PROGRAMMER'S REFERENCE MANUAL

Fortran

AUTOMATIC CODING SYSTEM

FOR THE IBM 704

Programmer's
Reference Manual
October 15, 1956

THE FORTRAN AUTOMATIC CODING SYSTEM FOR THE IBM 704 EDPM®

This manual supersedes all earlier information about the FORTRAN system. It describes the system which will be made available during late 1956, and is intended to permit planning and FORTRAN coding in advance of that time. An Introductory Programmer's Manual and an Operator's Manual will also be issued.

APPLIED SCIENCE DIVISION
AND PROGRAMMING RESEARCH DEPT.
International Business Machines Corporation
590 Madison Ave., New York 22, N. Y.

WORKING COMMITTEE

J. W. BACKUS
R. J. BEEBER
S. BEST
R. GOLDBERG
H. L. HERRICK
R. A. HUGHES
University of California Radiation Laboratory, Livermore, Calif.
L. B. MITCHELL
R. A. NELSON
R. NUTT
United Aircraft Corp., East Hartford, Conn.
D. SAYRE
P. B. SHERIDAN
H. STERN
I. ZILLER

THE FORTRAN SYSTEM

The IBM Mathematical Formula Translating System FORTRAN *is an automatic coding system for the IBM 704 EDPM. More precisely, it is a 704 program which accepts a* source program *written in a language – the* FORTRAN *language – closely resembling the ordinary language of mathematics, and which produces an* object program *in 704 machine language, ready to be run on a 704.*

FORTRAN *therefore in effect transforms the 704 into a machine with which communication can be made in a language more concise and more familiar than the 704 language itself. The result should be a considerable reduction in the training required to program, as well as in the time consumed in writing programs and eliminating their errors.*

Among the features which characterize the FORTRAN *system are the following.*

Size of Machine Required

The system has been designed to operate on a "small" 704, but to write object programs for any 704. (For further details, see the section on Source and Object Machines in Chapter 7.) If an object program is produced which is too large for the machine on which it is to be run, the programmer must subdivide the program.

Efficiency of the Object Program

Object programs produced by FORTRAN *will be nearly as efficient as those written by good programmers.*

Scope of Applicability

The FORTRAN *language is intended to be capable of expressing any problem of numerical computation. In particular, it deals easily with problems containing* *large sets of formulae and many variables, and it permits any variable to have up to three independent subscripts.*

However, for problems in which machine words have a logical rather than a numerical meaning it is less satisfactory, and it may fail entirely to express some such problems. Nevertheless, many logical operations not directly expressible in the FORTRAN *language can be obtained by making use of the provisions for incorporating library routines.*

Inclusion of Library Routines

Pre-written routines to evaluate any single-valued functions of any number of arguments can be made available for incorporation into object programs by placing them on the master FORTRAN *tape.*

Provision for Input and Output

Certain statements in the FORTRAN *language cause the object program to be equipped with its necessary input and output programs. Those which deal with decimal information include conversion to or from binary, and permit considerable freedom of format in the external medium.*

Nature of Fortran Arithmetic

Arithmetic in the object program will generally be performed with single-precision 704 floating point numbers. These numbers provide 27 binary digits (about 8 decimal digits) of precision, and may have magnitudes between approximately 10^{-38} *and* 10^{38}*, and zero. Fixed point arithmetic, but for integers only, is also provided.*

TABLE OF CONTENTS

CHAPTER 1. GENERAL PROPERTIES OF A FORTRAN SOURCE PROGRAM

A FORTRAN source program consists of a sequence of FORTRAN *statements.* There are 32 different types of statement, which are described in detail in the chapters which follow.

Example of a Fortran Program

The following brief program will serve to illustrate the general appearance and some of the properties of a FORTRAN program. It is shown as coded on a standard FORTRAN coding sheet.

C ← FOR COMMENT / STATEMENT NUMBER 1 5	CONTINUATION 6	FORTRAN STATEMENT 7 72	IDENTIFICATION 73 80
C		PROGRAM FOR FINDING THE LARGEST VALUE	
C	X	ATTAINED BY A SET OF NUMBERS	
		BIGA = A(1)	
		DO 20 I = 2,N	
		IF (BIGA - A(I)) 10, 20, 20	
10		BIGA = A(I)	
20		CONTINUE	

This program examines the set of n numbers a_i $(i=1,\ldots,n)$ and sets the quantity BIGA to the largest value attained in the set. It begins (after a comment describing the program) by setting BIGA equal to a_1. Next the DO statement causes the succeeding statements to and including statement 20 to be carried out repeatedly, first with $i=2$, then with $i=3$, etc., and finally with $i=n$. During each repetition of this loop the IF statement compares BIGA with a_i; if BIGA is less than a_i, statement 10, which replaces BIGA by a_i, is executed before continuing.

Keypunching the Program

Each statement is punched on a separate card. If a statement is too long to fit on a single card it can be continued over as many as 9 additional *continuation* cards. For each statement the initial card must contain either a zero or a blank in column 6; on continuation cards column 6 must not contain a zero or a blank, and it should be used to number the continuation cards consecutively from 1 to 9.

If a statement is too long to fit on a single line of the coding form, the programmer can signal to the keypuncher that he has continued on to the next line by placing a mark in the column labeled CONTINUATION.

The order of the statements is governed solely by the order of the cards. However, any number less than 2^{15} (=32768) may be associated with any statement by punching it in columns 1-5 of the initial card bearing that statement. Thereupon this number becomes the *statement number* of that statement. Statement numbers, besides permitting cross-references within the source program, also help the programmer to correlate the object program with his source program.

Punching the character C in column 1 will cause the card to be ignored by FORTRAN. Such cards may therefore be used to carry comments which will appear when the deck is listed.

Columns 73-80 are not read by FORTRAN and may be punched with any desired identifying information.

The statements themselves are punched in columns 7-72, both on initial and continuation cards. Thus a statement consists of not more than 10 x 66=660 characters. A table of the admissible characters in FORTRAN is given in Appendix A.

Blank characters, except in column 6, are simply ignored by FORTRAN, and the programmer may use blanks freely to improve the readability of his FORTRAN listing.

The actual input to FORTRAN is either this deck of cards or a BCD tape written from it on the 704 peripheral card-to-tape equipment with the standard SHARE 80 x 84 board. On the tape an End of File mark after the last card is required.

Preview of the Fortran Statements

The 32 types of statement, of which every FORTRAN program is composed, may be classified as follows.

1. The *arithmetic formula,* which causes the object program to carry out a numerical computation. Chapter 2 discusses the symbols available for referring to constants and variables, and Chapter 3 the combining of these into arithmetic formulas.
2. The 15 *control statements,* which govern the flow of control in the object program. These are discussed in Chapter 4.
3. The 13 *input-output statements,* which provide the object program with its necessary input and output routines. These are discussed in Chapter 5.
4. Finally, the 3 *specification statements,* which provide various information required or desirable to make the object program efficient, are discussed in Chapter 6.

Chapters 7 and 8, which conclude the manual, give additional detail on various topics and examples of FORTRAN programming.

CHAPTER 2. THE FORTRAN LANGUAGE: *Constants, Variables, and Subscripts*

Any programming language must provide for expressing numerical constants and variable quantities. FORTRAN also provides a subscript notation for expressing 1, 2, or 3-dimensional arrays of variables.

Constants

Two types of constant are permissible: fixed point (restricted to integers) and floating point (characterized by being written with a decimal point).

Fixed Point Constants.

GENERAL FORM	EXAMPLES
1 to 5 decimal digits. A preceding + or − sign is optional. The magnitude of the constant must be less than 32768.	3 +1 −28987

Any unsigned fixed point constant may be used as a statement number.

Floating Point Constants.

GENERAL FORM	EXAMPLES
Any number of decimal digits, with a decimal point at the beginning, at the end, or between two digits. A preceding + or − sign is optional. A decimal exponent preceded by an E may follow.	17. 5.0 −.0003 5.0E3 (= 5.0 x 10^3) 5.0E+3 (= 5.0 x 10^3) 5.0E−7 (= 5.0 x 10^{-7})

The magnitude of the number thus expressed must be zero, or must lie between the approximate limits of 10^{-38} to 10^{38}. The number will appear in the object program as a normalised single-precision floating point number.

Variables

Two types of variable are also permissible: fixed point (restricted to integral values) and floating point. Fixed point variables are distinguished by the fact that their first character is I, J, K, L, M, or N.

Fixed Point Variables.

GENERAL FORM	EXAMPLES
1 to 6 alphabetic or numeric characters (not special characters) of which the first is I, J, K, L, M, or N.	I M2 JOBNO

A fixed point variable can assume any integral value whose magnitude is less than 32768. However, see the section on Fixed Point Arithmetic in Chapter 7.

WARNING. The name of a variable must not be the same as the name of any *function* used in the program after the terminal F of the function name has been removed. Also, if a *subscripted* variable has 4 or more characters in its name, the last of these must not be an F. (For the meaning of "function" and "subscripted" see Chapter 3 and the last section of this chapter.)

Floating Point Variables.

GENERAL FORM	EXAMPLES
1 to 6 alphabetic or numeric characters (not special characters) of which the first is alphabetic but not I, J, K, L, M, or N.	A B7 DELTA

A floating point variable can assume any value expressible as a normalised 704 floating point number; i.e. zero, or with magnitude between approximately 10^{-38} and 10^{38}.

WARNING. The restrictions on naming fixed point variables also apply to floating point variables.

Subscripts and Subscripted Variables

A variable can be made to represent any member of a 1, 2, or 3-dimensional array of quantities by appending to it 1, 2, or 3 *subscripts;* the variable is then a *subscripted variable.* The subscripts are fixed point quantities whose values determine which member of the array is being referred to.

Subscripts.

GENERAL FORM		EXAMPLES
Let v represent any fixed point variable and c (or c') any unsigned fixed point constant. Then a subscript is an expression of one of the forms:	v	I
	c	3
	v+c or v—c	MU+2
	c*v	MU—2
	c*v+c' or c*v—c'	5*J
		5*J+2
		5*J—2

The symbol * denotes multiplication. The variable v must not itself be subscripted.

Subscripted Variables.

GENERAL FORM	EXAMPLES

A fixed or floating point variable followed by parentheses enclosing 1, 2, or 3 subscripts separated by commas.	A(I)
	K(3)
	BETA(5*J—2, K+2,L)

For each variable that appears in subscripted form the size of the array (i.e. the maximum values which its subscripts can attain) must be stated in a DIMENSION statement (see Chapter 6) preceding the first appearance of the variable.

The minimum value which a subscript may assume in the object program is + 1.

Arrangement of Arrays in Storage.

A 2-dimensional array A will, in the object program, be stored sequentially in the order $A_{1,1}$, $A_{2,1}$, ..., $A_{m,1}$, $A_{1,2}$, $A_{2,2}$, ..., $A_{m,2}$, ..., $A_{m,n}$. Thus it is stored "columnwise", with the first of its subscripts varying most rapidly, and the last varying least rapidly. The same is true of 3-dimensional arrays. 1-dimensional arrays are of course simply stored sequentially. All arrays are stored backwards in storage; i.e. the above sequence is in the order of decreasing absolute location.

CHAPTER 3. THE FORTRAN LANGUAGE: *Functions, Expressions, and Arithmetic Formulas*

Of the various FORTRAN statements it is the *arithmetic formula* which defines a numerical calculation which the object program is to do. A FORTRAN arithmetic formula resembles very closely a conventional arithmetic formula; it consists of the variable to be computed, followed by an = sign, followed by an arithmetic *expression*. For example, the arithmetic formula

Y = A–SINF(B–C)

means "replace the value of y by the value of a-sin(b-c)".

Functions

As in the above example, a FORTRAN expression may include the name of a *function* (e.g. the sine function SINF), provided that the routine for evaluating the function is either built into FORTRAN or is accessible to it as a pre-written subroutine in 704 language on the master FORTRAN tape.

GENERAL FORM	EXAMPLES

The name of the function is 4 to 7 alphabetic or numeric characters (not special characters), of which the last must be F and the first must be alphabetic. Also, the first must be X if and only if the value of the function is to be fixed point. The name of the function is followed by parentheses enclosing the arguments (which may be expressions), separated by commas.	SINF(A+B)
	SOMEF(X,Y)
	SQRTF(SINF(A))
	XTANF(3.*X)

Mode of a Function and its Arguments. Consider a function of a single argument. It may be desired to state the argument either in fixed or floating point; similarly the function itself may be in either of these modes. Thus a function of a single argument has 4 possible mode configurations; in general a function of n arguments will have 2^{n+1} mode configurations.

A separate name must be given, and a separate routine must be available, for each of the mode configurations which is used. Thus a complete set of names for the sine function might be

SIN0F	Fixed argument, floating function
SIN1F	Floating " , " "
XSIN0F	Fixed " , fixed "
XSIN1F	Floating " , " "

where the X's and F's are compulsory, but the remainder of the naming is arbitrary.

Built-In Functions. The FORTRAN system has the routines for evaluating certain functions built in. The list is as follows.

TYPE OF FUNCTION	DEFINITION	NO. OF ARGS	NAME	MODE OF ARGUMENT	MODE OF FUNCTION
Absolute value	$\lvert \mathrm{Arg} \rvert$	1	ABSF	Floating	Floating
			XABSF	Fixed	Fixed
Truncation	Sign of Arg times largest integer $\leq \lvert \mathrm{Arg} \rvert$	1	INTF	Floating	Floating
			XINTF	Floating	Fixed
Remaindering (see note below)	$\mathrm{Arg}_1 \pmod{\mathrm{Arg}_2}$	2	MODF	Floating	Floating
			XMODF	Fixed	Fixed
Choosing largest value	Max $(\mathrm{Arg}_1, \mathrm{Arg}_2, \ldots)$	$\geqslant 2$	MAX0F	Fixed	Floating
			MAX1F	Floating	Floating
			XMAX0F	Fixed	Fixed
			XMAX1F	Floating	Fixed
Choosing smallest value	Min $(\mathrm{Arg}_1, \mathrm{Arg}_2, \ldots)$	$\geqslant 2$	MIN0F	Fixed	Floating
			MIN1F	Floating	Floating
			XMIN0F	Fixed	Fixed
			XMIN1F	Floating	Fixed

NOTE. The function MODF $(\mathrm{Arg}_1, \mathrm{Arg}_2)$ is defined as $\mathrm{Arg}_1 - [\mathrm{Arg}_1/\mathrm{Arg}_2]\ \mathrm{Arg}_2$, where $[x] =$ integral part of x.

These 14 built-in subroutines are always compiled into the object program as open subroutines.

Functions on the Library Tape. Besides the built-in routines, any single-valued function of any number of arguments can be made available to the programmer by placing the appropriate routine on the master FORTRAN tape.

Any such routine will be compiled into the object program as a closed subroutine. In the section on Writing Subroutines for the Master Tape in Chapter 7 are given the specifications which any such routine must meet.

Expressions

An expression is any sequence of constants, variables (subscripted or not subscripted), and functions, separated by operation symbols, commas, and parentheses so as to form a meaningful mathematical expression.

However, one special restriction does exist. A FORTRAN expression may be either a fixed or a floating point expression, but it must not be a mixed expression. This does not mean that a floating point quantity can not appear in a fixed point expression, or vice versa, but rather that a quantity of one mode can appear in an expression of the other mode only in certain ways. Briefly, a floating point quantity can appear in a fixed point expression only as an argument of a function; a fixed point quantity can appear in a floating point expression only as an argument of a function, or as a subscript, or as an exponent.

Formal Rules for Forming Expressions. By repeated use of the following rules, all permissible expressions may be derived.

1. Any fixed point (floating point) constant, variable, or subscripted variable is an expression of the same mode. Thus 3 and I are fixed point expressions, and ALPHA and A(I,J,K) are floating point expressions.
2. If SOMEF is some function of n variables, and if E, F, ... , H are a set of n expressions of the correct modes for SOMEF, then SOMEF (E, F, ... , H) is an expression of the same mode as SOMEF.
3. If E is an expression, and if its first character is not + or –, then +E and –E are expressions of the same mode as E. Thus –A is an expression, but +–A is not.
4. If E is an expression, then (E) is an expression of the same mode as E. Thus (A), ((A)), (((A))), etc. are expressions.
5. If E and F are expressions of the same mode, and if the first character of F is not + or –, then

```
E + F
E – F
E * F
E / F
```

are expressions of the same mode. Thus A–+B and A/+B are not expressions. The characters +, –, *, and / denote addition, subtraction, multiplication, and division.

6. If E and F are expressions, and F is not floating point unless E is too, and the first character of F is not + or –, and neither E nor F is of the form A**B, then

```
E**F
```

is an expression of the same mode as E. Thus A**(B**C) is an expression, but I**(B**C) and A**B**C are not. The symbol ** denotes exponentiation; i.e. A**B means A^B.

Hierarchy of Operations. When the hierarchy of operations in an expression is not completely specified by parentheses, then it is understood to be in the following order (from innermost operations to outermost):

Exponentiation

Multiplication and Division

Addition and Subtraction

For example, the expression

```
A+B/C+D**E*F–G
```

will be taken to mean

$$A+(B/C)+(D^E*F)-G.$$

Ordering within a Hierarchy. Parentheses which have been omitted from a sequence of consecutive multiplications and divisions (or consecutive additions and subtractions) will be understood to be grouped from the left. Thus, if . represents either * or / (or either + or –), then

```
A.B.C.D.E
```

will be taken to mean

```
((((A.B).C).D).E)
```

Verification of Correct Use of Parentheses. The following procedure is suggested for checking that the parentheses in a complicated expression correctly express the desired operations.

Label the first open parenthesis "1"; thereafter, working from left to right, increase the label by 1 for each open parenthesis and decrease it by 1 for each closed parenthesis. The label of the last parenthesis should be 0; the mate of an open parenthesis labeled n will be the next parenthesis labeled n-1.

Optimisation of Arithmetic Expressions. The efficiency of the object program into which an arithmetic expression is translated may depend upon how the arithmetic expression is written. The section on Optimisation of Arithmetic Expressions in Chapter 7 mentions some of the considerations which affect object program efficiency.

Arithmetic Formulas

GENERAL FORM	EXAMPLES
"a=b" where a is a variable (subscripted or not subscripted) and b is an expression.	A(I)=B(I)+SINF(C (I))

The = sign in an arithmetic formula has the meaning "is to be replaced by". An arithmetic formula is therefore a command to compute the value of the right-hand side and to store that value in the storage location designated by the left-hand side.

The result will be stored in fixed or floating point according as the variable on the left-hand side is a fixed or floating point variable.

If the variable on the left is fixed point and the expression on the right is floating point, the result will first be computed in floating point and then truncated and converted to a fixed point integer. Thus, if the result is ±3.569 the fixed point number stored will be ±3, not ±4.

Examples of Arithmetic Formulas.

FORMULA	MEANING
A=B	Store the value of B in A.
I=B	Truncate B to an integer, convert to fixed point, and store in I.
A=I	Convert I to floating point and store in A.
I=I+1	Add 1 to I and store in I. This example illustrates the point that an arithmetic formula is not an equation but a command to replace a value.
A=MAX1F(SINF(B), COSF(B))	Replace A by the larger of the quantities sinB and cosB. This example illustrates the use of a function as an argument of a function.
A=3.0*B	Replace A by 3B.
A=3*B	Not permitted. The expression is mixed.
A=I*B	Not permitted. The expression is mixed.

CHAPTER 4. THE FORTRAN LANGUAGE: *Control Statements*

The second class of FORTRAN statements is the set of 15 control statements, which enable the programmer to state the flow of his program.

Unconditional GO TO

GENERAL FORM	EXAMPLES
"GO TO n" where n is a statement number.	GO TO 3

This statement causes transfer of control to the statement with statement number n.

Assigned GO TO

GENERAL FORM	EXAMPLES
"GO TO n, (n_1, n_2, ..., n_m)" where n is a non-subscripted fixed point variable appearing in a previously executed ASSIGN statement, and n_1, n_2, ..., n_m are statement numbers.	GO TO N, (7, 12, 19)

This statement causes transfer of control to the statement with statement number equal to the value of n last assigned by an ASSIGN statement. The n_1, n_2, ... , n_m are a list of the values which n may be assigned.
The assigned GO TO is used to obtain a pre-set many-way fork.

NOTE: When an assigned GO TO exists in the range of a DO, there is a restriction on the n_1, n_2, . . . , n_m. (See the section on DOs in this chapter.)

ASSIGN

GENERAL FORM	EXAMPLES
"ASSIGN i TO n" where i is a statement number and n is a non-subscripted fixed point variable.	ASSIGN 12 TO N

This statement causes a subsequent GO TO n, (n_1, . ., i, . ., n_m) to transfer control to the statement whose statement number is i.

The statement ASSIGN 12 TO N and the arithmetic formula N = 12 are *not* the same. A variable which has been assigned can be used only for an assigned GO TO until it is re-established as an ordinary variable.

Computed GO TO

GENERAL FORM	EXAMPLES
"GO TO $(n_1, n_2, \ldots\ldots, n_m)$, i" where $n_1, n_2, \ldots\ldots, n_m$ are statement numbers and i is a non-subscripted fixed point variable.	GO TO (30, 40, 50, 60), I

If at the time of execution the value of the variable i is j, then control is transferred to the statement with statement number n_j. Thus, in the example, if I has the value 3 at the time of execution, a transfer to statement 50 will occur. This statement is used to obtain a computed many-way fork.

IF

GENERAL FORM	EXAMPLES
"IF (a) n_1, n_2, n_3" where a is any expression and n_1, n_2, n_3 are statement numbers.	IF (A(J,K) —B) 10, 20, 30

Control is transferred to the statement with statement number n_1, n_2, or n_3 according as the value of a is less than, equal to, or greater than zero.

SENSE LIGHT

GENERAL FORM	EXAMPLES
"SENSE LIGHT i" where i is 0, 1, 2, 3, or 4.	SENSE LIGHT 3

If i is 0, all Sense Lights on the 704 console will be turned OFF; otherwise Sense Light i will be turned ON.

IF (SENSE LIGHT)

GENERAL FORM	EXAMPLES
"IF (SENSE LIGHT i) n_1, n_2" where n_1 and n_2 are statement numbers and i is 1, 2, 3, or 4.	IF (SENSE LIGHT 3) 30, 40

Control is transferred to the statement with statement number n_1 or n_2 according as Sense Light i is ON or OFF, and the Sense Light is turned OFF.

IF (SENSE SWITCH)

GENERAL FORM	EXAMPLES
"IF (SENSE SWITCH i) n_1, n_2" where n_1 and n_2 are statement numbers and i is 1, 2, 3, 4, 5, or 6.	IF (SENSE SWITCH 3) 30, 40

Control is transferred to the statement with statement number n_1 or n_2 according as Sense Switch i on the 704 console is DOWN or UP.

IF ACCUMULATOR OVERFLOW

GENERAL FORM	EXAMPLES
"IF ACCUMULATOR OVERFLOW n_1, n_2" where n_1 and n_2 are statement numbers.	IF ACCUMULATOR OVERFLOW 30, 40

Control is transferred to the statement with statement number n_1 or n_2 according as the Accumulator Overflow trigger of the 704 is ON or OFF, and the trigger is turned OFF.

IF QUOTIENT OVERFLOW

GENERAL FORM	EXAMPLES
"IF QUOTIENT OVERFLOW n_1, n_2" where n_1 and n_2 are statement numbers.	IF QUOTIENT OVERFLOW 30, 40

Control is transferred to the statement with statement number n_1 or n_2 according as the Multiplier-Quotient Overflow trigger of the 704 is ON or OFF, and the trigger is turned OFF.

IF DIVIDE CHECK

GENERAL FORM	EXAMPLES
"IF DIVIDE CHECK n_1, n_2" where n_1 and n_2 are statement numbers.	IF DIVIDE CHECK 30, 40

Control is transferred to the statement with statement number n_1 or n_2 according as the Divide Check trigger of the 704 is ON or OFF, and the trigger is turned OFF.

PAUSE

GENERAL FORM	EXAMPLES
"PAUSE" or "PAUSE n" where n is an unsigned **octal** fixed point constant.	PAUSE PAUSE 77777

The machine will HALT, with the octal number n displayed on the 704 console in the address field of the storage register. (If n is not stated it is taken to be 0.) Pressing the START button causes the program to resume at the next FORTRAN statement.

STOP

GENERAL FORM	EXAMPLES
"STOP" or "STOP n" where n is an unsigned **octal** fixed point constant.	STOP STOP 77777

This statement causes the machine to HALT in such a way that pressing the START button has no effect. Therefore, in contrast to the PAUSE, it is used where a get-off-the-machine stop, rather than a temporary stop, is desired. The octal number n is displayed on the 704 console in the address field of the storage register. (If n is not stated it is taken to be 0.)

DO

GENERAL FORM	EXAMPLES
"DO n i = m_1, m_2" or "DO n i = m_1, m_2, m_3" where n is a statement number, i is a non-subscripted fixed point variable, and m_1, m_2, m_3 are each either an unsigned fixed point constant or a non-subscripted fixed point variable. If m_3 is not stated it is taken to be 1.	DO 30 I = 1, 10 DO 30 I = 1, M, 3

The DO statement is a command to "DO the statements which follow, to and including the statement with statement number n, repeatedly, the first time with $i = m_1$ and with i increased by m_3 for each succeeding time; after they have been done with i equal to the highest of this sequence of values which does not exceed m_2 let control reach the statement following the statement with statement number n".

The *range* of a DO is the set of statements which will be executed repeatedly; it is the sequence of consecutive statements immediately following the DO, to and including the statement numbered n.

The *index* of a DO is the fixed point variable i, which is controlled by the DO in such a way that its value begins at m_1 and is increased each time by m_3 until it is about to exceed m_2. Throughout the range it is available for computation, either as an ordinary fixed point variable or as the variable of a subscript. During the last execution of the range, the DO is said to be *satisfied.*

Suppose, for example, that control has reached statement 10 of the program

```
10    DO 11 I = 1, 10
11    A(I) = I*N(I)
12
```

The range of the DO is statement 11, and the index is I. The DO sets I to 1 and control passes into the range. N(1) is computed, converted to floating point, and stored in A(1). Now, since statement 11 is the last statement in the range of the DO and the DO is unsatisfied, I is increased to 2 and control returns to the beginning of the range, statement 11. 2N(2) is computed and stored in A(2). This continues until statement 11 has been executed with I = 10. Since the DO is satisfied, control now passes to statement 12.

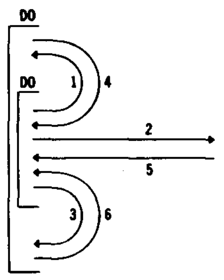
DOs within DOs. Among the statements in the range of a DO may be other DO statements. When this is so, the following rule must be observed.

Rule 1. If the range of a DO includes another DO, then all of the statements in the range of the latter must also be in the range of the former.

A set of DOs satisfying this rule is called a *nest* of DOs.

Transfer of Control and DOs. Transfers of control by IF-type or GO TO-type statements are subject to the following rule.

Rule 2. No transfer is permitted into the range of any DO from outside its range. Thus, in the configuration below, 1, 2, and 3 are permitted transfers, but 4, 5, and 6 are not.



EXCEPTION. There is one situation in which control *can* be transferred into the range of a DO from outside its range. Suppose control is somewhere in the range of one or more DOs, and that it is transferred to a section of program, completely outside the nest to which those DOs belong, which makes no change in any of the indexes or indexing parameters (m's) in the nest. Then after the execution of this section of program control can be transferred back to the same part of the nest from which it originally came. This provision makes it possible to exit temporarily from the range of a DO to execute a subroutine. An example is given in Chapter 8.

Restriction on Assigned GO TOs in the Range of a DO. When an assigned GO TO exists in the range of a DO, the statements to which it may transfer must all lie in one single part of the nest, or must all be completely outside the nest. If this condition cannot be met, a computed GO TO may be used.

Preservation of Index Values. When control leaves the range of a DO in the ordinary way (i.e. by the DO becoming satisfied and control passing on to the next statement after the range) the exit is said to be a *normal* exit. After a normal exit from a DO occurs, the value of the index controlled by that DO is not defined, and the index can not be used again until it is redefined. (See, however, the section on Further Details about DO Statements in Chapter 7.)

However, if exit occurs by a transfer out of the range, the current value of the index remains available for any subsequent use. If exit occurs by a transfer which is in the ranges of several DOs, the current values of all the indexes controlled by those DOs are preserved for any subsequent use.

Restriction on Calculations in the Range of a DO. Almost every type of calculation is permitted in the range of a DO. Only one type of statement is not permitted, namely any statement which redefines the value of the index or of any of the indexing parameters (m's). In other words, the indexing of a DO loop must be completely set before the range is entered.

CONTINUE

GENERAL FORM	EXAMPLES
"CONTINUE"	CONTINUE

CONTINUE is a dummy statement which gives rise to no instructions in the object program. Its most frequent use is as the last statement in the range of a DO, where it provides a statement number which can be referred to in transfers which are desired to become, in the object program, transfers to the indexing instuctions at the end of the loop.

As an example of a program which requires a CONTINUE, consider the table search program

```
10    DO 12 I = 1, 100
11    IF(ARG-VALUE(I)) 12, 20, 12
12    CONTINUE
13
```

This program will examine the 100-entry VALUE table until it finds an entry which equals ARG, whereupon it will exit to statement 20 with the successful value of I available for fixed point use; if no entry in the table equals ARG a normal exit to statement 13 will occur. The program

```
10    DO 11 I = 1, 100
11    IF(ARG-VALUE(I)) 11, 20, 11
12
```

would not work since, as stated in the next section, DO-sequencing does not occur if the last statement in the range of a DO is a transfer.

Summary of Fortran Sequencing

The precise laws which govern the order in which the statements of a FORTRAN program will be executed, and which have been left unstated till now, may be stated as follows.

1. Control begins at the first executable statement.

2. If control is in a statement S, then control will next go to the statement dictated by the *normal sequencing* properties of S. (The normal sequencing for each type of FORTRAN statement is given in Appendix B.)

3. EXCEPTION. If, however, S is the last statement in the range of one or more DOs which are not yet satisfied, and if S is not a transfer (IF-type or GO TO-type statement), then the normal sequencing of S is ignored and *DO-sequencing* occurs; i.e. control will next go to the first statement of the range of the nearest of the unsatisfied DOs, and the index of that DO will be raised.

NOTE. The statements FORMAT, DIMENSION, EQUIVALENCE, and FREQUENCY, which are discussed in the next two chapters, are non-executable statements, and in any question of sequencing are simply to be ignored.

WARNING. Every executable statement in the source program (except the first) must have some path of control leading to it; otherwise errors will occur in the compilation of the object program.

CHAPTER 5. THE FORTRAN LANGUAGE: *Input-Output Statements*

There are 13 FORTRAN statements available for specifying the transmission of information, during the running of the object program, between core storage on the one hand and tapes, drums, and attached card reader, card punch, and printer on the other hand. These 13 statements may be grouped as follows.

1. Five statements (READ, PUNCH, PRINT, READ INPUT TAPE, and WRITE OUTPUT TAPE) call for the transmission of a list of quantities between cores and an external storage medium — cards, printed sheet, or tape — in which information is expressed in Hollerith punch, decimal print, or BCD code, respectively.

2. One statement (FORMAT) is a non-executed statement which defines the information format in the external medium for the above 5 statements.

3. Four more statements (READ TAPE, READ DRUM, WRITE TAPE, and WRITE DRUM) call for ordinary binary transmission of a list of quantities.

4. Finally, there are 3 statements (END FILE, REWIND, and BACKSPACE) for manipulating tapes.

This chapter will first discuss the formation of a list of quantities for transmission, since such a list must appear in each of the 9 statements in groups 1 and 3 above. Next the method of writing a FORMAT statement will be described, and the format which input data to the object program must have. Finally, the statements in groups 1, 3, and 4 will be discussed.

Specifying Lists of Quantities

Each of the 9 statements which call for the transmission of information includes a list of the quantities to be transmitted. This list is ordered, and its order must be the same as the order in which the words of information exist (for input), or will exist (for output), in the external medium.

The formation and meaning of a list is best described by an example.

```
A, B(3), (C(I), D(I,K), I = 1,10), ((E(I,J), I = 1,10,1), F(J,3), J = 1,K)
```

Suppose that this list is used with an output statement. Then the information will be written in the external medium in the order

```
A, B(3), C(1), D(1,K), C(2), D(2,K), . . . . ., C(10), D(10,K),
E(1,1), E(2,1), . . . . ., E(10,1), F(1,3),
E(1,2), E(2,2), . . . . ., E(10,2), F(2,3), . . . . . . ., F(K,3).
```

Similarly, if this list were used with an input statement, the successive words, as they were read from the external medium, would be placed into the sequence of storage locations just given.

Thus the list reads from left to right and with repetition of variables enclosed within parentheses. Indeed *the repetition is exactly that of DO-repetition,* as if each open parenthesis (except subscripting parentheses) were a DO, with indexing given immediately before the mated closed parenthesis, and with range extending up to that indexing information. Thus the order of the above list is the same as of the "program"

```
1   A
2   B(3)
3   DO 5 I = 1, 10
4   C(I)
5   D(I,K)
6   DO 9 J = 1, K
7   DO 8 I = 1, 10, 1
8   E(I,J)
9   F(J,3)
```

Only variables, and not constants, may be listed.

Notice that indexing information, as in DOs, consists of 3 constants or fixed point variables, and that the last of these may be omitted, in which case it is taken to be 1.

For a list of the form K, A(K) or K, (A(I), I = 1,K), where an index or indexing parameter itself appears *earlier* in the list of an input statement, the indexing will be carried out with the newly read-in value.

When it is desired to transmit an *entire* array, and in its natural order (see the section on Arrangement of Arrays in Storage in Chapter 2), then an abbreviated notation may be used; only the name of the array need be given and the indexing information may be omitted. FORTRAN will look to see if a DIMENSION statement (see Chapter 6) has been made about that name; *if it has, indexing information to transmit the entire array in natural order* will be supplied automatically, while if it has not, only a single variable will be transmitted. Thus, in the example, the entire A-array will be transmitted, including the special case where the array consists of only a single quantity.

WARNING. The information given in this section applies in its full generality only to lists which are given with the 5 *decimal* statements of group 1. For the *binary* statements WRITE DRUM and READ DRUM only the abbreviated notation mentioned immediately above is permitted; the restrictions which affect lists for WRITE TAPE and READ TAPE are discussed in the section on Lists for Binary Tape Operations in Chapter 7.

FORMAT

GENERAL FORM	EXAMPLES
"FORMAT (Specification)" where Specification is as described below.	FORMAT (12/(E12.4, F10.4))

The 5 decimal input-output statements of group 1 contain, in addition to the list of quantities to be transmitted, the statement number of a FORMAT statement, which describes the information format which exists, or is to be produced, in the external medium. It also specifies the type of conversion between binary and decimal which is to be performed. FORMAT statements are not executed, their function being merely to supply information in the object program. Therefore they may be placed anywhere in the source program.

The Specification. For the sake of definiteness the details of writing a FORMAT Specification are given for use with a PRINT statement. However, the description is valid for any case simply by generalizing the concept of "printed line" to that of unit record in the external medium. Thus a unit record may be

1. A printed line with a maximum of 120 characters.
2. A punched card with a maximum of 72 characters.
3. A BCD tape record with a maximum of 120 characters.

Three basic types of decimal-to-binary or binary-to-decimal conversion are available:

	INTERNAL	TO OR FROM EXTERNAL
E	Floating point variable	Floating point decimal
F	Floating point variable	Fixed point decimal
I	Fixed point variable	Decimal integer

The FORMAT specification describes the line to be printed by giving, for each field in the line (from left to right, beginning with the first type wheel):

1. The type of conversion (E, F, or I) to be used;
2. The width (w) of the field; and
3. For E- and F-conversion, the number of places (d) after the decimal point that are to be rounded and printed. If d is not less than 10 it is treated mod 10.

These basic field specifications are given in the forms

Iw, Ew.d, and Fw.d

with the specification for successive fields separated by commas. Thus the statement FORMAT (I2, E12.4, F10.4) might give the line

```
27 -0.9321E 02    -0.0076
```

As in this example the field widths may be made greater than necessary, so as to ensure spacing blanks between numbers. In this case there is 1 blank following the 27, 1 blank automatically supplied after the E, and 3 blanks after the 02. Within each field the printed output is always pushed to the extreme right.

It may be desired to print n successive fields in the same fashion. This may be done by giving n before the E, F, or I. Thus the statement FORMAT (I2, 3E12.4) might give

```
27 -0.9321E 02 -0.7580E-02  0.5536E 00
```

To permit the repetition of *groups* of field specifications a limited parenthetical expression is permitted. Thus FORMAT (2(F10.6, E10.2), I4) is equivalent to FORMAT (F10.6, E10.2, F10.6, E10.2, I4). No provision is made for parentheses within parentheses.

To permit more general use of F-conversion, a *scale factor* followed by the letter P may precede the specification. The scale factor is so defined that

$$\text{Printed number} = \text{Internal number} \times 10^{\text{Scale factor}}$$

Thus the statement FORMAT (I2, 1P3F11.3), used with the data of the preceding example, would give

```
27    -932.096      -0.076       5.536
```

while FORMAT (I2, -1P3F11.3) would give

```
27      -9.321      -0.001       0.055
```

A positive scale factor may also be used with E-conversion to increase the number and decrease the exponent. Thus FORMAT (I2, 1P3E12.4) would give with the same data

```
27 -9.3210E 01 -7.5804E-03  5.5361E-01
```

NOTE. The scale factor is assumed to be zero if no other value has been given. However, once a value has been given it will hold for all F- and E-conversions until a new one is given. The scale factor has no effect on I-conversion.

Hollerith Fields. A field may be designated as a Hollerith field, in which case English text will be printed in it. The field width, followed by the desired characters, should appear in the appropriate place in the specification. For example, FORMAT (3HXY= F8.3, 4H Z = F6.2, 7H   W/AF = F7.3) would give with the same data

```
XY=  -93.210  Z=  -0.01  W/AF=   0.554
```

Notice that any Hollerith characters, including blanks, may be printed. This is the sole exception to the statement made in Chapter 1 that FORTRAN ignores blanks.

It is possible to print Hollerith information only, by giving no list with the input-output statement and setting up no I, E, or F fields in the FORMAT statement.

Consider a Hollerith field in a FORMAT statement at the time of execution of the object program. If the FORMAT statement is being used with an input statement, the Hollerith text in the FORMAT statement will be replaced with whatever text is read from the corresponding field in the external medium. When the FORMAT statement is used with an output statement, whatever text is currently in the FORMAT statement will be outputted. Thus text can be originated either at source time or at object time by not using, or using, the FORMAT statement with an input statement.

Multi-Record Formats. To deal with a *block* of printing a FORMAT specification may have several different line formats, separated by the slash /. Thus FORMAT (3F9.2, 2F10.4/8E14.5) would specify a block in which lines 1, 3, 5, ... have format 3F9.2, 2F10.4 and lines 2, 4, 6, ... have format 8E14.5.

If a block format is desired having the first two lines of some special formats and all the remaining lines of another format, the last line of the format should be enclosed in parentheses; e.g. FORMAT (I2, 3E12.4/2F10.3, 3F9.4/(10F12.4)).

In general, if there are items in the list still remaining to be transmitted after the format specification has been completely used, the format repeats from the last open parenthesis or (if no parentheses are present) from the beginning.

Blank lines may be introduced into a block by omitting format information; thus // and /// will give 1 and 2 blank lines respectively.

Carriage Control. When WRITE OUTPUT TAPE is being used to write a BCD tape for subsequent printing, the question of carriage control of the printer must be considered. The peripheral printer can operate in 3 modes: single space, double space, and Program Control, of which the last gives the greatest flexibility.

Under Program Control the carriage is controlled by the first character of each BCD record, and that character is not printed. The various control characters and their effects are

Blank	Single space before printing
0	Double space before printing
+	No space before printing
1-9	Skip to channels 1-9
J-R	Short skip to channels 1-9

Thus a FORMAT Specification for WRITE OUTPUT TAPE for printing with Program Control will usually begin with 1H followed by the appropriate control character. The same is true of PRINT since in FORTRAN printing on the attached printer simulates Program Control printing on the peripheral printer.

Data Input to the Object Program

Decimal data to be read by a READ or READ INPUT TAPE at the time of execution of the object program must be in essentially the same format as given in the examples of the preceding section. Thus a card to be read with FORMAT (I2, E12.4, F10.4) might be punched

```
27 -0.9321E 02    -0.0076
```

Within each field all information must be pushed to the extreme right. Positive signs may be indicated either by a blank or a +; – signs may be punched with an 11-punch or an 8-4 punch (see Appendix A). Blanks in numeric fields are regarded as zeroes. Numbers for E- and F-conversion may contain any number of digits, but only 8 digits of accuracy will be retained. Numbers for I-conversion will be treated mod 32768.

To permit economy in keypunching certain relaxations in input data format are permitted.

1. Numbers for E-conversion need not have 4 columns devoted to the exponent field. The start of the exponent field must be marked by an E, or if that is omitted, by a + or – (not a blank). Thus E2, E02, +2, +02, E 02, and E+02 are all permissible exponent fields.

2. Numbers for E- or F-conversion need not have the decimal point punched. If it is not punched the FORMAT Specification sets its effective position; for example, −09321+2 with E12.4 will be treated as if the decimal point had been punched 4 places before the start of the exponent field, that is between the 0 and the 9. If the decimal point is punched, its position overrides the value of d given in the FORMAT Specification.

READ

GENERAL FORM	EXAMPLES
"READ n, List" where n is the statement number of a FORMAT statement, and List is as previously described.	READ 30, K, A(K)

The READ statement causes the object program to read cards from the attached card reader. Record after record (card after card) is read until the complete list has been brought in, converted, and stored in the locations specified by the list. The FORMAT statement describes the arrangement of information on the cards and the type of conversion to be done.

If an End of File is encountered (the program attempts to read a card and finds that there is none in the card reader) the object program HALTS. Placing more cards in the card reader and pressing the START button causes the program to continue the reading from the point in the list which it had reached.

A partial check is made in the object program for incorrectly punched columns. Such a column causes a HALT. Pressing the START button causes the faulty column to be treated as a zero, and the program to continue.

READ INPUT TAPE

GENERAL FORM	EXAMPLES
"READ INPUT TAPE i, n, List" where i is an unsigned fixed point constant between 1 and 10 inclusive or a fixed point variable, n is the statement number of a FORMAT statement, and List is as previously described.	READ INPUT TAPE 3, 30, K, A(K) READ INPUT TAPE I, 30, K, A(K)

The READ INPUT TAPE statement causes the object program to read BCD information from tape unit i, where i = 1, 2, . . ., 10. Record after record is brought in, in accordance with the FORMAT statement, until the complete list has been placed in storage.

An End of File causes a HALT in the object program. Pressing the START button causes the program to continue the reading from the point in the list which it had reached.

The object program redundancy checks the tape reading. If a record fails twice the program HALTS. Pressing the START button causes the information read on the second attempt to be accepted and the program to continue.

PUNCH

GENERAL FORM	EXAMPLES
"PUNCH n, List" where n is the statement number of a FORMAT statement and List is as previously described.	PUNCH 30, (A(J), J = 1, 10)

The PUNCH statement causes the object program to punch cards on the attached card punch. Card after card is punched in accordance with the FORMAT statement until the complete list has been punched.

No checking is done, and there are no HALTS in the object program.

PRINT

GENERAL FORM	EXAMPLES
"PRINT n, List" where n is the statement number of a FORMAT statement and List is as previously described.	PRINT 30, (A(J), J = 1, 10)

The PRINT statement causes the object program to print on the attached printer. Line after line is printed in accordance with the FORMAT statement until the complete list has been printed.

The printing is echo checked. A printing error so detected causes the object program to HALT. Pressing the START button causes the program to continue. Pressing the RESET button and then the START button causes the line to be printed again and the program to continue.

WRITE OUTPUT TAPE

GENERAL FORM	EXAMPLES
"WRITE OUTPUT TAPE i, n, List" where i is an unsigned fixed point constant between 1 and 10 inclusive or a fixed point variable, n is the statement number of a FORMAT statement, and List is as previously described.	WRITE OUTPUT TAPE 3, 30, (A(J), J = 1, 10) WRITE OUTPUT TAPE I, 30, (A(J), J = 1, 10)

The WRITE OUTPUT TAPE statement causes the object program to write BCD information on tape unit i, where i = 1, 2, . . ., 10. Record after record is written in accordance with the FORMAT statement until the complete list has been written. No End of File is written after the last record.

No checking is done, and there are no HALTS in the object program.

READ TAPE

GENERAL FORM	EXAMPLES
"READ TAPE i, List" where i is an unsigned fixed point constant between 1 and 10 inclusive or a fixed point variable, and List is as described in Chapter 7.	READ TAPE 3, (A(J), J = 1, 10) READ TAPE I, (A(J), J = 1, 10)

The READ TAPE statement causes the object program to read binary information from tape unit i, where i = 1, 2, ..., 10. Only one record is read, and it will be completely read only if the list contains as many words as the record. The tape, however, always moves all the way to the next record.

If the list is longer than the record, the object program will stop with a Read-Write Check, and the program will not be able to continue.

An End of File causes a HALT in the object program. Pressing the START button causes the program to read the next record. A READ TAPE may, however, be given without a list, in which case it will simply skip over a record or an End of File.

For reasons of timing there are limitations on the complexity of the list. See the section on Lists for Binary Tape Operations in Chapter 7.

The object program redundancy checks the tape reading. (The longitudinal check is applied only if the whole record is read.) If a record fails twice the program HALTS. Pressing the START button causes the information read on the second attempt to be accepted and the program to continue.

READ DRUM

GENERAL FORM	EXAMPLES
"READ DRUM i, j, List" where i and j are each either an unsigned fixed point constant or a fixed point variable, with the value of i between 1 and 8 inclusive, and List is as described below.	READ DRUM 2, 1000, A, B, C, D READ DRUM I, J, A, B, C, D

The READ DRUM statement causes the object program to read words of binary information from consecutive locations on drum i, where i = 1, 2, ... , 8, beginning with the word in drum location j, where j = 0, 1, ... , 2047. (If j $\geqslant$ 2048 it is interpreted mod 2048.) Reading continues until the complete list has been read in.

For reasons of timing there are stringent limitations on the complexity of the list. In fact, the list can employ only the abbreviated notation described earlier in this chapter; it may consist only of variables without subscripts, as A, B, C, D, . . . Those variables which are simple will be read into storage in the ordinary way; those which are arrays will be read with indexing obtained from their DIMENSION statements (see Chapter 6). Thus with READ DRUM the full array must be read in, and in natural order. For example, if there is a DIMENSION statement saying that A is a 2-dimensional array with maximum indexes 5, 10, but no DIMENSION statements for B, C, D, the list above will be treated as if it were written

((A(I,J), I = 1, 5), J = 1, 10), B, C, D

No checking is done and there are no HALTS in the object program.

WRITE TAPE

GENERAL FORM	EXAMPLES
"WRITE TAPE i, List" where i is an unsigned fixed point constant between 1 and 10 inclusive or a fixed point variable, and List is as described in Chapter 7.	WRITE TAPE 3, (A(J), J = 1, 10) WRITE TAPE I, (A(J), J = 1, 10)

The WRITE TAPE statement causes the object program to write binary information on tape unit i, where i = 1, 2, ... , 10. Only one record is written; its length will be that of the list.

For reasons of timing, there are limitations on the complexity of the list. See the section on Lists for Binary Tape Operations in Chapter 7.

No checking is done, and there are no HALTS in the object program.

WRITE DRUM

GENERAL FORM	EXAMPLES
"WRITE DRUM i, j, List" where i and j are each either an unsigned fixed point constant or a fixed point variable, with the value of i between 1 and 8 inclusive, and List is as described for READ DRUM.	WRITE DRUM 2, 1000, A, B, C, D WRITE DRUM I, J, A, B, C, D

The WRITE DRUM statement causes the object program to write words of binary information into consecutive locations on drum i, where i = 1, 2, ..., 8, beginning with drum location j, where j = 0, 1, ..., 2047. (If $j \geqslant 2048$ it is interpreted mod 2048.) Writing continues until the complete list has been written.

The list is subject to the same restrictions as for READ DRUM.

No checking is done and there are no HALTS in the object program.

END FILE

GENERAL FORM	EXAMPLES

"END FILE i" where i is an unsigned fixed point constant between 1 and 10 inclusive or a fixed point variable.	END FILE 3
	END FILE I

The END FILE statement causes the object program to write End of File on tape unit i, where i = 1, 2, ..., 10.

REWIND

GENERAL FORM	EXAMPLES
"REWIND i" where i is an unsigned fixed point constant between 1 and 10 inclusive or a fixed point variable.	REWIND 3 REWIND I

The REWIND statement causes the object program to rewind tape unit i, where i = 1,2, ..., 10.

BACKSPACE

GENERAL FORM	EXAMPLES
"BACKSPACE i" where i is an unsigned fixed point constant between 1 and 10 inclusive or a fixed point variable.	BACKSPACE 3 BACKSPACE I

The BACKSPACE statement causes the object program to backspace tape unit i by one record, where i = 1, 2, ..., 10.

Error Halts

The several HALTS which can occur during input or output operation in the object program can be identified by the contents of the storage register on the 704 console. The FORTRAN Operator's Manual contains a list of these HALTS and their identifications.

CHAPTER 6. THE FORTRAN LANGUAGE: *Specification Statements*

The last class of FORTRAN statements is the set of 3 specification statements DIMENSION, EQUIVALENCE, and FREQUENCY. These are statements which are not executed, but which furnish information for use by FORTRAN to make the object program efficient.

DIMENSION

GENERAL FORM	EXAMPLES
"DIMENSION v, v, v, . . ." where each v is a variable subscripted with 1, 2, or 3 unsigned fixed point constants. Any number of v's may be given.	DIMENSION A(10), B(5, 15), C(3, 4, 5)

The DIMENSION statement provides the information necessary to allocate storage in the object program for arrays of quantities.

Every variable which appears in the program in subscripted form must appear in a DIMENSION statement, and the DIMENSION statement must precede the first appearance of the variable. In the DIMENSION statement are given the desired dimensions of the array; in the executed program the subscripts of that variable must never take on values larger than those dimensions.

Thus the example states that B is a 2-dimensional array and that the subscripts of B will never exceed 5 and 15; it causes 75 words of storage to be set aside for the B array.

A single DIMENSION statement may be used to dimension any number of arrays.

EQUIVALENCE

GENERAL FORM	EXAMPLES
"EQUIVALENCE (a, b, c, ...), (d, e, f, ...), .." where a, b, c, d, e, f, ... are variables optionally followed by a single unsigned fixed point constant in parentheses.	EQUIVALENCE (A, B(1), C(5)), (D(17), E(3))

The EQUIVALENCE statement enables the programmer, if he wishes, to control the allocation of data storage in the object program. In particular, it permits him to economise on data storage requirements by causing storage locations to be shared by two or more quantities, when the logic of his program permits. It also permits him, if he wishes, to call the same quantity by several different names, and then ensure that those names are treated as equivalent.

An EQUIVALENCE statement may be placed anywhere in the source program. Each pair of parentheses encloses the names of two or more quantities whose storage locations are to be made the same in the object program; any number of equivalences (pairs of parentheses) may be given.

In an EQUIVALENCE statement the meaning of C(5), for example, is "the 4th storage location in the object program after the cell containing C, or (if C is an array) after C(1) or C(1,1) or C(1,1,1)". In general A(p) is defined for $p \geqslant 1$ and means the $p-1^{th}$ location after A or the beginning of the A array; i.e. the p^{th} location in the array. If p is not given, it is taken to be 1.

Thus the example statement causes A, B, and C (or the beginnings of the A, B, and C arrays) to be so placed in storage that the location containing A, the location containing B, and the 4th location after that containing C, are the same location. Similarly, it causes the 16th location after D and the 2nd after E both to be another location.

A quantity or array which does not appear in any EQUIVALENCE statement will have storage exclusively to itself.

Locations can be shared only among variables, not among constants.

The sharing of storage locations cannot be planned safely without a knowledge of which FORTRAN statements, when executed in the object program, will cause a new value to be stored in a storage location. There are 7 such statements.

1. Execution of an arithmetic formula stores a new value of the variable on its left-hand side.

2. Execution of an ASSIGN i TO n stores a new value in n.

3. Execution of a DO will in general store a new value of the index. (It will not always do so, however; see the section on Further Details about DO Statements in Chapter 7.)

4. Execution of a READ, READ INPUT TAPE, READ TAPE, or READ DRUM stores new values of the variables listed.

FREQUENCY

GENERAL FORM	EXAMPLES
"FREQUENCY n(i, j, . . .), m(k, l, . . .), . . ." where n, m, . . . are statement numbers and i, j, k, l, . . . are unsigned fixed point constants.	FREQUENCY 30(1, 2, 1), 40(11), 50(1, 7, 1, 1)

The FREQUENCY statement permits the programmer to give his estimate, for each branch-point of control, of the frequencies with which the several branches will actually be executed in the object program. This information is used to optimise the use of index registers in the object program.

A FREQUENCY statement may be placed anywhere in the source program, and may be used to give the frequency information about any number of branch-points. For each branch-point the information consists of the statement number of the statement causing the branch, followed by parenthesis enclosing the estimated frequencies separated by commas.

Consider the example. This might be a FREQUENCY statement in a program in which statement 30 is an IF, 40 is a DO, and 50 is a computed GO TO. The programmer estimates that the argument of the IF is as likely to be zero as non-zero, and when it is non-zero it is as likely to be negative as positive. The DO statement at 40 is presumably one for which at least one of the indexing parameters (m's) is not a constant but a variable, so that the number of times the loop must be executed to make a normal exit is not known in advance; the programmer here estimates that 11 is a good average for that number. The computed GO TO at 50 is estimated to transfer to its four branches with frequencies 1, 7, 1, 1.

All frequency estimates, except those about DOs, are *relative;* thus they can be multiplied by any constant. The example statement, for instance, could equally well be given as FREQUENCY 30(2,4,2), 40(11), 50(3,21,3,3). A frequency may be estimated as 0; this will be taken to mean that the frequency is very small.

The following table lists the 8 types of statement about which frequency information may be given.

TYPE	NO OF FREQS	REMARKS
Computed GO TO	$\geqslant 2$	Order of frequencies: same as order of branches
IF	3	" " " " " " " "
IF (SENSE SWITCH)	2	" " " " " " " "
IF ACCUMULATOR OVERFLOW	2	" " " " " " " "
IF QUOTIENT OVERFLOW	2	" " " " " " " "
IF DIVIDE CHECK	2	" " " " " " " "
PAUSE	2	" " " number of times START will be pressed, number of times it will not.
DO	1	To be given only when m_1, m_2, or m_3 is variable.

It is not necessary to give frequency information about any branch-point. If none is given, it will be taken that the probabilities of all branches are equal.

A frequency estimate concerning a DO will be ignored except when at least one of the indexing parameters of that DO is variable. Moreover, the frequency estimate should be based only on the expected values of those parameters; in other words, even if the range of the DO contains IFs or GO TOs which may transfer outside the range, the frequency estimate should be the number of times the range must be executed to cause a normal exit.

A DO for which the indexing parameters are variable and for which no FREQUENCY statement is given will be treated as if a frequency of 5 had been estimated.

CHAPTER 7. MISCELLANEOUS DETAILS ABOUT FORTRAN

Source and Object Machines

The *source* machine is the 704 on which the source program is translated into the object program. The *object* machine is that on which the object program is run.

The source machine must be at least as large as a "small" 704; i.e. a 704 possessing 4096 words of core storage, floating point instructions, CPA (copy and add logical) instruction, 1 drum unit, 4 tape units, attached card punch, attached or peripheral card reader, and attached or peripheral printer.

The object machine may be of any size. Among the information produced by FORTRAN is a count of the storage locations required by the object program, from which it can be easily decided whether the object program is too large for any given object machine.

Arrangement of the Object Program

The instructions and constants of the object program begin in lower memory and extend upwards. Data and other storage locations required for the operation of the program begin at location 77777_8 and extend downwards. Thus these latter locations are always at the top of memory, regardless of the size of the object machine.

The topmost section of data storage is occupied by those variables which appear in DIMENSION or EQUIVALENCE statements. The arrangement of this region is such that two programs, whose DIMENSION and EQUIVALENCE statements are identical, will have this region allocated identically. This fact makes it possible to write families of programs which deal with the same data.

The successively lower sections of storage are occupied by variables not mentioned in DIMENSION or EQUIVALENCE statements, then certain storage locations required for the operation of the program, and finally a section of erasable storage.

For each object program FORTRAN produces a printed description of the exact arrangement of storage locations.

Fixed Point Arithmetic

The use of fixed point arithmetic is governed by the following considerations.

1. Fixed point constants specified in the source program must have magnitudes less than 2^{15}.
2. Fixed point data read in by the object program ~~are treated mod 2^{15}~~. must be smaller than 2^{15}
3. Fixed point arithmetic in the object program is arithmetic mod 2^{15}.
4. Indexing in the object program is mod (size of the object machine).

Writing Subroutines for the Master Tape

Library subroutines exist on the master FORTRAN tape in relocatable binary form. Placing a new subroutine on that tape involves (1) producing the routine in the form of relocatable binary cards, and (2) transcribing those cards on to the master tape by means of a program furnished for that purpose. Further details will be found in the FORTRAN Operator's Manual.

In the object program transfer to the subroutine is by the sequence

```
TSX    Subroutine, 4
Return
```

The subroutine itself and any constants that it requires should be located in relocatable locations 0, 1, 2, ... It may also make use of a *common storage region* of any desired length n, beginning with relocatable location $77777_8-(n-1)$ and ending with relocatable location 77777_8.

At the moment of transfer to the subroutine Arg_1 will have been placed in the AC, Arg_2 (if it exists) in the MQ, Arg_3 (if it exists) in relocatable location 77775_8 of the common storage region, Arg_4 (if it exists) in relocatable location 77774_8, etc. The common storage region may also be used for erasable storage by the subroutine.

The output of the subroutine is to be left in the AC, and index registers 1 and 2 must be returned with their original contents.

Fixed point quantities in the object program exist in the following format: sign in sign bit, magnitude in decrement field, remainder of word all zeroes.

It is suggested that error HALTS in subroutines be coded as HPR instructions, permitting the tag and address fields to contain identifying numbers which can be recognised at the console.

Optimisation of Arithmetic Expressions

Considerable attention is paid in FORTRAN to ensure that the object program arising from an arithmetic expression shall be efficient, regardless of how the expression has been written. For example, a sequence of multiplications and divisions not grouped by parentheses will automatically be reordered if necessary to minimise the number of storage accesses in the object program.

However, one important type of optimisation, concerned with *common subexpressions,* takes place only if the expression has been suitably written. As an example, consider the arithmetic formula.

```
Y = A*B*C+SINF(A*B)
```

An efficient object program would form the product A*B only once; yet if the arithmetic formula is written as above, the multiplication of A by B will occur twice. The correct way to write this arithmetic formula is

```
Y = (A*B)*C+SINF(A*B)
```

The common subexpression A*B has been displayed by the extra pair of parentheses, and an object program will be formed which multiplies A by B only once.

In general, when common subexpressions exist in an expression, parentheses should be used to display them.

There is one case where the programmer need not write the parentheses, because FORTRAN will understand that they are there. The parentheses discussed in the section Hierarchy of Operations in Chapter 3 are of this type, and need not be given. Thus

```
Y = A*B+C+SINF(A*B)
```

is as suitable for optimisation as

```
Y = (A*B)+C+SINF(A*B)
```

However, the parentheses discussed in the section Ordering within a Hierarchy in Chapter 3 must be supplied if common subexpression optimisation is to occur.

Further Details about DO Statements

This section contains further details about DOs, which may be of interest to the advanced programmer.

Triangular Indexing. Indexing such as

```
DO   I = 1,10
DO   J = I,10
```

or

```
DO   I = 1,10
DO   J = 1,I
```

is permitted and simplifies work with triangular arrays. These are simply special cases of the fact that an index under control of a DO is available for general use as a fixed point variable.

The diagonal elements of an array may be picked out by the following type of indexing:

```
DO   I = 1,10
A(I,I,I) = some expression
```

Status of the Cell Containing I. A DO loop with index I does not affect the contents of the object program storage location for I except under certain circumstances, namely if

1. An IF-type or GO TO-type transfer exit occurs from the range of the DO;
2. I is used as a variable in the range of the DO; or
3. I is used as a subscript in combination with a *relative constant* whose value changes within the range of the DO. (A relative constant is a subscript the fixed point variable of which is not currently under control of a DO.)

Therefore, if a normal exit occurs from a DO to which cases 2 and 3 do not apply, the I cell contains what it did before the DO was encountered. After normal exit where 2 or 3 do apply, the I cell contains the first value of the I-sequence which exceeds m_2. After a transfer exit the I cell contains the current value of I.

Lists for Binary Tape Operations

There are restrictions on the complexity of the parenthesised parts of lists for the READ TAPE and WRITE TAPE statements which must be observed if the object program indexing is not to exceed 288 microseconds, the maximum safe time for calculation between CPY instructions on the 704.

Unfortunately this matter is exceedingly complicated and therefore no complete discussion will be given. Instead, certain rules will be given which will permit the construction of lists which can be relied upon to work; there will be other lists, excluded by these rules, which would also work.

Define the term *subscript combination* as follows. Consider each subscript of each subscripted variable to be specified by the four symbols, c, v, c′ and d, where d is the maximum value of the subscript as given in the DIMENSION statement about the variable, and where $c*v \pm c'$ is the full form of the subscript. (The full form of the subscript I would be $1*I+0$; that of the subscript 3 would be $1*0+3$, etc.). Then the subscript combination of a subscripted variable is one of the ordered sets

$$c_1, v_1$$
$$c_1, v_1, c_2, v_2, d_1$$
$$c_1, v_1, c_2, v_2, c_3, v_3, d_1, d_2$$

depending upon whether the variable has 1, 2, or 3 subscripts. Thus A(I,J,K), B(I,K,J), C(5*I,J,K), and D(I,J) all have different subscript combinations, but A(I,J,K) and B(I+5,J,K) have the same subscript combination if the first two dimensions of A and B are the same.

Also, define as an *element* of a list a part bounded by principal commas in the list.

Elements which Contain no Subscripted Variable. There are no restrictions on such elements.

Elements which Contain no Variable with Three Subscripts. Let

N_1 = number of different 1-dimensional subscript combinations
N_2 = " " " 2- " " "
$M_1 = 0$ if $N_1 = 0$; otherwise $M_1 = 1$
$M_2 = 0$ if $N_2 = 0$; otherwise $M_2 = 5$

Then the restriction is that $4N_1+6N_2+\max(M_1,M_2)$ must not exceed 18.

For example, the element ((A(I,J), B(I,J), J = 1,10), C(I), I = 1,10) has $N_1 = 1$, $N_2 = 1$ (provided that the first dimensions of A and B are the same); hence $4N_1+6N_2+\max(M_1,M_2) = 15$, and the element is permissible.

Elements which Contain Variables with Three Subscripts. Consider a variable A whose three subscripts s_1, s_2, s_3 involve as variables v_1, v_2, v_3. Then the skeleton elements

$$(((\quad, v_3 = \quad), v_2 = \quad), \quad , v_1 = \quad)$$

or

$$(((\quad, v_3 = \quad), v_1 = \quad), \quad , v_2 = \quad)$$

or

$$(((\quad, v_2 = \quad), v_3 = \quad), \quad , v_1 = \quad)$$

will work. The innermost parentheses (controlling v_3, v_3, v_2 respectively) may contain $A(s_1,s_2,s_3)$ and any number of other variables with the same subscript combination; similarly the outermost parentheses may contain variables of any one subscript combination. The middle parentheses may not contain any subscripted variables. Thus for example

```
((A(I,J,K), B(I,J,K), K = 1,10), J = 1,5), C(I), D(I), I = 1,10)
```

will work provided that the first two dimensions of A and B are the same. This example makes use of the first of the skeleton elements just given.

In the special case where the next element in the list is neither subscripted nor enclosed in controlling parentheses, the first of the above skeleton elements may also contain variables of any one subscript combination in its middle parentheses.

Finally, the skeleton element

$$(((\quad, v_1 = 1,d_1), v_2 = 1,d_2), v_3 = \quad)$$

which unlike the others indexes in the natural order, will work. Notice, however, that the indexing parameters for v_1 and v_2 must be such that the array is swept through consecutively. The innermost parameters may contain $A(s_1,s_2,s_3)$ and any number of other variables with the same subscript combination.

Variable Indexing Parameters and Relative Constants. Another restriction affecting lists for binary tape operations concerns the use of relative constants and of indexing parameters which are variables. (A relative constant is a subscript, the variable of which is not currently under the control of a DO or a controlling parenthesis.)

The restriction is, that variables which have a subscript involving either a relative constant or an index governed by variable indexing parameters may appear only in the *first* element of a list. Furthermore, all such variables must also have among their subscripts one whose index is controlled by the first parenthesis.

For example, in ((A(I,J), I = 1,L), B(K,J), J = 1,M) both the subscripts of A are governed by variable indexing parameters; for B the same is true of one of its subscripts, while the other is a relative constant (unless the READ TAPE or WRITE

TAPE is itself in the range of a DO for K). However, since J, which is the index controlled by the first parenthesis, is a subscript of both A and B, this element will work if it is the first element in the list.

Limits on the Size of the Source Program

In performing the translation from source to object program, FORTRAN forms and uses tables which summarise various aspects of the information contained in the source program. These tables are limited in size, with corresponding limitations in the amount of information which the source program may contain. If a table size is exceeded the FORTRAN program will HALT at one of a list of locations given in the FORTRAN Operator's Manual.

In what follows, the phrase "literal appearance" means that if the same thing appears more than once it must be counted more than once.

1. (TEIFNO Table). The number of FORTRAN statements which have statement numbers must not exceed 1500. (An input or output statement which has a statement number and whose list contains controlling parentheses counts double.)
2. (FIXCON Table). The number of different fixed point constants must not exceed 100. (In this count constants which differ only in sign are not considered different.)
3. (FLOCON Table). The number of different floating point constants must not exceed 450. (In this count constants which differ only in sign are not considered different, nor are numbers such as 4., 4.0, 40.E-1, etc. which are really the same number.)
4. (TDO Table). The number of DO statements must not exceed 150.
5. The number of DO statements in any one nest must not exceed 50.
6. (TIFGO Table). The total number of ASSIGNS plus IF-type and GO TO-type statements must not exceed 300.
7. (TRAD Table). The total number of statement numbers mentioned in assigned GO TO and computed GO TO statements must not exceed 250.
8. (FRET Table). The total number of numbers mentioned in FREQUENCY statements must not exceed 750. (Such a statement as FREQUENCY 30(1,2,1) has 4 numbers.)
9. (DIM Tables). The number of 1, 2, and 3-dimensional variables which appear in DIMENSION statements must not exceed 100, 100, and 90 respectively.
10. (EQUIT Table). The number of literal appearances of variables in EQUIVALENCE statements must not exceed 750.
11. (LAMBDA Table). This table, and the BETA Table which follows, limit the size of arithmetic expressions on the right-hand side of arithmetic formulas and as the arguments of IF statements. In any one expression let

n = number of literal appearances of variables and constants, except those in subscripts;

b = number of open parentheses, except those introducing subscripts;

p = number of appearances of + or –, except in subscripts or as unitary operators. (The + in A*(+B) is a unitary operator.);

t = number of appearances of * or /, except in subscripts;

e = number of appearances of **:

f = number of literal appearances of function names; and

a = number of arguments of functions. (For SINF(SINF(X)), a = 2.)

Then λ, which equals $n+4b+4a-3f+3p+2t+e+3$, must not exceed 400.

12. (BETA Table). With the same definitions, $\beta = \lambda + 1 - n - f$ must not exceed 300.

13. (CLOSUB Table). In the entire program the number of literal appearances of functions must not exceed 1500.

14. (FORVAL Table). The number of arithmetic formulas whose left-hand sides are non-subscripted fixed point variables must not exceed 500.

15. (FORVAR Table). The total number of literal appearances of non-subscripted fixed point variables on the right-hand side of arithmetic formulas and in the arguments of IFs must not exceed 750.

16. (FORTAG Table). The total number of literal appearances of subscripted variables must not exceed 1500.

17. (TAU Tables). The total number of different 1, 2, and 3-dimensional subscript combinations must not exceed 100, 90, and 75 respectively. (See the preceding section on Lists for Binary Tape Operations for the definition of subscript combination.)

18. (SIGMA Tables). Consider a variable with 3 subscripts, and let the additive parts of these subscripts (when written in full form) be c_1', c_2', c_3'. Then the number of distinct triples c_1', c_2', c_3' must not exceed 100. Similarly for 2- and 1-dimensional variables, neither the number of distinct c_1', c_2' nor of distinct c_1' may exceed 100.

19. There are a few more limitations on table sizes, which are too complicated to set forth. In every case, however, a violation causes an identifiable HALT in the FORTRAN program.

CHAPTER 8. EXAMPLES OF FORTRAN PROGRAMMING

A Complete but Simple Program

The example below is the same as that given in Chapter 1, but expanded into a complete program. The purpose of the program is to discover the largest value attained by a set of numbers A(I) and to print that number on the attached printer. The numbers A(I) exist on punched cards, 12 to a card, each number occupying a field of 6 columns. There are not more than 999 numbers; the actual number N is punched on a lead card.

C ← FOR COMMENT	STATEMENT NUMBER 1–5	CONTINUATION 6	FORTRAN STATEMENT 7–72	IDENTIFICATION 73–80
C			PROGRAM FOR FINDING THE LARGEST VALUE	
C		X	ATTAINED BY A SET OF NUMBERS	
			DIMENSION A(999)	
			FREQUENCY 30(2,1,10), 5(100)	
			READ 1, N, (A(I), I = 1,N)	
	1		FORMAT (I3/(12F6.2))	
			BIGA = A(1)	
	5		DO 20 I = 2,N	
	30		IF (BIGA-A(I)) 10,20,20	
	10		BIGA = A(I)	
	20		CONTINUE	
			PRINT 2, N, BIGA	
	2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)	
			STOP 77777	

The first executable statement is the READ; therefore the program begins there. The READ causes first N and then A(1), A(2), ..., A(N) to be brought in from the card reader, in accordance with the FORMAT statement 1. Notice that the indexing of the loop bringing in A will work correctly, since the indexing parameter N occurs *earlier* in the list.

The FORMAT statement says that there is first a single card with format I3, followed by any number of cards with format 12F6.2. On the single card N is punched as a decimal integer in columns 1-3 and is also to be converted into a fixed point number. In the remainder of the deck the numbers A(I) are punched 12 to a card in columns 1-6, 7-12, etc. Each number is presumably punched as xxxxx or –xxxxx with the decimal point understood to precede the last two digits. If, however, a column is used for a decimal point, its position overrides this understanding.

After the READ is executed control moves to the next executable statement, the arithmetic formula BIGA = A(1). The cell BIGA now contains A(1).

Next the DO statement sets I to 2 and creates a loop starting with the IF and ending with the CONTINUE. The first time the IF is executed it transfers control either to statement 10 or statement 20, according as $A(2) > BIGA$ or $A(2) \leqslant BIGA$. In the first case, therefore, BIGA becomes A(2); otherwise it remains A(1).

Control is now in the CONTINUE, which is the last statement in the range of a DO which is still unsatisfied (provided that $N > 2$), and which is not a transfer. Therefore DO-sequencing occurs: I is increased to 3 and control goes back to the IF. The comparison now is between BIGA and A(3); if A(3) is the larger it becomes the new BIGA.

This process continues until control is in the CONTINUE with $I = N$. The DO is satisfied; therefore normal sequencing occurs and control moves to the PRINT. BIGA now contains the largest value in the set of A's; N and the A-array have not been altered.

The PRINT statement causes N and BIGA to be printed on the attached printer in accordance with the FORMAT statement 2. This statement causes a line to be printed as follows. The carriage control character is a Hollerith "1" which causes a skip to channel 1, bringing the paper to the top of a new sheet. Type wheels 1-21 receive the text 'THE LARGEST OF THESE '; wheels 22-24 receive N converted back to decimal integer form; wheels 25-36 receive ' NUMBERS IS '; and wheels 37-43 receive BIGA converted into fixed decimal form, with a decimal point preceding the last two digits. Notice that an extra column is allotted to BIGA to allow for the decimal point, which was presumably omitted in the data input.

The program ends with a HALT and 77777_8 in the address field of the storage register on the console. Pressing the START button will have no effect.

The only subscripted variable used in the program is A. Hence DIMENSION A(999) is the only dimension information required.

The only statements about which a FREQUENCY statement can be made are the IF and the DO. The programmer anticipates that BIGA will usually be greater than A(I), a plausible guess if the A(I) are randomly arranged but not if they tend to increase with increasing I. He also predicts for the DO that 100 is a reasonable average for N.

A DO Nest with Exit and Return

Given an N x N square matrix A, to find those off-diagonal elements which are symmetric and to write them on binary tape.

```
C FOR COMMENT
STATEMENT  CONT-                        FORTRAN STATEMENT                        IDENTI-
 NUMBER    INUATION                                                              FICATION
1      5   6      7                                                          72  73     80

                  REWIND 3
                  DO 3 I = 1,N
                  DO 3 J = 1,N
                  IF(A(I,J)-A(J,I)) 3,20,3
    3             CONTINUE
                  END FILE 3
                  MORE PROGRAM

   20             IF(I-J) 21,3,21
   21             WRITE TAPE 3,I,J, A(I,J)
                  GO TO 3
```

After rewinding tape 3, a nested pair of DO loops scans the entire matrix for elements A(I,J) equal to A(J,I). Whenever such an element is found an exit completely out of the nest is made to a routine which for off-diagonal elements only writes a 3-word record (I, J, and A(I,J)) in binary on tape 3. Both for on- and off-diagonal elements this routine makes no change in the indexes or indexing parameters of the nest, and so it is permissible to re-enter the nest and continue the scan.

This program actually finds each element twice. This could be avoided by writing the second DO as DO 3 J = I,N.

APPENDIX A. TABLE OF FORTRAN CHARACTERS

	CARD	BCD TAPE	704		CARD	BCD TAPE	704		CARD	BCD TAPE	704		CARD	BCD TAPE	704
1	1	01	01	A	12 1	61	21	J	11 1	41	41	/	0 1	21	61
2	2	02	02	B	12 2	62	22	K	11 2	42	42	S	0 2	22	62
3	3	03	03	C	12 3	63	23	L	11 3	43	43	T	0 3	23	63
4	4	04	04	D	12 4	64	24	M	11 4	44	44	U	0 4	24	64
5	5	05	05	E	12 5	65	25	N	11 5	45	45	V	0 5	25	65
6	6	06	06	F	12 6	66	26	O	11 6	46	46	W	0 6	26	66
7	7	07	07	G	12 7	67	27	P	11 7	47	47	X	0 7	27	67
8	8	10	10	H	12 8	70	30	Q	11 8	50	50	Y	0 8	30	70
9	9	11	11	I	12 9	71	31	R	11 9	51	51	Z	0 9	31	71
Blank		20	60	+	12	60	20	—	11	40	40	0	0	12	00
=	8-3	13	13	.	12 8-3	73	33	$	11 8-3	53	53	,	0 8-3	33	73
—	8-4	14	14	)	12 8-4	74	34	*	11 8-4	54	54	(	0 8-4	34	74

NOTE 1. There are two — signs. Only the 11-punch minus may be used in source program cards. Either minus may be used in input data to the object program; object program output has the 8-4 minus.

NOTE 2. The $ character can be used in FORTRAN only as Hollerith text in a FORMAT statement.

APPENDIX B. TABLE OF FORTRAN STATEMENTS

STATEMENT	NORMAL SEQUENCING
a = b	Next executable statement
GO TO n	Statement n
GO TO n, $(n_1,n_2,\ldots,n_m)$	Statement last assigned
ASSIGN i TO n	Next executable statement
GO TO $(n_1,n_2,\ldots,n_m)$, i	Statement n_i
IF (a) n_1,n_2,n_3	Statement n_1,n_2,n_3 as a less than, =, or greater than 0
SENSE LIGHT i	Next executable statement
IF (SENSE LIGHT i) n_1,n_2	Statement n_1,n_2 as Sense Light i ON or OFF
IF (SENSE SWITCH i) n_1,n_2	" " " as Sense Switch i DOWN or UP
IF ACCUMULATOR OVERFLOW n_1,n_2	" " " as Accumulator Overflow trigger ON or OFF
IF QUOTIENT OVERFLOW n_1,n_2	" " " as MQ Overflow trigger ON or OFF
IF DIVIDE CHECK n_1,n_2	" " " as Divide Check trigger ON or OFF
PAUSE or PAUSE n	Next executable statement
STOP or STOP n	Terminates program
DO n i = m_1,m_2 or DO n i = m_1,m_2,m_3	Next executable statement
CONTINUE	" " "
FORMAT (Specification)	Not executed
READ n, List	Next executable statement
READ INPUT TAPE i, n, List	" " "
PUNCH n, List	" " "
PRINT n, List	" " "
WRITE OUTPUT TAPE i, n, List	" " "
READ TAPE i, List	" " "
READ DRUM i, j, List	" " "
WRITE TAPE i, List	" " "
WRITE DRUM i, j, List	" " "
END FILE i	" " "
REWIND i	" " "
BACKSPACE i	" " "
DIMENSION v, v, v, ...	Not executed
EQUIVALENCE (a,b,c,..), (d,e,f,..), ...	" "
FREQUENCY n(i,j,..), m(k,l,..), ...	" "

INDEX

IBM®

INTERNATIONAL BUSINESS MACHINES CORPORATION, 590 MADISON AVE., NEW YORK 22, NEW YORK