ELLIOTT

900

Volume 2: PROGRAMMING INFORMATION

Part 1: PROGRAMMING LANGUAGES

Section 3: FORTRAN

Contents

Page

Page

Page

Page

Printed in England by
Engineering Unit, English Electric Computers Ltd.

Chapter 1: SYSTEM SPECIFICATION

1.1 Minimum Configuration

The 900 FORTRAN system described in this document is designed for use on the 903 computer. The minimum configuration is a 903B with 8192 words of store, paper tape reader and punch.

1.2 Method of Operation

The compiler translates FORTRAN programs into SIR. The resultant SIR program is subsequently translated and run in conjunction with a special FORTRAN functions package.

1.3 The Language

The language incorporates most of the features of basic ASA FORTRAN. It incorporates, in addition, certain features of full ASA FORTRAN.

1.4 Form of Distribution

900 FORTRAN is distributed as a set of two binary tapes:-

(1) The Translator

(2) The Run Time Package

Further details are given in Chapter 9.

1.5 The 900 FORTRAN Representation

900 FORTRAN is paper tape orientated. The characters recognised by the SIR internal code (2.1.1 para 1.5) may be used in punching FORTRAN programs.

Certain characters, which are not part of FORTRAN, may only be used in Alphanumeric text and in SIR instructions within the FORTRAN program.

1.5.1 The Characters Erase, Runout and Halt

The characters Erase and Runout may appear anywhere and are always ignored. The character Halt is used for stopping the computer at the end of a data tape or tape of FORTRAN text, in order that the data or FORTRAN text may be continued on another tape. Halt should only be used at the beginning of a new line.

1. 5. 2 Punching Instructions

An example is given in 1. 5. 6.

(1) A 900 FORTRAN program may be punched on any one of several types of tape perforation equipment operating in the ISO code. Whatever equipment is used on the punched tape should either be verified on a verifier-punch by a second operator, or be printed up. In the latter case the print-up produced from the tape should be carefully checked against the original program manuscript. On some equipment new-line is punched as a single character, while other equipment uses separate carriage-return and line feed characters. On the latter equipment N consecutive new-lines should be punched as:-

carriage-return, N line-feeds, blank

(2) FORTRAN programs may be written either on a pre-printed form (The ELLIOTT FORTRAN Program Sheet) or on lined paper with a vertical line added about $1\frac{1}{2}$" (4 cm) from the left-hand margin.

(3) Punch exactly what is necessary to produce a print-out like the written program, i. e. all blank lines, spaces, etc.

(4) If anything is written to the left of the vertical line it must be punched and at least two spaces left between it and the rest of the information on that line.

(5) Care must be taken to avoid confusion between characters, in particular between

figure 0 and letter O

figure 1 and letter I

figure 2 and letter Z

figure 5 and letter S

These must be punched correctly and punch operators should familiarise themselves with the difference in print of these characters and with the conventions used by the various authors of the manuscripts which they are called upon to punch.

(6) About 6" (15 cm) of blank tape should be runout at the beginning of every tape punched.

(7) A wrong character may be cancelled by over-punching with 'erase'. This erase character does not count towards the maximum number of characters that may be punched on a line (see 9 below).

(8) Every tape should be ended by

new line

about 2" (5 cm) blank tape

Halt code

about 6" (15 cm) blank tape

(9) There may not be more than 120 characters on a line of text (Blank and erase do NOT count towards this total).

(10) Some tape preparation devices can only print about the first 70 characters on a line. If a line is longer than this continue punching on the same line. Do not insert additional new-line characters. Note the circumstance, since especial care may be needed in checking that the line has been correctly punched.

1. 5. 3 Names Starting with Q

If the name of a program, sub-program or identifier starts with the letter Q, its next character must be the letter U. Additional SIR identifiers are introduced as labels, during the translation of FORTRAN. These identifiers start with the letter Q followed by a character other than U. Consequently, if this restriction is not observed, clashes may occur.

1.5.4 Program Writing

(1) Each FORTRAN statement starts on a new line.

(2) Continuation lines are permitted only for FORMAT and GLOBAL statements and begin with a currency symbol ($ or £).

(3) Comment lines start with the character C, followed by two or more spaces. The FORTRAN word CALL must be terminated by space. Elsewhere spaces are not significant, except in alphanumeric strings.

(4) Programs may be written either on a pre-printed form (Elliott FORTRAN Program Sheet) or on lined paper on which a vertical line has been drawn about $1\frac{1}{2}''$ (4 cm) from the left-hand margin.

To the left of this line are written:

(i) The letter C indicating a comment

(ii) Statement numbers.

(iii) A currency symbol, indicating a continuation statement.

(5) There must not be more than 120 characters on a line.

1.5.5 The Program Title

A 900 FORTRAN program or sub-program must be preceded by a GLOBAL statement which comprises the name of the program, followed by a list of the names of any sub-programs it uses. This statement is compulsory for programs, but may be omitted for any sub-program which does not itself use sub-programs.

The list is preceded by the word GLOBAL and the names comprising it are separated by commas, it may extend over several lines (and these additional lines must start with a currency sign) and it is terminated by].

Examples

```
GLOBAL, PROG]

GLOBAL, PROG3, SUB1, SUB2, MAXMULT, MXDIV

£ INVMX]
```

1.5.6 Example

An example of a Program:

(a) As Written

```
        SUBROUTINE MXMULT (A, B, C, I, J, K)
        DIMENSION A(I, K), B(I, J), C(J, K)
C       A=B*C
        DO 1 II = 1, I
        DO 1 KK = 1, K
        AA = 0
        DO 2 JJ = 1, J
2       AA = AA + B(II, JJ)* C(JJ, K)
1       A(II, KK) = AA
        RETURN

        END
```

(b) As Punched

(i) First Layout

```
        SUBROUTINE MXMULT (A, B, C, I, J, K)
        DIMENSION A(I, K), B(I, J), C(J, K)
        C    A=B*C
        DO 1 II = 1, I
        DO 1 KK = 1, K
        AA=0
        DO 2 JJ = 1. J
        2  AA = AA + B(II, JJ)* C(JJ, KK)
        1  A(II, KK) = AA
        RETURN

        END
```

(ii) Alternative layout

```
        SUBROUTINE MXMULT (A, B, C, I, J, K)
        DIMENSION A(I, K)B(I, J), C(J, K)
C       A = B*C
        DO 1 II = 1, I
        DO 1 KK = 1, K
        AA = 0
        DO 2 JJ = 1, J
2       AA = AA + B(II, JJ)* C(JJ, KK)
1       A(II, KK) = AA
        RETURN

        END
```

1. 5. 7 Correction of FORTRAN Programs

Corrections may be made in either the original FORTRAN, or in the SIR program, which is produced from it. To assist the user in making corrections in the second way each group of SIR statements is preceded by a comment line comprising the FORTRAN statement which give rise to them.

To assist the programmer in editing his tapes the halt code character is available. To insert a statement or group of statements into a program after a given statement S say, punch a halt code onto the tape on the blank that forms part of the new-line sequence that terminates S. On reading the halt code the computer will come to a systems wait. Then translate the additional statements (the last one being followed by a halt code on a new line), and finally continue translating the original program.

Alternatively, a 900 FORTRAN program may be modified by the use of the 903 Edit program. (see Volume 2. 3. 2)

Chapter 2: LANGUAGE SPECIFICATION

2.1 Constants

2.1.1 Integer or Fixed Point Constants

An integer constant must lie in the range

$-131071 \leq constant \leq +131071$

If it is negative it must be preceded by a minus sign, if it is positive it may, but need not be, preceded by a plus sign.

2.1.2 Real or Floating Point Constant

A real constant possesses one or both of two features that distinguish it from integer constants

(i) A decimal point

(ii) An exponent

An exponent consists of E followed by an integer. It follows a number which may, but need not, contain a decimal point and indicates the power of ten to which the number as written is to be raised, to obtain the intended number. On a data tape an exponent part occurring by itself is treated as though it was preceded by 1. This form may not occur in the text of a program.

A real constant must lie in range

$-2^{63} < constant < 2^{63}$. i. e. approximately $-9_{10}18 < constant < 9_{10}18$

Within the machine it is expressed to a precision of about one part in 10^8.

A constant may not contain a space.

On data tapes $_{10}$ and @ are permissible alternatives to E.

2.1.3 Examples

314 -19 0	are integer constants	
3. -.3 0.0		are real constants
-2E3 -2000.0 -20000E-1	these represent the same number	
1E2 100	these represent the same number	

2.2 Names or Identifiers

(1) A name is a string of letters and digits that begins with a letter.

(2) An integer name starts with I, J, K, L, M or N.

(3) A real name starts with any other letter.

(4) If a real name starts with the letter Q its second character must be the letter U. (The reason for this special restriction of 900 FORTRAN is explained in 1.5.3).

(5) Names may comprise any number of characters. However they are only distinguished by their first six characters.

2.3 Variables

2.3.1 Integer or Fixed Point Variables

These variables are the names of quantities that only take integer values.

They are defined by integer names (see 2.2)

The values assigned to them during computation must be permissible values for integer constants (see 2.1.1).

Unless a COMMON statement (see chapter 4) requires otherwise, each integer variable is allocated a unique store location.

2. 3. 2 Real or Floating Point Variables

These variables are the names of continuously varying quantities.

They are defined by real names (see 2. 2).

The values assigned to them during computation must be permissible values for real constants (see 2. 1. 2).

Unless a COMMON or EQUIVALENCE statement (see Chapter 4) required otherwise, each real variable is allocated to a unique pair of adjacent store locations.

2. 4 Arrays or Subscripted Variables

2. 4. 1 Subscripts

A subscript comprises an integer expression of one of the following forms:

```
I
2
I + 2
I - 2
3 * I
3 * I + 4
3 * I - 4
```

Must be this way round, a not I*3 etc.

where I may be replaced by any integer variable (see 2. 3. 1) and 2, 3 and 4 may be replaced by any integer constants. A unary plus or minus sign may be prefixed to any of these forms e. g. -I, + 3 * I.

2. 4. 2 Subscripted Variables

Before it is used a subscripted variable must be declared as such in a DIMENSION statement (see 4. 1).

It may be a real or an integer variable (see 2. 3).

It may have up to two subscripts which are written in parentheses after its name.

If it has two subscripts they are separated by a comma.

Whenever a subscripted variable is referred to during the execution of the program its subscripts are evaluated. The values they are found to have must be positive and be less than the maximum values specified by the DIMENSION statement in which it was declared. (In the case of a subscripted parameter of a sub-program the relevant maximum value is that of the <u>actual</u> parameter specified in the current call of the sub-program).

2.4.3 Storage of Arrays

The elements of arrays with two subscripts are stored with the value of the first subscript varying most rapidly.

2.4.4 Examples

A(1) DOG (3, 7*ITEM-5000)	elements of real arrays

LIST(4*JULIET)
JIG(2*I-9, -4*J+100)
	elements of integer arrays

2.5 Expressions

FORTRAN expressions comprise a string of constants, variables and functions separated by operators and delimiters.

The operators are:

+ which indicates positivity or addition.

- which indicates negation or subtraction.

* which indicates multiplication. (It is used rather than x to avoid confusion with the letter X).

/ which indicates division.

** which indicates exponentiation.

The delimiters are:

() which enclose subscripts and parameter lists and modify the order of the evaluation of the terms of expressions.

which separates items in a list.

<space> which may make the expression easier for a human being to read but is ignored by the compiler inside an expression.

<newline> which terminates the expression.

2. 5. 1 Order of Evaluation of Terms

(1) The current values of all the variables in the expression are first determined. (This may require the evaluation of subscripts or functions before the evaluation of the main expression can continue).

(2) Any exponentiation operations are next performed. The desired order of multiple exponentiation should be indicated by parentheses. If these parenthese are omitted 903 FORTRAN will evaluate the expression from the left, other compilers may however reject the construction or evaluate it differently.

(3) Multiplication and division occur next.

(4) Addition and subtraction are performed last of all.

Parentheses may be freely used, to indicate the grouping of terms. Sub-expressions within parentheses are evaluated in the same manner but not necessarily in same order as subscripts or functions and the value so found is used in further determination of value of the main expression.

Note 1. The effect of division extends only over the next element. For example:

A/B*C is equivalent to (A/B)*C or A*C/B
A/B/C is equivalent to A/(B*C)

Note 2. If the terms of an expression involving real variables are of widely differing magnitudes the result may, because of the finite precision of representation of intermediate results, depend on the order in which various operations are performed. Thus, if $A=B=10^{10}$ and $C=10^{-10}$ then $(A-B)+C=10^{-10}$ while $A-(B-C)=0$ If in such cases, the desired order of evaluation cannot be specified by the judicious use of parentheses then the expression must be rewritten as several separate expressions.

2.5.2 The Types of Results

(1) If, in evaluating the parts of an expression it is necessary to add together, subtract or multiply terms of which some are real and others are integer, all the integer terms are converted to real before the evaluation preceeds.

(2) Division and exponentiation always yield real results.

2.5.3 Exponentiation

The validity of a**b depends on the values of a and b and on the type of b. The result, where it exists, is always real.

If b is an integer variable, constant or expression then the operation is valid unless both a and b are zero. In all other cases the result is that obtained by multiplying a by itself $|b|$ times, and, if b is negative, taking the reciprocal.

If b is a real variable, constant or expression then the operation is only valid if a is positive. The result, when it exists has the value $exp(b*log_e(a))$

If, when a and b are of type integer and b is negative, exponentiation is required to give a zero answer, it is suggested that the following function sub-program be introduced:

```
      FUNCTION IEXP(I, J)

      IF (J) 1, 2, 3
1     IEXP = 0
      RETURN
2     IEXP = 1
      RETURN
3     IEXP = I**J
      RETURN
      END
```

2. 5. 4 Division

If integer division is required it is suggested that the following function sub-program be introduced:

```
FUNCTION IDIV(I, J)
IDIV = I/J
RETURN
END
```

This will give an integer result rounded towards zero, as exemplified by the following table:

K	IDIV (K, 3) and IDIV (- K, - 3)	IDIV (- K, 3) and IDIV (K, - 3)
0, 1 or 2	0	0
3, 4 or 5	1	- 1
6, 7 or 8	2	- 2
. . . .	. . . .	. . . .
3n, 3n+1 or 3n+2	n	n

Chapter 3: STATEMENTS

3.1 Statements.

A FORTRAN program consists of a sequence of statements. These are two types:

(1) Executable statements, which are obeyed when the program is run.

(2) Non-executable statements, which further define the meaning of executable statements. Non-executable statements are of four types:

(a) Those which give information to the compiler e.g. COMMON statements.

(b) Those which further define run-time operations, e.g. FORMAT statements.

(c) Those which contain information used both during compilation and at run-time. e.g. DIMENSION statements.

(d) Those which may contain information of value to human beings reading the program, but which have no effect whatsoever on its compilation or running. i.e. Comments.

3.2 Arithmetic Assignment Statements

An arithmetic assignment statement is an executable statement of the form

v=e

where v represents an arbitrary variable and e represents an arbitrary expression. If v and e are of different types automatic type conversion occurs. When a real result is assigned to an integer variable:

(a) It is always rounded towards zero.

(b) Integer overflow may occur.

3. 3 Control Statements

Control statements indicate the order in which the executable statements of a program are to be obeyed. Usually they indicate departures from obeying executable statements in the order in which they are written.

3. 3. 1 Statement Numbers or Labels

A statement number is an unsigned integer in the range 1 to 99999 inclusive that is prefixed to a statement to allow it to be referred to by other statements. A given statement number may only be used once within a given program or sub-program. Not more than one statement number may be prefixed to a statement.

3. 3. 2 GOTO including IF.

GOTO statements are control statements that indicate explicitly that executable statements are not to be obeyed in sequence. They are themselves executable statements. A GOTO statement gives the number of the statement that is to be obeyed next. This next obeyed statement must be an executable statement.

The next executable statement after a GOTO statement must bear a statement number.

Three types of GOTO statements are available in 900 FORTRAN.

(1) The unconditional GOTO statement

(2) The computed GOTO statement

(3) The IF statement

3. 3. 2. 1 Unconditional GOTO

These statements are of the form

GO TO n

where n is the number of the statment which is to be obeyed next.

Examples.

```
               GO TO 7
5              GO TO 8
```

3.3.2.2 Computed GOTO.

These statements have the form

GO TO(n_1, n_2, n_3... n_m), i

where n_1, n_2, n_3 ... n_m are m statement numbers (which need not all be different) and i is a non-subscripted integer variable which, whenever the statement is obeyed, must have a value in the range 1, 2, 3 ... m.

Examples.

```
               GO TO (53, 1, 53, 437), INDEX
53             GO TO (150, 151), NY
```

3.3.2.3 IF

These statements have the form

IF(e)n_1, n_2, n_3

where e is an expression and n_1, n_2 and n_3 are three, not necessarily distinct, statement numbers. Control is transferred to n_1, n_2 or n_3 according to the sign of e.

Sign of e	Label of the Next Executable Statement
NEGATIVE	n_1
ZERO	n_2
POSITIVE	n_3

Examples.

```
        IF(INDEX-2)53, 1, 500
500     IF(INDEX-3)999, 53, 437
53      IF(NY-2) 150, 151, 999
```

Provided that INDEX and NY only take their expected values ((1 to 4) and (1 or 2) respectively) this sequence of IF statements has the same effect as that of the sequence of two computed GOTO statements given as an example in 3.3.2.2. Control can never be transferred to the label 999, and in practice this would either label a dummy statement such as CONTINUE or would be replaced in the IF statements by the adjacent statement number so that, for example, the last statement above would be written

```
53          IF(NY-2) 150, 151, 151
```

3.3.3 CONTINUE

A CONTINUE statement is a dummy executable statement which allows a statement number to be placed at some point where it might not otherwise be placed.

Since it is itself an executable statement its use does not permit several numbers to be attached to the same statement.

3.3.4 DO

A DO statement indicates that the following group of statements, up to and including the statement whose reference number is given in the DO statement, are to be obeyed a number of times while an integer variable takes a succession of values in arithmetic progression. It is written in the form

$$\text{DO } n \ i{=}m_1, m_2, m_3$$

where

- n is the statement number of the last statement to be obeyed each time the DO loop is traversed,
- i is an un-subscripted integer variable,
- m_1 is the value to be assumed by i the first time the DO loop is is traversed,
- m_3 is the constant interval of the arithmetic progression of values assumed by i. It must be greater than zero.
- m_2 which must be greater than m_1, is the greatest value which i may assume. (It need not be attained nor, if $m_3>1$, need it be attainable)

m_3 is added to i at the end of each traverse of the DO loop. If the new value of i is less than or equal to m_2 the loop is traversed again. Otherwise the loop is terminated and the next executable statement (following the statement labelled n) is obeyed.

m_1, m_2 and m_3 may be unsubscripted integer variables or unsigned integer constants.

If m_3 has the value 1 it may be omitted and the DO statement written in the form

DO n i=m_1, m_2

When the DO loop is left because if i were again incremented it would exceed m_2, the value of i is undefined.

If the DO loop is left by means of a GOTO statement the value of i is preserved.

The values of i, m_2 and m_3 may not be altered within the DO loop.

With one exception a GOTO statement may not cause a jump into a DO loop that bypasses its initial DO statement. The exception is when the DO loop was itself left by a jump and the values of i, m_2 and m_3 have not since been altered.

DO loops may occur within DO loops to a depth of 20.

DO loops may not intersect each other.

The first statement of a DO loop must be an executable statement.

The terminating statement of a DO loop must not be a GOTO statement (an IF statement is a GOTO statement). Where this restriction arises a CONTINUE statement should be used after the GOTO statement.

3. 3. 4. 1 An Example of Correctly Nested DO Loops.

```
              DO 307 I=1, 10
              DO 307 J=I, 30, I
              DO 314 INDEX=5, 15
                 A(I, J) = A(I, J)- B(INDEX)
314           CONTINUE
              DO 330 INDEX=-10, 0, 3
              DO 329 K=LLB, 12
              IF(B(INDEX+10)) 345, 329, 345
329           A(INDEX+10, K)=0.
330           CONTINUE
              DO 307 K=1, 10
              .  .  .  .
              .  .  .  .
              GOTO 307
345           G = G+1
              GOTO 329
307           CONTINUE
```

3. 3. 5 PAUSE

An executable statement that causes the computer to wait until the operator indicates that computation is to continue. It should normally be preceded by a statement that causes a message to be displayed telling the operator to take some special action (e. g. to load the next data tape).

3. 3. 6 STOP

An executable statement that causes the computer to abandon the calculation. The program cannot be continued, but it will normally remain intact in store, so that it can be re-entered at the beginning with a fresh set of data.

3. 3. 7 END

An executable statement which

(1) Tells the compiler that it has reached the end of the program or sub-program that it is translating into SIR.

(2) Has, at run time, the same effect as STOP.

3.3.8 RETURN

An executable statement which may only occur in a sub-program and indicates that the sub-program has completed its task and that control is to be returned to the calling program or sub-program at the executable statement following its call.

Chapter 4: SPECIFICATION STATEMENTS

Three types of specification statement are available in 900 FORTRAN:

(1) DIMENSION statements

(2) COMMON statements

(3) EQUIVALENCE statements

4. 1 DIMENSION

A DIMENSION statement is of the form

DIMENSION $v_1(i_1)$, $v_2(i_2)$, . . . , $v_n(i_n)$

where each v is the name of a subscripted variable and the corresponding i indicates the permissible range of values for its subscripts. Each i is either:

a) an integer constant c indicating that v is a one-dimension array (a vector) of c elements, which may be referred to as v(1), v(2) . . . v(c), or

b) a pair of integer constants c_1, c_2 indicating that v is a two-dimension array of $c_1 * c_2$ elements which may be referred to as v(1, 1), v(2, 1). . . v(c_1, 1), v(1, 2). . . v(c_1, c_2).

If and only if, v is the name of an express formal parameter of a sub-program (see 5. 2. 5(3)) i may comprise the names of one or two integer variables which are also formal parameters of the sub-program. (see also 5. 2. 2; and 5. 2. 5.)

Example.

```
DIMENSION A(10), B(5, 15)CAT (99)
DIMENSION A(I), B(5, J), C(I, J)
```

The second example could only occur in a sub-program which numbered A, B, C, I and J among its parameters.

4. 2 COMMON

A program or sub-program may contain one or more statements of the form:

COMMON $v_1, v_2, \ldots, v_n$

where each v is the name of a variable (which may be an array name) which is used elsewhere in that program or sub-program.

Subject to the rule that a real variable occupies two store locations while an integer variable occupies one location the effect is that the kth item in COMMON in a given program or sub-program occupies the same location in a special store area (the COMMON area) as the kth item in COMMON in any other program or sub-program. Variables are allocated store space in the COMMON area in the order in which they are listed in COMMON statements. If two or more COMMON statements occur in one program or sub-program then the first variable of each COMMON statement except the first is placed immediately after the last variable in the preceding COMMON statement.

COMMON serves two purposes:

a) it provides an alternative method of communication between a program and its associated sub-programs to that provided by the use of express formal parameters (see 5. 3)

b) it permits the economic use of storage space. If arrays in a 900 FORTRAN program are placed in COMMON larger programs may be loaded, since the COMMON area may overwrite the program loader, whereas local arrays are embedded within each program unit.

Examples of the use of COMMON statements and of the resulting allocation of store locations are given in 4. 6.

4. 3 EQUIVALENCE Statements

An equivalence statement is of the form:

EQUIVALENCE $(k_1), (k_2), \ldots\ldots (k_n)$

where each k is a list of the form:

$a_1, a_2, \ldots\ldots a_m$

and each a is the name of a variable or an array element.

In Basic 900 FORTRAN the occurrence of an EQUIVALENCE statement is noted by output of a query message, but no other action is taken. Run time store space is not saved by its use, but providing it has not been misused to equate entities,programs which use EQUIVALENCE will be correctly executed. Note that the definition of ASA FORTRAN specifically forbids the use of EQUIVALENCE to equate entities.

4.4 Ordering of Specifications

A DIMENSION statement must precede the occurrence of the variables listed in it in any other statement.

A COMMON statement must precede the occurrence of the variables listed in it in any statement other than a DIMENSION statement.

An EQUIVALENCE statement must precede the occurrence of the variables listed in it in any statement other than a DIMENSION statement, a COMMON statement or other EQUIVALENCE statement. (This rule is inserted for compatibility with other dialects of FORTRAN. Its breach will not be detected and will not lead to failure of the program).

4.5 Inconsistent Specifications

4.5.1 DIMENSION and COMMON

Within a program or sub-program (see 5.2) a variable name may occur in at most one DIMENSION statement and one COMMON statement.

4.6 Examples

The specifications in program CAT:

```
DIMENSION A(5), I(3, 2), B(2), L(3)
COMMON A, I, J, G
COMMON K, LL
```

coupled with the specifications is sub-program DOG:

```
DIMENSION TAIL(6), L(3)
COMMON EAR, BARK, TAIL, F, K, BK
```

would lead to the following allocation of space in the first 20 locations of the COMMON area.

Location	Variables declared in	
	CAT	DOG
COMMON	A(1)	EAR
+1		
+2	A(2)	BARK
+3		
+4	A(3)	TAIL(1)
+5		
+6	A(4)	TAIL(2)
+7		
+8	A(5)	TAIL(3)
+9		
+10	I(1, 1)	TAIL(4)
+11	I(2, 1)	
+12	I(3, 1)	TAIL(5)
+13	I(1, 2)	
+14	I(2, 2)	TAIL(6)
+15	I(3, 2)	
+16	J	F
+17	G	
+18		K of DOG
+19	K of CAT	BK
+20	LL	

4. 7 Warning

Any user of COMMON statements is strongly advised to draw a store map such as that in 4. 6. Effects such as the overlap of J and G of CAT with F and K of DOG are not erroneous but the effect is unlikely to be that desired by the programmer.

COMMON statements if carefully used can save space and simplify the calling and construction of sub-programs. If mis-used they can cause chaos.

Chapter 5: FUNCTIONS AND SUBROUTINES

A FORTRAN function is a single-valued function of one or more variables.

Three types of functions are available in Basic 900 FORTRAN:

(1) Standard Library Functions

(2) FUNCTION sub-programs

(3) Additional Library Function (see Chapter 9)

A SUBROUTINE sub-program has the same form as a FUNCTION sub-program, but it returns the results, if any, of its computations to the program, or sub-program that called it by altering the values of one or more of its parameters and of variables in COMMON (see Chapter 4).

5. 1 Library Functions

The following functions, which are all real functions of a single real argument, are automatically available to all users of 900 FORTRAN. Collectively they form the basic library on the 900 FORTRAN systems tape. Any of these Functions used within a program unit (Main program, subroutine or function) must be declared in a GLOBAL statement at the head of that unit (see 1. 5. 5)

ALOG (E)	The natural logarithm of E
SIN (E) COS (E)	The sine and cosine of E, where E is measured in radians.
EXP (E)	e^E
ATAN (E)	The principal value in radians of the arctangent of E
ABS (E)	The absolute value (modulus) of E. (Intended for use with real E).
SQRT (E)	The square root of E.

In addition the following function is automatically available: it is an integer function of an integer variable:

Function	Definition
IABS (IE)	The absolute value (modulus) of IE. (Intended for use with integer IE).

Notes

1. In SIN (E) and COS (E), ABS (E) must be less than 2^{18}. This condition is imposed to ensure that, after multiples of 2π have been removed, enough significant figures remain for the evaluation to be meaningful.

2. ATAN can take any value between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ inclusive.

To the (finite) precision of the representation both values are attained.

5. 2 FUNCTION and SUBROUTINE Sub-programs

In basic 900 FORTRAN all functions must be written as FUNCTION sub-programs. A value must be assigned to the function name within the sub-program body. (see 5. 2. 3)

The computation performed by a SUBROUTINE sub-program, however may result in the assignment of many or no values. Thus one SUBROUTINE might invert a matrix while another prints the results of a computation performed by the main program.

5. 2. 1 Sub-Program Titles

A sub-program should have a GLOBAL statement as a title before its heading unless it does not use any sub-programs of library Function. (see 1. 5. 5) The first name in the GLOBAL list must be the sub-program name.

5. 2. 2 The Sub-Program Head

A sub-program head takes the form

FUNCTION $f(m_1, \ldots m_k)$

SUBROUTINE $s(m_1, m_2, \ldots m_k)$

or SUBROUTINE s

where

(1) f is the name of the FUNCTION and specifies its type in accordance with the usual rules (see 2. 2).

(2) s is the name of the SUBROUTINE. (A subroutine does not have a type associated with it consequently, apart from the Q rule (see 2. 2) and the necessity of avoiding clashes of names the choice of s is completely arbitary)

(3) The $m_1, m_2, \ldots m_k$ are express formal parameters. Each m_i must be the name of a variable or an array. (e. g. an express parameter may not be a sub-program).

(4) Each m_i which represents an array must appear in a DIMENSION statement within the body of the sub-program (see 4. 1). In this DIMENSION statement the upper bounds of its suffices may be given either as integer constants or as integer variables which are themselves express formal parameters.

(5) In addition to the express formal parameters a sub-program may refer to variables in COMMON, these may be regarded as implicit parameters.

5. 2. 3 The Sub-program Body

(1) A sub-program body is, subject to certain special rules, a normal FORTRAN program.

(2) A SUBROUTINE does not have a value and no assignment may be made to its name, s. It may communicate information to the program that called it (the main program or another sub-program) by altering the values of one or more of its parameters.

(3) Within a FUNCTION sub-program its name, f, acts as an ordinary variable, of the appropriate type. It is undefined on entry to the FUNCTION and a value must be assigned to it before the FUNCTION is left. This value is the value of the FUNCTION.

(4) The alteration by a FUNCTION of any of its parameters is not considered to be an error in 900 FORTRAN. However the evaluation of a FUNCTION may not validly alter the value of any other element within any expression, assignment statement or CALL statement in which the FUNCTION appears.

(5) A sub-program body may not itself contain a declaration of a sub-program.

(6) An explicit formal parameter may not occur in a COMMON or EQUIVALENCE statement (see Chapter 4).

(7) When a sub-program has completed its task it returns control to the program that called it by means of a RETURN statement. This comprises the word RETURN on a line by itself.

(8) The body of a sub-program is terminated by an END statement. This comprises the word END on a line by itself.

5.2.4 Examples

```
      FUNCTION MAX (I, J)
      IF(I-J)1, 1, 2
1     MAX=J
      RETURN
2     MAX=I
      RETURN
      END
```

```
      GLOBAL, TMAX, MAX]
      FUNCTION TMAX (K, M)
C     TMAX TAKES THE GREATEST VALUE
C     OF K, M and A NUMBER READ FROM TAPE
      M=MAX (K, M)
      READ (1) TMAX
      IF (TMAX-M) 1, 2, 2
1     TMAX=M
2     RETURN
      END
```

```
      SUBROUTINE MTXMLT(A, N, M, B, L, C)
      DIMENSION A(N, M), B(M, L), C(N, L)
C     C BECOMES A TIMES B
      DO 1 I = 1, N
      DO 1 K = 1, L
      D = 0.0
      DO 2 J=1, M
2     D = D+A(I, J)*B(J, K)
1     C(I, K)=D
      RETURN
      END
```

5.2.5 Calling Sub-Program

(1) A FUNCTION sub-program is activated by writing

$$f(m_1; m_2, \ldots m_k)$$

in some statement which can make use of the value of f.

(2) A SUBROUTINE sub-program is activated by a CALL statement, e.g.:

$$\text{CALL } s(m_1, m_2, \ldots m_k)$$

The FORTRAN word CALL must be terminated by at least one space

(3) If an express formal parameter is an array the corresponding actual parameter must be an array of the same type with the same number of dimensions.

(4) If an express formal parameter which is an integer variable is used as a subscript bound then the corresponding actual parameter must be an integer variable to which the correct value of the subscript bound has been assigned prior to the call of the procedure. (This rule is inserted for compatibility with other dialects of FORTRAN. Whatever bounds are quoted for the subscript of an express formal parameter which is an array the true bounds of the actual parameter will be used in verifying that any reference to that array is valid).

How to get variable subscripts.

(5) If an express formal parameter is a simple variable the corresponding actual parameter must be a simple variable, array element, or constant or expression of the same type. If an actual parameter is a constant or expression then the corresponding formal parameter:

(a) must not occur in a DIMENSION statement

(b) must not have a value assigned to it during the execution of the sub-program.

(6) The actual parameters need not all be distinct.

5.2.6 Examples

This example is based upon the example of sub-programs in 5. 3. 4.

```
      GLOBAL, EXAMPLE, MAX, MTXMLT]
      DIMENSION K(50), A(5,10), B(10,20), C(5,20)
      . . . . . . .
C     THE ELLIPSIS INDICATES THE ASSIGNMENT OF
C     VALUES TO THE ELEMENTS OF K, A AND B
      I=MAX(K(1), K(2))
      DO 1 J=3, 50
1     I=MAX(I, K(J))
      I1 = 5
      I2 = 10
      I3 = 20
      CALL MTXMLT(A, I1, I2, B, I3, C)
      END
      Ⓗ
```

5.2.7 Functions as Actual Parameters

Where the actual parameter to a sub-program is an expression, this expression may include Function calls. Where a function name with parameters, e. g. SIN(X) is used as an actual parameter by itself, it should be converted to an expression by enclosing in a set of parentheses. e. g.

```
B = TMAX(X, (SIN(Y)))
```

SIN(Y) is evaluated before TMAX is entered. This rule is given for compatibility with other dialects of Fortran. In fact the expression

```
B = TMAX(X, SIN(Y))
```

will have the same effect in Basic 900 Fortran, but not with other compilers.

Chapter 6: INPUT, OUTPUT and FORMAT STATEMENTS

6.1 Input and Output Statements

Input and output statements take the form

```
READ (u, f)k    or READ (u)k
WRITE (u, f)k   or WRITE (u)k
```

where:

u is an integer constant or variable indicating a device

f is the statement number of a FORMAT statement (see 6.3)

k is a list of items to be input or output

If'f'is absent the statements are known as unformatted statements, otherwise they are known as formatted statements.

Value of	Device referred to in a	
u	READ statement	WRITE statement
1	The tape reader	<error>
2	<error>	The tape punch
3	The teleprinter	The teleprinter
4-5	<error>	<error>
6-8	see note 2	see note 2

Notes:

1. If u is less than 1 or greater than 8 or the word error appears against the value of u then reading will take place from the tape reader or writing to the tape punch.
2. The numbers 6-8 are reserved for special input/output routines.

6. 2 Unformatted Input and Output

6. 2. 1 Unformatted READ

In the statement sequence

```
DIMENSION Y(2, 3), B(10)
J=3
READ (1) X, Y, J, B(J)
```

The effect of the READ statement is to assign the first number on the data tape to the variable X, the next six numbers to the six elements of Y in the order Y (1, 1), Y (2, 1).... Y (2, 3) the eighth to J and, irrespective of the new value of J, the ninth number to B(3).

6. 2. 2. The Data Tape

Each number on the data tape must be written in one of the forms specified for a constant in Chapter 2. 1 in accordance with the following rules:

(1) A number is terminated by one or more spaces, tabs or line feeds.

(2) To provide compatibility with Elliott Algol and to improve the legibility of data tapes $_{10}$ and @ are acceptable alternatives to E.

(3) Blank tape, carriage return and erase are always neglected.

(4) If a Halt code is encountered while a number is being read, the effect is as for a PAUSE statement (see 3. 3. 5).

(5) Except as described in 6. 3. 1. 3 (which deals with the input of alphanumeric strings) no symbol other than the digits +, -, . and those listed in (1) to (4) above may occur on a data tape.

6. 2. 3 Unformatted WRITE

An unformatted WRITE statement causes numbers to be output five to a line. Thus the statement

WRITE (2) I** 2 , Y, 14,B(J)

would cause output according to the following pattern

I^2 Y(1, 1) Y(2, 1) Y(1, 2) Y(2, 2)
Y(1, 3) Y(2, 3) 14 B(J)

Unformatted output obeys the following rules

(1) It is restricted to numerical values

(2) These are output five to a line in floating point format, e. g. :

$$-0.19300_{10}{+03}$$

(3) Both integers and real numbers are output in this format which is equivalent to FORMAT (5E12. 5) (see 6. 3. 1. 1.).

6. 3 FORMAT Statements

If a READ or WRITE statement refers to a FORMAT statement then its effect is determined by the FORMAT statement.

A FORMAT statement specifies the desired relationship between internal representations and external character strings.

It has the form

FORMAT ($q_1 t_1 z_1 t_1 z_2 \dots t_n z_n q_2$)

where

q represents a possibly empty sequence of oblique strokes (see 6. 3. 2. 2)

t represents a field descriptor or group of field descriptors (see 6. 3. 1 and 6. 3. 3)

z represents a field separator (see 6. 3. 2). There need not be any pairs t_i z_i

If the FORMAT statement is continued over several lines, each line must end with a field separator, and each continuation line must start with the character $ or £.

6. 3. 1 Field Descriptors

The following field descriptors are available in 900 FORTRAN

rFw. d	or	Fw. d
rEw. d	or	Ew. d
rIw	or	Iw
$nHh_1 h_2 \ldots h_n$		
nX		

where the letters F, E, I, H and X represent the nature of the conversion to be performed between the internal and external representations. They are called conversion codes.

w and n — are unsigned non-zero integer constants representing the total number of characters occupied by the field in the external string.

d — is an integer constant representing the number of characters following the decimal point

r — the repeat count, is an integer constant which indicates that the effect is to be as though the following field descriptor was repeated r times (see 6. 3. 3).

$h_1 h_2 \ldots h_n$ — n characters other than string quotes ʼand ˋ or newline, chosen from the internal character set (see Chapter 1. 5).

6. 3. 1. 1 Numeric Field Descriptors

Descriptor	Effect on Output
Ew. d $(W \geqslant d+7)$	Output a real number with a total of w characters. This will take the form: (w-d-7) spaces, sign, zero, decimal point, d decimal digits, ${}_{10}$,sign of exponent, two digit decimal exponent.
Fw. d $(W \geqslant d+2)$	Output a real number with a total of w characters. This will take the form sign, w-d-2 decimal digits, decimal point, d decimal digits. If the number is sufficiently small the sign will be floated, being preceded by the requisite number of spaces. If w>d+2 then at least one digit will be punched before the decimal point.
Iw $(W \geqslant 2)$	Output an integer with a sign and w-1 digits. The sign may be floated as described under Fw. d.

Effect on Input

On input these descriptors indicate that a number is to be input at this point, but do not imply that that number is a real number or an integer, that it is punched in any particular style or that it occupies w places.

TABLE

Effect of Numeric Items in READ and PUNCH Lists

Item	Effect in a READ List	Effect in a WRITE List
A simple variable (which does not occur in a DIMENSION statement) e.g. I	A number is input from the specified device and its value is assigned to the variable.	The value of the variable is output to the specified device.
A subscripted variable e.g. B(7, J)	As for a simple variable (but see note 1)	As for a simple variable (but see note 1)
The name of a DIMENSIONED variable e.g. B	The appropriate number of values is read from the specified device and they are assigned, in order, to the elements of B.	The values of the elements of B are output in order.
A constant	NOT PERMITTED	As for a simple variable.
An expression (see note 2)	Data is read and ignored.	The expression is evaluated and its value is output
The name of a FUNCTION		
(i) Inside its defining sub-program. (The name by itself without parameters)	As for a simple variable.	As for a simple variable.

Item	Effect in a READ List	Effect in a WRITE List
(ii) Elswhere (A call of the FUNCTION with a parameter list)	NOT PERMITTED	NOT PERMITTED
The name of a subroutine	NOT PERMITTED	NOT PERMITTED

Note 1: The identity of the items in a READ or PRINT list is determined before the values of any items in that list are input or output. Consequently, if the next two items on a data tape are 3 and 3.14159, one effect of

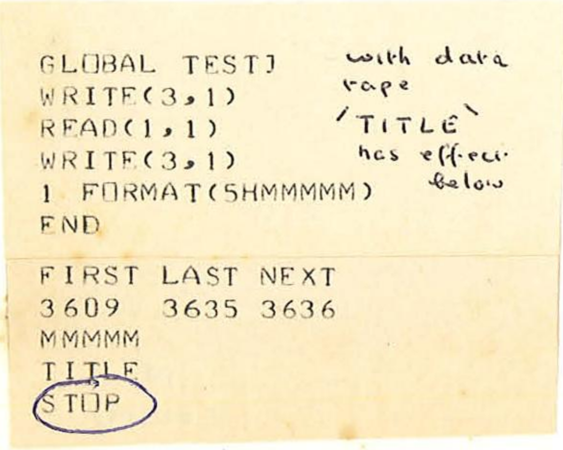
```
I=7
READ(1)I, A(I)
```

is to assign the value 3.14159 to A(7). The value of A(3) is unaffected.

Note 2: An expression in a READ or WRITE list may only refer to the name of a FUNCTION if it is inside the defining sub-program of that FUNCTION. Any other reference is an error.

6.3.1.2 Alphanumeric Field Descriptors

Descriptor	Effect on Input	Effect on Output
$nHh_1 h_2 \ldots h_n$	Replace the n characters by the string which is the next item of data.	Output the n characters or the characters which have replaced them as the result of an input operation.



```
GLOBAL TESTJ
WRITE(3,1)
READ(1,1)
WRITE(3,1)
1 FORMAT(5HMMMMM)
END
```

with data tape 'TITLE' has effect below

```
FIRST LAST NEXT
3609  3635 3636
MMMMM
TITLE
STOP
```

6.3.1.3 Alphanumeric Data

Alphanumeric strings on data tapes must be surrounded by the string quotes ´ and ` . If the string on the data tape is shorter than is specified by the FORMAT descriptor terminal spaces are authomatically inserted to bring it up to the correct length. If the string is longer it is truncated and tape is skipped up to the terminating `. ´ and ` may not themselves appear within an alphanumeric string.

6.3.2 Field Separators

The field descriptors are separated by oblique strokes or commas.

6.3.2.1 Comma

A comma placed between field descriptors serves to separate them. It has no other effect.

6.3.2.2 Oblique Stroke.

One or more oblique strokes placed before, after or between field descriptors serves, if placed between descriptors, to separate them. Furthermore each oblique stroke indicates that a new-line sequence is to be output.

6.3.3 Repeat Counts

If a field descriptor or a sequence of the form:

$$(q_1 t_1 q_1 t_2 z_2 \dots t_n z_n q_n)$$

is preceded by an unsigned indicator the descriptor or sequence is repeated the indicated number of times.

In 900 Fortran sequences may not occur within sequences.

6.3.4 Exhaustion of a READ, WRITE or FORMAT List

If a formatted READ or WRITE list is exhausted before the end of its associated FORMAT statement input or output ceases. If the FORMAT statement is again referred to, whether by the same or another input/output statement, it is once more obeyed from the beginning.

If a FORMAT statement is exhausted while elements still remain in the READ or PUNCH list, which refers to it, it is obeyed again from the repeat count, if any, preceding its last left bracket.

6. 3. 5 Records

A new record is begun

a) At the beginning of a FORMAT statement

b) Whenever / is encountered

c) When going back to the repeat count, if any, preceding the last left bracket after exhaustion of a FORMAT statement. Cases (a) and (c) are implicit. Case (b) is explicit.

A new record normally starts on a new line. This effect of an implicit new record may be cancelled by the control character (Field Descriptor) Ƶ immediately following the appropriate left bracket. If a Ƶ is encountered at any other time it is ignored. e. g.

FORMAT (I3, 4(I5))

prints five integers on the first line and four on each succeeding line. The first integer is preceded by a new line.

FORMAT (Ƶ, I3, 4(Ƶ, I5))

prints integer indefinitely on the current line.

6. 3. 6 The implied FORMAT statement

The FORMAT statement used where none is specified is

FORMAT (5E12. 5)

SIR CODE FOR FORTRAN SECTION

Chapter 7: CODE

7.1 Code Sections

Certain operations can be performed faster and require less machine instructions when written directly in SIR than when written in FORTRAN and translated into SIR. This is because the Fortran compiler must cater for all possibilities, but the programmer writing in machine code need consider only his special case. Code sections can be particularly advantageous in matrix work.

The examples in this chapter have been chosen to illustrate the method of writing SIR coding as part of a FORTRAN program. They are not necessarily suitable for use unmodified. Further examples may be found by examining the object code of a Fortran source program.

The reader of this chapter is assumed to be familiar with SIR and in particular with QF, QFMATH and the precautions necessary in performing machine code multiplication and division of integers.

7.1.1 Format

A code section may be a complete sub-program or may be part of a program, the rest of which is written in Fortran source text. It is preceded by the statement CODE, and is terminated by the character string FORTRAN on a newline.

The statement CODE may be numbered: the label will refer to the first word in the code body. A SIR label may appear on the line before the word FORTRAN: the label will refer to the following FORTRAN statement. The word CODE must be followed immediately by newline.

Example

```
        FUNCTION INT1 (N)
C       AT FIRST ENTRY N SHOULD BE ZERO, LEVEL 1, S-REG IS SET.
C       THE MANUAL LEVEL 1 INTERRUPT MAY THEN BE USED.
C       INT1 MAY BE CALLED FROM A PROGRAM LOOP, AND WILL BE
C       NON-ZERO IF AN INTERRUPT HAS OCCURRED.
        IF (N) 1.2.1
1       INT 1 = IW
3       IW = 0
        RETURN
2       CODE
        4    ADLI     (SET LEVEL 1 S-REGISTER)
        5    0
        FORTRAN
        GOTO 3
        CODE
ADLI      0    LI
LI        5    A
          3    Q
        FORTRAN
        IW = 1
        CODE
          0    Q
         14    1
          4    A
         15    7168
          8    LI
          A    +0
          Q    +0
          FORTRAN
          END
```

7.1.2 SIR Facilities Available in Code Bodies

The following facilities may be used in a code body without any syntactic restriction:

Constants: integer e.g. -79

fixed-point fraction e.g. +.317

octal e.g. &770123

alphanumeric e.g. £A23

Pseudo instructions e.g. 0 ;+1

Addresses:

absolute	e.g.	32
block relative	e.g.	5;
SIR relative	e.g.	;-3
identifier	e.g.	LIST +95
literal	i.e.	any constant or a quasi-instruction e.g. =6 8191

Skip e.g. >5

Comments e.g. (THIS IS A COMMENT)

(Comments must not contain the word FORTRAN)

Other facilities must be used with care, as described below.

7.1.2.1 Labels

Labels used in the code body must not duplicate local labels used in the object code of the program or sub-program of which the code body is part. Duplication will be avoided by observance of the following rules:

(i) labels whose first letter is Q must have U as their second letter (but see 7.2.1.1);

(ii) the names of variables, used in the program or sub-program of which the code body is part, must not be used as labels in the code body.

(iii) the word FORTRAN must not be used as an identifier in a code body;

(iv) a global name must not be located more than once.

7.1.2.2 Global Identifier List

A global identifier list may appear at the head of the code body if and only if the body forms a complete sub-program. The list must include the name of the sub-program and the names of any other sub-programs used in the block. If any of the following identifiers are used they must also be included:

QYO
QF
QZCOM

No other identifiers may be included in the list.

The remaining facilities, including the following, may not be used in a code body:

sub-global labels;

end-of-tape symbol (<halt>);

end-of-program symbol (%)

options;

7.2 Communication with Fortran Text

The following paragraphs explain how code sections in a 903 Fortran program may communicate with identifiers in sections of the program written in Fortran text. Examples are given showing the code equivalents of various Fortran statements. However, the code shown does not necessarily correspond to that produced by the Translator when processing these statements. Examples of actual translations may be obtained by printing the object code tapes output by the Translator.

The descriptions are full at the beginning of the section. If a particular form of parameter word is used repeatedly reference is made back to the first example.

7.2.1 Labels

7.2.1.1 Statement numbers as SIR labels

If n is an unsigned integer between 1 and 99999 inclusive, then the Fortran statement number n is equivalent to the SIR label Qn. This fact may be used to transfer control between Fortran text and code bodies in the same program or sub-program.

7.2.1.2 Location of variables

Three types of variable may be used:

Local Variable: used in only one Fortran program or sub-program or SIR block. Every local variable must be located in the SIR block (either object code or code body) in which it is used. (See also below)

Common Variable: a variable named in a Fortran COMMON statement, or operated on in a sub-program written entirely in SIR as though it were so named. These variables are not located by name.

Formal Parameter: a variable named in parentheses in a Fortran FUNCTION or SUBROUTINE statement, or operated on in a sub-program written entirely in SIR as though it were so named. These variables are not located by name.

Care must be excercised in locating the local variables used in the code body. If the body is a complete sub-program then all communication with Fortran text must be via parameters (either explicit or implicit). Space is reserved for the actual parameters by the object code; only local workspace (and constants) for the sub-program are located in the code body. If the code body is only part of a program or sub-program then the user must locate, in a code body, any variables or constants that are referred to only in code bodies. Local variables used in Fortran text are located by the Fortran compiler when END is read.

Example

```
      SUBROUTINE ERROR (N)
C     THIS PUNCHES 20 BLANKS, N, 20 BLANKS
      A=0
      CODE
        Q1      4     -5120
                1     CHAR
        L1      15    6144
                1     +256
                9     L1
                4     A
                9     Q2
      FORTRAN
      WRITE (1) N
      A=-1
      GOTO 1
2     RETURN
      CODE
        CHAR +0
      FORTRAN
      END
```

7.2.2 Simple local variables

Real variables are held in packed format, occupying two locations each. They are operated upon via QF. (See 900 Manual, Vol. 2.2.8, for further details).

Integer variables are held at 2^{-17} occupying one location each. Both are accessed by identified addresses.

7.2.3 Array variables

Two-dimensional arrays are stored with the first suffix varying most rapidly. Real arrays occupy two locations per element and integer arrays one per element.

An element is accessed by using the name of the array as a base address, and an offset determined by its type and position. For arrays in COMMON or array parameters see 7.2.3.2 and 7.4.

7.2.3.1 Unchecked access

Example

The Fortran statements

```
DIMENSION  A(8),  I(6,8)
B = A(M)
J = I(K, L)
 A(8) = C
 INT = I(4, 3)
```

may be reduced to the following code body. A store map is given below to aid interpretation of the coding.

```
                                        Notes

(B = A [M])

                                        Real numbers occupy 2
 4      M                               locations each
 1      M                               RMOD = M*2
 5      RMOD
11      QF
 8      QF+1
 0      RMOD
/4      A-2                             Suffix count starts at 1
 5      B
+0

(J = I [K, L])
 4      L
 1      -1
12      +6                              (L-1) * 6
14      17                              (1st. bound = 6)
 1      K
 1      -1                              IMOD = (L-1)*6+(K-1)
 5      IMOD
 0      IMOD
/4      I
 5      J
 (A [8] =C)
 11     QF
  8     QF+1
  4     C
  5     A+14
 +0
 (INT=I [4, 3].)
  4     I+15
  5     INT
```

The order of the array elements in store is shown in the following table;

Location	Element	Location	Element
A	A_1	I	I_{11}
A+1		I+1	I_{21}
A+2	A_2	I+2	I_{31}
A+3		⋮	⋮
A+4	A_3		
A+5		I+7	I_{81}
⋮	⋮	I+8	I_{12}
		I+9	I_{22}
		⋮	⋮
A+14	A_8	I+15	I_{82}
A+15		I+16	I_{13}
		⋮	⋮
		I+63	I_{88}

7.2.3.2 Checked Access

The above example makes no check that a subscript is in the permitted range. If the user wishes to check the subscript, he may use the array "map" which is present in the store at run-time if the array has been declared in a DIMENSION statement.

The map starts at the location QYM. It contains consecutive 3-word entries stored in the order in which the arrays were declared. The meaning of each word is described in the following example which demonstrates the map produced by the statements:

```
SUBROUTINE  MTRXSR (R2, I, J)
DIMENSION   I1(4), R1(8, 8), I2(6, 2)
DIMENSION   R2(I, J), R3(9, 10)
COMMON      I2, R3
```

Example

Description of array	Map		Significance	Refer to
I1 (4 × 0)	QYM 0	I1	Pointer to base address	Note 1
	0	4	Dimension 1	
	0	0	Dimension 2 = 0	
R1 (8 × 8)	0	R1	Pointer to base address	
	0	8	Dimension 1	
	/0	8	Dimension 2; with real indicator	Note 3
I2 (6 × 2) in COMMON	+0		Offset in common area	7.2.4.
	/0	6	Dimension 1; with COMMON indicator	Note 2
	0	2	Dimension 2	
R2 (I × J) Formal parameter	1	QYP+0	Indirect pointer to actual map	Note 1
	0	0	Dummy dimension	
	/0	0	Dummy dimension, real indicator	
R3 (9 × 10) in COMMON	+12		Offset in common area	7.2.4
	/0	9	Dimension 1; with COMMON indicator	
	/0	10	Dimension 2; with real indicator	

Notes

1. The content of word 1 is one of 3 types:

   (i) If the array is in COMMON, it gives the address of the first element relative to the start of the common area. (For further details see 7. 2. 4).

   (ii) If the array is a formal parameter, bit 14 is set to 1 and the address bits contain the address of a location in the QYP stack. This location points toward word 1 of the map of the actual parameter. (see 7. 2. 5)

   (iii) If the array is an actual variable, bit 14 is set to 0 and the address bits point to the first word of the array.

2. Bit 18 of word 2 is set to 1 if and only if the array is in COMMON.

3. Bit 18 of word 3 is set to 1 if and only if the array contains real elements.

7. 2. 4 Variables in COMMON

Variables declared in a COMMON statement must be accessed by a B-lined instruction. The location QZCOM holds an address pointing to the first location of the common area.

Example

The Fortran statements

```
DIMENSION  ARR (3, 4)
COMMON     X, I, ARR

Y = X
I = 2
ARR (J, K) =  Y
Z=ARR (1, 2)
```

may be reduced to the following code body. A map of the common area is given to aid interpretation of the coding.

```
11      QF
 8      QF+1

 0      QZCOM       (Y=X)
/4      0
 5      Y
+0
 4      +2          (I=2)
 0      QZCOM
/5      2

 4      K           (ARR [J, K] =Y)
 1      - 1
12      +6          (*6 because real)
14      17
 1      J
 1      J
 1      - 2
 1      QZCOM
 5      R
11      QF
 8      QF+1
 0      R
 4      Y
/5      3
 0      QZCOM       (Z=ARR [1, 2])
/4      9
 5      Z
+0
```

Address relative to start of common area	Variable
0;	X
1;	
2;	I
3;	ARR(1, 1)
4;	
5;	ARR(2, 1)
6;	
25;	ARR(3, 4)
26;	

7.2.5 Parameters

No distinction is made between functions and sub-routines in the object code.

7.2.5.1 Defining sub-programs

A sub-program may be written entirely in code. If however an executable Fortran statement is included in the sub-program, then the sub-program head and any specification statements must also be written as Fortran text. (This is to ensure parameters are treated correctly in the executable statement.) The object code produced is described below.

<u>Example</u>

A, AP are real arrays

I is an integer
P, R1, R2 are real numbers

FUNC is a Fortran function

Using the above information, the Fortran statement

```
FUNCTION FUNC(I, R2, A, AP, P)
```

may be compiled to the following object code:

Code			Significance	Refer to
[FUNC				
QF				
QYO				
QZCOM]				
FUNC	+ 0		Link	
	11	QYO	Get actual parameters	
	8	QYO+1		
	+ 5		Number of parameters	
	&360000		Real/Integer display	7. 2. 5. 2
	+ 0		Space for array display	7. 2. 5. 2
QYP	+ 0			
	+ 0		Space for actual	Note 1
	+ 0		parameter pointers	7. 2. 5. 2
	+ 0			
	+ 0			
	+ 0			

Notes

1. The sub-routine entry

```
11  QYO
 8  QYO+1
```

places the addresses of the actual parameters in the locations QYP et seq. Even if an actual parameter is, in fact, a formal parameter of the calling sub-program (and the reference may involve a chain of indefinite length) this sub-routine will still place the address of the ultimate actual parameter in the appropriate QYP location.

If a formal parameter is an array the subroutine places the address of the first word of its map in the appropriate QYP location.

The last word in the QYP stack is used to hold the address where the result is to be stored, in the case of a FUNCTION sub-program.

7. 2. 5. 2 Calling sub-programs

The name of the sub-program must appear in a GLOBAL statement at the head of the block using it.

The call is a standard entry, followed by a string of parameter words, defining the actual parameters of the call and their type, and the store for the result.

Example

If a Fortran coded subprogram with heading:

```
SUBROUTINE   SUB (AP, P, I)

DIMENSION    A(100), AP (I)
```

Contains the Fortran statement

```
R1 = FUNC (I, R2, A, AP, P)
```

this statement may be reduced to the following object code:

Code	Significance	Refer to
11 FUNC 8 FUNC+1	Standard entry	
+5	Number of actual parameters	
&360000	Real/Integer display	Note 1
&140000	Array display	Note 2
0 I	Pointer to actual variable	Note 3
0 R2	Pointer to actual variable	Note 3
0 QYM+0	Pointer to map of A	Note 3;7. 2. 3. 2
1 QYP+0	Indirect pointer to map of AP	Note 3;7. 2. 3. 2
1 QYP+1	Indirect pointer to P	Note 3
0 R1	Pointer to store for result	

Simple variables in COMMON should not be used as express actual parameters.

Notes

1. Bit 18 = 1 if parameter 1 is real, = 0 if integer or non-existent; bit 17 = 1 if parameter 2 is real, = 0 if integer or non-existent; etc.

2. Bit 18 = 1 if parameter 1 is an array, = 0 if simple variable or non-existent; bit 17 = 1 if parameter 1 is an array, = 0 if simple variable or non-existent; etc.

3. If the actual parameter is also a formal parameter, bit 14 is set to 1 and the address bits point to the location (in the QYP stack) where the actual address may be found. Otherwise bit 14 is 0 and the address bits point to the actual parameter, or to a map of the actual parameter.

7. 3. 1 Floating Point Arithmetic

QF and QFMATH may be used as described in 903 Manual, vols. 2. 2. 8 and 2. 2. 14. However the following names must be used in code bodies for reference to QF and QFMATH:

Name used in description of subroutine	Name used in code body of 903FORTRAN program.
QF	QF
LN	QFLOG
EXP	QFEXP
SQRT	QFSQRT
SIN	QFSIN
COS	QFCOS
ARCTAN	QFATAN

These names must be included in a global identifier list at the head of any block which uses them, except QF which is automatically included in any global list punched by FORTRAN.

7. 3. 2 Library Functions

The library functions may be called in the same way as any other sub-program. In the 16K (LP) system the labels QFLOG, QFEXP, QFATAN etc. are not available for use by CODE statements, and the Library function entries must be used.

7. 3. 3 Input and Output

The 903 FORTRAN input and output package may be used in a code body. A description of data formats is given in Chapter 6. 3. The FORMAT statement must be written as Fortran text.

Example

Using the same variables as in 7. 2. 5. 2 the Fortran statement

WRITE (2, 999) A, P, AP, I, R1

may be compiled to the following object code

Coding	Significance	Refer to
11 QYO 8 QYO+2	Entry to input/output package	
/0 + 2	Channel number	Note 1
0 Q999+1	Pointer to format code	
+5	Number of parameters	
&720000	Real-Integer display	7. 2. 5. 2
&500000	Array display	7. 2. 5. 2
0 QYM+0	Pointer to map of A	7. 2. 5. 2; 7. 2. 3. 2
1 QYP+0	Indirect pointer to P	7. 2. 5. 2
1 QYP+1	Indirect pointer to map of AP	7. 2. 5. 2; 7. 2. 3. 2
0 I	Pointer to I	7. 2. 5. 2
0 R1	Pointer to R1	7. 2. 5. 2

Note 1 In the word after the subroutine entry:

(i) The sign bit is set to one if and only if the statement is a WRITE (output) statement.

(ii) The address part points to a location holding the integer value of the channel number.

7.4 Example

```
        GLOBAL, MXTRCE ]

        FUNCTION MXTRCE (A, I BOUND)

C       FINDS SUM OVER I OF A (I, I) ** 2
C       DIMENSION STATEMENT OMMITTED AS  A IS
C       NOT USED IN FORTAN CODE
        CODE
                   0     QYP+0          (FIND BASE ADDRESS OF A)
                  /4     1
                   9     COMMON
                  /4     0
                   6     +8191

                                                           X IS SQUARE)

                                                           R INDICATION)
```

PROBLEMS NOTED WITH
EXAMPLE 7.4

MXTRCE implies an integer function, but a real result is returned.

No check is made that IBOUND is less than the dimension of the array.

AJHerbert 22/03/2012

```
OK         0   QYP+1    (FIND BOUND)
          /4   0
           5   BOUND
           2   +0       (SET UP CONTROL UNIT)
           5   COUNT
           4   INCREM   (SET UP INCREMENT TO)
          14   1        (ACCESS DIAGONAL)
           1  -2
           5   INCREM
           4   +0       (SUM = 0.0)
           5   SUM
           5   SUM+1
           8   LOOPIN

LOOPBK     4   MODA
           1   INCREM
           5   MODA

LOOPIN    11   QF
           8   QF+1
           0   MODA     (SUM = SUM+A[I, I]*A[I, I])
          /4   0
         /12   0
           1   SUM
           5   SUM
          +0

          10   COUNT
           4   COUNT
           9   LOOPBK

           0   QYP+3    (MXTRCE = SUM)
           4   SUM
          /5   0
           4   SUM+1
          /5   1

FORTRAN

RETURN
END
```

LAST BIT OF SIR CODE SECTION

```
CODE
MODA      +0          (POINTER TO ARRAY ELEMENT)
BOUND     +0          (SIZE OF ARRAY)
COUNT     +0          (CONTROL COUNT)
INCREM    +0          (INCREMENT FOR MODA)
SUM       +0          (CUMULATIVE TOTAL)
          +0
LINK      +0          (LINK OF MXMULT FOR ERROR)
FORTRAN
Ⓗ
```

P 39 – 56 SIR~~&~~ CODE SECTION. LEFT ~~INB~~ IN BOOK.

Chapter 8: ERROR MESSAGES

8.1 Error Detected by the Translator

Errors are displayed in the form

E n+N
<statement>

where E is the error number (as listed below)
n is the last statement number
and N is the number of statements since n.

6 inches of blank tape are run-out on the punch before the first error in a program. When an error is detected compilation continues in the report mode (9.2.1) until the operator restarts compiling.

Error No.	Cause
1	Unacceptable form to left of = sign in an arithmetic statement.
3	Two operators in succession in an arithmetic expression.
5	Parentheses do not match.
6	Subscripted variable has not been declared in a DIMENSION statement, or a FUNCTION has not been declared in GLOBAL.
7	Unacceptable subscript form.
8	An unsubscripted array name has appeared in an arithmetic statement.
DO errors:	(In DO n $I=m_1, m_2, m_3$ or DO n $I=m_1, m_2$)
9	n is omitted, or is not an acceptable statement number.
10	I is not an unsubscripted integer variable, or is omitted.
11	There is an impermissible number of m_i's.
12	One of the m_i's is of impermissible form.
13	Do statement has no = sign.
14	DO statements have intersecting loops.
15	A DO loop terminates with a GOTO or IF statement.

Error No.	Cause
16	There is at least one unterminated DO loop when END is reached.
18	A number found where a variable expected.
19	No variable where one expected.
20	A statement number has more than 5 digits.
21	An integer or real constant is out of range.
22	A CALL statement has an unacceptable format.
23	A FUNCTION or SUBROUTINE statement has an unacceptable format.
24	The word FORTRAN has not been preceded by a CODE statement.
25	An error in GOTO or an IF statement.
26	A mispelt or otherwise corrupt or unrecognisable statement.
27	Statement too long or too complex to compile.
28	Program too long or too complex to compile.
29	Error in FORMAT statement.
31	Error in DIMENSION statement.
32	Error in COMMON statement.
33	A variable has been declared twice in a COMMON list.
38	In a DIMENSION statement, a bound exceeds 8192.
39	A variable has been declared twice in a DIMENSION list.
40	A sub-program has more than 18 parameters.
41	A continuation line has been used other than after a GLOBAL or FORMAT statement.
42	Too many sub-programs declared in a GLOBAL statement.
43	Error in a READ or WRITE statement.
44	A FUNCTION or SUBROUTINE statement has appeared in a sub-program.

Error No.	Cause
45	Error in a GLOBAL statement.
46	Too many variables have been used in a program or sub-program.
47 (i)	A sub-program does not contain a RETURN statement.
(ii)	RETURN has occurred outside a sub-program.
48	A number appears against a statement that should not be labelled.
49	Channel number in a READ or WRITE statement is not an integer.
50	<Halt code> not preceded by newline, therefore the last statement was not processed.

8.2 Queries raised by the translator.

The translator detects certain constructions that are not in themselves FORTRAN errors, but whose effect in 900 FORTRAN is probably not that desired by the programmer. If one of these constructions is detected a query is displayed, if the compiler is in Report Mode (9.2.1).

Queries are output in parentheses.
They take the form

(Q n+N)

where Q is the query number (as listed below)
n is the last statement number
N is the number of statements since N

If no on line teleprinter is fitted, queries are output on the punch, as comments in the SIR object code. Otherwise queries are displayed.

Query No.	Cause	Action taken by translator
101	Variable name has more than 6 characters	The first 6 are taken as significant
105	CONTINUE statement is not numbered	Statement ignored
106	Statement immediately following a GOTO statement is unnumbered	Statement is compiled normally
108	A FORMAT statement is unnumbered	Statement is ignored
109	An EQUIVALENCE statement has been used	The statement is ignored
110	An executable statement has occurred before a COMMON or a DIMENSION statement	Statement is compiled normally However, if the executable statement contained a variable mentioned in the specification statement it will have been compiled wrongly.
113	More than 1 GLOBAL statement in a program or sub-program	Statement is compiled normally.

8.3 Errors detected by the assembler.

Certain errors in the FORTRAN source text will not be detected at translation, but will be detected when the object code is assembled. A version of 903 SIR within the run-time package is used to assemble the object code. The following notes interpret some SIR error messages, giving possible reasons for their output.

EU Messages (Unlocated labels)

(1) Unlocated labels of the form Qn, where n is an integer:
The statement number n is referred to by one of the following types of statement, but n is not used as a statement number:

GOTO
IF
READ
WRITE

(2) Unlocated labels introduced by the user in GLOBAL statements:

EU messages giving sub-program names indicate that the sub-program has not been loaded.

(3) The unlocated label is a mis-spelling of a variable name:
The variable has been mis-spelt in one of the following positions:

the right-hand side of an arithmetic statement;
a WRITE list;
a CODE body

(4) The unlocated label is a variable name:
The variable has not been assigned a value by program;
the variable has been mis-spelt in one of the following positions:

the left-hand side of an arithmetic statement;
a READ list;
a CODE body

E3 Messages (label used twice)

(1) The label (with the first 6 characters significant) has been used as the name of at least two of the following in the same sub-program:

an array
a variable
a sub-program
a library function
a label in a CODE body

(2) Labels of the form Qn, where n is an integer:
the integer, n, has been used twice as a statement number.

E5 Messages (Store full)

If running the program in batch mode (9. 3. 1) re-run in relocatable mode (9. 3. 2). If in relocatable mode extra space may be gained by adding COMMON Statements to place arrays in the COMMON area.

Other error messages may be caused by mis-punched tape produced at translate time or by a mistake in a CODE body.

8.4 Errors Detected at Run-time.

Run-time errors are displayed on a newline. Continuation, if any is possible, is made by entry at 9.

Error indications take the form

En A n_1 n_2 n_3

where n is the error number as listed below
A is the address in the object program where the error was detected
n_1, n_2, n_3 are integers giving further information on the particular error; the significance of these are described below - less than 3 integers may be output.

E1 A P_1 P_2

Number of parameters in call of sub-program is not equal to number of parameters in its definition

P_1 Number of parameters in definition of sub-program
P_2 Number of parameters in call

No continuation possible.

E2 A Q_1 Q_2

Type of parameters in call of sub-program is not same as type of parameters in its definition

Q_1 Real/Integer display in definition of sub-program
Q_2 Real/Integer display in call of sub-program

Q_1, Q_2 represent a bit pattern indicating which parameters are real;

bit 18 = 1 if parameter 1 is real
= 0 if parameter 1 is integer
bit 17 = 1 if parameter 2 is real
= 0 if parameter 2 is integer

ie 131,000 ap

65,000 ap

So if Q_2 was 65536
bit 18 = 0 (param 1 integer)
bit 17 = 1 (" " 2 real)
in this case, output from subprogram was real, should have been integer.

No continuation is possible.

E3 A 1 S_1 S_2

Array subscript is out of range

S_1 Actual value of subscript
S_2 Maximum value allowed

On continuation S_1 is taken equal to 1 if $S_1 < 1$
or to S_2 if $S_1 > S_2$

E4 A 1 S_1 S_2

Computed GOTO variable out of range

S_1 Actual value of variable
S_2 Maximum value allowed

On continuation S_1 is taken equal to 1 if $S_1 < 1$
or to S_2 if $S_1 > S_2$

E6 A C

Parity error on input

C Decimal value of character read

On continuation the character is ignored.

E7 A C

Illegal character input in a number

C Decimal value of the character code

On continuation, the character is treated as a terminator.

E8 A C

First character not ′ on input of a string

C Decimal value of character input

On continuation, the character is ignored.

E9 A

Error in the parameters supplied to a coded subprogram (e.g. magnetic tape subroutine). Otherwise a corrupt code section.

E10 A M_3

Increment in a DO statement, $M_3 \leq 0$

On continuation, M_3 is taken to be +1.

E11 A N

Attempt to output a non-standardised real number to channel N.

On continuation, the number is ignored

E12 0

Compiler overwritten
Operator must re-input compiler (Tape 2 on an 8K system).

E13 0

Program incorrectly compiled.
The operator must re-compile the program, taking care to perform all operations, particularly the entry at 10 to indicate that program is complete.

E14 0

Object code and systems incompatible.
A program has beentranslated by one issue or version of Fortran and attempted to run on an incompatible version.

E15 X A

Real (floating point) overflow.
Ignore X. A is the address in object code where the error occurred.

E16 X A

Illegal instruction interpreted by QF.
(This means that the program has been corrupted in some way).
X and A have the meaning of E15.

E17 A

Overflow on conversion from real to integer value.
(Outside the range −131072 to +131071)

E18 A Z

Error in ALOG, EXP or SQRT.

Z = 78, Argument of ALOG(X), $X \leq 0$
On continuation result = 0

Z = 81, Argument of SQRT(X), $X<0$
On continuation result = SQRT(ABS(X))

Z= 88, Argument of EXP(X), $X>2^{16}$
(Also may be caused by exonentiation Y**X, see Chapter 2.5.3).
On continuation, result of EXP(X) is X.

8.4.1 Number Too Large For Style of Printing

******

A string of asterisks is output to maintain the format of results, if a number cannot be output in the format demanded: the number is ignored.

Continuation occurs automatically.

Chapter 9: OPERATION OF THE FORTRAN (BASIC SYSTEM)

9.1 General

On a basic 903 with 8192 word core store and no backing store:

Fortran programs are translated and run in two stages. In the first stage one or more programs are checked for syntactic errors and translated to SIR code on paper tape. In the second stage, the translated programs are loaded and run. The original Fortran text is referred to as "source code"; the output from the Translator is "object code".

9.1.1 Distribution

The basic 903 Fortran is distributed as a set of two tapes.

1) 903 FORTRAN TAPE 1 (The Translator).

2) 903 FORTRAN TAPE 2 (Run time routines).

9.2 Translation

The binary Translator tape (TAPE 1) is read in by Initial Instructions (entry at 8181). *

Fortran source code programs may be translated to paper tape, or checked for syntactic errors using the Report Mode (9.2.1).

1) To translate source code to paper tape enter at 8.

2) To check source code by Report Mode enter at 10.

3) To continue reading subsequent tapes of the same program in either mode re-enter at 9.

If any error is detected during translation, no more object code is produced, but the rest of the program is checked, as for Report Mode. A legible X is punched on the tape at the end of every sub-program containing an error, and the program waits. Queries are ignored during normal translation.

* Continuous output on the punch or automatic lighting of the paper tape "READ" lamp indicates misread or damaged tape.

When translation or checking of a program is complete, the operator should run out some blanks; and return to step (1) or (2) with another program, or proceed to run the translated program (9. 3).

Complete programs are normally translated as a whole, but individual programs and sub-programs may be translated or checked independently in the same manner. This may be done where a library of pre-translated sub-programs is used, or when testing a large program (to avoid re-compiling the whole program when an error is corrected in one sub-program).

9. 2. 1 Report Mode

When a tape is read in Report Mode, the same syntactic checks are carried out as in Translation mode. Errors and queries (see Chapter 8) are displayed, but no object code is punched.

The use of report mode saves time whenever a program does contain an error, and it is recommended that all programs are checked in this way before the first attempt to translate them.

9. 3 Loading and Running

There are two modes available for running 903 Basic Fortran programs, the Batch Mode and the Relocatable Mode. The Batch Mode should be used for all programs containing up to about 120 statements. Larger programs must be run in Relocatable Mode.

The operator must not attempt to run any program where the object code tape ends with a legible X (see 9. 2).

When loading an object code program a store map may be obtained. This lists the absolute addresses of all SIR object code identifiers. This map may be used to interpret the addresses output when run time errors are displayed. Fortran statement numbers (n) are printed in the form Qn on the store map.

9. 3. 1 Batch Mode

The binary run time package (TAPE 2) is input by initial instructions (entry at 8181). *

1) Load the first object code tape in the reader. If a store map is required go to step (3).

* Continuous output on the punch or automatic lighting of the read lamp indicates mis-read or damaged tape.

Tape ~~may stop in reader. If so, [illegible]~~?.

2) Enter at 8 to load the program. Go to step (4).

3) Enter at 12 to load the program and display a store map.

4) If the object code contains a halt code or is continued on further tapes, load the next tape and re-enter at 9.

5) Enter at 10 to indicate that the program is complete, including all sub-programs.

6) To run the program, load a data tape (if required) and enter at 11.

7) To continue after a PAUSE, halt code on data tape, or run time error, re-enter at 9. To re-run the program (e.g. with different data) go back to (6).

8) Return to step (1) to load the next program. However, if the program used variables in COMMON the assembler may be overwritten, and TAPE 2 must be re-input before loading another program.

Note 1.

The first piece of object code loaded must be the main program of the Fortran to be run. After this the sub-programs (if any) can be loaded in any order.

Note 2.

When a Fortran program has been tested in Batch Mode, the Relocatable mode may be used to produce a binary tape for repeated runs of the tested program.

9.3.2 Relocatable Mode

1) Load the binary run time package (TAPE 2) by initial instructions (entry at 8181) as for Batch Mode.

2) Load the first object code tape, run out some blanks on the punch, and enter at 13. The tape is translated to relocatable binary by a version of the SIR assembler.

3) If the program is continued on further tapes, load the next tape and re-enter at 9.

4) Enter at 10 to indicate that the program is complete, including all sub-programs.

5) Run out some blanks and tear off the tape. If further programs are to be run in Relocatable Mode, return to step (2).

6) Load the relocatable binary tape produced at step (2) and enter at 14.

7) Load the data tape (if any) and enter at 11 to run the program.

8) To continue after a wait: due to PAUSE, halt code on data tape, or run time error, re-enter at 9. To re-run the program return to step (7).

9. 3. 3 User's Library

When a 903 Fortran user has a library of commonly used sub-programs these may be kept as a set of pre-translated relocatable binary tapes.

Each sub-program should be translated and then converted to relocatable binary as in steps (1) to (5) of 9. 3. 2.

These library tapes can then be added to programs run in either Batch or Relocatable Mode:

To add a library sub-program in Batch Mode (9. 3. 1) after step (5) load the tape and enter at 15.

To add a library sub-program in Relocatable Mode (9. 3. 2) after step (6) load the tape and enter at 15.

Chapter 10: STORE USED ON THE BASIC SYSTEM

10. 1 Store Use at Translation Time.

The Translator (TAPE 1) with its workspace and lists, occupies 8150 words of store.

10. 2 Store Used at Run-time

The run-time routines occupy 3500 words approximately. In Batch Mode the Assembler occupies 2600 words, which may be used for the COMMON area. In relocatable Mode the Loader occupies 1100 words, which may be used for the COMMON area.

Thus on a basic 903, with 8192 words of store, the following table shows the store available for the Fortran programs:

	Batch Mode	Relocatable Mode
Object code + local variables and arrays	2000	3500
Object code + all variables and arrays including COMMON	4600	4600

The example below shows the store layout for a typical program run in Relocatable Mode.

0 to 7	Registers
8 to 3500	Entry points and run-time routines.
3500 to 6100	Object code of Fortran program.
6100 to 7300	COMMON area for Fortran program.
6120 to 7000	Dictionary list assembled when loading program (overwritten at run-time).
7000 to 8179	SIR loader (Fortran version).

Chapter 11. OPERATION OF THE FORTRAN 16K (LG) SYSTEM

11.1 General

FORTRAN programs may be translated and run "load-and-go" using this system, which uses a 903B or 903C configuration with at least 16384 words of core store. The chief objective of this system is to provide an efficient operating system when running large numbers of small programs.

11.1.1 Distribution

The 903 FORTRAN compiler for this system consists of one sum-checked binary tape.

11.2 Operation of the system

(1) The binary tape is read in by Initial Instructions (entry at 8181)*

(2) FORTRAN source code programs may be compiled load-and-go (11.2.2) or checked for syntactic errors using the Report Mode (11.2.1)

* Continuous output on the punch or automatic lighting of the paper tape "READ" lamp indicates misread or damaged tape.

11.2.1 Report Mode

To check source code by Report Mode enter at 16. When a tape is read in Report Mode, the same syntactic checks are carried out as in the Translation Mode. Errors and queries (see Chapter 8) are displayed, but no object code is produced.

The use of Report Mode saves time whenever a program does contain a syntactic error.

1) To check source code by Report Mode enter at 16

2) To continue after halt code or after a legible X is output on the punch, enter at 9.

11.2.2 Load and Go Mode

The Load and Go Mode may be used for all programs containing up to about 120 statements. Larger programs must be run in Relocatable Mode on the basic system or the 16K (LP) System.

When compiling an object code program load-and-go a store map may be obtained. This lists the absolute addresses of all SIR object code identifiers. This map may be used to interpret the addresses output when run time errors are displayed. FORTRAN statement numbers (n) are printed in the form Qn on the store map.

If any error is detected during translation the compiler enters Report Mode (see 11. 2. 1 above) automatically. A legible X is output on the punch at the end of every program unit containing an error, and the compiler waits.

(1) Load the first source code tape in the reader (see Note 1 below) If a store map is required go to step (3)

(2) Enter at 8 to compile and load the program. Go to step (4)

(3) Enter at 12 to compile the program and display a store map.

(4) If the source code contains a halt code or is continued on further tapes; load the next tape and re-enter at 9.

(5) Enter at 10 to indicate the program is complete, including all sub-programs.

(6) To run the program , load a data tape (if required) and enter at 11. (see Note 2 below)

(7) To continue after a PAUSE, halt code on a data tape, or run-time error, re-enter at 9.

(8) Return to step (1) to compile the next program (see Note 3 below)

Note 1. The first source code tape loaded must be the main FORTRAN program. After this sub-programs(if any) can be compiled in any order.

Note 2. The operator must not attempt to run the program if a legible X has been output during compilation.

Note 3. If the compiler has been overwritten by running the program, an error indication is output (see 11. 2. 4).

11. 2. 3 User's Library

When a 903 FORTRAN user has a library of commonly used sub-programs these may be kept as a set of pre-translated relocatable tapes. However, these have to be translated on the basic (8K) FORTRAN system (see Chapter 9. 3. 3 for details of this).

Once translated, these library tapes can be added to programs run in 'load-and-go' mode.

To add a library sub-program , after step (5) in 11. 2. 2 load the library tape and enter at 15.

11.2.4 Overwriting of compiler

To allow the maximum possible size of programs, certain parts of the compiler may be overwritten when running a large program. If the operator subsequently attempts to activate a part of the compiler that has been corrupted then the message

E 12 0

is displayed.

If this message is displayed the compiler must be reloaded. It will be displayed if address a3 (see 11.3.3 below) was greater than 5400 when the previous program was loaded.

11.3 Store Used

11.3.1 During Translation

0 to 7	Registers
8 to 3608	Runtime Routines*
3609 to 5400 (max)	Available for FORTRAN object code
5475 to 8100	SIR Assembler
8224 to 16330	FORTRAN Translator and its workspace

* These are not actually used during the compilation process.

11.3.2 At Runtime

0 to 7	Registers
8 to 3608	Runtime Routines
3609 to 5400 (max)	FORTRAN object code

COMMON area runs on from the end of object code.

11.3.3 Message Displayed

At the end of compilation, a message is displayed showing the store used by the program:

FIRST	LAST	COMM
a1	a2	a3

Programs and local variables occupy locations a1 to a2 inclusive; The COMMON are follows on from locations (a2 + 1) to a3.

In the current issue of the tape a3 is printed as a four digit number.

Where a3 exceeds 9999 the initial character is a SIR internal code character with a value greater than 9, i.e. the characters : ; < = $>_{10}$ represent 10,000, 11000, 12000, 13000, 14000, 15000 and 16000 respectively. An alphabetic character indicates that COMMON is too large for a 16384 word store.

11.4 SIZE of COMMON area

The total COMMON area should not normally exceed 8192 words of store, allowing 1 word each for integer and two words each for real values. However this limit may be exceeded if the last item in COMMON for any subprogram is an array which starts before relative location 8192, relative to the beginning of the COMMON area. For example if X, Y, K, I are simple variables and A, IA, B are arrays:

DIMENSION A(3000), IA(2000). B(2000).

COMMON X, Y, A, IA, J, K, B would be acceptable

but COMMON A., B, IA
and COMMON A, IA, B, X

would not be acceptable, since the last item begins above 8192, of the COMMON area.

If the above restriction is observed, the COMMON area may extend into further store modules.

Chapter 12. OPERATION OF THE FORTRAN 16K (LP) SYSTEM.

12.1 General.

Large FORTRAN programs may be translated and run in 2 passes using this system which uses a 903 configuration with at least 16384 words of core store. The object code produced must be run using this system, as described below. The objective of this system is to run the largest possible FORTRAN programs on a 16K machine.

12.1.1 Distribution.

The 903 FORTRAN compiler for this system consists of one sum-checked binary tape.

12.2 Operation of the system.

(1) The binary tape is read in by Initial Instructions (entry at 8181).*

(2) FORTRAN source code programs may be compiled to paper tape (12.2.2) or checked for syntactic errors using the Report Mode (12.2.1).

* Continuous output on the punch or automatic lighting of the paper tape "READ" lamp indicates misread or damaged tape.

12.2.1 Report Mode.

To check source code by Report Mode enter at 16. When a tape is read in Report Mode, the same syntactic checks are carried out as in the Translation Mode. Errors and queries (see Chapter 8) are displayed, but no object code is produced.

The use of Report Mode saves time whenever a program does contain a syntactic error, and it is recommended that all programs are checked in this way before the first attempt to translate them.

(1) To check source code by Report Mode, enter at 16.
(2) To continue after halt code or after a legible X is output on the punch, enter at 9.

12.2.2 Translation Mode (Relocatable Mode)

All programs must be translated in relocatable mode using this system.

If an error is detected during translation the compiler enters Report Mode (see 12.2.1 above) automatically.

A legible X is output on the punch at the end of every program unit containing an error, and the compiler waits. The operator must not attempt to run such a program.

(1) Load the first source code tape (see Note 1, below), run out some blanks on the punch, and enter at 13. The tape is compiled and translated to relocatable binary by a version of the SIR assembler.

(2) If the program is continued on further tapes, load the next tape and re-enter at 9.

(3) Enter at 10 to indicate the compilation is complete, (more tape will be punched).

(4) Run out some blanks and tear off the tape. If further programs are to be translated, return to step (1).

(5) Load the relocatable binary tape produced at step (4) and enter at 14. (see Note 1, below).

(6) Load the data tape (if any) and enter at 11 to run the program.

(7) To continue after a wait; due to a PAUSE, halt code on a data tape, or run-time error, re-enter at 9. To re-run the program return to step (6)

(8) To load another object code program return to step (5). If the COMMON area of the last program occupied more than 3000 locations the compiler must first be reloaded

Note 1. The main program and subprograms may be translated independantly, at different times, though in this case the complete program will occupy slightly more store. At step (5) the main program tape must be loaded first, followed by subprogram tapes at entry point 15.

12.2.3 User's Library.

When a 903 FORTRAN user has a library of commonly used sub-programs these may be kept as a set of pre-translated relocatable binary tapes.

Each sub-program should be translated to relocatable binary object code as in steps (1) to (4) of 9.2.2.

These library tapes can then be added to programs run in Relocatable Mode after step (5) load the tape and enter at 15.

Only library tapes compiled from the source code by the 16K (LP) version of 903 FORTRAN may be used.

12. 3 Store Used

12. 3. 1 Compile Time.

The Translator with its workspace and lists occupies 8150 words of module 1. A version of the SIR assembler, used to output the relocatable binary object code, occupies 2500 words of module 2. 2000 words of module 2 are shared between the SIR dictionary and the Translator's list of COMMON items.

12. 3. 2 Run-time.

Locations 32 to 8039 of module 1 are available for object code. 4392 words of module 2 are available as the COMMON area. The run-time routines occupy approximately 3800 words of module 2 and 140 words of module1. The SIR loader occupies 1000 words of module 2.

Thus on a 903 with 16384 words of store, the following table shows the store available for FORTRAN programs.

Object Program (including local variables and arrays)	8000 words (approx.)
COMMON area (starting at location 11992 (3800 + 8192))	4192 words

See also Chapter 11. 4 which applies to the 16K (LP) system.