mantissa. This mantissa consists of a maximum of 10 digits. It may be followed by an exponent preceded by the letter E.

This exponent may be preceded by a sign, the absence of a sign being understood as +. The absolute value of the exponent must be less than 38.

Examples: 5.4E2

0.7E-3

20.

The absolute value of a floating point number is zero or understood between $10^{-38}$ and $10^{+38}$.

2. **VARIABLES**

Two types of variables are allowed.

a. Fixed Point Variables

A fixed point variable is represented by a name of 1 to 7 alphanumeric characters, with the first character one of the letters I, J, K, L, M, N.

Examples: IN

K3

JEME

The absolute value of a fixed point variable may be zero or any integer less than 2097152.

b. Floating Point Variables

A floating point variable is represented by a name of 1 to 7 alphanumeric characters, with the first character some letter other than I, J, K, L, M or N.

Examples: PI

A4

TIME

The absolute value of a floating point variable may lie anywhere within the range from $10^{-38}$ to $10^{+38}$.

NOTE: A fixed or floating point variable may be represented by a name longer than 7 characters. The characters beyond the seventh are ignored by the compiler and do not help to distinguish between two different variables. The name of a variable must not cause confusion with some other quantity; in particular, it must be different from the name of a function.

## 3. SUBSCRIPTS AND SUBSCRIPTED VARIABLES

Any element of an array with 1, 2 or 3 dimensions may be represented as a variable with 1, 2 or 3 subscripts. Such a variable is said to be subscripted.

Subscripts may assume only the integral values greater than zero.

A subscript is an arithmetic expression composed of fixed point constants or non-subscripted fixed point variables which are separated by the arithmetic operators + - * / and **.

A subscripted variable is represented by a fixed or floating point variable followed by 1, 2 or 3 subscripts separated by commas and enclosed by a single pair of parentheses.

Examples: ALPHA (I+5*J,K,LAMBDA)

ARRAY ( (IND+7) *J,K+1,MU-7)

For each variable which appears in a subscripted form, the dimension of the array (that is, the maximum values that the subscripts may assume) must be defined in a dimension statement preceding the first appearance of the variable in the source program.

In the object program (after compilation) a two-dimensional array will be stored in contiguous storage locations in the order $A_{11}$, $A_{21}, \ldots,$ $A_{1m}$, $A_{2m}, \ldots,$ that is to say, by column, with the first subscript varying most rapidly and the last least rapidly. The same applies to three-dimensional arrays. The storage order is such that the highest address is associated with the first term of an array, with decreasing addresses for subsequent terms.

## 4. ARITHMETIC EXPRESSIONS

An arithmetic expression is composed of a set of constants, variables (subscripted or not) and functions separated by operators, commas and parentheses.

The operators are:

| | |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| ** | Exponentiation |

### Rules for the formation of arithmetic expressions

a. All constants, variables and subscripted variables appearing in an arithmetic expression must be of the same mode (fixed or floating point). Only special functions permit the use of differing modes in the same arithmetic expression.

However:

A floating point quantity or expression may be an argument of a function used in a fixed point expression.

A fixed point quantity or expression may be a function argument, a subscript (if the item consists only of constants, non-subscripted variables and the operators + - * / **) or an exponent in a floating point expression.

b. Two operators ( + - * / ** ) may not appear in adjacent positions in an arithmetic expression.

An arithmetic expression must begin with a minus sign, a left parenthesis, a variable, a constant or a function.

A plus sign must not immediately follow a left parenthesis, nor may two operators be separated only by left parenthesis.

c. The result of a function is always of a mode defined by the name of the function.

d. The operators + - * / may be used only with operands of the same mode. The mode of the result of the operation is the same as the mode of the operands.

e. If E is an expression, (E) is an expression of the same mode.

f. Exponentiation: Given the expression E**F where E and F are themselves expressions:

  - Neither E nor F may be of the form A**B, but one can write A**(B**C).
  - If E is fixed point, F must be fixed point and the result is also fixed point.
  - If E is floating point, F may be fixed or floating point and the result is floating point

g. Hierarchy of Operations: The following give the hierarchy of operations:

   Exponentiation

   Multiplication and Division

   Addition and Subtraction

That is, in an expression lacking parentheses to define precisely the hierarchy of operations, exponentiations will be executed first, then multiplications and divisions, and finally, additions and subtractions.

Example: A+B*C**F/G means $A + \frac{B \mathrm{x} C^F}{G}$

If there are no parentheses in a set of operations of the same hierarchy, the execution of the operations takes place from left to right.

Example: A*B/C*D means $\frac{A \mathrm{~x~} B}{C} \mathrm{~x~} D$

# CHAPTER II

# FUNCTIONS AND SUBPROGRAMS

Raytheon 250 FORTRAN II allows three types of functions and one type of subprogram.

## 1. STANDARD LIBRARY FUNCTIONS

| Function | Name | Argument | Result |
|---|---|---|---|
| Absolute value | ABSF | floating | floating |
| Square root | SQRTF | floating | floating |
| Sine | SINF | floating | floating |
| Cosine | COSF | floating | floating |
| Tangent | TANF | floating | floating |
| Arc tangent | ATANF | floating | floating |
| Exponential | EXPF | floating | floating |
| Natural ~~Decimal~~ logarithm | LOGF | floating | floating |
| Integer part | INTF | floating | floating |
| Fixed to floating | FLOATF | fixed | floating |
| Floating to fixed | FIXF | floating | fixed |

These standard functions are referred to in arithmetic expressions by the name followed by the argument enclosed within parentheses. The argument may itself be an arithmetic expression of a convenient type.

Example: `A+SINF(ALPHA**2+BETA)/D`

## 2. ARITHMETICALLY DEFINED FUNCTIONS

It is possible to define one or several functions in a source program, by means of a simple arithmetic expression. The definition of this function will be retained throughout the program. This permits definition of functions that are insufficiently used to justify their incorporation in the library. A definition is achieved as follows:

function name (list of arguments) = arithmetic expression

The function name may consist of 1 to 7 characters; as with variables, the first character must be a letter.

If the first letter of the name is I, J, K, L, M or N, the function value will be in fixed point, otherwise it will be in floating point.

Arguments may be fixed or floating point variables (according to the mode established by the first letter) but are never subscripted. They are separated by commas.

The arithmetic expression for an arithmetically defined function follows the previously explained rules; however, it must not contain subscripted variables and may call only standard functions or those functions already arithmetically defined.

Any variable appearing in the expression at the right may be an argument at the left.

At the time of a call upon an arithmetically defined function, the variables in the expression at the right which are not arguments at the left are used with their values at the particular moment of execution. The values of the arguments are computed before the execution of the arithmetically defined function.

```
    Example:      POLYN2(X) = A*X**2+B*X+C

                             or

                  POLYN2(A,B,C,X) = A*X**2+B*X+C
```

At the time of a call upon an arithmetically defined function, the arguments may be any arithmetic expression (but of the mode indicated at the time of the function definition).

The arithmetic definitions of functions must precede the first executable statement of the program.

## 3. FORTRAN FUNCTIONS

This type of function permits definition in FORTRAN language of functions which may not consist of a simple arithmetic expression but which, like arithmetically defined functions, compute only a single result. They are called upon in arithmetic expression, like standard or arithmetically defined functions.

The mode of the function, its name, and the mode of the arguments follow the same rules as the arithmetically defined function. Chapter V discusses the definition of FORTRAN functions.

## 4. SUBROUTINES

It is possible to organize in subroutine form certain actions that are needed at several different points in a single FORTRAN program or in several different programs.

Subroutines are written in FORTRAN.

Unlike functions, subroutines normally do not compute a single result. Moreover, they are not called by reference in an arithmetic expression but by the special "call" statement.

The name of a subroutine consists of 1 to 7 characters, the first of which is a letter.

The mode of arguments is established by the normal conventions for writing in FORTRAN II.

Chapter V discusses the definition of subroutine.

# CHAPTER III

# THE ARITHMETIC FORMULA

The following gives the syntax of an arithmetic formula:

V = E

- where E is any arithmetic expression and V is a subscripted or non-subscripted variable which is fixed or floating in accordance with the first letter.
- V is set equal to the computed value of the expression E.
- If E is floating and V fixed, E is computed in floating point and the integer part of the result (with rounding) is stored in V.
- Conversely, if E is fixed and V floating, the result is computed in fixed point, converted to floating and stored in V.

```
Examples:      A = B

               I = A*X+B

               TAB(I+1,J*K) = A(SINF(PHI)+COSF(PHI))/FLOATF(I)
```

# CHAPTER IV

# CONTROL STATEMENTS AND THE END STATEMENT

These 12 statements and the end statement allow the programmer to define the flow of the program.

## 1. UNCONDITIONAL GO TO

go to stat

Statement stat will be executed next.

Example: go to 15

## 2. ASSIGNED GO TO

go to var, (stat1, stat2,..., statm)

where var is a non-subscripted fixed point variable whose value has been preset by an assign statement, stat1, stat2, etc. are statement names. The statement that will be executed next is the one whose relative position in the list corresponds to the value of var.

Example: go to I, (10,7,15,11)

The statement executed next will be 10,7,15, or 11 depending on values of 10,7,15, or 11 for I.
1 2 3 4 ?

## 3. ASSIGN

Assign stat to var

where stat is a statement number and var is a non-subscripted fixed point variable.

Example: assign 11 to I

This statement presets a variable for the eventual execution of an "assigned go to" statement. Stat must also appear in the list of the corresponding "assigned go to" statement.

The statement "assign 11 to I" and the arithmetic formula "I = 11" are not equivalents. A variable to which a value has been assigned may be used only with an assigned go to until it has been re-established as an ordinary variable.

## 4. COMPUTED GO TO

go to (stat1, stat2,..., statm), var

where stat1, stat2,..., statm are statement numbers and var is a non-subscripted fixed point variable. If, at the moment of execution, the value of var is k, the statement executed next will be the one occupying the $k^{th}$ position in the list.

Example: go to (10,30,7), I

The statement executed next will be 10, 30 or 7, depending on values of 1, 2 or 3 for I.

5. <u>IF</u>

if (expr) stat1, stat2, stat3,

where expr is any arithmetic expression and stat1, stat2, and stat3 are numbers referring to statements. The next statement executed will be stat1, stat2 or stat3, depending on values of less than, equal, or greater than zero for expr.

Example: if (ABSF(U(N)-EPS) )10,11,11

6. <u>SENSE LIGHT</u>

sense light n

where n is 0,1,..., 20

This statement permits the setting of a sense light to "on" or "off". There are, of course, no sense lights on the console to permit visual recognition of status.

If n = 0, all sense lights are set to "off", otherwise sense light n is set "on".

Example: sense light 4

When the Library is loaded, all sense lights are in the OFF state.

7. <u>IF (SENSE LIGHT)</u>

if (sense light n) stat1, stat2

where stat1 and stat2 are statement names and n is 1,2,..., 20.

Stat1 or stat2 is the next statement, depending on an "on" or "off" status for switch n. After execution of the "if (sense light n)" statement, sense light n is always "off".

Example: if (sense light 4) 30,40

8. <u>IF (SENSE SWITCH)</u>

if (sense switch n) stat1, stat2

where stat1 and stat2 are statement names and n is 8, 9,..., 15. The choice of stat1 or stat2 as the next statement depends on the status of the sense switch tested (see the chapter on operating procedures).

## 9. PAUSE

```
pause n
```

where n is an unsigned decimal integer less than 32. The computer stops on this statement and displays in the Operand Lights the octal equivalent of n. Operation of the Enable Switch permits resumption of program execution.

Example: `pause 20`

The computer stops on this statement and displays octal 24 on the console.

## 10. STOP

```
stop n
```

where n is an unsigned decimal integer less than 32. This statement causes a stop without the possibility of resumption. It is used when the programmer desires a final stop. The octal equivalent of n is displayed on the console.

Example: `stop 18`

## 11. DO

```
do stat ind = n1, n2
```

or

```
do stat ind = n1, n2, n3
```

where stat is a statement number. Ind is a fixed point variable. n1, n2, n3 are each either an unsigned fixed point constant or a non-subscripted fixed point variable. If n3 is omitted, it is assumed to be 1. The "do" statement means:

"Do the following statements through stat and repeat them, the first time with ind = n1, then incrementing ind by n3 each time (that is n1 + n3, n1 + 2n3,...) until ind has assumed the greatest value of this series which is not greater than n2."

In other words, this statement group (called "do loop") will be executed N times where N is the integer part of:

$$\left(\frac{n2 - n1}{n3} + 1\right)$$

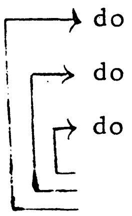
n1 may not be greater than n2 at the start.

During execution of a "do loop" the loop index (the variable being incremented) may be used in calculation, whether in formulas or subscripts.

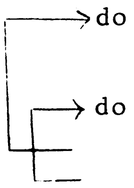
Example:
```
10 do 11  I = 1,N
11 SIGMA = SIGMA + TAB (I)
```

Other "do" statements may appear within a "do loop". It is necessary, however, to respect the following rule:

If a "do loop" contains another "do loop" all the statements of the latter must also be contained within the former. This "nesting" continues in the same manner and may be represented by the following diagram:



The following organization is forbidden:



Similarly, entry into a "do loop" by an "if" or "go to" statement must be to the proper "do" statement.

On the other hand, exit from a "do loop" by an "if" or "go to" statement is allowed without restriction.

> Exception: It is possible to return into a "do loop" after an exit if the following conditions apply:
>
> a. The transfer goes to a statement situated outside of the outermost loop encompassing the transfer statement.
>
> b. The section of program executed outside of the loops modifies no indices or indexing parameters related to control of the loops.

Index values upon exit from a loop: When control passes from a "do loop" in the normal manner (that is, when the looping is finished, the control passes to the statement after the last statement of the loop), the exit is said to be normal. After a normal exit from a "do loop" the value of the indexing variable is equal to the value immediately greater than the last value for execution.

If the exit is made by a transfer from within the loop, the current value of the index is available for any desired use. If the exit takes place from the interior of nested loops, the values of all the loop indices are available.

Last Statement of a do loop: The last statement of a "do loop" must not be a transfer statement. In case of this need, the "continue" statement described in the following paragraph permits proper closing of the loop.

## 12. CONTINUE

continue

This is a statement which causes generation of no instructions for the object program. It is normally used as the last statement of a "do loop".

As an example of a program requiring the "continue" statement, consider the following table search:

```
10 do 12 I = 1,100
      if (ARG - VAL(I) ) 12,20,12
12 continue
13 ...
```

This program examines the table of 100 values of VAL until it finds an element equal to ARG.

At this moment, control is transferred to statement 20, with the reference value of I available. Flow passes to statement 13 only if no element equal to ARG is detected.

## 13. END

end

This statement has no effect upon the object program.

It must be the last statement of a program.

It signals the FORTRAN compiler that the preceding statements constitute the entire source program.

# CHAPTER V

# STATEMENTS RELATED TO FUNCTIONS AND SUBPROGRAMS

It is possible to write FORTRAN subprograms to be used with other programs. These subprograms may themselves call upon subprograms written in FORTRAN.

This chapter discusses two possible types of subprograms written in FORTRAN, defined respectively by the "subroutine" statement and the "function" statement, and the two statements which control transfer to and from subprograms: "call" and "return".

In spite of the similarities between subprograms of the "function" or "subroutine" types, it is convenient to remember the following differences:

a. "Function" subprograms allow the computation of only a single resulting value.

b. The calling of a "function" is taken care of by the FORTRAN compiler when the name of the function appears in an arithmetic expression. By contrast, a "subroutine" may be called only through use of a "call" statement.

Each of these two types allows construction of independent subprograms and conforms to the programming rules of FORTRAN.

Subprograms may be compiled and debugged separately, but each time they are used with a main program they must be compiled beforehand.

## 1. FUNCTION

function name (arg1, arg2,..., arg n)

where "name" is the symbolic name of the function.

The arguments (there must be at least one) are non-subscripted variables.

The name consists of 1 to 7 alphanumeric characters, with the first one a letter. If the first letter is I, J, K, L, M or N, the function value will be given in fixed point, otherwise it will be in floating point.

The name of the function must not appear in a "dimension" statement for the subprogram or in such a statement for the programs using the subprogram.

The "function" statement is the first of the statements forming a subprogram. This subprogram must not contain any other "function" or "subroutine" statement or any arithmetically defined functions.

A subprogram beginning with the "function" statement is called in the main program by an arithmetic formula which contains the function name followed by the list of arguments enclosed within parentheses. These arguments must agree in number, order and mode with the list of formal arguments appearing in the "function" statement. Within a function, the name of the function must appear either at the entry point or at least on the left side of an arithmetic formula.

Therefore, the result obtained by execution of the function may be sent back to the calling program.

The arguments enumerated between the parentheses following the function name being called may have different names than those that appear in the list of formal parameters in the "function" statement. However, it is proper to respect the following rules:

a. The list of arguments appearing in the "function" statement (actual arguments) must agree in number, mode and order with the corresponding list in the calling program.

b. If an argument is an array, this array must be declared (with the same number of subscripts) in a "dimension" statement in the subprogram and in the main program.

c. No argument appearing in the list of a "function" statement may appear in a "common" statement for the "function" subprogram.

   The actual arguments in the calling list for a function may be arithmetic expressions or arrays.

## 2. <u>SUBROUTINE</u>

```
subroutine name (arg1, arg2,..., arg n)

     Example:     subroutine MULMAT (A,B,C,L,M,N)
```

The arguments (if any) are non-subscripted variables or arrays. The name of the subprogram consists of 1 to 7 characters, the first being a letter.

The name of the subprogram should not conflict with any other quantity, either in the subprogram itself or in the calling program. The "subroutine" statement must be the first of those forming the subprogram. The subprogram must not contain any other "subroutine" or "function" statement or an arithmetically defined function.

Transfer to the subprogram is effected by the "call" statement.

When an argument is the name of a dimensioned variable, it must appear in a "dimension" statement of the subprogram. The corresponding argument of the "call" statement must appear in a "dimension" statement of the main program.

The results of "subroutine" subprogram can be transmitted to the calling program, either by variables declared in a "common" statement (see Chapter VII) and necessarily appearing on the left side of an arithmetic formula belonging to the subprogram, or by arrays appearing in the list of arguments.

## 3. CALL

call name (arg1, arg2,..., arg n)

where "name" is the name of a subprogram defined by a "subroutine" statement.

Example: call MULMAT (X,Y,Z,10,11,15)

An argument may be any arithmetic expression or an array.

The "call" statement permits initiation of the execution of the subprogram in question. The list of arguments in the "call" statement must agree in order and mode with the list of the corresponding "subroutine" statement.

## 4. RETURN

return

This statement terminates the execution of a "subroutine" or "function" subprogram and returns control to the main program. It is not necessarily the last statement of a subprogram, and any number of "return" statements may appear in the same subprogram.

# CHAPTER VI

## INPUT/OUTPUT STATEMENTS

1. READ TAPE

read tape A,B,C,D,...

where A,B,C,D,... are fixed or floating point variables with or without subscripting.

This statement causes reading of a set of values into storage from the Flexowriter paper tape reader. The number of values is equal to the number of variables in the "read tape" statement list. These values belong (in the order of their appearance) to A,B,C,D,...

2. READ KEYBOARD

read keyboard A,B,C,D,...

where A,B,C,D,... are fixed or floating point variables with or without subscripting.

This statement causes reading of a set of values into storage from the Flexowriter keyboard. The number of values is equal to the number of variables in the "read keyboard" list. These values belong (in the order of their appearance) to A,B,C,D,...

3. PRINT

print E1 E2 E3 ...

where E1 E2 etc. are elements of the "print" statement list.

An element for printing may have 3 different forms:

a. [literal]

where the literal consists of less than 30 characters.

b. A [literal, punctuation,x,y]

where A is a fixed or floating point variable with or without subscripting.

"Literal" is any list of less than 30 characters and will be printed in front of the value for the variable A.

"Punctuation" is any character which will signal the end of the number being printed (note that for input the only allowable characters for designating the end of a number are: space, tabulation and carriage return).

"x" is the number of digits to the left of the decimal point.

"y" is the number of digits to the right of the decimal point.

c. A [punctuation,x,y]

This format is identical to the one just described except that the literal has been omitted.

Any characters are allowed as punctuation or in a literal except comma and carriage return; spaces are not eliminated; the decimal point character is executed as a carriage return.

x and y are always positive or zero.

$x + y$ must be less than 11.

$x = 0$ and $y \neq 0$ cause floating point printing of a floating point variable, with y digits of mantissa.

$x \neq 0$ and $y \neq 0$ cause fixed point printing of a floating point variable, with x digits before the point and y digits after.

$x \neq 0$ and $y = 0$ cause fixed point printing (without a decimal point) of a fixed point variable, with x the number of digits.

The "print" statement causes printing on the typewriter.

Example: print [STOP --] I [INTERATION NUMBER,.,4,0]

4. PUNCH

punch E1 E2 E3 E4 ...

This statement, whose syntax is like that of "print", causes paper tape punching.

Note: The sign is always printed and punched: space for +, explicit −.

# CHAPTER VII

# DECLARATIVE STATEMENTS

These two statements are not executable. They give information to the compiler. They must appear at the front of a FORTRAN program.

## 1. COMMON

```
common A,B,...
```

where A,B,... are names of variables or arrays.

The variables and the arrays appearing in "common" statements are assigned storage relative to a fixed address (whose value depends on the size of the machine's storage). This permits assignment of the same storage to variables used by a main program and by various subprograms.

The array names appearing in a "common" statement must appear in a "dimension" statement within the same program.

The sequence of assignment of storage corresponds to the order of appearance of the variables in the "common" statement.

There may be only one "common" statement in a FORTRAN program. It must be the first statement of a main program. In a subprogram, it must appear immediately after the "subroutine" or "function" statement.

## 2. DIMENSION

```
dimension V1, V2, V3,...
```

where each $V_i$ is the name of a variable subscripted by 1, 2 or 3 positive integers.

Example: `dimension A(10), B(5,15), C(3,4,5)`

The "dimension" effects assignment of object program storage for arrays.

Each variable which appears in a subscripted form within a program must appear in a "dimension" statement.

The maximum dimensions of arrays are specified in a "dimension" statement. In an object program, subscript values must never exceed the values specified in the "dimension".

For example, dimension MAT (10,11) means that MAT has two dimensions, that the first subscript never exceeds 10, and that the second subscript never exceeds 11.

A single "dimension" statement can be used to define several arrays.

All "dimension" statements of a FORTRAN program must appear at the front, immediately after the "common" statement (if any).

# CHAPTER VIII

# PROGRAM AND DATA FORMATS

## 1. PROGRAM

A FORTRAN statement occupies a maximum of one line and is terminated by a carriage return.

Statement key words are always written in small letters:

go to, if, if (sense light),

if (sense switch), assign,

do, pause, stop, end,

continue, function, subroutine,

call, return, dimension, common,

read tape, read keyboard, sense light, print and punch.

All other letters must be large.

Spaces, delete codes and blank tape are ignored by the compiler.

The source program may be punched in a paper tape. The compiler reads it with the Flexowriter reader. It may also be typed directly into the compiler from the keyboard.

A line of commentary may appear between two FORTRAN statements. A comments line must begin with a small c. The comments may be written in any large or small characters. The end of the comments line is indicated by a carriage return.

## 2. DATA

Data may be read from the Flexowriter reader or keyboard by the use of read tape or read keyboard statements.

Data format is as follows:

### a. Fixed point variable

| | | |
|---|---|---|
| sign | value | TC |

| | |
|---|---|
| Sign: | + or - sign. Absence of a sign is interpreted as +. |
| Value: | The value of the fixed point variable (without a decimal point). The value must be less than 2097152. |

TC:        A terminal character which can be:

space

tabulation

carriage return

Leading spaces or zeros are eliminated.

Example: ␣ - 13 ␣

b. Floating point variable

| sign | value | E | sign | exponent | TC |
|---|---|---|---|---|---|

Sign: + or - sign. Absence of a sign is interpreted as +.

Value: The value of the variable, with a decimal point normally required.

E: The letter E.

Exponent: An integer (no decimal point), with an absolute value less than 38. The exponent gives the power of 10 which is the scale factor for the value.

TC: A terminal character which can be:

space

tabulation

carriage return

Leading spaces and zeros are eliminated. The exponent (letter E, optional sign and exponent value) may be omitted. In this case, the assumed exponent is zero. If the exponent is expressed, the decimal point may be omitted and will then be assumed at the right of the mantissa.

Examples of representations of the value 1:

␣ 1. ␣

␣ + 1 E 0 ␣

␣ . 1 E 1 ␣

During reading of a number, delete codes and blank tape are ignored. Reading of the lower case code causes cancellation of all that has just been read and allows re-initialization of the reading of the number in question.

## 3. COMPILATION OF A FORTRAN PROGRAM

The operating procedures are described in Chapters IX-XII.

Compilation takes place in two passes with inter-pass representation of the program in an intermediate language tape.

The object program is in binary.

The "subroutine" and "function" subprograms must be compiled before the main program.

```
C         NUMBER GAME DEMONSTRATION ROUTINE
          SUBROUTINE LIST MATRIX
          DIMENSION N(7,3)
          COMMON N
          PRINT [.ROW 1 ROW 2 ROW 3.]
          DO 300 I=1,7
          PRINT [.]
          DO 300 J=1,3
300       PRINT N(I,J)[ ,4,0]
          RETURN
          END
          SUBROUTINE INTERCHANGE ELEMENTS
          DIMENSION N(7,3),NTEMP(7,3)
          COMMON N
10        PRINT [.ROW? ]
          READ KEYBOARD L
          IF(L-1)10,1,20
20        IF(L-2)10,3,30
30        IF(L-3)10,3,10
1         L=4
3         M=1
          L=L-1
          DO 5 I=1,7
          DO 5 J=1,3
          NTEMP(I,J)=N(M,L)
          IF(M-7)5,4,4
4         M=0
          L=L+1
          IF(L-3)5,5,6
6         L=1
5         M=M+1
          DO 2 I=1,7
          DO 2 J=1,3
2         N(I,J)=NTEMP(I,J)
          RETURN
          END
          DIMENSION N(7,3)
          COMMON N
          K=10
          DO 200 I=1,7
          DO 200 J=1,3
          K=K+1
200       N(I,J)=K
100       PRINT [..PICK A NUMBER AND TELL ME][ WHICH ROW IT IS IN.]
          NN=0
400       CALL LIST MATRIX
          NN=NN+1
          IF(NN-3)500,600,600
500       CALL INTERCHANGE ELEMENTS
          PRINT [.NOW WHICH ROW IS YOUR NUMBER IN?.]
          GO TO 400
600       PRINT [.ROW? ]
          READ KEYBOARD L
          I=4
          IF(L-3)700,800,600
700       IF(L-1)600,800,800
800       PRINT N(I,L)[.THE NUMBER YOU SELECTED WAS  ,.,4,0]
          PAUSE 7
          GO TO 100
          END
```

# CHAPTER IX

# OPERATION OF FORTRAN II ON THE RAYTHEON 250

## I. GENERAL

The compilation of a FORTRAN program requires two passes. The subsequent phase is the execution of the program.

The two passes take place under control of two compiler segments; and the execution phase requires that the standard subroutines and functions be in the machine (from the FORTRAN function tape).

During the first pass the source program is read and a tape is punched with an intermediate language more readily understood by the machine.

During the second pass, the intermediate language is translated to machine language and the resultant program is punched into another tape.

## II. OPERATING INSTRUCTIONS FOR PROGRAM COMPILATION

The loading of the first pass, second pass, and FORTRAN subroutines is accomplished with the bootstrap (fill) procedure.

Two versions of the compiler are available; one which makes use of the high-speed Photoreader, and one which uses the Flexowriter only.

### A. Description of loading with the bootstrap procedure.

1. Depress Enable and Breakpoint switches.

2. Mount the Phase I tape in the selected reader. Verify that the corresponding bootstrap plug is affixed to the rear of the computer.

    a. Phase I Flexowriter version must be mounted in the Flexowriter reader.

    b. Phase I Photoreader version must be mounted in the HSR Photoreader.

3. Raise the console Fill switch. After the tape begins moving, raise either or both the Enable and/or Breakpoint switches.

4. After the tape stops, depress the Fill switch.

5. Depress both the Enable and Breakpoint switches, strike the I key on the Flexowriter keyboard, and then raise both the Enable and Breakpoint switches.

Yes, with parity on

The tape should then be read to the end and the computer should halt with the console Operand Lights displaying octal 36.

If the computer halts during loading with the console Operand Lights displaying octal 10, incorrect reading has occurred and it is necessary to restart.

B. Operating procedures for the first pass.

1. If the first compiler section is not in storage, load it with the bootstrap procedure.

2. Advance the tape in the Flexowriter Punch so as to obtain about eight inches of leader, then make sure that the Flexowriter Punch switch is off. Position the symbolic tape in the mechanical reader.

3. To start the compilation process, depress the Enable switch and the Breakpoint switch, strike the I key, then raise the Enable switch. The symbolic tape is then read and translated into the intermediate language.

4. If the Breakpoint switch is down, the compiler halts upon recognition of an end statement occurring within a subprogram (to permit loading of another tape). The compilation process resumes when the Breakpoint switch is raised (at which time it should again be depressed).

5. The compilation process may be interrupted by detection of source program errors (see Chapter X).

   When the program is completely translated (signaled by an end statement in the main program), the computer halts. The Operand Lights then contain octal 36 and the Parity Light is on (HLT 36). Punch U on tape.

6. Advance the tape in the Flexowriter Punch so as to obtain about 10 inches of trailer, then remove the punched program and make sure the Flexowriter Punch switch is off.

7. To start the compilation process of another symbolic program, it is not necessary to reload the first pass. Do again only those operating steps associated with the first pass (Step B).

C. Operating procedures for the second pass.

1. Load the second pass with the bootstrap procedure.

   a. If Phase II has loaded correctly, the entire tape will have been read and control will go to the Octal Utility Package, thereby causing the Flexowriter keyboard input light to come on.

b. If the computer halts during loading and the console Operand Lights display octal 10, incorrect reading has occurred and it is necessary to restart.

2. After Phase II is in memory, two cases are possible.

a. First Case. The first pass translation of a symbolic program has just been accomplished, and the machine has not been turned off. Then:

i. Advance the tape in the Flexowriter Punch so as to obtain about eight inches of leader. Then make sure that the Punch switch is off.

ii. Place the intermediate tape in the selected reader.

a. If the Flexowriter version is being used, the intermediate tape must be mounted in the Flexowriter reader.

b. If the HSR Photoreader version is being used, the intermediate tape must be mounted in the HSR Photoreader.

Strike the "T" key on the Flexowriter typewriter. The second pass of the compilation then begins. The computer processes 256-word blocks of the intermediate tape and punches out the compiled program with O.U.P. pre-addressing for each block.

If a block of the intermediate language is not read properly, an octal 37 appears on the console, and the Parity Light is turned on. In this event, reposition the tape at the beginning of the block, depress the Enable switch and the Breakpoint switch and then raise both switches. The compilation process resumes normally. (The blocks are separated by about ten inches of blank tape.)

At the end of the compilation, HLT 35 appears on the console.

b. Second Case. The first pass translation of a symbolic program has been followed by the translation of one or more source programs or by the turning off of the machine. It is then necessary to perform an extra operation on the intermediate tape.

First depress the Breakpoint switch, then find the last block on the intermediate language tape and read it in the selected reader. Remember that each block is preceded by about ten inches of blank tape and starts with the letter G (code 47).

In order to achieve this reading, type the following characters:

```
11F         (if using Flexowriter Reader)
                                          or
16$,11F     (if using Photoreader)
```

It is then possible to proceed as with the first case.

## III. OPERATING INSTRUCTIONS FOR PROGRAM EXECUTION

A. Load the standard subroutines by bootstrap method if they are not already in storage.

1. If the program execution phase (Library of subroutines and functions) has loaded correctly, the Flexowriter keyboard input light will come on since the octal utility package is in control.

2. If the computer halts during loading and the console Operand Lights display octal 10, incorrect reading has occurred and it is necessary to restart.

B. Load the compiled program in the selected reader. To accomplish this, it is necessary to type the following characters:

| | | |
|---|---|---|
| F | (if using Flexowriter Reader) | or |
| 07$,F | (if using Photoreader) | |

After each of these steps, the Flexowriter light should be on. If not, restart the operation.

C. To begin execution strike the T key.

# CHAPTER X

# ERROR DETECTION

## I. GENERAL

Prior to compilation, the program should be closely checked for errors. Print statements should be carefully checked for proper format (commas, brackets, etc.) and maximum number of literal characters.
A carriage return must be used only at the end of a statement.

The first pass of the compilation process is interrupted in case of a syntax error.

The Parity Light is turned on and an octal number is displayed in the Operand Lights. This number is indicative of the error.

To continue the compilation process, depress the Enable switch and the Breakpoint switch, then raise the Enable switch. (In some cases, it is necessary to do this operation several times.)

The second pass of the compilation indicates if the program is too large for the machine. In the case where it is too large, the number of words lacking is printed out in octal.

## II. LIST OF FIRST PASS ERROR INDICATIONS

| Octal Code | Significance |
|---|---|
| 02 | Subscript raised to a floating point power |
| 03 | Mode error |
| 04 | Too few or too many right parentheses |
| 05 | Mode error in a do statement |
| 06 | Unknown statement (the statement is read as if commentary) |
| 01,07,11 | Incorrectly positioned operator (for example, two consecutive operators) |
| 12 | Statement name or dimension error |
| 13,15,17 | Too many statement names, constants, functions |
| 14,22 | Too few or too many parameters |
| 16 | Number of formal parameters greater than 9 |
| 20 | Too many functions |
| 21 | Punctuation error, incorrectly placed comma |
| 23 | Number of names greater than 111. |

Except in the case of error message 23, the compilation process may be resumed as previously described. The compilation process ends with HLT 36.

# CHAPTER XI

# COMPILER LIMITS

The compiler allows 111 names of variables, FORTRAN functions, FORTRAN subprograms, and arithmetically defined functions. However, in FORTRAN functions and subprograms, the names of variables not declared in a common statement are local, except for one case: within a FORTRAN function, the name which is itself the name of the function. The number of local names does not count in the total number of names in the program. Formal parameters are not counted either.

Example:

```
F (A, B) = INTEGER (A**2 + B**2)
function FACT (X)
FACT = 1.
3 if  (X)  1, 1, 2
2 FACT = X*FACT
X = X - 1
go to 3
1  return
end
read Q , R
Z  =  FACT  (F(Q,R))
print Z [ . , 6 , 1]
end
```

The number of names counted for the above program is 6. (These names are underlined.)

On the other hand let:

- E be the number of statement names
- L be the number of lists of statement names (such lists are found in the commands - if, computed go to, and assigned go to.)
- EL be the number of statement names in these lists
- B be the number of do loops
- F be the number of FORTRAN functions, FORTRAN subprograms and arithmetically defined functions.
- C be the number of constants
- M be the number of array names

Then the following relation must hold:

$$E + L + EL + B + F + C + M < 124$$

Finally, if PF is the number of formal parameters, then a necessary relation is:

$$4F + PF < 70$$

# CHAPTER XII

## COMPUTER REQUIREMENTS

The following equipment is required to compile programs with the Raytheon 250 FORTRAN II system.

A. A Raytheon 250 computer with:

1. The capability of executing an instruction from the "A" register.
2. A 16 word line 00.
3. Lines 1 - 16 (octal) of 256 words each.
4. Either a 16 or 256 word line 17 (octal).
5. Flexowriter.

B. Optional equipment that the FORTRAN II compiler can make use of includes:

1. An HSR-1 Photoreader.
2. A bank of 8 external sense switches.

The complete Raytheon 250 FORTRAN II system consists of three separate sections of binary tape. These sections are referred to as Phase I, Phase II, and the Execution Phase respectively.

There are two systems, one using the Flexowriter paper tape reader, the other using the HSR Photoreader. Each system is further divided into three sub-systems which determine the means of inputting the source program. The first allows the source program to be typed in through the Flexowriter. The second allows the source program to be input via the Flexowriter paper tape reader. The third allows either of the preceding two and the option is made by setting sense switch 0 false for typed input and true for paper tape input. If sense switches are to be used, they should be connected to the computer in such a way as to exhibit their condition on lines 10-17 (octal).