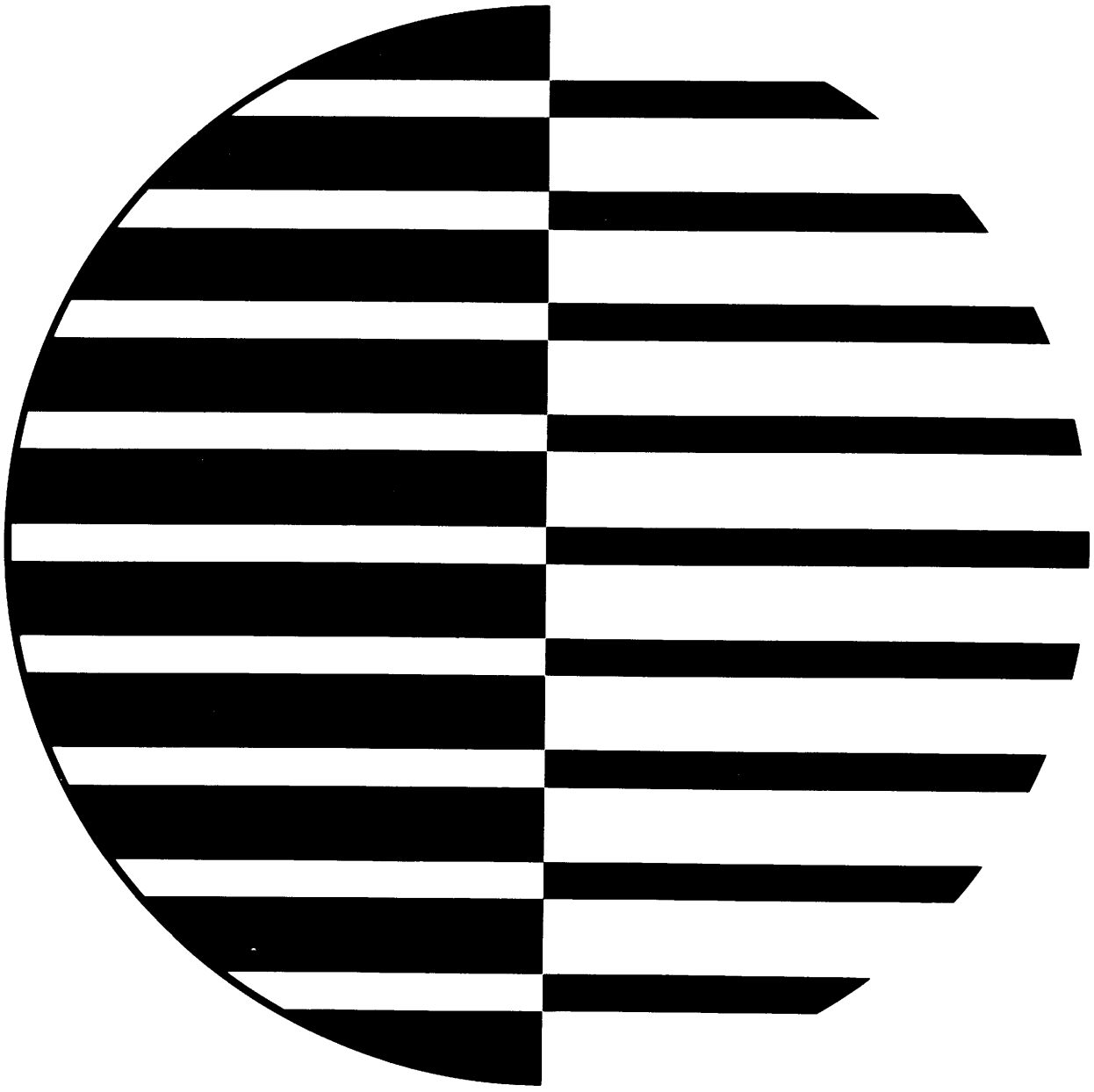
CONTROL DATA® 6000 SERIES COMPUTER SYSTEMS

Chippewa Operating System FORTRAN Reference Manual



Additional copies of this manual may be obtained from the nearest Control Data Corporation Sales office listed on the back cover.

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

May, 1966
Pub. No. 60132700, Rev. A

PREFACE

This document describes the FORTRAN† language and compiler for the CONTROL DATA 6000 Series Chippewa Operating System. The statement language is compatible with FORTRAN II and FORTRAN IV; but new programs should be written in FORTRAN IV.

The compiler can operate as load-and-go and produce 6000 Series machine language output. It operates as an independent program under control of the operating system and can be called to use only the storage required for compilation of a particular program. Several compilations may be processed simultaneously using the normal Chippewa Operating System multiprogramming features.

FORTRAN accepts main programs and subprograms written either in FORTRAN source language or 6000 Series assembly language as well as binary decks of previously compiled subprograms. These features permit a flexible program arrangement for each particular job.

This document assumes a knowledge of the FORTRAN language and the CONTROL DATA 6000 Series Computer System.

†FORmula TRANslation originally developed for International Business Machines equipment.

CONTENTS

CODING PROCEDURES 1

1.1 CODING LINE

A FORTRAN coding line contains 80 columns in which FORTRAN characters are written one per column. The four types of coding lines are listed below:

	Column		Content
Statement	1-5	or	statement number
	1		D,I,B,F, - FORTRAN II
	6		blank or zero
	7-72		FORTRAN statement
	73-80		identification field
Continuation	1-5		blank
	6		FORTRAN character other than blank or zero
	7-72		continued FORTRAN statement
	73-80		identification field
Comment	1		C . $ or *
	2-80		comments
Data	1-80		data

1.1.1 STATEMENT

Statement information is written in columns 7 through 72. Statements longer than 66 columns may be continued to the next line. Blanks are ignored by the FORTRAN compiler except in H fields. The character $ may be used to separate statements when more than one is written on a coding line, however, it may not be used with FORMAT or DATA statements. A blank card may be used to separate the statements.

1.1.2 CONTINUATION

The first line of every statement must have a blank or zero in column 6. If statements occupy more than one line, all subsequent lines must have a FORTRAN character other than blank or zero in column 6. Continuation cards may be separated by cards whose first 72 columns are blank. A statement may have up to 19 continuation lines.

1.1.3 STATEMENT NUMBER

Any statement may have an identifier, statement number, but only statements referred to elsewhere in the program require identifiers. A statement number is a string of 1 to 5 digits occupying any column positions 1 through 5.

1.1.4 IDENTIFICATION FIELD

Columns 73 through 80 are always ignored in the compilation process. They may be used for identification when the program is to be punched on cards. Usually these columns contain sequencing information provided by the programmer.

1.1.5 COMMENTS

Each line of comment information is designated by a C, *, . , or $ in column 1. Comment information appears in the source program and the source program listing, but it is not translated into object code. The continuation character in column 6 is not applicable to comments cards.

1.2 PUNCHED CARDS

Each line of the coding form corresponds to one 80-column card; the terms "line" and "card" are often used interchangeably. Source programs and data can be read into the computer from cards; a relocatable binary deck or data can be punched directly onto cards.

When cards are being used for data input, all 80 columns may be used.

ELEMENTS OF CHIPPEWA FORTRAN 2

2.1 FORTRAN CHARACTER SET

Alphabetic:	A to Z			
Numeric:	0 to 9			
Special:	=	equals	)	right parenthesis
	+	plus	,	comma
	-	minus	.	decimal point
	*	asterisk	$	dollar sign
	/	slash	(space)	blank
	(	left parenthesis		

All characters appear internally in 6000 Series display code (Appendix A). A blank is ignored by the compiler except in Hollerith fields within DATA statements; otherwise it may be used freely to improve program readability.

2.2 IDENTIFIERS

2.2.1 ALPHANUMERIC IDENTIFIER

An alphanumeric identifier can be any combination of 1-7 characters beginning with a letter, with one exception. The combination of the letter O and 6 digits is recognized as an octal constant. Embedded blanks within an identifier are ignored.

Example:

O123456	illegal (as an identifier)
O12KK3	Legal
O123	Legal

Alphanumeric Identifiers are used for:

- Constants
- Variables
- Subprograms
- Main programs
- Input/output units
- Labeled common blocks
- Name list names

2.2.2 STATEMENT IDENTIFIER

Statements are identified by unsigned numbers, 1-5 digits, which can be referred to from other sections of the program. A statement identifier (from 1-99999) may be placed anywhere in columns 1-5 of the initial line of a statement. Leading zeros are ignored. In any given program or subprogram, each statement identifier must be unique.

2.3 CONSTANTS

Seven types of constants are used in FORTRAN: integer, octal, real, double precision, complex, Hollerith, and logical. Complex and double precision constants are formed from real constants. The type of a constant is determined by its form. The computer word structure for each type is listed in Appendix E.

2.3.1 INTEGER CONSTANTS

An integer constant, N, is a string of up to 18 decimal digits in the range $-(2^{59}-1) \leq N \leq (2^{59}-1)$.

Examples:

63	3647631
247	464646464
314159265	574396517802457165

During execution, the maximum allowable value is $2^{48}-1$ when an integer constant is converted to real. The maximum value of the result of integer multiplication or division must be less than $2^{48}-1$. High order bits will be lost if the value is larger; but, no diagnostic is provided.

2.3.2 OCTAL CONSTANTS

An octal constant consists of 6 to 20 octal digits preceded by the letter O or 1 to 20 octal digits suffixed with a B. The form is:

$On_1 . . . n_i$

$n_1 . . . n_i B$

A constant of the second form is assigned logical mode and may be used only in arithmetic or DATA statements.

If the constant exceeds 20 digits; or if a nonoctal digit appears, a compiler diagnostic is provided.

Examples:

O00007777777700000000	2374216B
O7777700077777	777776B
O2323232323232323	777000777000777B
O000077	
O7777777777777700	

2.3.3 REAL CONSTANTS

A real constant is represented by a string of up to 15 digits; it contains a decimal point and may contain an exponent representing a power of 10. Real constants may be in the following forms:

n.n n. .n n.nE±s n.E±s .nE±s

The base is n; s is the exponent to the base 10; the plus sign may be omitted if s is positive, the range of s is 0 through 308. If the range of the real constant is exceeded, a compiler diagnostic is provided.

All real numbers are carried in normalized form.

Examples:

3.E1 (means 3.0×10^1;i.e., 30.)

3.1415768	31.41592E-01
314.0749162	.31415E01
-3.141592E+279	.31415E+01

2.3.4 DOUBLE PRECISION CONSTANTS

A double precision constant is represented by a string of up to 18 digits. Double precision constants are represented internally by two words; the second is always zero. The forms are similar to real constants, the base is n; s is the exponent to the base 10.

.nD±s n.nD±s n.D±s

The D must always appear. The plus sign may be omitted for positive s; the range of s is 0 through 308. If the range is exceeded, a compiler diagnostic is provided.

Examples:

3.1415927D	3141.593D3
3.1416D0	31416.D-04
3141.593D-03	

2.3.5 COMPLEX CONSTANTS

A complex constant is represented by a pair of real constants separated by a comma and enclosed in parentheses (r_1,r_2); r_1 represents the real part of the complex number, r_2 the imaginary part. Either constant may be preceded by a minus sign.

If the range of the real numbers comprising the constant is exceeded, a compiler diagnostic is provided. Diagnostics also occur when the pair consists of integer constants, including (0,0).

Examples:

FORTRAN Representation	Complex Number
(1. , 6.55)	1. + 6.55i
(15. , 16.7)	15. + 16.7i
(-14.09, 1.654E-04)	-14.09 + .0001654i
(0. , -1.)	0 - 1.0i

2.3.6 HOLLERITH CONSTANTS

A Hollerith constant is a string of FORTRAN characters of the form hHf; h is an unsigned decimal integer between 1 and n representing the length of the field f. n is limited to the number of characters that can be contained in up to 19 continuation lines. Spaces are significant in the field f. When h is not a

multiple of 10, the last computer word is left justified with blank fill. Alternate forms are nLf (left justified) and nRf (right justified) Hollerith constants with zero fill for incomplete words. They may be used in arithmetic and DATA statements. Hollerith constants are stored internally in 6000 Series console display code.

Examples:

6HCOGITO	12HCONTROL DATA
4HERGO	5LSUMbb = SUMbb00000
3HSUM	1H)
5RSUMbb = 00000SUMbb	3LbTT = bTT0000000

A statement of the form: T=(+5HABCDE) is permitted as a Hollerith constant.

2.3.7 LOGICAL CONSTANTS

Logical constants may be in the forms:

.TRUE. or .T.

.FALSE. or .F.

A false constant is stored internally as binary zero. A true constant is stored internally as the one's complement of binary zero (all bits).

2.4 VARIABLES

FORTRAN recognizes simple and subscripted variables; a simple variable represents a single quantity; it references a storage location. The value specified by the name is always the current value stored in the location. Variables are identified by a symbolic name of 1-7 alphanumeric characters, the first of which must be alphabetic.

The type of variable is defined in one of two ways:

EXPLICIT. Variables may be declared a particular type with the FORTRAN TYPE declarations.

IMPLICIT. A variable not defined in a FORTRAN TYPE declaration is assumed to be integer if the first character of the symbolic name is I, J, K, L, M, or N.

Example:

I15, JK26, KKK, NP362L, M

All other variables not declared in a FORTRAN TYPE declaration are assumed to be real.

Examples:

TEMP, ROBIN, A55, R3P281

2.4.1 INTEGER VARIABLES

Integer variables can be defined explicitly or implicitly; values may be in the range $-(2^{59}-1) \leq I \leq (2^{59}-1)$. Each integer variable occupies one word in storage.

Examples:

N	NEGATE
ITEM	K2S04
M58A	M58

2.4.2 REAL VARIABLES

Real variables can be defined explicitly or implicitly; they may be values in the range $10^{-308} \leq r \leq 10^{308}$ with 15 significant digits. Each real variable is stored in 6000 Series floating-point format and occupies one word.

Examples:

VECTOR	A62597	X
YBAR	BARMIN	X74A

The variable, r, may have any of the following values:

$-10^{+308} \leq r \leq -10^{-308}$, $r = 0$, $10^{-308} \leq r \leq 10^{308}$.

2.4.3 DOUBLE PRECISION VARIABLES

Double precision variables must be defined explicitly by a TYPE declaration. Each double precision variable occupies two words of storage and can assume values in the range $10^{-308} \leq d \leq 10^{308}$ with 15 significant digits.

2.4.4 COMPLEX VARIABLES

Complex variables must be explicitly defined by a TYPE declaration. A complex variable occupies two words in storage. Each word contains a number in real variable format. This ordered pair of real variables (C_1,C_2) represents the complex number: $C_1 + i\ C_2$

2.4.5 LOGICAL VARIABLES

Logical variables must be defined explicitly by a TYPE declaration. Each logical variable occupies one word of storage; it can assume the value true or false. A logical variable with a plus zero value is false; any other value true. When logical variable appears in an expression whose dominant mode is real, double, or complex, it is not packed and normalized prior to its use in the evaluation of an expression (as is the case with an integer variable).

2.5 SUBSCRIPTED VARIABLE

A subscripted variable may have one, two, or three subscripts enclosed in parentheses. More than three subscripts produce a compiler diagnostic. The subscripts can be expressions in which operands are simple integer variables and integer constants and operators are addition, subtraction, multiplication, and division only. A subscripted variable represents a single quantity within an array of quantities.

When a subscripted variable represents the entire array, the subscripts are the dimensions of the array. When a subscripted variable references a single element in an array, the subscripts describe the relative location of the element in the array.

Simple Variable	Subscripted Variable
FRAN	A(I,J)
P	B(I+2,J+3,2*K+1)
Z14	Q(14)
EVAL	STRING (3*K*ILIM+3)
MAX3	Q(1,4,2)
I	

2.6 ARRAYS

An array is a block of successive storage locations which is divided into areas for storage of variables. The entire array may be referenced by the array name without subscripts (I/O lists and Implied DO-loop notation). Arrays may have one, two, or three dimensions; the array name and dimensions must be declared in a DIMENSION, COMMON, or TYPE declaration prior to the first program reference to that array.

Each element in an array may be referenced by the array name plus a subscript notation. Program execution errors may result if subscripts are larger than the dimensions initially declared for the array. The maximum number of elements in an array is the product of the dimensions.

2.6.1 ARRAY STRUCTURE

Array elements are stored by columns in ascending locations. In the array declared as A(3,3,3):

A_{111}	A_{121}	A_{131}
A_{211}	A_{221}	A_{231}
A_{311}	A_{321}	A_{331}

A_{112}	A_{122}	A_{132}
A_{212}	A_{222}	A_{232}
A_{312}	A_{322}	A_{332}

A_{113}	A_{123}	A_{133}
A_{213}	A_{223}	A_{233}
A_{313}	A_{323}	A_{333}

The planes are stored in order, starting with the first, as follows:

$A_{111} \rightarrow L$ $\quad$ $A_{121} \rightarrow L+3 \ldots A_{133} \rightarrow L+24$

$A_{211} \rightarrow L+1$ $\quad$ $A_{221} \rightarrow L+4 \ldots A_{233} \rightarrow L+25$

$A_{311} \rightarrow L+2$ $\quad$ $A_{321} \rightarrow L+5 \ldots A_{333} \rightarrow L+26$

Array allocation is discussed under DIMENSION declaration. The location of an array element with respect to the first element is a function of the maximum array dimensions and the type of the array.

Given DIMENSION A(L,M,N), the location of A(i,j,k), with respect to the first element A of the array, is given by A+(i-1+L*(j-1 +M *(k-1)))*E.

The quantity enclosed by the outer parentheses is the subscript expression. E is the element length—the number of storage words required for each element of the array. For real, logical, and integer arrays, E = 1. For complex and double precision arrays, E = 2.

Example:

In an array defined by DIMENSION A(3,3,3), the location of A(2,2,3) with respect to A(1,1,1) is:

$$\text{Locn } A(2,2,3) = \text{Locn } A(1,1,1) + (2-1+3(1+3(2)))$$
$$= L + 22$$

FORTRAN permits the following relaxation of the representation of subscripted variables:

Given	$A(D_1,D_2,D_3)$, where the D_i are integer constants,	
then	A(I,J,K)	implies A(I,J,K)
	A(I,J)	implies A(I,J,1)
	A(I)	implies A(I,1,1)
	A	implies A(1,1,1)
similarly,	for $A(D_1,D_2)$	
	A(I,J)	implies A(I,J)
	A(I)	implies A(I,1)
	A	implies A(1,1)
and for	$A(D_1)$	
	A(I)	implies A(I)
	A	implies A(1)

The elements of a single-dimension array $A(D_1)$ may not be referred to as A(I,J,K) or A(I, J). Diagnostics occur if this is attempted.

EXPRESSIONS 3

An expression is a constant, variable (simple or subscripted), function, or any combination of these separated by operators and parentheses. The four kinds of expressions in FORTRAN are: arithmetic and masking (Boolean) expressions which have numerical values, and logical and relational expressions which have truth values. Each type of expression is associated with a group of operators and operands.

3.1 ARITHMETIC EXPRESSIONS

An arithmetic expression can contain the following operators:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Operands are:

Constants

Variables (simple or subscripted)

Evaluated functions

Any unsigned constant, variable, or function is an arithmetic expression. If X is an expression, then (X) is an expression. If X and Y are expressions, then the following are expressions:

X + Y	X - Y
X * Y	X / Y
-X	X**Y

If op is valid operator and X and Y are valid expressions, then X op op Y is never a valid expression.

Examples:

```
A
3.14159
B + 16.427
(XBAR+(B(I,J+I,K)/3) )
-(C+DELTA*AERO)
(B-SQRT(B**2-(4*A*C) ) )/(2.0*A)
GROSS-(TAX*0.04)
(TEMP+V(M,MAXF(A,B) )*Y**C)/(H-FACT(K+3) )
```

3.1.1 ARITHMETIC EVALUATION

The hierarchy of arithmetic evaluation is:

**	exponentiation	class 1
/	division	class 2
*	multiplication	
+	addition	class 3
-	subtraction	

In an expression with no parentheses or within a pair of parentheses in which unlike classes of operators appear, evaluation proceeds in the above order. In expressions containing like classes of operators, evaluation proceeds from left to right. For example, A**B**C is evaluated as (A**B)**C.

Parenthetical and function expressions are evaluated first in a right-to-left scan of the entire statement. In parenthetical expressions within parenthetical expressions, evaluation begins with the innermost expression. Parenthetical expressions are evaluated as they are encountered in the right-to-left scanning process.

When writing an integer expression, it is important to remember not only the left-to-right scanning process but also that dividing an integer quantity by an integer quantity always yields a truncated result; thus 11/3 = 3. The expression I*J/K may yield a different result than the expression J/K*I.

For example, 4*3/2 = 6 but 3/2*4 = 4.

Examples:

In the following examples, R indicates an intermediate result in evaluation:

A**B/C+D*E*F-G is evaluated:

$A^{**}B \rightarrow R_1$

$R_1/C \rightarrow R_2$

$D*E \rightarrow R_3$

$R_3*F \rightarrow R_4$

$R_4+R_2 \rightarrow R_5$

$R_5-G \rightarrow R_6$ evaluation completed

A**B/(C+D)*(E*F-G) is evaluated:

$E*F-G \rightarrow R_1$

$C+D \rightarrow R_2$

$A^{**}B \rightarrow R_3$

$R_3/R_2 \rightarrow R_4$

$R_4*R_1 \rightarrow R_5$ evaluation completed

H(I3)+C(I,J+2)*(COS(Z))**2 is evaluated:

$COS(Z) \rightarrow R_1$

$R_1{}^{**}2 \rightarrow R_2$

$R_2*C(I,J+2) \rightarrow R_3$

$R_3+H(13) \rightarrow R_4$ evaluation completed

The following is an example of an expression with embedded parentheses.

A*(B+((C/D)-E)) is evaluated:

$C/D \rightarrow R_1$

$R_1-E \rightarrow R_2$

$R_2+B \rightarrow R_3$

$R_3*A \rightarrow R_4$ evaluation completed

(A*(SIN(X)+1.)-Z)/(C*(D-(E+F))) is evaluated:

$E+F \rightarrow R_1$

$D-R_1 \rightarrow R_2$

$C*R_2 \rightarrow R_3$

$SIN(X) \rightarrow R_4$

$R_4+1. \rightarrow R_5$
$A*R_5 \rightarrow R_6$
$R_6-Z \rightarrow R_7$
$R_7/R_3 \rightarrow R_8$ evaluation completed

3.1.2 MIXED-MODE ARITHMETIC EXPRESSIONS

Mixed-mode arithmetic with the exception of exponentiation is completely general; however, most applications probably mix operand types, real and integer, real and double, or real and complex. The relationship between the mode of an evaluated expression and the types of operands it contains is established as follows.

Order of dominance of the operand types within an expression from highest to lowest:

Complex

Double

Real

Integer

Logical

Simple double precision expressions are not evaluated by closed subroutines, but by in-line arithmetic instructions.

The type of an evaluated arithmetic expression is the mode of the dominant operand type.

In expressions of the form A**B, the following rules apply:

If B is preceded by a unary minus operator, the form is A**(-B).

B is treated as an integer if type logical.

For the various operand types, the type relationships of A**B are:

		Type B				
		I	R	D	C	L
T	I	I	n	n	n	I
y	R	R	R	D	n	R
p	D	D	D	D	n	D
e	C	C	n	n	n	C
A	L	I	n	n	n	I

mode of A**B

n indicates an invalid operation

For example, if A is real and B is integer, the mode of A**B is real.

Examples:

1) Given real A, B; integer I, J. The type of expression A*B-I+J is real because the dominant operand type is real.

   The expression is evaluated:

   Convert I to real
   Convert J to real
   $A*B \rightarrow R_1$ real
   $R_1-I \rightarrow R_2$ real
   $R_2+J \rightarrow R_3$ real

2) The use of parentheses can change the evaluation. A,B,I,J are defined as above. A*B-(I-J) is evaluated:

   $I-J \rightarrow R_1$ integer
   $A*B \rightarrow R_2$ real
   Convert R_1 to real
   $R_2-R_1 \rightarrow R_3$ real

3) Given complex C,D, real A,B. The type of the expression A* (C/D)+B is complex because the dominant operand type is complex. The expression is evaluated:

   $C/D \rightarrow R_1$ complex
   Convert A to complex
   $A*R_1 \rightarrow R_2$ complex
   Convert B to complex
   $R_2+B \rightarrow R_3$ complex

4) Consider the expression C/D+(A-B) where the operands are defined in 3 above. The expression is evaluated:

   $A-B \rightarrow R_1$ real
   $C/D \rightarrow R_2$ complex
   Convert R_1 to complex
   $R_1+R_2 \rightarrow R_3$ complex

5) Mixed-mode arithmetic with all types is illustrated by this example:

Given: the expression C*D+R/I-L

C	Complex
D	Double
R	Real
I	Integer
L	Logical

The dominant operand type in this expression is complex; therefore, the evaluated expression is complex.

Evaluation:

Round D to real and affix zero imaginary part.

Convert D to	complex
$C*D \rightarrow R_1$	complex
Convert R to	complex
Convert I to	complex
$R/I \rightarrow R_2$	complex
$R_2+R_1 \rightarrow R_3$	complex
$R_3-L \rightarrow R_4$	complex

If the same expression is rewritten with parentheses as C*D+(R/I-L) the evaluation proceeds:

Convert I to	real
$R/I \rightarrow R_1$	real
$R_1-L \rightarrow R_2$	real
Convert D to	complex
$C*D \rightarrow R_3$	complex
Convert R_2 to	complex
$R_2+R_3 \rightarrow R_4$	complex

3.2 RELATIONAL EXPRESSIONS

A relational expression has the form:

a_1 op a_2

The a's are arithmetic expressions; op is an operator belonging to the set:

.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE.	Greater than or equal to
.LT.	Less than
.LE.	Less than or equal to

A relation is true if a_1 and a_2 satisfy the relation specified by op; otherwise, it is false. A false relational expression is assigned the value plus zero; a true relational expression is assigned the value minus zero (all one bits).

Relations are evaluated as illustrated in the relation p.EQ.q. This is equivalent to the question: does p - q = 0?

The difference is computed and tested for zero. If the difference is zero or minus zero, the relation is true. If the difference is not zero or minus zero, the relation is false. Relational expressions are converted internally to arithmetic expressions according to the rules of mixed-mode arithmetic. These expressions are evaluated and compared with zero to determine the truth value of the corresponding relational expression. When complex expressions are tested for zero or minus zero, only the real part is used in the comparison.

Relational expressions of the following forms are allowed:

I.LT.R

I.LT.D

I.LT.C

I is integer, R is real, D is double precision and C is complex.

Order of dominance of the operand types within an expression is the order stated in mixed mode arithmetic expressions.

The relation of the form I.GE.0 is treated as true if I assumes the value -0.

a_1 op a_2 op a_3. . . is not a valid expression.

A relation of the form a_1 op a_2 is evaluated from left to right. The relations a_1 op a_2, a_1 op (a_2), (a_1) op a_2, (a_1) op (a_2) are equivalent.

Examples:

```
A   .GT.  16.              R(I).GE.R(I-1)
R-Q(I)*Z.LE.3.141592       K  .LT.  16
B-C .NE.  D+E              I  .EQ.  J(K)
                           (I) .EQ. (J(K) )
```

3.3 LOGICAL EXPRESSIONS

A logical expression has the general form:

L_1 op L_2 op L_3. . .

The terms L_i are logical variables, logical constants, or relational expressions and op is the logical operator .AND. indicating conjunction or .OR. indicating disjunction.

The logical operator .NOT. indicating negation appears in the form:

.NOT. L_1

The value of the expression is examined. If the value is equal to plus zero, the logical expression has the value false. All other values are considered true.

The hierarchy of logical operations is:

First	.NOT.	or	.N.
then	.AND.	or	.A.
then	.OR.	or	.O.

A logical variable, logical constant, or a relational expression is, in itself, a logical expression. If L_1, L_2 are logical expressions, then the following are logical expressions:

.NOT.L_1
L_1.AND.L_2
L_1.OR.L_2

If L is a logical expression, then (L) is a logical expression.

If L_1, L_2 are logical expressions and op is .AND. or .OR., then L_1 op op L_2 is never legitimate.

.NOT. may appear in combination with .AND. or .OR. only as follows:

L_1.AND. .NOT.L_2
L_1.OR. .NOT.L_2
L_1.AND.(.NOT. . . .)
L_1.OR.(.NOT. . . .)

.NOT. may appear with itself only in the form .NOT.(.NOT.(.NOT. L))
Other combinations cause compilation diagnostics.

If L_1, L_2 are logical expressions, the logical operators are defined as follows:

.NOT.L_1	is false only if L_1 is true
L_1.AND.L_2	is true only if L_1, L_2 are both true
L_1.OR.L_2	is false only if L_1, L_2 are both false

Examples:

1) $B - C \leq A \leq B + C$ is written
   B-C.LE.A.AND.A.LE.B+C
2) FICA greater than 176.0 and PAYNMB equal to 5889.0 is written
   FICA.GT.176.0.AND.PAYNMB.EQ.5889.0
3) An expression equivalent to the logical relationship (P $\longrightarrow$ Q) may be written in two ways:
   .NOT.(P.AND.(.NOT.Q))
   .N.(P.A.(.N.Q))

3.4 MASKING EXPRESSIONS

The masking expression is a generalized form of the logical expression in which the variables may be types other than logical.

In a FORTRAN masking expression, 60-bit logical arithmetic is performed bit-by-bit on the operands within the expression. The operands may be any type variable, constant, or expression. No mode conversion is performed during evaluation. If the operand is complex, operations are performed on the real part. Although the masking operators are identical in appearance to the logical operators, their meanings are different. They are listed according to hierarchy. The following definitions apply:

.NOT. or .N.	complement the operand
.AND. or .A.	form the bit-by-bit logical product of two operands
.OR. or .O.	form the bit-by-bit logical sum of two operands

The operations are described below:

p	v	p .AND. v	p .OR. v	.NOT. p
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Let B_i be masking expressions, variables or constants of any type. The following are masking expressions:

.NOT. B_1 B_1 .AND. B_2 B_1 .OR. B_2

If B is a masking expression, then (B) is a masking expression.

.NOT. may appear with .AND. or .OR. only as follows:

.AND. .NOT.

.OR. .NOT.

.AND. (.NOT. . . .)

.OR. (.NOT. . . .)

Masking expressions of the following forms are evaluated from left to right.

A .AND. B .AND. C . . .

A .OR. B .OR. C . . .

Masking expressions must not contain parenthetical arithmetic expressions or functions.

Examples:

A	77770000000000000000	octal constant
D	00000000777777777777	octal constant
B	00000000000000001763	octal form of integer constant
C	20045000000000000000	octal form of real constant

.NOT. A	is	00007777777777777777
A .AND. C	is	20040000000000000000
A .AND. .NOT. C	is	57730000000000000000
B .OR. .NOT. D	is	77777777000000001763

The last expression could also be written as B .O. .N. D

REPLACEMENT STATEMENTS 4

4.1 ARITHMETIC REPLACEMENT

The general form of the arithmetic replacement statement is A = E, where E is an arithmetic expression and A is any variable name, simple or subscripted. The operator = means that A is replaced by the value of the evaluated expression, E, with conversion for mode if necessary.

Examples:

```
      A = -A
      B(B,4) = CALC(I+1)*BETA+2.3478
   39 XTHETA=7.4*DELTA+(A(I,J,K)**BETA)
      RESPSNE=SIN(ABAR(INV+2,JBAR)/ALPHA(J,KAPL(I) ) )
    4 JMAX=19
      AREA = SIDE1 * SIDE2
      PERIM = 2.*(SIDE1 + SIDE2)
```

4.2 MIXED-MODE REPLACEMENT

The type of an evaluated expression is determined by the type of the dominant operand. This, however, does not restrict the types that identifier A may assume. A complex expression may replace A, even if A is real. The following chart shows the A = E relationship for all the standard modes. The mode of A determines the mode of the statement.

When all the operands in the expression E are logical, the expression is evaluated as if all the logical operands were integers.

For example, if L_1, L_2, L_3, L_4 are logical variables, R is a real variable, and I is an integer variable, then $I = L_1 * L_2 + L_3 - L_4$ is evaluated as if the L_i were all integers and the resulting value is stored as an integer in I.

$R = L_1 * L_2 + L_3 - L_4$ is evaluated as stated above, but the result is converted to a real (a floating point quantity) before it is stored in R.

Type of A	Type of Expression E			
	Complex	Double Precision	Real	Integer
Complex	A = E	Set A = most significant half of E $A_{real} = E$ $A_{imag} = 0$	$A_{real} = E$ $A_{imag} = 0$	Convert E to Real $A_{real} = E$ $A_{imag} = 0$
Double Precision	$A = E_{real}$ less significant is set to zero	A = E	A = E less significant is set to zero	Convert E to Real A = E
Real	$A = E_{real}$	Set A = most significant half of E A = E	A = E	Convert E to Real A = E less significant is set to zero
Integer	Truncate E_{real} to Integer A = E	Truncate E to Integer A = E	Truncate E to Integer A = E	A = E
Logical	If $E_{real} \neq 0$, A = 1 If $E_{real} = 0$, A = 0	If $E \neq 0$, A = 1 If E = 0, A = 0	If $E \neq 0$, A = 1 If E = 0, A = 0	If $E \neq 0$, A = 1 If E = 0, A = 0

Examples:

Given:

C_i, A_1	Complex
D_i, A_2	Double
R_i, A_3	Real
I_i, A_4	Integer
L_i, A_5	Logical

1. $A_1 = C_1 * C_2 - C_3 / C_4$ (6.905, 15.393) = (4.4, 2.1)* (3.0, 2.0) - (3.3, 6.8) / (1.1, 3.4)

The expression is complex; the result of the expression is a two-word, floating point quantity. A_1 is complex, and the result replaces A_1.

2. $A_3 = C_1$ 4.4000+000 = (4.4,2.1)

The expression is complex. A_3 is real; therefore, the real part of C_1 replaces A_3.

3. $A_3 = C_1 * (0.\ ,-1.)$ 2.1000+000 = (4.4,2.1)* (0. ,-1.)

The expression is complex. A_3 is real; the real part of the result of the complex multiplication replaces A_3.

4. $A_4 = R_1/R_2 * (R_3 - R_4) + I_1 - (I_2 * R_5)$ 13=8.4/4.2*(3.1-2.1)+ 14-(1*2.3)

The expression is real. A_4 is integer; the result of the expression evaluation, a real, is converted to an integer replacing A_4.

5. $A_2 = D_1 ** 2 * (D_2 + (D_3 * D_4)) + (D_2 * D_1 * D_2)$ 4.96800000000000+001= 2.0D**2*(3.2D+(4.1D*1.0D)) +(3.2K*2.0D*3.2D)

The expression is double precision. A_2 is double precision; the result of the expression evaluation, a double precision floating quantity, replaces A_2.

6. $A_5 = C_1 * R_1 - R_2 + I_1$ 1=(4.4,2.1)*8.4-4.2+14

The expression is complex. Since A_5 is logical, the real part of the evaluated expression replaces A_5. If the real part is zero, zero replaces A_5.

4.3 LOGICAL REPLACEMENT

The general form of the logical replacement statement is L = E, where L is a logical variable and E may be a logical, relational, or arithmetic expression:

Examples:

```
      LOGICAL A, B, C, D, E, LGA, LGB, LGC
      REAL F, G, H
      A = B .AND. C .AND. D
      A = F .GT. G .OR. F .GT. H
    5 A = .N. (A.A. .N. B) .AND. (C.O.D)
      LGA = .NOT. LGB
 2109 LGC = E .OR. LGC .OR. LGB .OR. LGA .OR. (A .AND. B)
```

4.4 MASKING REPLACEMENT

The general form of the masking replacement statement is M = E. E is a masking expression, and M is a variable of any type except logical. No mode conversion is made during the replacement.

Examples:

```
      INTEGER I,J,K,L,M,N(16)
      REAL B,C,D,E,F(15)

      N(2) = I .AND. J
      B = C .AND. L
84    F(J) = I .OR. .NOT. L .AND. F(J)
      N(1) = I.O.J.O.K.O.L.O.M
      I = .N.I
      D = (B.LS. C) .AND. (C .LE. E) .AND. .NOT. I
```

TYPE DECLARATIONS AND STORAGE ALLOCATION 5

5.1 TYPE DECLARATION

The type declaration statement provides the compiler with information on the structure of variable and function identifiers.

Statement	Characteristics	
COMPLEX list	2 words/element	Floating Point
DOUBLE PRECISION list or DOUBLE list	2 words/element	Floating Point
REAL list	1 word/element	Floating Point
INTEGER list	1 word/element	Integer
LOGICAL list	1 word/element	Logical

TYPE list may be used for compatibility during compilation time.

DOUBLE may replace DOUBLE PRECISION in any FORTRAN statement in which the latter is allowed.

List is a string of identifiers separated by commas; integer constant subscripts are permitted. For example:

A, B1, CAT, D36F, GAR (1, 2, 3)

The type declaration is non-executable and must precede the first reference to the variable or function in a given program. If an identifier is declared in two or more type declarations, the first declaration holds until the second is read, the second holds until the third, etc.

An identifier not declared in a type declaration is type integer if the first letter of the name is I, J, K, L, M, N; for any other letter, it is type real.

When subscripts appear in the list, the associated identifier is the name of an array, and the product of the subscripts determines the amount of storage to be reserved for that array.

Examples:

```
COMPLEX A412,DATA,DRIVE,IMPORT
DOUBLE PRECISION PLATE,ALPHA(20,20),B2MAX,F60,JUNE
REAL I,J(20,50,2),LOGIC,MPH
INTEGER GAR(60),BETA,ZTANK,AGE,YEAR,DATE
LOGICAL DISJ,IMPL,STROKE,EQUIV,MODAL
DOUBLE RL,MASS(10,10)
```

5.2 DIMENSION DECLARATION

A subscripted variable represents an element of an array of variables. Storage is reserved for arrays by the non-executable statements DIMENSION, COMMON, or a type statement.

The standard form of the DIMENSION declaration is:

DIMENSION $v_1, v_2, \ldots, v_n$

The variable names v_i may have 1, 2, or 3 integer constant subscripts separated by commas, as in SPACE (5, 5, 5). Under certain conditions within subprograms only, the subscripts may be constants or variables.

Example:

```
SUBROUTINE X(A,L,M)
DIMENSION A(L,10,M)
```

The DIMENSION declaration is non-executable and it must precede the first reference to the array in a given program.

The number of computer words reserved for an array is determined by the product of the subscripts in the subscript string and the type of the variable. A maximum of $2^{17}-1$ elements may be reserved in any one array. If the maximum is exceeded, a diagnostic is provided.

```
COMPLEX ATOM
DIMENSION ATOM (10,20)
```

In the above declarations, the number of elements in the array ATOM is 200. Two words are used to contain a complex element; therefore, the number of computer words reserved is 400. This is also true for double precision arrays. For real, logical, and integer arrays, the number of words in an array equals the number of elements in the array.

If an array is dimensioned in more than one declaration statement, a diagnostic is provided.

Examples:

```
DIMENSION A(20,2,5)
DIMENSION MATRIX(10,10,10),VECTOR(100),ARRAY(16,27)
```

5.2.1 VARIABLE DIMENSIONS

When an array identifier and its dimensions appear as formal parameters in a function or subroutine, the dimensions may be assigned through the actual parameter list accompanying the function reference or subroutine call. The dimensions must not exceed the maximum array size specified by the DIMENSION declaration in the calling program. (See Variable Dimensions in Subprograms in Chapter 7.)

5.3 COMMON DECLARATION

The COMMON declaration provides blocks of storage that can be referenced by more than one subprogram. The declaration reserves both numbered and labeled blocks. Only labeled common blocks may be preset. Data may be stored in labeled common blocks by the DATA declaration and is made available to any subprogram, using the appropriate labeled block.

The starting addresses for both numbered and labeled blocks are indicated on the map listing.

Areas of common information may be specified by the declaration:

COMMON/i_1/list/i_2/list. . .

The common block identifier, i, may be 1-7 characters. If the first character is alphabetic, the identifier denotes a labeled common block; remaining characters may be alphabetic or numeric. If the first character is numeric, remaining characters must be numeric and the identifier denotes a numbered common block. Leading zeros in numeric identifiers are ignored. Zero by itself is an acceptable numbered common block identifier.

Example:

```
COMMON/200/A, B, C
```

Such common assignments are treated as blank common assignments; they are made to the high end of the field in which the program runs. Data may not be entered into these blocks by the DATA declaration. The following are common identifiers:

Labeled	Numbered
AZ13	1
MAXIM	146
Z	6600
XRAY	0

List is a string of identifiers representing simple and subscripted variables. If a non-subscripted array name appears in the list, the dimensions must be defined by a type or DIMENSION declaration in that program. If an array is dimensioned in more than one declaration, a computer diagnostic is provided. The order of array storage within a common block is determined by the COMMON declaration.

The common block identifier with or without the separating slashes may be omitted for blank common. Blank common is treated as numbered common by the compiler. Numbered common blocks must be declared in the main program.

COMMON is non-executable and can appear anywhere in the program. Any number of COMMON declarations may appear in a program. If DIMENSION, COMMON or type declarations appear together, the order is immaterial.

Since labeled and numbered common block identifiers are used only within the compiler, they may be used elsewhere in the program as other kinds of identifiers except subroutine names in the same job. An identifier in one common block may not appear in another common block. (If it does, the name is doubly defined.)

Numbered common blocks are treated as blank common assignments. Data may not be entered into these blocks by the DATA statement. At the beginning of program execution, the contents of all common areas are zero except labeled common areas specified in a DATA declaration.

Examples:

```
COMMON A,B,C
COMMON/ /E,F,G,H
COMMON/BLOCKA,/A1(15),B1,C1/BLOCKD/DEL(5,2),ECHO
COMMON/VECTOR/VECTOR(5),HECTOR,NECTOR
```

The length of a common block in computer words is determined from the number and type of the list variables. In the following statements, the length of common block A is 12 computer words. The origin of the common block is Q (1).

```
COMMON/A/Q(4), R(4), S(2)
REAL Q,R
COMPLEX S
```

	Block A	
origin	Q(1)	
	Q(2)	
	Q(3)	
	Q(4)	
	R(1)	
	R(2)	
	R(3)	
	R(4)	
	S (1)	real part
	S (1)	imaginary part
	S (2)	real part
	S (2)	imaginary part

If a subprogram does not use all of the locations reserved in a common block, unused variables may be necessary in the COMMON declaration to insure proper correspondence of common areas.

```
COMMON/SUM/A,B,C,D   (main program)
COMMON/SUM/E(3),D    (subprogram)
```

In the above example, only the variable D is used in the subprogram. The unused variable E is necessary to space over the area reserved by A,B, and C.

Each subprogram using a common block assigns the allocation of words in the block. The identifiers used within the block may differ as to name, type, and number of elements; but the block identifier must remain the same.

Example:

```
PROGRAM MAIN
COMPLEX C
COMMON/TEST/C(20)/36/A,B,Z
.
.
.
```

The length of the block named TEST is 40 computer words. The length of the block numbered 36 is 3 computer words.

The subprogram may rearrange the allocation of words as in:

```
SUBROUTINE ONE
COMMON/TEST/A(10),G(10),K(10)
COMPLEX A
:
:
```

The length of TEST is 40 words. The first 10 elements (20 words) of the block represented by A are complex elements. Array G is the next 10 words, and array K is the last 10 words. Within the subprogram, elements of G are treated as floating point quantities; elements of K are treated as integer quantities.

The length of a common block must not be changed by subprograms using the block. The symbolic names used within the block may differ, however, as shown above.

5.4 EQUIVALENCE DECLARATION

The EQUIVALENCE declaration permits variables to share locations in storage. The general form is:

```
EQUIVALENCE (A,B, . . .), (A1,B1, . . .), . . .
```

(A,B, . . .) is an equivalence group of two or more simple or singly subscripted variable names. A multiply subscripted variable can be represented only by a singly subscripted variable. The correspondence is:

A(i,j,k) is the same as A(the value of (i+(j-1)*I+(k-1)*I*J))

i,j,k are integer constants; I and J are the integer constants appearing in DIMENSION A(I,J,K). For example, in DIMENSION A(2,3,4), the element A(1,1,2) is represented by A(7).

EQUIVALENCE is most commonly used when two or more arrays can share the same storage locations. The lengths need not be equal.

Example:

```
      DIMENSION A(10,10),I(100)
      EQUIVALENCE (A,I)
5     READ 10, A
                :
                :
6     READ 20,I
```

The EQUIVALENCE declaration assigns the first element of array A and array I to the same storage location. The READ statement 5 stores the A array in consecutive locations. Before statement 6 is executed, all operations using A should be completed since the values of array I are read into the storage locations previously occupied by A.

Variables requiring two memory positions which appear in EQUIVALENCE statements must be declared to be COMPLEX or DOUBLE prior to their appearance in such statements.

Example:

```
COMPLEX DAT,BAT
DIMENSION DAT(10,10),BAT(10,10),CAT(10,10)
DOUBLE PRECISION CAT
COMMON/IFAT/FAT(2,2)
EQUIVALENCE (DAT(6,3),FAT(2,2) ),(CAT,BAT)
:
:
```

EQUIVALENCE is non-executable and can appear anywhere in the program or subprogram.

Any full or multiword variable may be made equivalent to any other full or multiword variable. The variables may be with or without subscript. In FORTRAN II, equivalence groups can reorder the common variables and arrays, and more than one variable in an equivalence group may be in common.

The following examples illustrate changes in block lengths caused by the EQUIVALENCE declaration.

Given: Arrays A and B

Sa subscript of A

Sb subscript of B

Examples:

1. A and C in common, B not in common

   Sb $\leq$ Sa is a permissible subscript arrangement

   Sb > Sa is not

		Block 1	
origin	A(1)		COMMON/1/A(4),C
	A(2)	B(1)	DIMENSION B(5)
	A(3)	B(2)	EQUIVALENCE (A(3),B(2))
	A(4)	B(3)	
	C	B(4)	
		B(5)	

5.5 DATA DECLARATION

Values may be assigned to program variables or labeled common variables with the DATA declaration:

DATA $d_1, \ldots, d_n/a_1, k*a_2, \ldots, a_n/, d_1, \ldots, d_n/a_1, \ldots, a_n/, \ldots$

- d_i identifiers representing simple variables, array names, or variables with integer constant subscripts or integer variable subscripts (implied DO-loop notation).
- a_i literals (any constant); they may be signed or unsigned.
- k integer constant repetition factor that causes the literal following the asterisk to be repeated k times. If k is noninteger, a compiler diagnostic occurs.

DATA is non-executable and can appear anywhere in the program or subprogram When DATA appears with DIMENSION, COMMON, EQUIVALENCE, or a type declaration, the order is immaterial. Variables in blank or numbered common or formal parameters may not be preset by a DATA declaration.

Single-subscript, DO-loop-implying notation is permissible. This notation may be used for storing constant values in arrays.

Example:

```
DIMENSION GIB(10)
DATA (GIB(I), I=1, 10)/1. , 2. , 3. , 7*4.32/
```

Array GIB:
1.
2.
3.
4.32
4.32
4.32
4.32
4.32
4.32
4.32

In the DATA declaration, the type of the constant stored is determined by the structure of the constant rather than by the variable type in the statement. In DATA A/2/, an integer 2 replaces A, not a real 2 as might be expected from the form of the symbolic name A.

There should be a one-one correspondence between the variable names and the list. This is particularly important in arrays in labeled common. For instance:

```
COMMON/BLK/A(3),B
DATA A/1. ,2. ,3. ,4./
```

The constants 1. ,2. ,3. , are stored in array locations A, A+1, A+2; the constant 4. is stored in location B. If this occurs unintentionally, errors may occur when B is referred to elsewhere in the program.

```
COMMON/TUP/C(3)
DATA C/1. ,2./
```

The constants 1. , 2. are stored in array locations C and C+1; the content of C(3), that is, location C+2, is not defined.

When the number of list elements exceeds the range of the implied DO, the excess list elements are not stored.

```
DATA (A(I), I=1, 5, 1)/1. , 2. , . . . ,10./
```

The excess values 6. through 10. are discarded.

Examples:

1) DATA LEDA, CASTOR, POLLUX/15,16.0,84.0/

LEDA	15
	.
	.
	.
CASTOR	16.0
	.
	.
	.
POLLUX	84.0

2) DATA A(1,3)/16.239/

ARRAY A	
A(1,3)	16.239

3) DIMENSION B(10)

DATA B/O000077, O000064, 3*O000005, 5*O000200/

ARRAY B	O77
	O64
	O5
	O5
	O5
	O200
	O200
	O200
	O200
	O200

4) COMMON/HERA/C(4)

DATA C/3.6, 3*10.5/

ARRAY C	3.6
	10.5
	10.5
	10.5

5) COMPLEX PROTER (4)

DATA PROTER/4*(1.0,2.0)/

ARRAY PROTER	1.0
	2.0
	1.0
	2.0
	1.0
	2.0
	1.0
	2.0

6) DIMENSION MESAGE (3)

DATA MESAGE/9HSTATEMENT,2HIS,10HINCOMPLETE/

ARRAY MESAGE	STATEMENT
	IS
	INCOMPLETE

Data declaration statements of the following forms may also be used to assign constant values to program or labeled variables at load time.

DATA (i_1=value list), (i_2=value list), . . .

DATA (i(j,k,l)=list), . . .

DATE (((i(I,J),I=n_1,n_2),J=m_1,m_2)=list), . . .

The variable identifier, i, may be:

non-subscripted variable

array variable with constant subscripts

array name

array element with integer variable quantifiers

The value list is either a single constant or set of constants whose number is equal to the number of elements in the named array.

List contains constants only and has the form:

a_1,a_2, . . . ,k(b_1,b_2, . . .),c_1,c_2, . . .

k is an integer constant repetition factor that causes the parenthetical list following it to be repeated k times. If k is non-integer a compiler diagnostic is provided.

Examples:

```
COMMON/DATA/GIB
DATA ( (GIB(I), I=1,10)=1. , 2. , 3. , 7(4.32) )
COMMON/DATA/ROBIN(5,5,5)
DATA (ROBIN(4,3,2)=16.)
COMMON/DATA/BAT(10,10)
DATA ( (BAT(10,N), N=1,3)=2. , 6. , 10.)
```

5.5.1 BLOCK DATA SUBPROGRAM

A block data subprogram may be used to enter data into labeled common prior to program execution in place of a DATA declaration and it may appear more than once in a FORTRAN program. A block data subprogram has the form:

```
BLOCK DATA
    .
    .
    .
FORTRAN declaration statements only
    .
    .
    .
END
```

All elements in the common blocks must appear in a COMMON declaration in the subprogram even if they are not in the DATA declaration.

Example:

```
BLOCK DATA
COMMON/ABC/A(5),B,C/DEF/D,E,F
COMPLEX D,E
DOUBLE PRECISION F
DATA (A(L),L=1,5)/2.3,3.4,3*7.1/B/2034.756/D,E,F/2*(1.0,2.5),
17.86972415872E30/
END
```

CONTROL STATEMENTS 6

Program execution normally proceeds from one statement to the statement immediately following it in the program. Control statements can be used to alter this sequence or cause a number of iterations of a program section.

Control may be transferred to an executable statement only; a transfer to a non-executable statement results in a program error.

6.1 GO TO STATEMENTS

Program control is transferred to a statement other than the next statement in sequence by the GO TO statements.

6.1.1 UNCONDITIONAL GO TO

GO TO n

An unconditional transfer is made to the statement labeled n.

6.1.2 ASSIGNED GO TO

GO TO m, (n_1,n_2, . . . ,n_m)

GO TO m

This statement acts as a many-branch GO TO; m is a simple integer variable assigned an integer value n in a preceding ASSIGN statement. The n_i are statement labels. As shown, the parenthetical statement label list need not be present.

The comma after m is optional; however, when the list is omitted, the comma must be omitted. m cannot be defined as the result of a computation. No compiler diagnostic is given if m is computed, but the object code is incorrect. If an assignment has not been made for an assigned GO TO statement during run time, a diagnostic is provided at object time.

6.1.3 ASSIGN STATEMENT

ASSIGN s TO m

This statement is used with the assigned GO TO statement; s is a statement label, m is a simple integer variable.

Example:

```
ASSIGN 10 TO LSWTCH
     .
     .
     .
GO TO LSWTCH, (5,10,15,20)
```

Control transfers to statement 10.

6.1.4 COMPUTED GO TO

GO TO $(n_1,n_2, \ldots ,n_m),i$

This statement acts as a many-branch GO TO; s is preset or computed prior to its use in the GO TO.

The n_i are statement labels and i is a simple integer variable. If $i < 1$ or if $i > m$, the transfer is undefined and an object time diagnostic will be issued indicating the point at which the error was detected. If $1 \leq i \leq m$, the transfer is to n_i.

The comma separating the statement number list and the index is optional.

Example:

```
N=3
  .
  .
  .
GO TO (100,101,102,103) N
```

Statement number 102 will be the selected control transfer.

For proper operations, i must not be specified by an ASSIGN statement. No compilation diagnostic is provided for this error, but the object code is incorrect.

Example:

```
      ISWICH = 1
      GO TO (10,20,30), ISWICH
            .
            .
            .
10    JSWICH = ISWICH + 1
      GO TO (11,21,31), JSWICH
```

Control transfers to statement 21.

6.2 IF STATEMENTS

Program control is transferred to a statement depending upon the condition of the computed results of the IF statements.

6.2.1 THREE-BRANCH ARITHMETIC IF

IF (A) n_1,n_2,n_3

A is an arithmetic expression, and the n_i are statement labels. This statement tests the evaluated expression A and jumps accordingly as follows:

A < 0	jump to statement n_1
A = 0	jump to statement n_2
A > 0	jump to statement n_3

In the test for zero, +0 = -0. When the mode of the evaluated expression is complex, only the real part is tested for zero.

```
      IF(A*B-SINF(X) ) 10,20,10
      IF (I)5,6,7
402   IF (A/B ** 2) 3, 6, 6
```

6.2.2 ONE-BRANCH LOGICAL IF

IF (L) s

L is a logical expression and s is a statement. If L is TRUE (negative, including minus zero), the statement is executed. If L is FALSE (plus zero) the statement immediately following the IF statement is executed.

```
IF(A.LE.2.5) A=2.0
IF (VALUE*4.73.GT.PRICE.OR.VALUE.LT.150.0)BUY=.TRUE.
IF(P.AND.Q)GO TO 427
```

6.2.3 TWO-BRANCH LOGICAL IF

IF (L) n_1,n_2

L is a logical expression; n_i are statement labels.

The evaluated expression is tested for true (nonzero) or false (plus zero) condition. If L is true, the jump is to statement n_1. If L is false, the jump is to statement n_2.

Example:

```
      IF(K)5,6
5     IF(K.EQ.100)70,60
6     IF(IJUMP.LT.K)10,11
```

6.3 DO STATEMENT

DO n i = m_1,m_2,m_3

This statement makes it possible to repeat groups of statements and to change the value of an integer variable during the repetition. n is the statement label ending the DO loop; i is the index variable (simple integer). m_i are the indexing parameters; they may be unsigned integer constants or simple integer variables. The initial value assigned to i is m_1, m_2 is the largest value assigned to i, and m_3 is the amount added to i after each time the DO loop is executed. If m_3 does not appear, it is assigned the value 1. If a statement number terminating a DO loop has not been previously referenced except in a DO statement, it is ignored. A later reference to test number causes a missing statement number indication.

The DO statement, the statement labeled n, and any intermediate statements constitute a DO loop. Statement n may not be an arithmetic IF or GO TO statement, FORMAT declaration, or another DO statement.

The indexing parameters m_1,m_2,m_3 are either unsigned integer constants or simple integer variables. Subscripted variables and negative or zero integer constants cause a diagnostic.

The indexing parameters m_1 and m_2, if variable, may assume positive or negative values or zero.

The values of m_1,m_2, and m_3 may be changed during the execution of the DO loop.

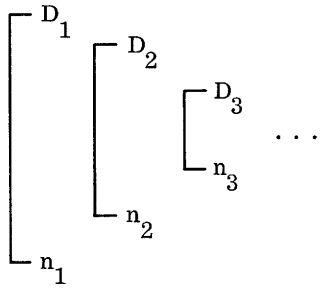
6.3.1 DO LOOP EXECUTION

The initial value of i, m_1, is increased by m_3 and compared with m_2 after executing the DO loop once, and if i does not exceed m_2, the loop is executed a second time. Then, i is again increased by m_3 and again compared with m_2; this process continues until i exceeds m_2. Control then passes to the statement immediately following n, and the DO loop is satisfied.

Should m_1 exceed m_2 on the initial entry to the loop, the loop is executed once and control is passed to the statement following n. When the DO loop is satisfied, the index variable i is no longer well defined. If a transfer out of the DO loop occurs before the DO is satisfied, the value of i is preserved and may be used in subsequent statements.

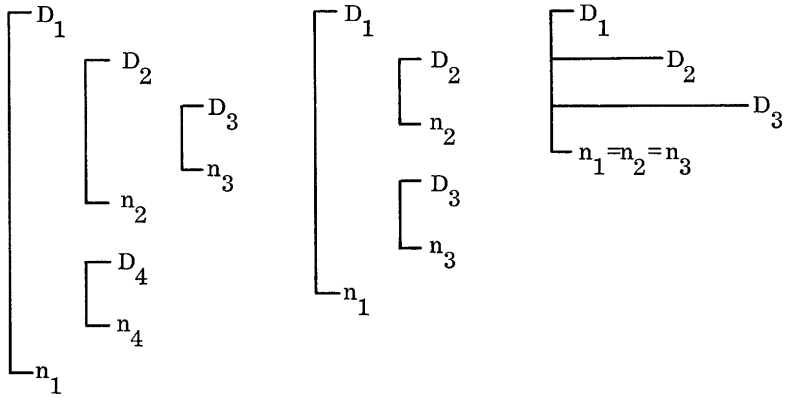
6.3.2 DO NESTS

When a DO loop contains another DO loop, the grouping is called a DO nest. Nesting may be to any level. The last statement of a nested DO loop must either be the same as the last statement of the outer DO loop or occur before it. If $D_1, D_2, \ldots D_m$ represent DO statements where the subscripts indicate that D_1 appears before D_2 appears before D_3 and $n_1, n_2, \ldots, n_m$ represent the corresponding limits of the D_i, then n_m must appear at or before n_{m-1}.



Examples:

DO loops may be nested in common with other DO loops:



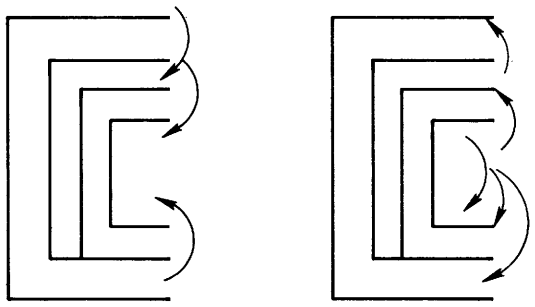
```
DO 1 I=1,10,2
    .
    .
    .
DO 2 J=1,5
    .
    .
    .
DO 3 K=2,8
    .
    .
    .
3 CONTINUE
    .
    .
    .
2 CONTINUE
    .
    .
    .
DO 4 L=1,3
    .
    .
    .
4 CONTINUE
    .
    .
    .
1 CONTINUE
```

```
DO 100 L=2,LIMIT
    .
    .
    .
DO 10 I=1,10
DO 10 J=1,10
    .
    .
    .
10 CONTINUE
    .
    .
    .
DO 20 K=K1,K2
    .
    .
    .
20 CONTINUE
    .
    .
    .
100 CONTINUE
```

```
DO 5 I=1,5
DO 5 J=I,10
DO 5 K=J,15
    .
    .
    .
5 A = B*C
```

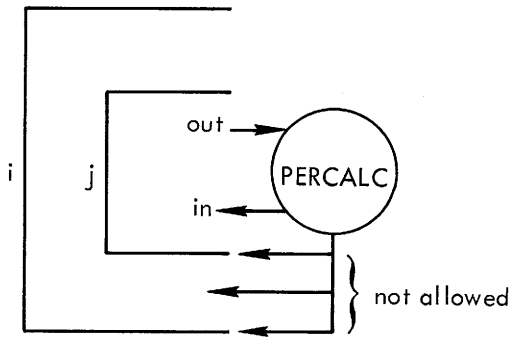
6.3.3 DO LOOP TRANSFER

In a DO nest, a transfer may be made from one DO loop into a DO loop that contains it, but not from the outer DO loop to the inner DO loop without first executing the DO statement of the inner DO loop.



One exception is allowed: when control is transferred completely out of the nested DO to perform some calculation and then transfers back into the range of the same DO from where the original transfer exit was made. For example: In a DO nest, if the range of i includes the range of j and a transfer out of j occurs, then a transfer into the range of j is permissible, but a transfer into the range of i or a transfer to either terminal statement is not permissible.

In the following diagram, PERCALC, represents a portion of the program being completely outside of the DO nest.



6.4 CONTINUE STATEMENT

CONTINUE

The CONTINUE statement is most frequently used as the last statement of a DO loop to provide a loop termination when a GO TO or IF would normally be the last statement of the loop. If CONTINUE is used elsewhere in the source program it acts as a do-nothing instruction and control passes to the next sequential program statement. The CONTINUE statement must contain a statement label in column 1-5.

6.5 PAUSE STATEMENT

PAUSE

PAUSE n

n ≤ 5 octal digits without an O prefix or B suffix. PAUSE n stops program execution with the words PAUSE n displayed as a day file message. An operator entry from the console can continue or terminate the program. Program continuation proceeds with the statement immediately following PAUSE. If n is omitted, it is understood to be zero.

6.6 STOP STATEMENT

STOP

STOP n

n ≤ 5 octal digits without an O prefix or B suffix. STOP terminates the program execution and returns control to the monitor. If n is omitted, it is understood to be zero.

6.7 RETURN STATEMENT

A subprogram normally contains one or more RETURN statements to indicate the end of logic flow within the subprogram and return control to the calling program.

In function subprograms, control returns to the statement containing the function reference. In a subroutine subprogram, control returns to the next executable statement following the CALL. A RETURN statement in the main program causes an exit to the monitor.

6.8 END STATEMENT

END must be the final statement in a program or subprogram. It is executable in the sense that it effects termination of the program.

The END statement may include the name of the program or subprogram which it terminates; the name is ignored, however.

PROGRAM, FUNCTION, AND SUBROUTINE 7

A Chippewa FORTRAN program consists of a main program with or without subprograms. Subprograms are of two kinds: subroutine and function. In the following discussions, the term subprogram refers to both. Subprograms may be compiled independently of the main program.

7.1 PROGRAM COMMUNICATION

The main program and subprograms communicate with each other via parameters and COMMON variables. Subprograms may call or be called by any other subprogram as long as the calls are nonrecursive; that is, if program A calls B, B may not call A. A calling program is a main program or subprogram that refers to another subprogram. A subroutine referenced by a program or segment may have the same name as the program or segment. However, a program or subprogram may not call itself. The following examples illustrate such calling reference:

Example:

```
PROGRAM MAIN (INPUT)
      .                          The program and the subroutine
      .                          have the same name.
      .
CALL MAIN
END
SUBROUTINE MAIN (OUTPUT)
      .
      .
      .
END
SEGMENT LOOP1 (INPUT)
      .                          The segment and the subroutine
      .                          have the same name.
      .
CALL LOOP1
END
SUBROUTINE LOOP1 (OUTPUT)
      .
      .
      .
END
```

7.2 SUBPROGRAM COMMUNICATION

Subprograms, functions, and subroutines use parameters as one means of communication. The parameters appearing in a subroutine call or a function reference are actual parameters. The corresponding arguments appearing with the program, subprogram, statement function, or library function name in the definition are formal parameters. One or more of the formal parameters or common variables can be used to return output to the calling program.

7.3 FORMAL PARAMETERS

Formal parameters may be the names of arrays, simple variables, library functions, and subprograms. Since formal parameters are local to the subprogram containing them, they may be the same as names appearing outside the procedure.

No element of a formal parameter list may appear in an EQUIVALENCE or DATA statement within the subroutine. If it does, a compiler diagnostic results.

When a formal parameter represents an array, it must be dimensioned within the subprogram. If it is not declared, the array name must appear without subscripts and only the first element of the array is available to the subprogram.

7.4 ACTUAL PARAMETERS

Permissible forms:

Arithmetic expression

Logical expression

Constant

Simple or subscripted variable

Array name

FUNCTION subprogram name

Library function and subroutine name

SUBROUTINE name

A calling program statement label, identified by suffixing the label with the character S and used only in MACHINE SUBROUTINE.

A function name or a function reference may be used as an actual parameter. The function reference is a special case of an arithmetic expression.

Function name:

```
SUBROUTINE PULL(X,Y,Z)
      .
      .
      .
Z = X(Y)
      .
      .
      .
```

Calling Program Reference

```
      .
      .
      .
EXTERNAL SIN
CALL PULL(SIN,R,Q)
```

Function reference:

```
SUBROUTINE PULL(X,Z)
      .
      .
      .
Z = X
      .
      .
      .
```

Calling Program Reference

```
      .
      .
      .
CALL PULL(SIN(R),Q)
      .
      .
      .
```

A subroutine name may appear as an actual parameter; any parameters to be associated with a call of this subroutine must appear as separate actual parameters.

Example:

Calling Routine

```
EXTERNAL ADDER
      .
      .
      .
CALL SUB (ADDER,A,B)
      .
      .
      .
```

Called Routine

```
SUBROUTINE SUB (X,Y,Z)
      .
      .
      .
CALL Y(X,Q,P,Z)
```

CALL SUB (ADDER(A,B)) implies that ADDER is a function, not a subroutine.

When an actual parameter is the name of a function or subroutine, that name must also appear in an EXTERNAL statement in the calling program.

Actual and formal parameters must agree in order, type, and number.

7.5 MAIN PROGRAM

The first statement of a main program must be one of the following forms where name is an alphanumeric identifier of 1-7 characters:

PROGRAM name ($f_1, \ldots, f_n$)
FORTRAN IV PROGRAM name ($f_1, \ldots, f_n$)
FORTRAN II PROGRAM name ($f_1, \ldots, f_n$)
MACHINE PROGRAM name ($f_1, \ldots, f_n$) } See
ASCENTF PROGRAM name ($f_1, \ldots, f_n$) } Appendix F

The first two forms cause the program to be compiled in FORTRAN IV mode. The third form causes compiling in FORTRAN II mode. The last two forms are the first statements of programs written in assembly language and compiled by the FORTRAN compiler. The parameter list is optional on all forms.

The parameter f_i must be the names of all input/output files required by the main program and its subprograms. Although these parameters may be changed at execution time, they must, at compile time, satisfy the following conditions:

1. The file name INPUT must appear if any READ statement is included in the program or its subprograms.

2. The file name OUTPUT must appear if any PRINT statement is included in the program or its subprograms.

3. The file name PUNCH must appear if any PUNCH statement is included in the program or its subprograms.

4. The file name TAPE i, must appear if a READ INPUT TAPE i, WRITE OUTPUT TAPE i, READ TAPE i, WRITE TAPE i, READ(i,n), WRITE (i,n), READ (i), or WRITE (i) statement is included in the program or its subprogram. (i is an integer)

5. If I is an integer variable name for a READ INPUT TAPE I, WRITE OUTPUT TAPE I, READ TAPE I, WRITE TAPE I, READ (I,n), WRITE (I,n), READ (I), or WRITE (I) statement which appears in the program or its subprograms, the file names TAPE i_1,..., TAPE i_k must appear. The integers i_1,..., i_k must include all values which are assumed by the variable I. The file name TAPE I may not appear in the list of arguments to the main program.

File names may be made equivalent and their buffer lengths may be specified at compile time.

Example:

PROGRAM name (INPUT,OUTPUT=10000,TAPE1=INPUT,TAPE2=OUTPUT)

All input normally provided by TAPE 1 would be extracted from INPUT and all listable output normally recorded on TAPE 2 would be transmitted to the OUTPUT file. Buffer length is specified by OUTPUT=10000 which establishes an output buffer length of 10000_8. If buffer length is not indicated, a standard buffer size is allocated. Buffer length may not be less than 1001 words. For instance, PROGRAM X(INPUT=20) will cause a buffer of 1001 words to be formed. Equivalenced file names must follow, in the list of parameters, those to which they are made equivalent. Their corresponding parameter positions may not be changed at the time the program is executed, although the names of the files to which they are made equivalent may be changed at this time.

7.6 SUBROUTINE SUBPROGRAM

A subroutine subprogram is a closed loop computational procedure which may return none, one or more values. A value or type is not associated with the subroutine name itself.

The first statement of a subroutine subprogram must have one of the following forms:

SUBROUTINE name (p_1, . . . ,p_n)

FORTRAN IV SUBROUTINE name (p_1, . . . ,p_n)

FORTRAN II SUBROUTINE name (p_1, . . . ,p_n)

MACHINE SUBROUTINE name (p_1, . . . ,p_n) } See

ASCENTF SUBROUTINE name (p_1, ...,p_n) } Appendix F

name is an alphanumeric identifier and p_i are formal parameters; n may be 1 to 60.

The parameter list is optional. If the parameter list is not specified, the following form is allowed:

SUBROUTINE name

7.7 CALL STATEMENT

The executable statement in the calling program for referring to a subroutine is:

CALL name

or

CALL name (p_1, . . . ,p_n)

name is the name of the subroutine being called, and p_i are actual parameters; n is 1 to 60. The name should not appear in any declarative statement in the calling program, with the exception of the EXTERNAL statement when name is also an actual parameter.

The CALL statement transfers control to the subroutine. When a RETURN statement is encountered in the subroutine, control is returned to the next executable statement following the CALL statement in the calling program. If the CALL statement is the last statement in a DO loop, looping continues until the DO loop is satisfied.

Examples:

```
1)    SUBROUTINE BLDX(A,B,W)
      W=2.*B/A
      END
```

Calls

```
CALL BLDX(X(I),Y(I),W)
CALL BLDX(X(I)+H/2. ,Y(I)+C(J),PROX)
CALL BLDX(SIN(Q5),EVEC(I+J),OVEC(L) )
```

2)

```
SUBROUTINE MATMULT
COMMON/ITRARE/X(20,20),Y(20,20),Z(20,20)
DO 10 I = 1,20
DO 10 J = 1,20
Z(I,J) = 0.
DO 10 K=1,20
10 Z(I,J) = Z(I,J) + X(I,K)*Y(K,J)
END
```

Operations in MATMULT are performed on variables contained in the common block ITRARE. This block must be defined in all calling programs.

```
COMMON/ITRARE/AB(20,20),CD(20,20),EF(20,20)
CALL MATMULT
```

3)

```
SUBROUTINE AGMT(SUB,ARG)
COMMON/ABL/XP(100)
ARG = 0.
DO 5 I = 1,100
5 ARG = ARG + XP(I)
CALL SUB
END
```

In this case the subprogram used as an actual parameter must have its name declared in an EXTERNAL statement in the calling program.

```
COMMON/ABL/ALST(100)
EXTERNAL RTEMTA, RTEMTB
CALL AGMT(RTEMTA,V1)
CALL AGMT(RTEMTB,V1)
```

7.8 EXTERNAL STATEMENT

When the actual parameter list which calls a function or subroutine subprogram contains a function or subroutine name, that name must be declared in an EXTERNAL statement.

EXTERNAL $name_1$, $name_2$, . . .

The EXTERNAL statement must precede the first statement of any program which calls a function or subroutine subprogram using the EXTERNAL name. When it is used, EXTERNAL always appears in the calling program; it may not be used with statement functions. If it is, a compiler diagnostic is provided.

7.9 LIBRARY SUBROUTINES

Chippewa FORTRAN contains several built-in subroutine subprograms which may be referenced by any program with a CALL statement. i must be an integer variable or constant; j is an integer variable.

CALL SLITE (i)

Turn on sense light i. If i = 0, turn all sense lights off. i is 0 to 6; if i > 6, the results are undefined and no diagnostic is provided.

CALL SLITET (i,j)

If sense light i is on, set j = 1, if sense light i is off, set j = 2; then turn sense light i off. i is 1 to 6. If i is out of the range, the results are undefined.

CALL SSWTCH (i,j)

If sense switch i is down, set j = 1; if sense switch i is up, set j = 2. i is 1 to 6. If i is out of the range, the results are undefined.

CALL OVERFL (j)

If a floating point overflow condition exists, set j = 1; if no overflow exists, set j = 2; and set the machine to a no overflow condition.

CALL DVCHK (j)

If division by zero occurred, set j = 1 and clear the indicator; if division by zero did not occur, set j = 2.

CALL EXIT

Terminate program execution and return control to the monitor.

CALL DUMP (a_1,b_1,f_1, . . . ,a_n,b_n,f_n)
CALL PDUMP (a_1,b_1,f_1, . . . ,a_n,b_n,f_n) $n \leq 20$

Dump storage on the OUTPUT file in the indicated format. If PDUMP was called, return control to the calling program; if DUMP was called, terminate program execution and return control to the monitor.

a_i and b_i, identifiers or statement numbers, indicate the first word and the last word of the storage area to be dumped. The statement numbers must be 1 to 5 digits trailed by an S; CALL DUMP (10S, 20S, 0).

The dump format indicators are as follows:

f = 0 or 3 octal dump

f = 1 real dump

f = 2 integer dump; if bit 48 is set (normalize bit), the dump is real (f =1).

If no parameters are provided, an octal dump of all storage occurs.

If b_i is the last statement of a DO loop, then b_iS is not allowed to be used as the last word of the storage area to be dumped.

7.10 FUNCTION SUBPROGRAM

A function is a computational procedure which returns a value associated with the function name. The mode of the function is determined by a type indicator or the name of the function.

The first statement of a function subprogram must be one of the following forms where name is an alphanumeric identifier and p_i are formal parameters. A FUNCTION statement must have at least one parameter. $1 \leq n \leq 60$.

FUNCTION name (p_1, . . . ,p_n)

type FUNCTION name (p_1, . . . ,p_n)

FORTRAN IV FUNCTION name (p_1, . . . ,p_n)

FORTRAN IV type FUNCTION name (p_1, . . . ,p_n)

FORTRAN II FUNCTION name (p_1, . . . ,p_n)

FORTRAN II type FUNCTION name (p_1, . . . ,p_n)

Type is REAL, INTEGER, DOUBLE PRECISION, DOUBLE, COMPLEX, or LOGICAL. When the type indicator is omitted, the mode is determined by the first character of the function name.

The name of a function must not appear in a DIMENSION declaration. The name must appear, however, at least once as any of the following:

The left-hand identifier of a replacement statement

An element of an input list

An actual parameter of a subroutine reference

7.11 FUNCTION REFERENCE

In the general form, name identifies the function referenced, it is an alphanumeric identifier, and its type is determined in the same way as a variable identifier. p_i are actual parameters, n is 1 to 60.

name (p_1, . . . ,p_n)

A function reference may appear any place in an expression that an operand may be used. The evaluated function has a single value associated with the function name.

When a function reference is encountered in an expression, control is transferred to the function indicated. When a RETURN statement in the function subprogram is encountered, control is returned to the statement containing the function reference.

Examples:

```
1)      FUNCTION GRATER(A,B)
        IF(A.GT.B)1,2
    1   GRATER=A-B
        RETURN
    2   GRATER=A+B
        RETURN
        END
```

A reference to the function GRATER might be:

```
W(I,J)=FA+FB-GRATER(C-D,3.*AX/BX)
```

2)
```
FUNCTION PHI (ALPHA,PHI2)
PHI=PHI2(ALPHA)
RETURN
END
```

This function can be referenced:

```
EXTERNAL SIN
C=D-PHI(Q(K),SIN)
```

The replacement statement in the function PHI will be executed as if it had been written PHI=SIN(Q(K))

7.12 STATEMENT FUNCTION

A statement function is defined by a single expression and applies only to the program or subprogram containing the definition. The name of the statement function is an alphanumeric identifier; a single value is always associated with the name.

name $(p_1, \ldots, p_n) = E$

p_i are formal parameters and must be simple variables; n is 1 to 60. The expression E may be any arithmetic or logical expression which may contain reference to library functions, statement functions, or function subprograms.

The nonparameter identifiers appearing in the expression have the same values as they have outside the function.

A statement function reference has the form:

name$(p_1, \ldots, p_n)$

name is the name of the statement function; the actual parameters p_i may be any arithmetic expressions.

During compilation, the arithmetic statement function definition is compiled once at the beginning of the program and a transfer is made to this portion of the program whenever a reference is made to the arithmetic statement function.

The statement function name must not appear in a DIMENSION, EQUIVALENCE, COMMON, or EXTERNAL statement; the name can appear in a TYPE declaration but cannot be dimensioned. Statement function names must not appear as actual or formal parameters.

Actual and formal parameters must agree in number, order, and mode. The types of the statement function name or formal parameters is ignored. The mode of the evaluated statement function is determined by the name of the arithmetic statement function. However, the mode of the right-hand expression is determined by the highest mode of the formal parameters of the function.

A statement function must precede the first statement in which it is used, but it must follow all declarative statements (DIMENSION, Type, etc.) which contain symbolic names referenced in the statement function.

Examples:

```
LOGICAL A,B
EQV(A,B)=(A.AND.B).OR.(.NOT.A.AND. .NOT.B)
COMPLEX Z
Z(X,Y)=(1. ,0.)*EXP(X)*COS(Y)+(0. ,1.)*EXP(X)*SIN(Y)
GROPAY (RATE,HRS,OTHRS)=RATE*HRS+RATE*.5*OTHRS
```

7.13 LIBRARY FUNCTIONS

Function subprograms that are used frequently have been stored in a reference library and are available to the programmer through the compiler. Library function references may appear in the main program, subprograms, and statement functions.

Chippewa FORTRAN contains the standard library functions available in earlier versions of FORTRAN. (Appendix C.) The parameter and result type of all library functions is also listed in Appendix C.

7.14 PROGRAM MODES

A Chippewa FORTRAN program or subprogram is compiled in one of four modes:

FORTRAN IV

FORTRAN II

Assembly language (MACHINE) } (Appendix F)

ASCENTF assembly language }

When a mode is not indicated, the program or subprogram is compiled in FORTRAN IV mode.

The compiling mode for the main program is the prevailing mode and is assumed for all subsequent subprograms unless specific subprograms are declared to be

of a different mode. A subprogram declared to be of a different mode is processed in its declared mode. The subprogram following it, unless declared to be of a different mode, is processed in the prevailing mode.

FORTRAN II and FORTRAN IV statements which are not inherently incompatible may be intermixed in a program to be compiled in either mode (Appendix D). Inherently incompatible statements are those involving function subprogram references and EQUIVALENCE statements, causing a reordering of variables in COMMON. However, any standard FORTRAN II or FORTRAN IV library function or subroutine reference may appear in a program to be compiled in either mode.

7.15 VARIABLE DIMENSIONS IN SUBPROGRAMS

In many subprograms, especially those performing matrix manipulation, the programmer may wish to vary array dimensions each time the subprogram is called.

This is accomplished by specifying the array name and its dimensions as formal parameters in the FUNCTION or SUBROUTINE statement. The corresponding actual parameters specified in the calling program are used by the called subprogram. The maximum dimensions that any given array may assume are determined by dimensions in a DIMENSION, COMMON, or type statement in the calling program at compile time.

The formal parameters representing the array dimensions must be simple integer variables. The array name must also be a formal parameter. The actual parameters representing the array dimensions must have integer values.

The total number of elements of the corresponding array in the subprogram may not exceed the total number of elements of a given array in the calling program.

Example:

Consider a simple matrix add routine written as a subroutine:

```
      SUBROUTINE MATADD (X,Y,Z,M,N)
      DIMENSION X (M,N),Y(M,N),Z(M,N)
      DO 10 I = 1,M
      DO 10 J = 1,N
10    Z(I,J)=X(I,J)+Y(I,J)
      END
```

The arrays X, Y, Z and the variable dimensions M, N must all appear as formal parameters in the SUBROUTINE statement and also in the DIMENSION

statement as shown. If the calling program contains the array allocation declaration

```
DIMENSION A(10,10),B(10,10),C(10,10),E(5,5),F(5,5),G(5,5),H(10,10)
```

the program may call the subroutine MATADD from several places within the main program as follows:

```
CALL MATADD(A,B,C,10,10)
CALL MATADD(E,F,G,5,5)
CALL MATADD(B,C,A,10,10)
CALL MATADD(B,C,H,10,10)
```

The compiler does not check to see if the limits of the array established by the DIMENSION statement in the main program are exceeded.

7.16 PROGRAM ARRANGEMENT

Chippewa FORTRAN assumes that all statements and comments appearing between a PROGRAM, SUBROUTINE, or FUNCTION statement and an END statement belong to one program. A typical arrangement of a set of main program and subprograms follows.

```
PROGRAM          WHAT
    .
    .
    .
END
FORTRAN II       SUBROUTINE S1(A,B)
    .
    .
    .
END
FORTRAN IV       SUBROUTINE S2
    .
    .
    .
END
REAL FUNCTION F1(P1)
    .
    .
    .
END
```

CHAINING 8

8.1 CHAINING

Chaining is a method used to execute a program which would otherwise exceed available storage. The program is separated into a main program with or without subprograms and any number of segments which may be called and executed as needed. Each segment must begin with the statement:

SEGMENT name (f_1, . . . ,f_n)

name is an alphanumeric identifier and transfer address for the segment; f_i are file names which appear in the PROGRAM statement in the main program.

The main program is the first portion loaded and executed; segments are loaded from the disk file when called. Only main or one segment may occupy storage at a given time. The main program may not be recalled once a segment has been called but a segment may be called for execution more than once. The main program must declare the largest value of common used by any one segment. Parameters may be transferred to a segment only through blank common. A subprogram beginning with the SEGMENT statement is compiled as one beginning with the PROGRAM statement except that blank common is not cleared when starting execution of the segment.

Segments are called by

CALL CHAIN (name)

CHAIN is a central FORTRAN subroutine which loads the called segment from the disk file. The only parameter to CHAIN must be the segment name to which control is transferred after loading.

See Appendix F for basic deck structure.

INPUT/OUTPUT FORMATS 9

Data transmission between storage and external units requires the FORMAT statement (BCD only) and the I/O control statement (chapter 10). The I/O statement specifies the input/output device and process READ, WRITE, etc., and a list of data to be moved. The FORMAT statement specifies the manner in which the data is to be moved. In binary statements no FORMAT statement is used.

9.1 INPUT/OUTPUT LIST

The list portion of an input/output statement indicates the data items and the order, from left to right, of transmission. The input/output list can contain any number of elements; list items may be array names, simple or subscripted variables, or an implied DO loop. Items are separated by commas, and their order must correspond to any FORMAT specification associated with the list. External records are always read or written until the list is satisfied.

Subscripts in an I/O list may be in the following forms:

(c*I ± d)

(I ± d)

(c*I)

(I)

(c)

c and d are unsigned integer constants, and I is a simple integer variable, previously defined, or defined within an implied DO loop.

Examples:

```
READ 100, A,B,C,D
READ 200, A,B,C(I),D(3,4),E(I,J,7),H
READ 101, J,A(J),I,B(I,J)
READ 102, DELTA(5*J+2,5*I-3,5*K),C,D(I+7)
READ 202, DELTA
READ 300, A,B,C,(D(I),I=1,10),E(5,7),F(J),(G(I),H(I),I=2,6,2)
READ 400, I,J,K,( ( (A(II,JJ,KK),II=1,I),JJ=1,J),KK=1,K)
READ 500, ( (A(I,J),I=1,10,2),B(J,1),J=1,5),E,F,G,(L+5,M-7)
```

9.1.1 ARRAY TRANSMISSION

Part of all of an array can be represented for transmission as a single I/O list item by using an implied DO notation in the form:

$$(((A(I,J,K),L_1=m_1,m_2,m_3),L_2=n_1,n_2,n_3),L_3=p_1,p_2,p_3)$$

m_i,n_i,p_i	unsigned integer constants or simple integer variables. If m_3, n_3, or p_3 is omitted, it is assumed equal to 1.
I,J,K	subscripts of A.
L_1,L_2,L_3	index variables I, J, K in some order.

During execution, each subscript (index variable) is set to the initial index value: $L_1 = m_1$, $L_2 = m_1$, $L_3 = p_1$. The first index variable defined in the list is incremented first, following the rules for a DO loop execution. When the first index variable reaches the maximum value, it is reset; the next index variable to the right is incremented, and the process is repeated until the last index variable has been incremented. If m_1 is greater than m_2 initially, one card is read.

An array name which appears without subscripts in an I/O list causes transmission of the entire array by columns.

An implied DO loop can be used to transmit a simple variable more than one time. For example, the list item (A(K),B,K=1,5) causes the transmission of variable B five times. However, in the case of (B,A(K),K=1,5), B is transmitted only once. A list of the form K,(A(I),I=1,K) is permitted and the input value of K is used in the implied DO loop.

Examples:

1) Simple implied DO loop list items.

```
    READ 400,(A(I),I=1,10)
400 FORMAT (E20.10)
```

This statement is equivalent to the following DO loop.

```
    DO 5 I=1,10
  5 READ 400, A(I)
```

```
    READ 100, ( (A(JV,JX),JV=2,20,2),JX=1,30)
```

```
    READ 200, (BETA(3*JON+7),JON = JONA,JONB,JONC)
```

```
    READ 300, ( ( (ITMLST(I,J+1,K-2),I=1,25),J=2,N),K=IVAR,IVMAX,4)
```

```
    READ 600, (A(I),B(I),I=1,10)
600 FORMAT (F10.2, E6.1)
```

The previous statement is equivalent to the DO loop.

```
    DO 17 I = 1,10
 17 READ 600, A(I),B(I)
```

2) Nested implied DO list items.

```
READ 100,((((A(I,J,K)B(I,L),C(J,N),I=1,10),J=1,5),K=1,8),
1L=1,15),N=2,7)
```

Data is transmitted in the following sequence:

```
A(1,1,1),B(1,1),C(1,2),A(2,1,1),B(2,1),C(1,2). . .
. . .A(10,1,1),B(10,1),C(1,2),A(1,2,1),B(1,1),C(2,2). . .
. . .A(10,2,1),B(10,1),C(2,2). . .A(10,5,1),B(10,1),C(5,2). . .
. . .A(10,5,8),B(10,1),C(5,2). . .A(10,5,8),B(10,15),C(5,2). . .
. . .A(10,5,8),B(10,15),C(5,7)
```

The following list item will transmit the array E(3,3) by columns:

```
READ 100,((E(I,J),I=1,3),J=1,3)
```

The following list item will transmit the array E(3,3) by rows:

```
READ 100,((E(I,J),J=1,3), I=1,3)
```

3)

```
DIMENSION MATRIX(3,4,7)
READ 100, MATRIX
100 FORMAT (I6)
```

The above items are equivalent to the following statements:

```
DIMENSION MATRIX(3,4,7)
READ 100,(((MATRIX(I,J,K),I=1,3),J=1,4),K=1,7)
```

The list is equivalent to the nest of DO loops:

```
DO 5 K=1,7
DO 5 J=1,4
DO 5 I=1,3
5 READ 100, MATRIX(I,J,K)
```

9.2 FORMAT DECLARATION

BCD input/output statements require a FORMAT declaration which contains conversion and editing information relating to internal/external structure of the corresponding I/O list items. A FORMAT declaration has the following form:

FORMAT ($spec_1$, . . . ,k($spec_m$, . . .),$spec_n$, . . .)

$Spec_i$ format specification

k optional repetition factor which must be an unsigned integer constant.

The FORMAT declaration is non-executable and may appear anywhere in the program. FORMAT declarations must have a statement label in columns 1-5.

The data items in an I/O list are converted from one representation to another (external/internal) according to FORMAT conversion specifications. FORMAT specifications may also contain editing codes.

Conversion specifications:

Ew.d	Single precision floating point with exponent
Fw.d	Single precision floating point without exponent
Dw.d	Double precision floating point with exponent
Iw	Decimal integer conversion
Ow	Octal integer conversion
Aw	Alphanumeric conversion
Lw	Logical conversion
nP	Scaling factor

Complex data items are converted on input/output according to a pair of consecutive Ew.d or Fw.d specifications.

Example:

```
      COMPLEX A,B
      PRINT 10,A
10    FORMAT (F7.2,F9.2)
      READ 11,B
11    FORMAT (E10.3,E10.3)
```

Editing specifications:

wX	Intraline spacing
wH	Heading and labeling
/	Begin new record

Both w and d are unsigned integer constants; w specifies the field width in number of character positions in the external record, and d specifies the number of digits to the right of the decimal within the field.

9.3 CONVERSION SPECIFICATIONS

9.3.1 Ew.d OUTPUT

Real numbers in storage are converted to the BCD character form for output with the E conversion. The field occupies w positions in the output record; with the real number right justified in the form:

ba.a. . .a±eee	$100 \leq eee \leq 308$
or	
ba.a. . .aE±ee	$0 \leq ee \leq 99$

b indicates blank character. a are the most significant digits of the integer and fractional part and eee are the digits in the exponent. If d is zero or blank, the decimal point and digits to the right of the decimal do not appear as shown above. Field w must be wide enough to contain the significant digits, signs, decimal point, E, and the exponent. Generally, $w \geq d+7$. Positive numbers need not reserve a space for the sign of the number.

If the field is not wide enough to contain the output value, an asterisk is inserted in the high order position of the field. If the field is longer than the output value, the quantity is right justified with blank fill to the left.

Examples:

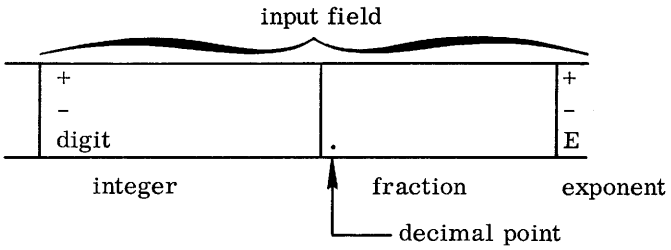
Ew.d Output

```
    PRINT 10,A                          A contains -67.32
10  FORMAT(E10.3)                             or  +67.32
          Result:  -6.732E+01 or b6.732E+01

    PRINT 10,A
10  FORMAT(E13.3)
          Result:  bbb-6.732E+01        bbbb6.732E+01

    PRINT 10,A                          A contains -67.32
10  FORMAT(E9.3)                        provision not made
          Result:  *.732E+01            for sign

    PRINT 10,A
10  FORMAT(E10.4)
          Result:  *.6732E+02
```

9.3.2 Ew.d INPUT

The E specification converts the number in the input field to a real number and stores it in the proper location.

Subfield structure of the input field:



The total number of characters in the input field is specified by w; this field is scanned from left to right; blanks are interpreted as zeros. A field may contain up to 15 significant digits.

The integer subfield begins with a sign (+ or -) or a digit and may contain a string of digits. The integer field is terminated by a decimal point, D, E, +, -, or the end of the input field.

The fraction subfield which begins with a decimal point may contain a string of digits. The field is terminated by D, E, +, -, or the end of the input field.

The exponent subfield may begin with D, E, + or -. When it begins with D or E, the + is optional between D or E and the string of digits of the subfield. The value of the string of digits in the exponent subfield must be less than 308.

Permissible subfield combinations:

+1.6327E-04	integer fraction exponent
-32.7216	integer fraction
+328+5	integer exponent
.629E-1	fraction exponent
+136	integer only
136	integer only
.07628431	fraction only
E-06 (interpreted as zero)	exponent only

In the Ew.d specification, d acts as a negative power-of-ten scaling factor when an external decimal point is not present. The internal representation of the input quantity is:

$$(\text{integer subfield}) \times 10^{-d} \times 10^{(\text{exponent subfield})}$$

For example, if the specification is E7.8, the input quantity 3267+05 is converted and stored as: $3267x10^{-8}x10^{5} = 3.267$.

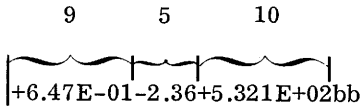
A decimal point in the input field overrides d. The input quantity 3.67294+5 read by an E9.d specification is always stored as $3.6729x10^{5}$. When d does not appear, it is assumed to be zero.

The field length specified by w in Ew.d should always be the same as the length of the field containing the input number. When it is not, incorrect numbers may be read, converted, and stored as shown below. The field w includes the significant digits, signs, decimal point, E or D, and exponent.

Example:

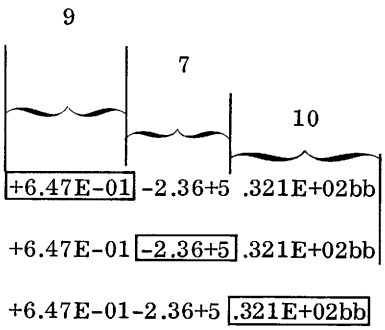
```
      READ 20,A,B,C
20    FORMAT (E9.3,E7.2,E10.3)
```

Input quantities on the card are in three contiguous fields columns 1 through 24:



The second specification (E7.2) exceeds the width of the second field by two characters.

Reading proceeds as follows:



First, +6.47-01 is read, converted, and placed in location A. Next, -2.36+5 is read, converted, and placed in location B. The number actually desired was -2.36, but the specification error (E7.2 instead of E5.2) caused the two extra characters to be read. The number read (-2.36+5) is a legitimate input representation under the definitions and restrictions.

Finally, .321E+0200 is read, converted, and placed in location C. Here again, the input number is legitimate and is converted and stored, even though it is not the number desired.

The above example illustrates a situation where numbers are incorrectly read, converted, and stored, and yet there is no immediate indication that an error has occurred.

Examples:

Ew.d Input Field	Specification	Converted Value	Remarks
+143.26E-03	E11.2	.14326	All subfields present
-12.437629E+1	E13.6	-124.37629	All subfields present
8936E+004	E9.10	.008936	No fraction subfield; input number converted as 8936. x 10^{-10+4}
327.625	E7.3	327.625	No exponent subfield
4.376	E5	4.376	No d in specification
-.0003627+5	E11.7	-36.27	Integer subfield contains - only
-.0003627E5	E11.7	-36.27	Integer subfield contains - only
blanks	Ew.d	-0.	All subfields empty
1E1	E3.0	10.	No fraction subfield; input number converted as $1.x10^{1}$
E+06	E10.6	0.	No integer or fraction subfield; zero stored regardless of exponent field contents
1.bEb1	E6.3	10.	Blanks are interpreted as zeros

9.3.3 Fw.d OUTPUT

The field occupies w positions in the output record; the corresponding list item must be a floating point quantity, which appears as a decimal number, right justified:

ba. . .a.a. . .a

b indicates a blank. The a's represent the most significant digits of the number. The number of decimal places to the right of the decimal is specified by d. If d is zero or omitted, the decimal point and digits to the right do not appear. If the number is positive, the + sign is suppressed. If the field is too short to accommodate the number, one asterisk appears in the high-order position of the output field. If the field is longer than required to accommodate the number, the number is right justified with blank fill to the left.

Examples:

A contains +32.694

```
    PRINT 10,A
10  FORMAT (F7.3)
```

Result: b32.694

```
    PRINT 11,A
11  FORMAT (F10.3)
```

Result: bbbb32.694

A contains -32.694

```
    PRINT 12,A
12  FORMAT (F6.3)
```

Result: *2.694 — no provision for - sign and most significant digit

A contains .32694

```
    PRINT 13,A,A
13  FORMAT (F4.3, F6.3)
```

Result: .327b0.327

9.3.4 Fw.d INPUT

This specification is a modification of Ew.d. The input field consists of an integer and a fraction subfield. An omitted subfield is assumed to be zero. The restrictions described under Ew.d input apply.

Examples:

Input Field	Specification	Converted Value	Remarks
367.2593	F8.4	367.2593	Integer and fraction field
37925	F5.7	.0037925	No fraction subfield; input number converted as 37925×10^{-7}
-4.7366	F7	-4.7366	No d in specification
.62543	F6.5	.62543	No integer subfield
.62543	F6.2	.62543	Decimal point overrides d of specification
+144.15E-03	F11.2	.14415	Exponents are legitimate in F input and may have P-scaling
5bbbb	F5.2	500.00	No fraction subfield; input number converted as 50000×10^{-2}

9.3.5 Dw.d OUTPUT

The field occupies w positions of the output record, the list item is a double precision quantity which appears as a decimal number, right justified:

ba.a. . .a±eee $100 \leq eee \leq 308$

or

ba.a. . .aD±ee $0 \leq ee \leq 99$

b indicates blank. D conversion corresponds to Ew.d Output.

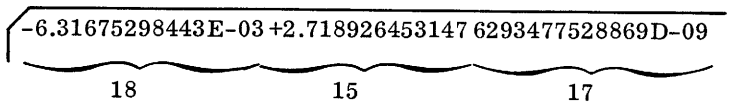
9.3.6 Dw.d INPUT

D conversion corresponds to E conversion except that 18 is the maximum number of significant digits permitted in the combined integer-fraction field. D is acceptable in place of E as the beginning of an exponent subfield.

Example:

```
      DOUBLE Z,Y,X
      READ1,Z,Y,X
1     FORMAT (D18.11,D15,D17.4)
```

Input Card:



9.3.7 Iw OUTPUT

I specification is used to output decimal integer values. The output quantity occupies w output record positions, right justified:

ba. . .a

b is a blank. The a's are the most significant decimal digits (maximum 18) of the integer. If the integer is positive, the + sign is suppressed.

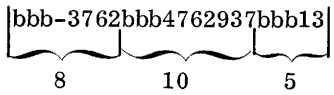
If the field w is larger than required, the output quantity is right justified with blank fill to the left. If the field is too short, characters are stored from the right, an asterisk occupies the leftmost position.

Example:

```
      PRINT 10,I,J,K
10    FORMAT (I8,I10,I5)
```

I contains -3762
J contains +4762937
K contains +13

Result:



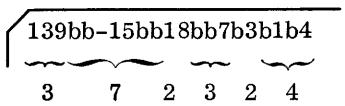
9.3.8 Iw INPUT

The field is w characters in length, and the list item is a decimal integer constant. The input field w consists of an integer subfield, containing +, -, 0 through 9, or blank. When a sign appears, it must precede the first digit in the field. Blanks are interpreted as zeros. The value is stored right justified in the specified variable.

Example:

```
      READ 10,I,J,K,L,M,N
10    FORMAT (I3,I7,I2,I3,I2,I4)
```

Input Card:



In storage:

I contains 139

J	-1500
K	18
l	7
M	3
N	104

9.3.9 Ow OUTPUT

O specification is used to output octal integer values. The output quantity occupies w output record positions right justified:

aa. . .a

The a's are octal digits. If w is 20 or less, the rightmost w digits appear. If w is greater than 20, the number is right justified in the field with blanks to the left of the output quantity. A negative number is output in its one's complement internal form.

9.3.10 Ow INPUT

Octal integer values are converted under O specification. The field is w characters in length, and the list item must be an integer variable.

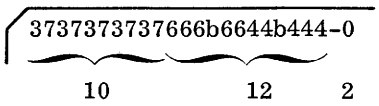
The input field w consists of an integer subfield only (maximum of 20 octal digits) containing +, -, 0 through 7 or blank.

Only one sign may precede the first digit in the field. Blanks are interpreted as zeros.

Example:

```
      TYPE INTEGER P,Q,R
      READ 10,P,Q,R
10    FORMAT (O10,O12,O2)
```

Input Card:



In storage:

P 0000000003737373737
Q 0000000666066440444
R 7777777777777777777
A negative number is represented in one's complement form.

A negative octal number is represented internally in seven's complement form (20 digits) obtained by subtracting each digit of the octal number from seven. For example, if -703 is an input quantity, its internal representation is 77777777777777777074.

That is, 77777777777777777777
-00000000000000000703
77777777777777777074

9.3.11 Aw OUTPUT

A conversion is used to output alphanumeric characters. If w is 10 or more, the quantity appears right justified in the output field, blank fill to left. If w is less than 10, the output quantity is represented by leftmost w characters.

9.3.12 Aw INPUT

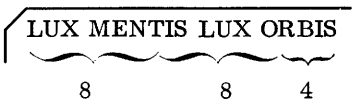
This specification accepts FORTRAN characters including blanks. The internal representation is 6000 Series display code; the field width is w characters.

If w exceeds 10, the input quantity is the rightmost 10 characters in the field. If w is 10 or less, the input quantity is stored as a left justified BCD word; the remaining spaces are blank filled.

Example:

```
      READ 10,Q,P,O
10    FORMAT (A8,A8,A4)
```

Input Card:



In storage:

```
Q   LUXbMENTbb
P   ISbLUXbObb
O   RBISbbbbbb
```

9.3.13 Rw OUTPUT

This specification is similar to the Aw Output with the following exception. If w is less than 10, the output quantity represents the rightmost characters.

9.3.14 Rw INPUT

This specification is the same as the Aw Input with the following exception. If w is less than 10, the input quantity is stored as a right justified binary zero filled word.

Example:

```
      READ 10,Q,P,O
10    FORMAT (R8,R8,R4)
```

Input Card:

In storage:

```
Q   00LUXbMENT
P   00ISbLUXbO
O   000000RBIS
```

9.3.15 Lw OUTPUT

L specification is used to output logical values. The output field is w characters long, and the list item must be a logical element.

A value of TRUE or FALSE in storage causes w-1 blanks followed by a T or F to be output.

Example:

```
      LOGICAL I,J,K,L          I contains -0     J contains  0
      PRINT 5,I,J,K,L          K contains -0     L contains -0
5     FORMAT (4L3)
```

Result: bbTbbFbbTbbT

9.3.16 Lw INPUT

This specification accepts logical quantities as list items. The field is considered true if the first nonblank character in the field is T or false if it is F. An all-blank field is considered false.

9.4 nP SCALE FACTOR

The D, E, and F conversion may be preceded by a scale factor which is: External number = Internal number x$10^{\text{scale factor}}$. The scale factor applies to Fw.d on both input and output and to Ew.d and Dw.d on output only. A scaled specification is written as shown below; n is a signed integer constant.

nPDw.d

nPEw.d

nPFw.d

nP

The scale factor is assumed to be zero if no other value has been given; however, once a value has been given, it holds for all D, E, and F specifications following the scale factor within the same FORMAT declaration. To nullify this effect in subsequent D, E, and F specifications, a zero scale factor, OP, must precede a D, E, or F specification. Scale factors for D, E, and F output specifications must be in the range $-8 \leq n \leq 8$.

Scale factors on D or E input specifications are ignored.

The scaling specification nP may appear independently of a D, E, or F specification; it holds for all subsequent D, E, and F specifications within the same FORMAT statement unless changed by another nP.

The specification (3P,3I9,F10.2) is the same as the specification (3I9,3PF10.2).

9.4.1 Fw.d SCALING

Input

The number in the input field is divided by 10^n and stored. For example, if the input quantity 314.1592 is read under the specification 2PF8.4, the internal number is $314.1592 \times 10^{-2} = 3.141592$.

Output

The number in the output field is the internal number multiplied by 10^n. In the output representation, the decimal point is fixed; the number moves to the left or right, depending on whether the scale factor is plus or minus. For example, the internal number 3.145926538 may be represented on output under scaled F specifications as follows:

Specification	Output Representation
F13.6	3.141593
1PF13.6	31.415927
3PF13.6	3141.592654
-1PF13.6	.314159

9.4.2 Ew.d OR Fw.d SCALING

Output

The scale factor has the effect of shifting the output number left n places while reducing the exponent by n. Using 3.1415926538, some output representations corresponding to scaled E specifications are:

Specification	Output Representation
E20.2	3.14 E+00
1PE20.2	31.42 E-01
2PE20.2	314.16 E-02
3PE20.2	3141.59 E-03
4PE20.2	31415.93 E-04
5PE20.2	314159.27 E-05
-1PE20.2	0.31 E+01

9.5 EDITING SPECIFICATIONS

9.5.1 wX

This specification may be used to include w blanks in an output record or to skip w characters on an input record to permit spacing of input/output quantities. 0X is not permitted; bX is interpreted as 1X. In the specification list, the comma following X is optional.

Examples:

```
      INTEGER A                          A contains 7
                                         B contains 13.6
      PRINT 10, A, B, C                  C contains 1462.37
10    FORMAT (I2, 6X, F6.2, 6X, E12.5)
            Result: b7bbbbbbb13.60bbbbbbb1.46237E+03
      READ 11, R, S, T
11    FORMAT (F5.2, 3X, F5.2, 6X, F5.2)
         or
11    FORMAT (F5.2, 3XF5.2, 6XF5.2)
```

Input Card:

```
14.62bb$13.78bCOSTb15.97
```

In storage:

```
R  14.62
S  13.78
T  15.97
```

9.5.2 wH OUTPUT

With this specification 6-bit characters, including blanks may be output in the form of comments, titles, and headings. w, an unsigned integer, specifies the number of characters to the right of H that are transmitted to the output record; w may specify a maximum of 136 characters. H denotes a Hollerith field; the comma following H is optional.

Examples:

Source program:

```
    PRINT 20
20  FORMAT (28HbBLANKSbCOUNTbINbANbHbFIELD.)
```

produces output record:

bBLANKSbCOUNTbINbANbHbFIELD.

Source program:

```
    PRINT 30, A                    A contains 1.5, comma is optional
30  FORMAT (6HbLMAX=,F5.2)
```

produces output record:

bLMAX = b1.50

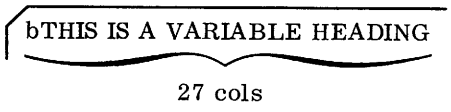
The H specification may be used to read Hollerith characters into an existing H field within the FORMAT specification.

Example:

Source program:

```
    READ 10
10  FORMAT (27Hbbbbbbbbbbbbbbbbbbbbbbbbbbb)
```

Input Card:



After READ, the FORMAT statement labeled 10 contains the alphanumeric information read from the input card; a subsequent reference to statement 10 in an output statement acts as follows:

```
PRINT 10
```

produces the print line:

bTHIS IS A VARIABLE HEADING

9.5.4 NEW RECORD

The slash (/) signals the end of a record anywhere in the specifications list. Consecutive slashes may appear in a list and they need not be separated from the other list elements by commas. During output, the slash is used to skip lines, cards, or tape records. During input, it specifies that control passes to the next record or card. K(/) results in K-1 lines being skipped.

Examples:

```
1)      PRINT 10
    10  FORMAT (6X, 7HHEADING/ / /3X, 5HINPUT, 2X, 6HOUTPUT)
```

Printout:

```
    HEADING _____________________ line 1
            ______ (blank) ______ line 2
            ______ (blank) ______ line 3
INPUTbbOUTPUT_____________________ line 4
```

Each line corresponds to a BCD record. The second and third records are null and produce the line spacing illustrated.

```
2)      PRINT 11, A, B, C, D
    11  FORMAT (2E10.2/2F7.3)
```

In storage:

```
A   -11.6
B   .325
C   46.327
D   -14.261
```

Printout:

```
b-1.16E+01bb3.25E-01
b46.327-14.261
```

```
3)      PRINT 11, A,B,C,D
    11  FORMAT (2E10.2/ /2F7.3)
```

Printout:

```
b-1.16E+01bb3.25E-01 _________ line 1
                   — (blank) — line 2
b46.327-14.261 _______________ line 3
```

```
4)      PRINT 15, (A(I), I=1, 9)
    15  FORMAT (8HbRESULTS2(/)  (3F8.2) )
```

Printout:

```
RESULTS _____________________________________ line 1
                                 _____ (blank)_____ line 2
  3.62     -4.03     -9.78     _____________________ line 3
 -6.33      7.12      3.49     _____________________ line 4
  6.21     -6.74     -1.18     _____________________ line 5
```

9.6 REPEATED FORMAT SPECIFICATIONS

FORMAT specifications may be repeated by using an unsigned integer constant repetition factor, k, as follows: k(spec), spec is any conversion specification except nP. For example, to print two quantities K, L:

```
    PRINT 10K,L
10  FORMAT (I2,I2)
```

Specifications for K, L are identical; the FORMAT statement may also be:

```
10  FORMAT (2I2)
```

When a group of FORMAT specifications repeats itself as in: FORMAT (E15.3, F6.1,I4,I4,E15.3,F6.1,I4,I4), the use of k produces: FORMAT (2(E15.3,F6.1, 2I4))

9.6.1 UNLIMITED GROUPS

FORMAT specifications may be repeated without using a repetition factor. The innermost parenthetical group that has no repetition factor is unlimited and will be used repeatedly until the I/O list is exhausted. Parentheses are the controlling factors in repetition. The right parenthesis of an unlimited group is equivalent to a slash. Specifications to the right of an unlimited group can never be reached, as in the following example:

Format specifications for output data:

```
(E16.3,F20.7,2(I4),(I3,F7.1),F8.2)
```

The first two fields are printed according to E16.3 and F20.7. Since 2(I4) is a repeated parenthetical group, the next two fields are printed according to I4 format. The remaining print fields follow (I3,F7.1), which does not have a repetition factor, until the list elements are exhausted. F8.2 is never reached.

9.7 VARIABLE FORMAT

FORMAT specifications may be specified at the time of program execution. The specification, including left and right parentheses but not the statement label or the word FORMAT, is read under A conversion or in a DATA statement and stored in an integer array. The name of the array containing the specifications may be used in place of the FORMAT statement labels in the associated input/output operation. The array name that appears without subscript specifies the location of the first word of the FORMAT information.

Examples:

1. Assume the following FORMAT specifications:

```
(E12.2,F8.2,I7,2E20.3,F9.3,I4)
```

This information can be punched in an input card and read by the statements of the program such as:

```
      DIMENSION IVAR(3)
      READ 1 (IVAR(I),I=1,3)
1     FORMAT (3A10)
```

The elements of the input card are placed in storage as follows:

```
IVAR(1):    (E12.2,F8.
IVAR(2):    2,I7,2E20.
IVAR(3):    3,F9.3,I4)
```

A subsequent output statement in the same program can refer to these FORMAT specifications as:

```
      PRINT IVAR, A, B, I, C, D, E, J
```

This produces exactly the same result as the program:

```
      PRINT 10, A, B, I, C, D, E, J
10    FORMAT (E12.2,F8.2,I7,2E20.3,F9.3,I4)
```

2.
```
DIMENSION LAIS1(3),LAIS2(2),A(6),LSN(3),TEMP(3)
DATA LAIS1/21H(2F6.3,I7,2E12.2,3I1)/LAIS2/20H(I6,6X,3F4.1,2E12.2)/
```

Output statement:

```
      PRINT LAIS1,(A(I),I=1,2),K,B,C,(LSN(J),J=1,3)
```

which is the same as:

```
      PRINT 1,(A(I),I=1,2),K,B,C,(LSN(J),J=1,3)
1     FORMAT (2F6.3, I7, 2E12.2, 3I1)
```

Output statement:

```
PRINT LAIS2,LA,(A(M),M=3,4),A(6),(TEMP(I),I=2,3)
```

which is the same as:

```
PRINT 2,LA,(A(M),M=3,4),A(6),(TEMP(L),L=2,3)
2   FORMAT (I6, 6X,3F4.1,2E12.2)
```

3. DIMENSION LAIS (3), VALUE(6)

```
DATA LAIS/26H(I3,13HMEANbVALUEbIS,F6.3)/
```

Output statement:

```
WRITE (10,LAIS)NUM,VALUE(6)
```

which is the same as:

```
WRITE (10,10)NUM,VALUE(6)
10  FORMAT (I3,13HMEANbVALUEbIS,F6.3)
```

INPUT/OUTPUT STATEMENTS 10

The following definitions apply to all I/O statements:

i	logical I/O unit number. i can be either an integer constant of one or two digits. (The first digit must not be a zero.) integer variable with a value from 1 to 99.
n	FORMAT declaration identifier as follows: statement number variable identifier which references the starting storage location of FORMAT information.
name	name of NAMELIST record
L	I/O list

10.1 OUTPUT STATEMENTS

PRINT n,L

Information is transferred from the storage locations in the list (L) to the standard output unit. Information is transferred as line printer images, 136 characters or less per line in accordance with the FORMAT declaration, n. The maximum record length is 136 characters, but the first character of every record is not printed as it is used for carriage control when printing on-line. Characters in excess of the print line appear on the succeeding line. Each new record starts a new print line.

Character	Action
Blank or any character other than 0, 1, +	Single-space after printing
0	Double-space after printing
1	Eject page before printing; followed by a single-space after printing
+	Suppress spacing after printing; print two successive records on the same line

The characters 0, 1, + , are not printed.

For off-line printing, the printer control is determined by the installation's printer routine.

PUNCH n,L

Information is transferred from the storage locations given by the list (L) identifiers to the standard punch unit. Information is transferred as Hollerith images, 80 characters or less per card in accordance with the FORMAT declaration, n.

WRITE (i,n)L

WRITE OUTPUT TAPE i,n,L

These forms are equivalent; they transfer information from storage locations given by the list (L) to a specified output unit (i) according to the FORMAT declaration (n).

With a half inch tape unit, a logical record containing up to 120 characters is recorded in even parity (BCD mode). Each logical record is one physical record. The number of words in the list and the FORMAT declaration determine the number of records that are written on a unit. If the logical record is less than 136 characters, the remainder of the record is filled with blanks (to the nearest multiple of ten characters).

With a one-inch tape unit, a packed 5120-character physical record is recorded in odd parity. Each physical record consists of as many logical record characters as required to fill the physical record. The information is recorded in 6000 series display code with no special control characters added, and it represents a continuous stream of logical output records. Trailing blanks on each logical record are removed and two consecutive characters with a value of zero separate logical records on the tape.

If the tape is to be printed, the first character of a record is not printed as it is a printer control. If the programmer fails to allow for a printer control character, the first character of the output data is lost on the printed listing.

WRITE (i)L

WRITE TAPE i,L

These equivalent forms transfer information from storage locations given by the list (L) to a specified output unit (i). If L is omitted, the WRITE (i) statement acts as a do-nothing statement. See READ (i)L.

The number of words in the list determines the number of physical records that are written on that unit. A physical record contains a maximum of 512 central storage words. The last physical record may contain from 1 to 512 words. The physical records written by one WRITE (i)L statement constitute one logical record. The information is recorded in odd parity (binary mode).

A logical record which is an exact multiple of 512 words is followed by a physical record of eight zero characters called a zero length record.

Examples:

```
      DIMENSION A(260), B(4000)
      WRITE(10)A,B
      DO 5 I = 1,10
5     WRITE TAPE 6 AMAX (I), (M(I,J),J=1,5)
      PRINT 50,A,B,C(I,J)
50    FORMAT (X 8HMINIMUM=F17.7,2X8HMAXIMUM=F17.7,
     2X10HVALUE IS $F8.2)
      PRINT 51, (A(I), I=1,20)
51    FORMAT(X23HTRUTH MATRIX VALUES ARE/(3X4L3))
      PUNCH 52,ACCT,LSTNME,FSTNME,TELNO,SHPDTE,ITMNO
52    FORMAT (I8,3X4A10,2XI10,XI5,F8.2)
```

The above format assumes the following dimension statement:

```
      DIMENSION LISTNME(2),FSTNME(2)
      WRITE (2,53)A,B,C,D
53    FORMAT (4E21.9)
      WRITE OUTPUT TAPE 2,52,A,B,C,D
      WRITE (2,54)
54    FORMAT (32HTHIS STATEMENT HAS NO DATA LIST.)
```

10.2 READ STATEMENTS

READ n,L

One or more card images are read from the standard input unit. Information is converted from left to right in accordance with FORMAT specification (n), and it is stored in the locations named by the list (L). Input may be on 80-column Hollerith cards or magnetic tapes prepared off-line, containing 80-character records in BCD mode.

Example:

```
      READ 10,A,B,C
10    FORMAT (3F10.4)
```

READ (i,n)L

READ INPUT TAPE i,n,L

These equivalent forms transfer one logical record of information from logical unit (i) to storage locations named by the list (L), according to FORMAT specification (n).

The number of words in the list and the FORMAT specifications must conform to the record structure on the logical unit.

READ (i)L

READ TAPE i,L

These equivalent forms transfer one logical record of information from a specified unit (i) to storage locations named by the list (L).

Records to be read by READ (i) should be written in binary mode. The number of words in the list of READ (i)L must not exceed the number of words in the corresponding WRITE statement.

If L is omitted, READ (i) spaces over one logical record. See WRITE (i)L.

Examples:

```
1)        DIMENSION C(264)
          READ (10)C
          DIMENSION BMAX (10), M2 (10,5)
          DO7I=1, 10
      7   READ TAPE 6, BMAX (I), (M2(I,J),J=1,5)
          READ (5) (skip one logical record on unit 5)
          READ (6) ((A(I,J),I=1,100),J=1, 50)
          READ TAPE 6,((A(I,J),I=1,100),J=1, 50)
```

```
2)      READ INPUT TAPE 10,50,X,Y,Z
   50   FORMAT (3F10.6)
        DOUBLE PRECISION DB(4)
        READ (10,51) DB
   51   FORMAT (4D20.12)
        READ 51,DB
        READ (2,52) (Z(J),J=1,8)
   52   FORMAT (F10.4)
```

10.3 TAPE HANDLING STATEMENTS

REWIND i

Magnetic tape unit i is rewound to load point. If the tape is already rewound, the statement acts as a do-nothing statement.

BACKSPACE i

Magnetic tape unit i is backspaced one logical record. (A logical record is a physical record, except for tape written by a WRITE (i)L statement.) If tape is at load point (rewound), this statement acts as a do-nothing statement.

END FILE i

An end-of-file is written on magnetic tape unit i.

IF (ENDFILE i)n_1,n_2

IF (EOF,i)n_1,n_2

These statements check the previous read operation to determine if an end-of-file has been encountered on unit i. If so, control is transferred to statement n_1; if not, control is transferred to statement n_2.

IF (IOCHECK,i)n_1,n_2

This statement checks for end-of-file and parity errors on previous input operations. If either condition occurred, control transfers to n_1; if not, to n_2.

IF (UNIT,i)n_1,n_2,n_3,n_4

n_1 not ready

n_2 ready and no previous error

n_3 EOF sensed on last input operation

n_4 parity or lost data error on last input operation

If errors are sensed during standard read or write operations, the FORTRAN I/O routines attempt to repeat the operation four times. On a buffer operation, only one attempt is made.

10.4 BUFFER STATEMENTS

The primary differences between buffer I/O and read/write I/O statements are given below:

1. The mode of transmission (BCD or binary) is tacitly implied by the form of the read/write control statement. In a buffer control statement, parity must be specified by a parity indicator.

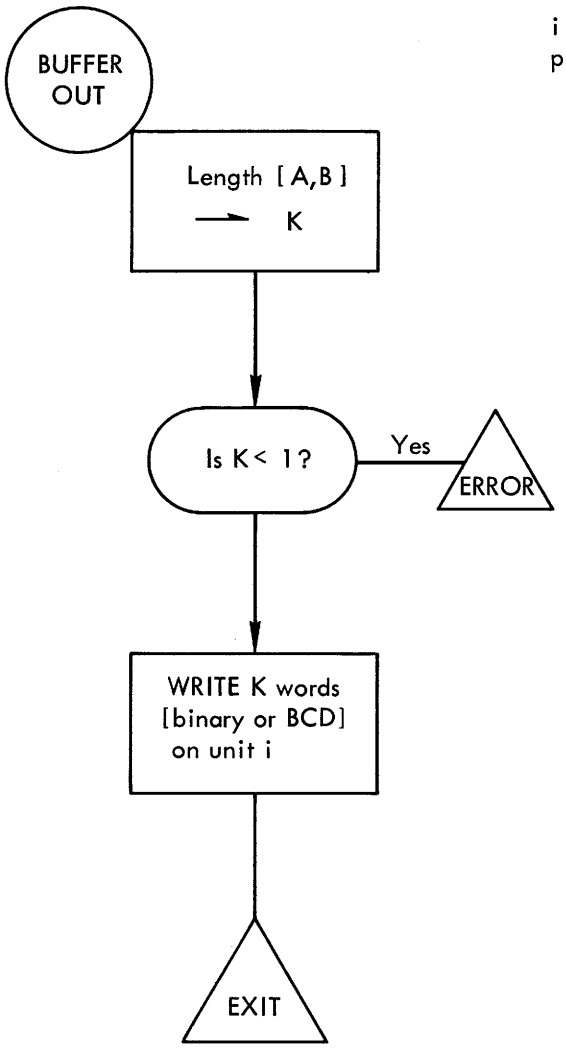
2. The read/write control statements are associated with a list and, in BCD transmission, with a FORMAT statement. The buffer control statements are not associated with a list; data transmission is to or from one area in storage.

3. A buffer control statement initiates data transmission, and then returns control to the program, permitting the program to perform other tasks while data transmission is in progress. Before buffered data is used, the status of the buffer operation should be checked. A read/write control statement completes the operation before returning control to the program.

In the descriptions that follow, these definitions apply.

u logical unit number

p parity key. May be specified by a simple variable (not subscripted). 0 for even parity (BCD); 1 for odd row binary; 2 for odd column binary.



A variable identifier: first word of data block to be transmitted.

B variable identifier: last word of data block to be transmitted.

In the BUFFER statements the address of B must be greater than that of A. A unit referenced in a BUFFER statement may not be referenced in other I/O statements.

BUFFER IN (u,p) (A,B)

Information is transmitted from unit u in mode p to storage locations A through B. Only one physical record is read for each BUFFER IN statement.

BUFFER OUT (u,p) (A,B)

Information is transmitted from storage locations A through B and one physical record is written on unit u in mode p containing all the words from A to B inclusive.

10.5 ENCODE/DECODE STATEMENTS

The ENCODE/DECODE statements are comparable to the BCD WRITE/READ statements; however, no peripheral equipment is involved. Information is transferred under FORMAT specifications from one area of storage to another. The parameters in these statements are defined as follows:

- n statement number, variable identifier, or formal parameter representing the FORMAT statement
- L input/output list
- v variable identifier or an array identifier which supplies the starting location of the BCD record.
- c unsigned integer constant or a simple integer variable (not subscripted) specifying the number of characters in the record. c may be an arbitrary number of BCD characters.

The first record within an encoded (decoded) area starts with the leftmost character position specified by v and continues c BCD characters, 10 BCD characters per computer word. For ENCODE, if c is not a multiple of 10, the last word in the record is blank-filled. For DECODE, if the record ends with a partial word the balance of the word is ignored.

Since each succeeding record begins with a new computer word, an integral number of computer words is allocated for each record with $\frac{c+9}{10}$ words. The total words allocated for the combined records in one encoded (decoded) area must not exceed 12.

Example:

A(1) = ABCDEFGHIJ

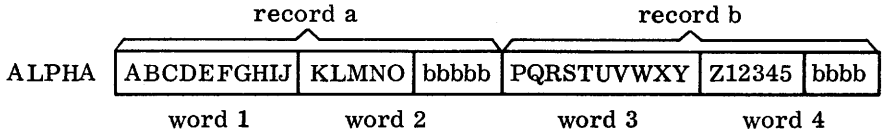
A(2) = KLMNO

B(1) = PQRSTUVWXY

B(2) = Z12345

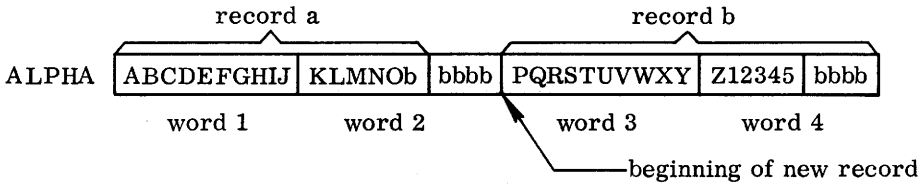
1) c = multiple of 10

```
      ENCODE (20, 1, ALPHA) A,B
    1 FORMAT (A10,A5/A10,A6)
```



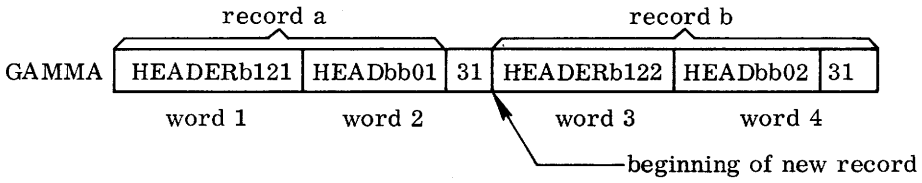
2) c ≠ multiple of 10

```
      ENCODE (16, 1, ALPHA) A,B
    1 FORMAT (A10,A6)
```



3) c ≠ multiple of 10

```
      DECODE (18, 1, GAMMA) A6,B6
    1 FORMAT (A10, A8)
```



A6(1) = HEADERb121

A6(2) = HEADbb01

B6(1) = HEADERb122

B6(2) = HEAD02

ENCODE (c,n,v)L

The information of the list variables, L, is transmitted according to the FORMAT (n) and stored in locations starting at v, c BCD characters per record. If the I/O list (L) and specification list (n) translate more than c characters per record, an execution diagnostic occurs. If the number of characters converted is less than c, the remainder of the record is filled with blanks.

DECODE (c,n,v)L

The information in c consecutive BCD characters (starting at address v) is transmitted according to the FORMAT and stored in the list variables. If the number of characters specified by the I/O list and the specification list (n) is greater than c (record length) per record, an execution diagnostic occurs. If DECODE attempts to process an illegal BCD code or a character illegal under a given conversion specification, an execution diagnostic occurs.

Examples:

1) The following illustrates one method of packing the partial contents of two words into one word. Information is stored in core as:

```
LOC(1)  SSSSxxxx
          .
          .
          .
LOC(6)  xxxxdddd
        10 bcd ch/wd
```

To form SSSSdddd in storage location NAME:

```
        DECODE(8,1 LOC(6) )TEMP
1       FORMAT (4X,A4)
        ENCODE(8,2,NAME) LOC(1)TEMP
2       FORMAT(2A4)
```

The DECODE statement places the last 4 BCD characters of LOC(6) into the first 4 characters of TEMP. The ENCODE statement packs the first 4 characters of LOC(1) and TEMP into NAME.

With the R specification; the program may be shortened to:

```
        ENCODE (8,1,NAME)LOC(1),LOC(6)
1       FORMAT (A4,R4)
```

2) DECODE may be used to calculate a field definition in a FORMAT specification at object time. Assume that in the statement FORMAT (2A8,Im) the programmer wishes to specify m at some point in the program, subject to the restriction $2 \leq m \leq 9$. The following program permits m to vary.

```
      IF(M.LT.10.AND.M.GT.1)1,2
    1 ENCODE (8,100,SPECMAT) M
  100 FORMAT (6H(2A8,I,I1,1H) )
      .
      .
      .
      PRINT SPECMAT,A,B,J
```

M is tested to insure it is within limits. If not, control goes to statement 2 which could be an error routine. If M is within limits, ENCODE packs the integer value of M with the characters: (2A8,I). This packed FORMAT is stored in SPECMAT. SPECMAT contains (2A8,Im).

A and B will be printed under specification A8, and the quantity J under specification I2, or I3, or . . . or I9 according to the value of m.

3) ENCODE can be used to rearrange and change the information in a record. The following example also illustrates that it is possible to encode an area into itself and that encoding will destroy information previously contained in an area.

```
      PROGRAM ENCO2
      I=7RBCDEFGH
      IA=1H1
      ENCODE (7,10,I)I,IA,I
   10 FORMAT (A2,A1,R4)
      PRINT 11,I
   11 FORMAT(O20)
      END
      PRINT OUT
              22012526273060
```

The BCD equivalent is

B1EFGHblank

4) In this example, accounting information is to be read from a magnetic tape prepared off-line from 80-column Hollerith card input. Each record on this tape will be 10 words (100 characters) long. The program is to initiate a read, decode the information of this read and initiate a second read while decoding the information obtained from the first read. Two 10-word buffers are used (AIN and CIN). The FORMAT specification in DECODE is:

(6A1,A1,8A1,A3,I2,A6,4I2,2A1,A8,A3,2A1)

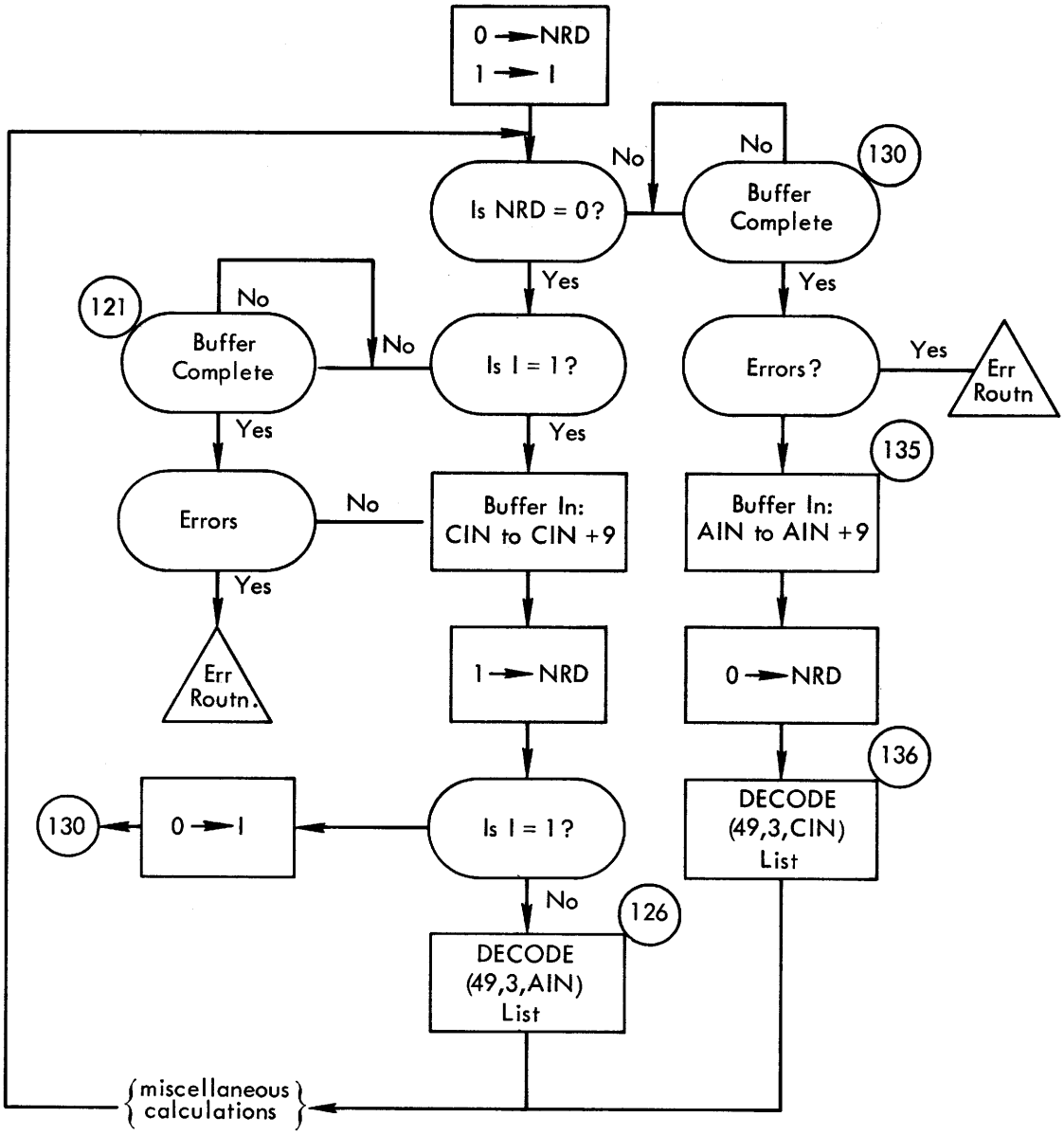
This specification breaks the first 49 characters of each BCD record read from magnetic tape. Let the list be the string of identifiers:

LIST: DT,CC,CN,PR,X,XM,N1,M1,N2,M2,CR,

ADJ,PER,RUN,ATT

DT is an array of length 6; CN is an array of length 8; the remaining identifiers name simple variables.

Flow chart of the basic procedure:



APPENDIX SECTION

6000 SERIES FORTRAN CHARACTER CODES

Source Language Character	Console Display Code	External BCD Code	Punch position in a Hollerith Card Column
A	01	61	12-1
B	02	62	12-2
C	03	63	12-3
D	04	64	12-4
E	05	65	12-5
F	06	66	12-6
G	07	67	12-7
H	10	70	12-8
I	11	71	12-9
J	12	41	11-1
K	13	42	11-2
L	14	43	11-3
M	15	44	11-4
N	16	45	11-5
O	17	46	11-6
P	20	47	11-7
Q	21	50	11-8
R	22	51	11-9
S	23	22	0-2
T	24	23	0-3
U	25	35	0-4
V	26	25	0-5
W	27	26	0-6
X	30	27	0-7
Y	31	30	0-8
Z	32	31	0-9
0	33	12	0
1	34	01	1
2	35	02	2
3	36	03	3
4	37	04	4
5	40	05	5
6	41	06	6
7	42	07	7
8	43	10	8
9	44	11	9
/	50	21	0-1
+	45	60	12
-	46	40	11
blank	55	20	space
.	57	73	12-8-3
)	52	74	12-8-4
$	53	53	11-8-3
*	47	54	11-8-4
,	56	33	0-8-3
(	51	34	0-8-4
=	54	13	8-3

FORTRAN STATEMENT LIST B

SUBPROGRAM STATEMENTS

Entry Points	SEGMENT name (f_1,f_2, . . .)	N
	PROGRAM name (f_1, . . . ,f_n)	N
	FORTRAN IV PROGRAM name (f_1, . . . ,f_n)	N
	FORTRAN II PROGRAM name (f_1, . . . ,f_n)	N
	MACHINE PROGRAM name (f_1, . . . ,f_n)	N
	ASCENTF PROGRAM name (f_1, . . . ,f_n)	N
	SUBROUTINE name (p_1, . . . ,p_n)	N
	FORTRAN IV SUBROUTINE name (p_1, . . . ,p_n)	N
	FORTRAN II SUBROUTINE name (p_1, . . . ,p_n)	N
	MACHINE SUBROUTINE name (p_1, . . . ,p_n)	N
	ASCENTF SUBROUTINE name (p_1, . . . ,p_n)	N
	FUNCTION name (p_1, . . . ,p_n)	N
	type FUNCTION name (p_1, . . . ,p_n)	N
	FORTRAN IV FUNCTION name (p_1, . . . ,p_n)	N
	FORTRAN II FUNCTION name (p_1, . . . ,p_n)	N
	FORTRAN IV type FUNCTION name (p_1, . . . ,p_n)	N
	FORTRAN II type FUNCTION name (p_1, . . . ,p_n)	N
Intersubroutine	EXTERNAL $name_1$,$name_2$. . .	N
*F	$name_1$,$name_2$, . . .	N
Transfer Statements	CALL name	E
	CALL name (p_1, . . . ,p_n)	E
	RETURN	

N = Non-executable E = Executable

DATA DECLARATION AND STORAGE ALLOCATION

Type Declaration	COMPLEX List	N
	DOUBLE PRECISION List	N
	DOUBLE List	N
	REAL List	N
	INTEGER List	N
	LOGICAL List	N
	TYPE DOUBLE List	N
	TYPE COMPLEX List	N
	TYPE REAL List	N
	TYPE INTEGER List	N
	TYPE LOGICAL List	N
Storage Allocations	DIMENSION V_1,V_2, . . . ,V_n	N
	COMMON/I_i/List	N
	EQUIVALENCE (A,B, . . .),(A1,B1, . . .). . .	N
	DATA1_1/List/,1_2/List/, . . .	N
	DATA (1_1=List), (1_2=List), . . .	N
	BLOCK DATA	N

ARITHMETIC STATEMENT FUNCTION

	name (p_1,p_2, . . .p_n) = Expression	E

SYMBOL MANIPULATION, CONTROL AND I/O

Replacement	A=E Arithmetic	E
	*D A=E	
	*I A=E	
	L=E Logical/Relational	E
	M=E Masking	E
	*B M=E	
Intraprogram Transfers	GO TO n	E
	GO TO m	E
	GO TO m, (n, . . .n_m)	E

	GO TO (n_1, . . . ,n_m),i	E
	IF (A) n_1,n_2,n_3	E
	IF (L) n_1,n_2	E
	IF (L) s	E
	IF (SENSE LIGHT i)n_1,n_2	E
	IF (SENSE SWITCH i)n_1,n_2	E
	IF (DIVIDE CHECK)n_1,n_2	E
	IF (ENDFILE i)n_1,n_2	E
	IF (EOF,i)n_1,n_2	E
	IF (IOCHECK,i)n_1,n_2	E
	IF (UNIT,i)n_1,n_2,n_3,n_4	E
	IF ACCUMULATOR OVERFLOW n_1,n_2	E
	IF QUOTIENT OVERFLOW n_1,n_2	E
LOOP CONTROL	DO n i = m_1,m_2,m_3	E
MISCELLANEOUS PROGRAM CONTROLS		
	ASSIGN s to m	E
	SENSE LIGHT i	E
	CONTINUE	E
	PAUSE	E
	PAUSE n	E
	STOP	E
	STOP n	E
I/O FORMAT		
	FORMAT (spec$_1$,spec$_2$, . . .)	N
I/O CONTROL STATEMENTS		
	READ n,L	E
	PRINT n,L	E
	PUNCH n,L	E

	READ (i,n)L	E
	READ INPUT TAPE i,n,L	E
	WRITE (i,n)L	E
	WRITE OUTPUT TAPE i,n,L	E
	READ (i)L	E
	READ TAPE i,L	E
	WRITE (i)L	E
	WRITE TAPE i,L	E
	ENCODE (c,n,v)L	E
	DECODE (c,n,v)L	E
	BUFFER IN (i,p) (fi,li)	E
	BUFFER OUT (i,p) (fi,li)	E
I/O Tape Handling	END FILE i	E
	REWIND i	E
	BACKSPACE i	E

PROGRAM AND SUBPROGRAM TERMINATION

	END	N

*Col. 1 indicator is used in FORTRAN II modes, I D B F

Form	Definition	Actual Parameter Type	Mode of Result
ABS(X)	Absolute value	Real	Real
AIMAG(C)	Obtain the imaginary part of a complex argument	Complex	Real
AIMT(X)	Truncation, integer	Real	Real
AMAX0(I_1,I_2, . . .)	Determine maximum argument	Integer	Real
AMAX1(X_1,X_2, . . .)	Determine maximum argument	Real	Real
AMIN0(I_1,I_2, . . .)	Determine minimum argument	Integer	Real
AMINI(X_1,X_2, . . .)	Determine minimum argument	Real	Real
AMOD(X_1,X_2)	X_1 module X_2	Real	Real
COMPLX(X_1,X_2)	Convert real to complex ($X_1 + iX_2$)	Real	Complex
CONJG(C)	Conjugate of C	Complex	Complex
DIM(X_1,X_2)	If $X_1 > X_2$: $X_1 - X_2$ If $X_1 \leq X_2$: 0	Real	Real
DMAX1(D_1,D_2, . . .)	Determine maximum argument	Double	Double
DMIN1(D_1,D_2, . . .)	Determine minimum argument	Double	Double
FLOAT(I)	Integer of real conversion	Integer	Real
IABS(I)	Absolute value	Integer	Integer
IDIM(I_1,I_2)	If $I_1 > I_2$: $I_1 - I_2$ If $I_1 \leq I_2$: 0	Integer	Integer

Form	Definition	Actual Parameter Type	Mode of Result	
IFIX(X)	Real-to-integer conversion	Real	Integer	E
INT(X)	Truncation, integer	Real	Integer	E
ISIGN(I_1,I_2)	Sign of I_1 times I_2	Integer	Integer	E
MAX0(I_1,I_2, . . .)	Determine maximum argument	Integer	Integer	E
MAX1(X_1,X_2, . . .)	Determine maximum argument	Real	Integer	E
MIN0(I_1,I_2, . . .)	Determine minimum argument	Integer	Integer	E
MIN1(X_1,X_2, . . .)	Determine minimum argument	Real	Real	E
MOD(I_1,I_2)	I_1 modulo X_2	Integer	Integer	E
REAL(C)	Obtain the real part of a complex argument	Complex	Real	E
SIGN(X_1,X_2)	Sign of X_2 times X_1	Real	Real	E
	LIBRARY FUNCTIONS			
ACOS(X)	Arccosine	Real	Real	N
ALOG(X)	Natural log of X	Real	Real	
ALOG10(X)	Log to the base 10 of X	Real	Real	
AND(X_1, . . .X_n)	Logical product of X_1 . . . X_n	Real	Logical	
ASIN(X)	Arcsine	Real	Real	
ATAN(X)	Arctangent X radians	Real	Real	
ATAN2(X_1,X_2)	Arctangent X_1/X_2	Real	Real	
CABS(C)	Absolute value	Complex	Real	
CCOS(C)	Complex cosine	Complex	Complex	
CEXP(C)	Complex exponent	Complex	Complex	
CLOG(C)	Complex log	Complex	Complex	
COMPL(X)	Complement of X	Real	Logical	

Form	Definition	Actual Parameter Type	Mode of Result
COS(X)	Cosine X radians	Real	Real
CSIN(C)	Complex sine	Complex	Complex
CSQRT(C)	Complex square root	Complex	Complex
DABS(D)	Absolute value	Double	Real
DATAN(D)	Double arctangent	Double	Double
DATAN2(D_1,D_2)	Double arctangent: D_1/D_2	Double	Double
DBLE(X)	Real to double	Real	Double
DCOS(D)	Double cosine	Double	Double
DEXP(D)	Double exponent	Double	Double
DLOG(D)	Natural log of D	Double	Double
DLOG10(D)	Log to the base 10 of D	Double	Double
DMOD(D)	D_1 modulo D_2	Double	Double
DSIGN(D_1,D_2)	Sign of: D_2 times D_1 in absolute value	Double	Double
DSIN(D)	Sign of double precision argument	Double	Double
DSQRT(D)	Square root of double	Double	Double
EXP(X)	e to Xth power	Real	Real
IDINT(D)	Double to integer	Double	Integer
LENGTH(I)	Returns number of words read on unit I	Integer	Integer
OR(X_1, . . . X_n)	Logical sum of X_1, . . . X_n	Real	Logical
RANF(X)	Random number generator	Real	Real
SECOND(I)	Returns time in seconds from dead start	Integer	Integer
SINGL(D)	Double to real	Double	Real
SIN(X)	Sine X radians	Real	Real

Form	Definition	Actual Parameter Type	Mode of Result
SQRT(X)	Square root of X	Real	Real
TAN(X)	Tangent X radians	Real	Real
TANH(X)	Hyperbolic tangent X radians	Real	Real

Following functions accept type A as a variable address name for an actual parameter:

Form	Definition	Actual Parameter Type	Mode of Result
LOCF(A)	Returns address of argument A	—	Integer
XLOCF(F)	Returns address of argument A	—	Real

SOME FORTRAN II, 63, 66, IV DIFFERENCES D

The following FORTRAN II statements are accepted by Chippewa FORTRAN:

1. In FORTRAN II arithmetic replacement statements, column 1 may contain either of the following characters

   D Double Precision mode

   I Complex mode

   When these characters are encountered, all variables and constants in the statement are assumed to be of the same type (double precision or complex).

2. FORTRAN II statements which contain a B in column 1 (Boolean) are evaluated as masking expressions. The operator equivalences are:

CHIPPEWA FORTRAN	FORTRAN II
. AND.	*
. NOT.	-
. OR.	+
	/

   Exclusive OR function defined as:

p	v	p/v
1	1	0
1	0	1
0	1	1
0	0	0

3. Mixed mode variables may appear in any FORTRAN II Boolean, B-type, Statement.

4. SENSE LIGHT STATEMENTS

   SENSE LIGHT i

   The statement turns on the sense light i; i may be a simple integer variable or constant (1 to 6). SENSE LIGHT 0 turns off all sense lights.

IF (SENSE LIGHT i)n_1,n_2

The statement tests sense light i. If it is on, it is turned off, and a jump occurs to statement n_1. If it is off, a jump occurs to statement n_2. The n_i are statement labels; i may be a simple integer variable or constant.

5. IF SENSE SWITCH STATEMENT

IF (SENSE SWITCH i)n_1,n_2

If sense switch i is set (on), a jump occurs to statement n_1. If it is not set (off), a jump occurs to statement n_2; i may be a simple integer variable or constant (1 to 6).

6. FAULT CONDITION STATEMENTS

At execute time, the computer may be set to interrupt on divide overflow or exponent fault. The fault indicator must be checked immediately after any statement that could possibly cause a fault condition.

IF DIVIDE CHECK n_1,n_2

A divide check occurs following division by zero. The statement checks for this condition; if it has occurred, the indicator is turned off and a jump to statement n_1 takes place. If no check exists, a jump to statement n_2 takes place.

IF QUOTIENT OVERFLOW n_1,n_2
IF ACCUMULATOR OVERFLOW n_1,n_2

An overflow occurs when the result of a real, double precision, or complex arithmetic operation exceeds the upper limits specified for these types. Results that are less than the lower limits are set to zero without indication. This statement is therefore a test for floating point overflow only. If the condition has occurred, the indicator is turned off, and a jump to statement n_1 takes place. If the condition does not exist, a jump to statement n_2 takes place.

7. Chippewa FORTRAN accepts the FORTRAN II version of the EXTERNAL statement. This form contains the same name list, but the word EXTERNAL has been replaced by the character F in column 1 of the statement.

```
1       7
F       name1,name2, . . .
```

8. The only inherently incompatible areas are the following:

COMMON-EQUIVALENCE Statement Relationships
In FORTRAN II, equivalence groups can reorder the common variables and arrays, and more than one variable in an equivalence group can be in common.

In Chippewa FORTRAN, equivalence groups do not reorder common, but may only extend the length of a common block.

Function-Naming Conventions

In FORTRAN II, the following rules apply for function subprogram, library function and statement function names:

The name is 4-7 alphanumeric characters, ending with the character F.

The first character must be X if, and only if, the value of the function is integer; for any other first character, the value of the function is real.

In Chippewa FORTRAN, the number of characters in the function name is 1-7; the first character must be alphabetic.

DO STATEMENT

If the terminal value of the DO statement (m_2) is less than the initial value (m_1):

FORTRAN IV	DO loop is executed once
FORTRAN 63	DO loop is not executed

END STATEMENT

FORTRAN IV	in subprograms, the END statement not preceded by a RETURN statement will cause the compilation of termination instruction (STOP)
FORTRAN 63	similar condition is compiled with an assumed RETURN statement

FORTRAN 66 AND FORTRAN IV DIFFERENCES

DO STATEMENT

If the terminal value of the DO statement (m_2) is less than the initial value of the index (m_1):

FORTRAN IV	DO loop is executed once
FORTRAN 66	DO loop is not executed

END STATEMENT

In subprograms:

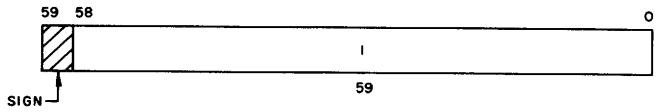
FORTRAN IV	causes compiling instructions to terminate
FORTRAN 66	compiles the instructions for returning control to the calling program

EQUIVALENCE STATEMENT

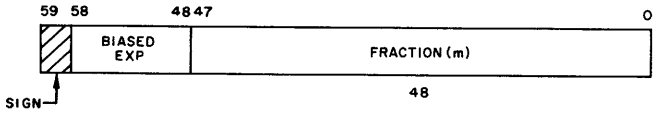
FORTRAN IV	programs and subroutines may not contain EQUIVALENCE statements which would cause a reordering of variables assigned to a COMMON region
FORTRAN 66	such reordering is possible

COMPUTER WORD STRUCTURE OF CONSTANTS–6600 E

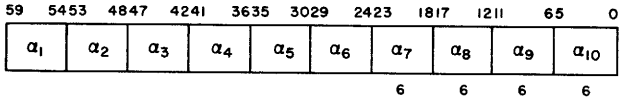
INTEGER



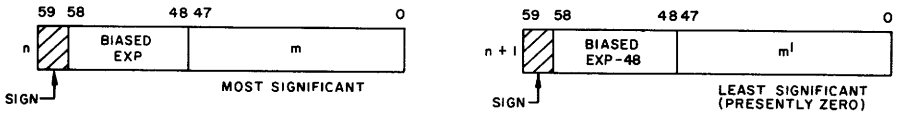
REAL



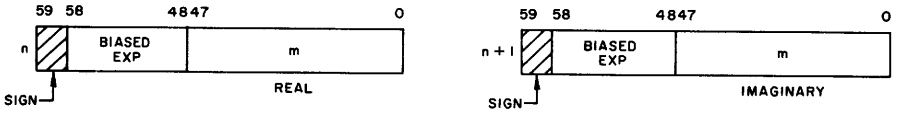
HOLLERITH BCD AND DISPLAY CODE



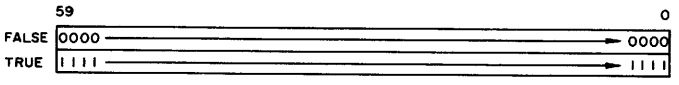
DOUBLE-PRECISION



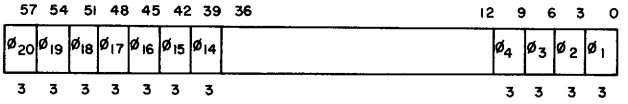
COMPLEX



LOGICAL



OCTAL



COMPILATION AND EXECUTION F

FORTRAN Control card

The FORTRAN compiler is called by the control card:

RUN (cm,fl,d,bl,if,of,fb)

cm compiler mode option; (if omitted, assume G)

- G compile and execute
- S compile, no execute
- P compile and punch, no execute
- L compile and list, no execute
- C chain mode
- B batch mode
- M (multiple mode) compiling is done as in batch mode, except octal versions of object programs, segments, and subroutines are produced for listing.
- I incomplete mode

fl object program field length (octal); if omitted, it is set equal to the field length at compile time. (24000_8 for compiler and constants)

d object program common length (octal); if omitted, it is set equal to the amount of common storage required for the main program being compiled.

bl object program I/O buffer lengths (octal); if omitted, assumed to be 2001 octal.

if file name for compiler input; if omitted, assumed to be INPUT.

of file name for compiler output; if omitted, assumed to be OUTPUT.

fb line-limit (octal) on the OUTPUT file of an object program. If not specified, it is set to 10000. If the line count exceeds the specified line limit, the job is terminated.

In if and of the file name may be followed by an equal sign and an octal constant to indicate a new buffer length. The length is normally 2001_8 words.

Example: Input = 10000_8

The starting addresses for I/O buffers appear on the map listing.

Compiler output, except in the G mode, includes a reproduction of the source program, a variable map, and indications of errors detected during compilation. If the G mode is selected, all output is suppressed unless errors are detected, in which case the output is the same as indicated for the other modes. If the L mode is selected, the output includes an octal list of the compiled instructions.

A copy of the compiled program is always left in disk storage as a binary file with the name of the program as file name. It may be called and executed repeatedly by name.

If the field length allocated by the system for a FORTRAN object program or compilation is less than the minimum required by the compiler, a jump to address 000000 occurs; the system indicates an arithmetic error.

In a Dayfile message at the end of compilation, the compiler indicates the amount of storage not used by the compiler and the object program. The unused spaces are filled with indefinite indication rather than with zeros. A count of errors detected and indicated during compilation appears in a Dayfile message at the end of compilation.

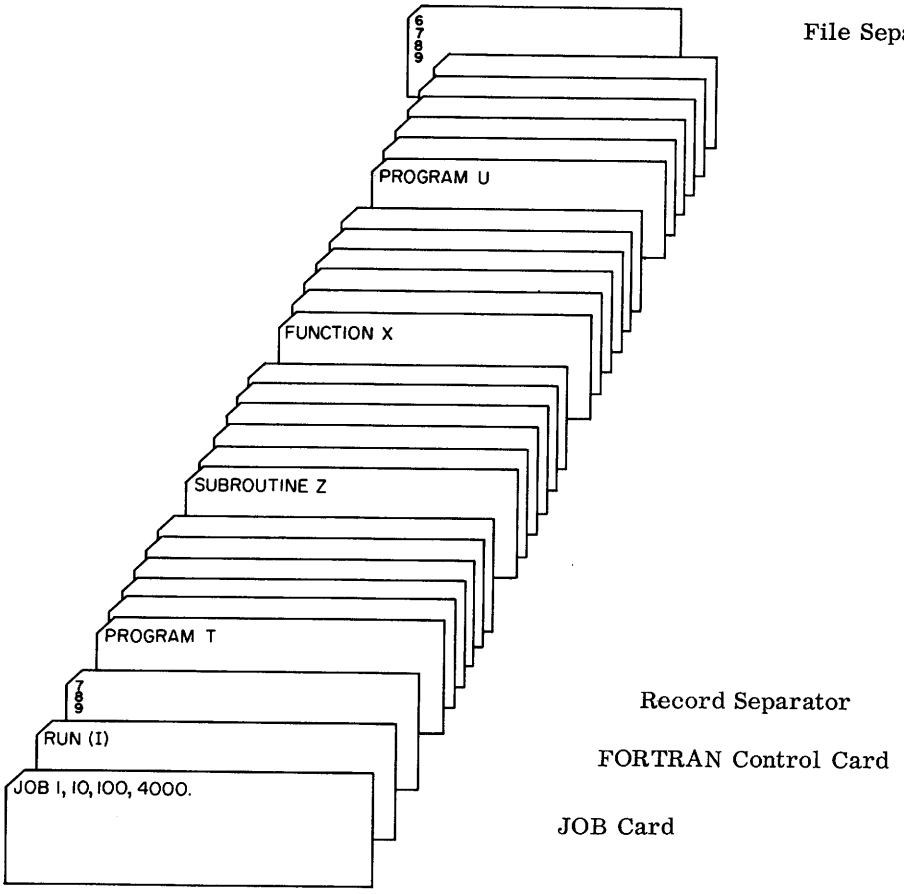
The compiler requires a minimum of 32000_8 locations.

Any program, segment, or subroutine punched in the incomplete mode (I mode) may produce a variable map if it is read back for incorporation in a running program or segment. When two or more subroutines have the same name, only the first one is assembled and the names of the deleted ones appear in the output listing. However, the variable map is not produced if the read back takes place during compiling in the G or C mode with no errors.

A FORTRAN coded or binary subroutine punched in the incomplete mode will override any other subroutine punched in the incomplete mode which appears later in the deck and which has the same name.

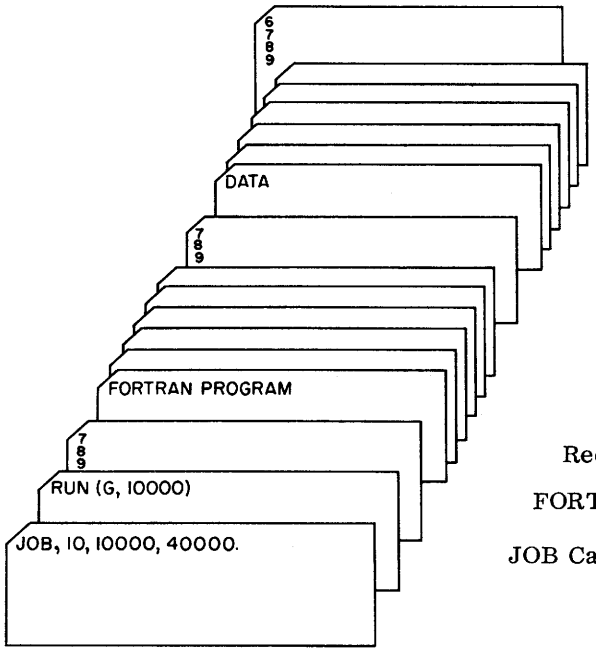
An I mode (an incomplete mode) compiles a program, subroutine, or function to binary cards in a form which can be reloaded by the compiler. Only instructions and constants of the object program or subroutine are punched. Compilation is continued until an end-of-record card is detected, and each program or subroutine punched is reloadable either by itself or along with others in the same mode.

Example:



When a RUN card specifies G or C mode and the object program field length differs from the length designated on the JOB card, the compiler does not request the system to change the job field length until completion of the compilation.

Example:



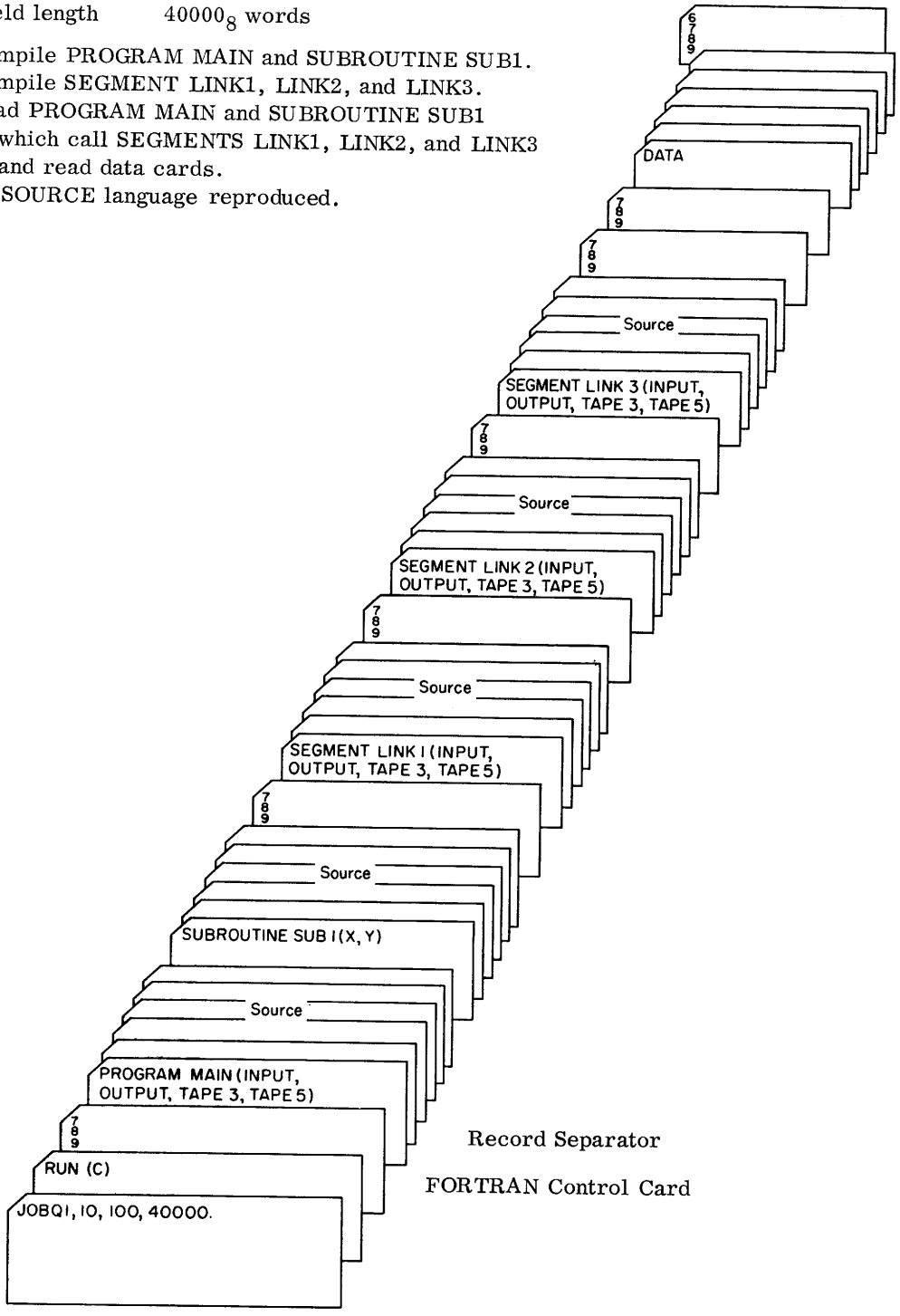
The above control card sequence will compile in a field of 40000 words and run in a field of 10000 words.

CHAIN COMPILATION AND EXECUTION

In C (chain) mode, a program followed by segments, separated by end-of-record cards, must follow the control statements. Source language is not reproduced, binary versions of the program and segments are compiled to disk, and compilation terminates upon detection of two successive end-of-record cards, or one end-of-file card, followed by a call to the first program of the chain.

Job name	JOBQ1
Priority	10
Time limit	approx. 1 minute
Field length	40000_8 words

Compile PROGRAM MAIN and SUBROUTINE SUB1.
Compile SEGMENT LINK1, LINK2, and LINK3.
Load PROGRAM MAIN and SUBROUTINE SUB1
which call SEGMENTS LINK1, LINK2, and LINK3
and read data cards.
No SOURCE language reproduced.



File Separator

Record Separator

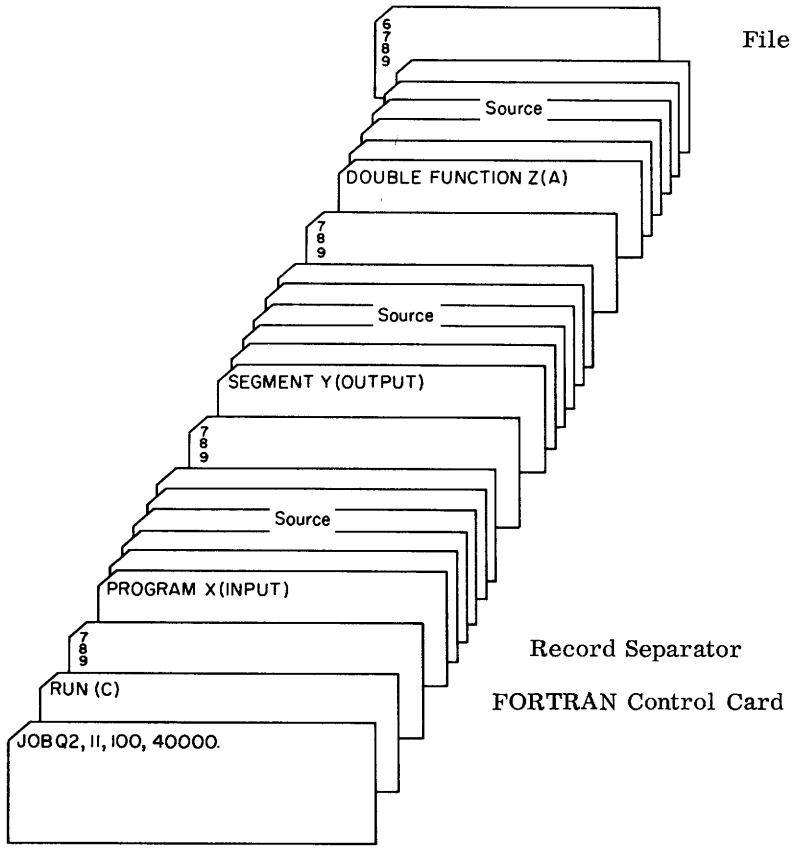
FORTRAN Control Card

BATCH COMPILATION

B mode (batch) compiles segments, subroutines, and functions, separated by record separator cards, to the disk without reloading the compiler each time. Source language is reproduced, and compilation is terminated upon detection of two successive end-of-record cards or one end-of-file card.

Job name	JOBQ2
Priority	11
Time limit	approx. 1 minute
Field length	40000_8 words

Source language reproduced.



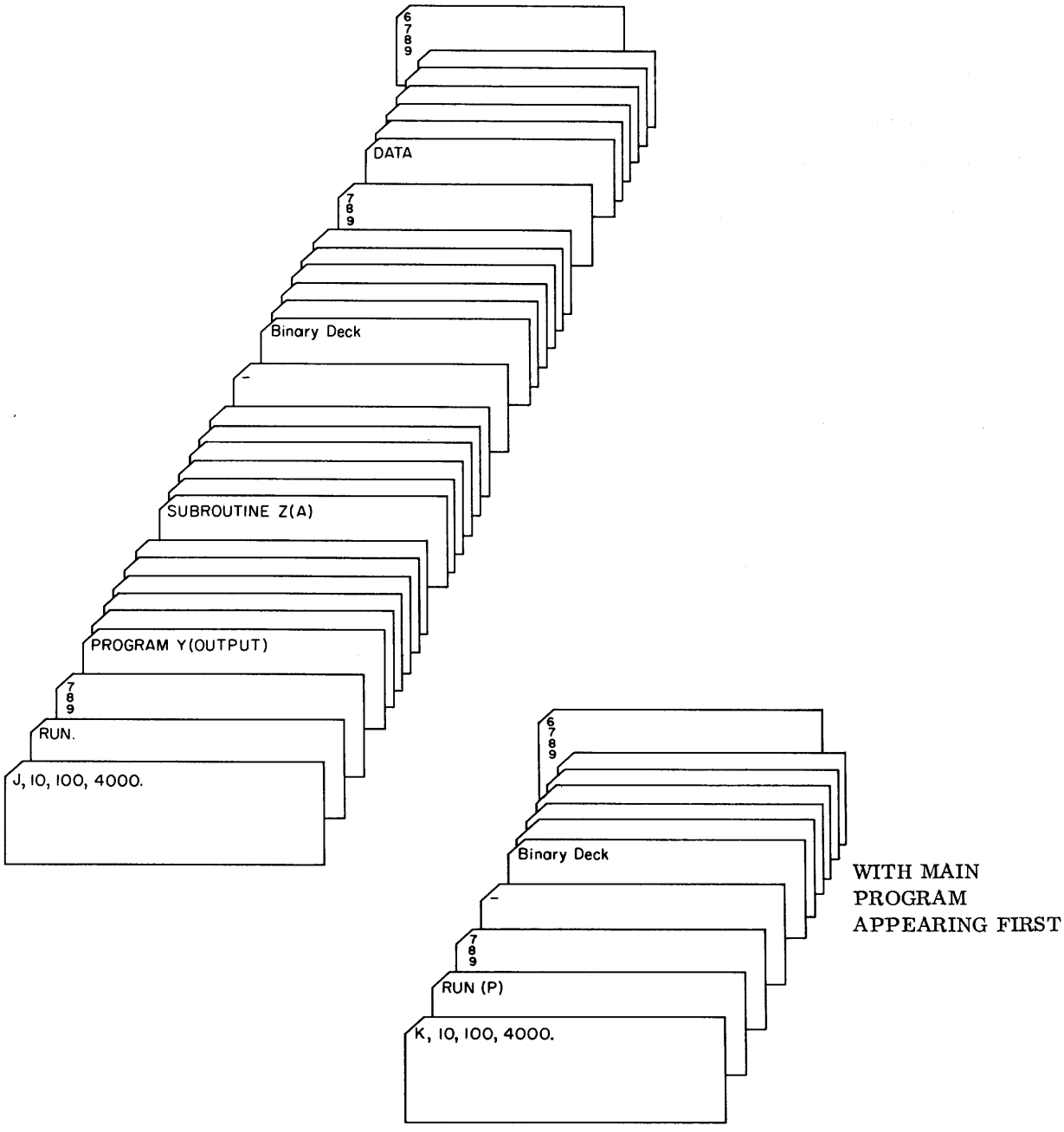
File Separator

Record Separator

FORTRAN Control Card

Binary cards to be reloaded by the compiler must be separated from the FORTRAN coded deck by a card with a minus sign (-) in column one. If the main program is also in a binary card form, the card with the minus sign must immediately follow the end-of-record card which separates the control cards from the data portion of the job.

Example:



Mixed Source Deck

The FORTRAN compiler, RUN, processes programs and subroutines written in assembly language or in a subset of ASCENT assembly language. Such programs or subroutines may be intermixed with regular FORTRAN programs and subprograms.

MACHINE PROGRAM name (f_1, . . . ,f_n)

MACHINE SUBROUTINE name (p_1, . . . ,p_n)

Either of the above must be the first statement of a program or subroutine coded in assembly language.

ASCENTF PROGRAM name (f_1, . . . ,f_n)

ASCENTF SUBROUTINE name (p_1, . . . ,p_n)

Either of the above must be the first statement of a program or subroutine coded in a subset of ASCENT.

name is an alphanumeric identifier.

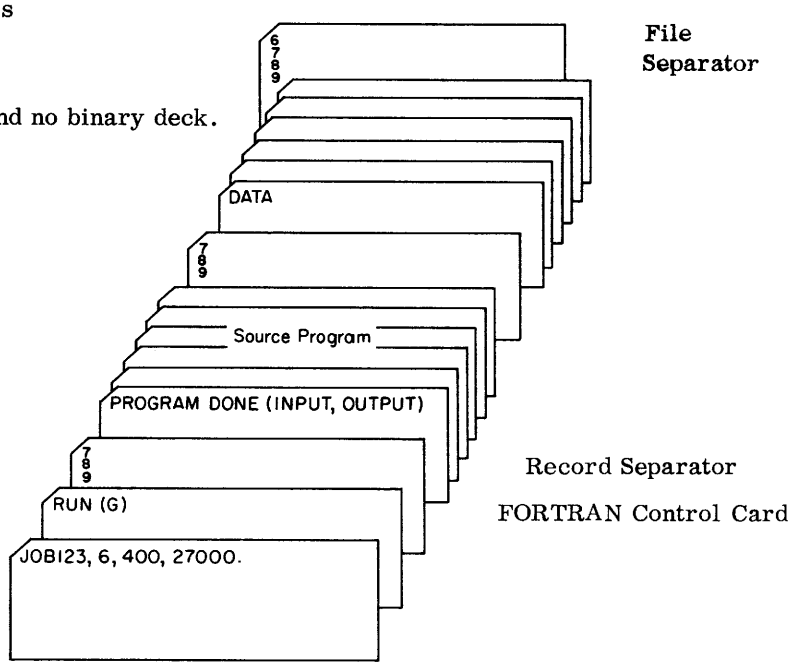
f_i are file names.

p_i are formal parameters.

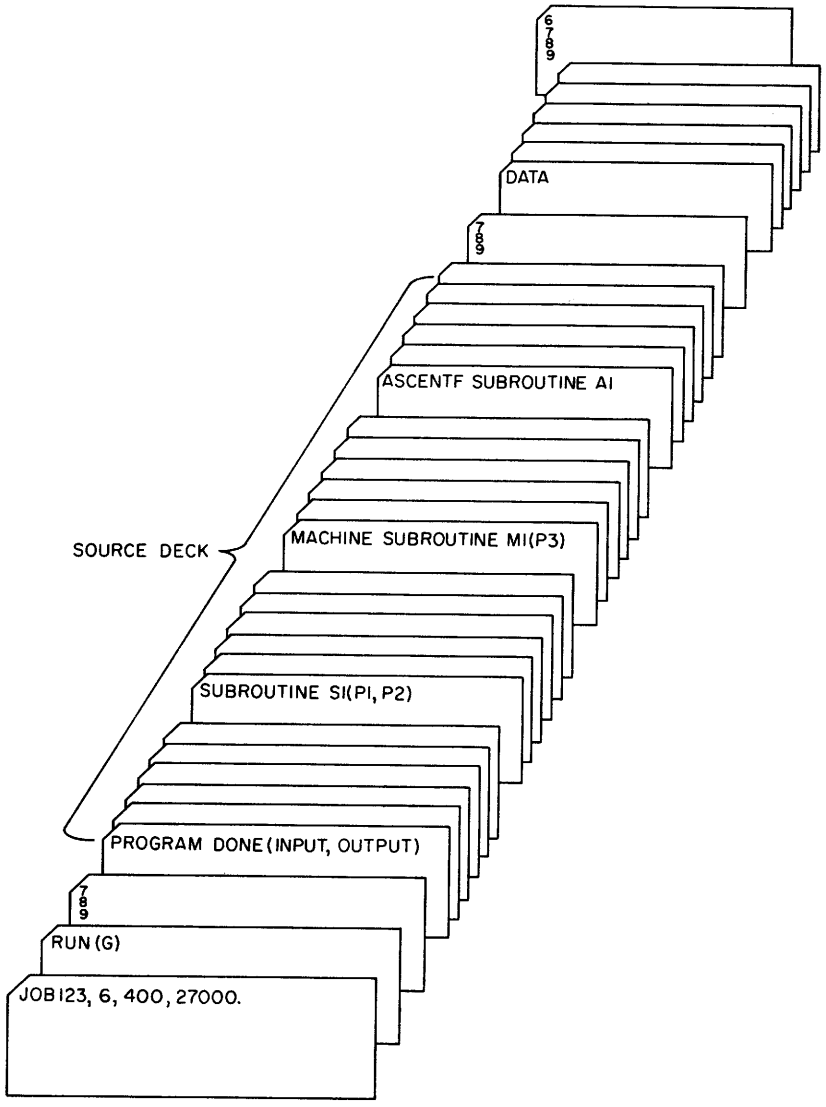
DECK STRUCTURE FOR A NORMAL COMPILE AND EXECUTE

Job name	JOB123
Priority	6
Time limit	approx. 4 minutes
Field length	27000_8 words

Compile and execute with no list and no binary deck.



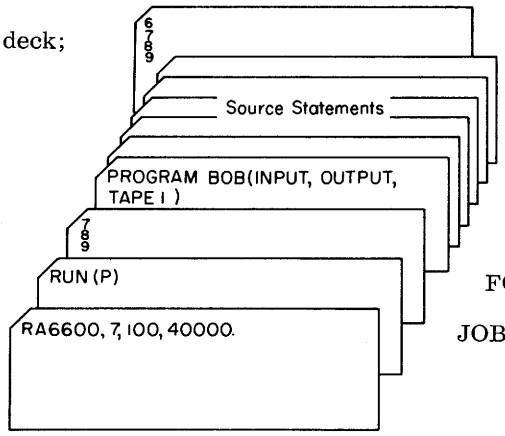
With a mixed source deck:



DECK STRUCTURE FOR COMPILE AND PRODUCE BINARY DECK

Job name	RA6600
Priority	7
Time limit	approx. 1 minute
Field length	40000_8 words

Compile program and punch binary deck; do not execute.



File Separator

Record Separator

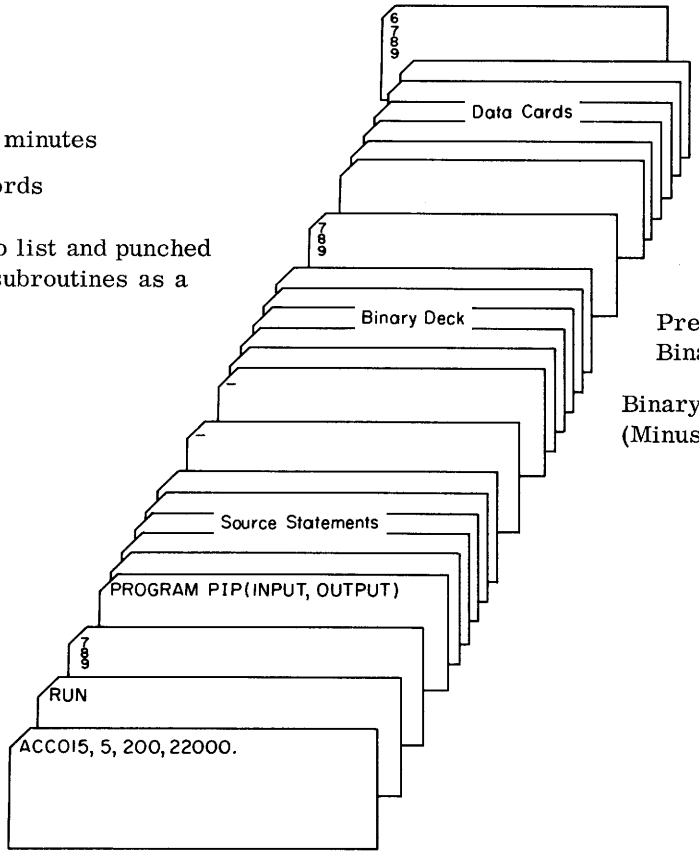
FORTRAN Control Card

JOB Card

DECK STRUCTURE FOR COMPILE WITH BINARY SUBROUTINES

Job name	ACC015
Priority	5
Time limit	approx. 2 minutes
Field length	22000_8 words

Compile and execute with no list and punched binary output plus a set of subroutines as a binary deck.



File Separator

Pre-Punched Binary Cards

Binary Deck Indicator (Minus Sign Column 1)

FORTRAN ERROR PRINTOUTS G

During a FORTRAN compilation, two-character error printouts follow statements which are incorrect; other printouts may follow the END statement, indicating other types of errors in the program. The two-character error indicators are explained below:

AC Argument Count

The number of arguments in a current reference to a subroutine differs from the number which occurred in a prior reference.

AL Argument List

Format error in a list of arguments.

AS Assign

Format error in an ASSIGN statement.

BC Boolean Constant

Format error in the designation of a FORTRAN Boolean constant in a B-type expression.

BI Binary Input

Incorrect header card in a subroutine of binary cards following the FORTRAN program.

BO Common Block Overflow

Current requirements for a labeled block of common storage exceed the block length established in a preceding COMMON statement.

BX Boolean Expression

Format error in a B-type Boolean statement.

CD Duplicate Common

A variable being assigned to the common region has been previously assigned to this region.

CE Common-Equivalence Error

Incorrect equivalence of two variables assigned to common storage.

CL CALL

Format error in a CALL statement.

CM COMMON

Format error in a COMMON statement.

CN Continuation

More than 19 continuation cards or one such card appears in an illogical sequence.

CO Common Overflow

Amount of common storage required by the main program or specified to the compiler is less than that required by the current program or subroutine.

CT CONTINUE

CONTINUE statement is missing a statement number.

DA Duplicate Argument

Duplicate dummy arguments appear in a function-definition statement.

DC Decimal Constant

Format error in the expression of a FORTRAN decimal constant.

DD Duplicate Dimension

A variable being dimensioned has been previously dimensioned.

DF Duplicate Function Name

The function name in the current function-definition statement has occurred as the name of a previously defined function.

DM DIMENSION

Format error in a DIMENSION statement.

DO DO

Format error in a DO statement.

DP Duplicate Statement Number

Current statement number has previously appeared in the statement-number field.

DS Missing DO Number

Current statement number has previously appeared in the statement number field.

DS Missing DO Number

In DO statements, non-existent numbers have been referenced.

DT DATA

Format error in a DATA statement.

EC Equivalence Contradiction

A variable cannot appear in an EQUIVALENCE statement because of an inherent contradiction in the statement.

EF End of File

End-of-file card detected before the END card encountered.

EM Exponential Mode

Mode of the base or the exponent is incorrect.

EQ Equivalence

Format error in the exponential statement.

FL Function List

Format error in an EXTERNAL statement or F-type statement.

FM Format

Format error in a statement whose type cannot be determined.

FN Format Statement Number

Statement number is missing from a FORMAT statement.

FS Format Specification

Format error in the specification portion of a FORMAT statement.

FT Function Type

Format error in a type statement.

GO GO-TO

Format error in a GO statement.

IF IF

Format error in an IF statement.

IL Indexed List

Format error in an indexed list of current input/output statement.

LR Library Reference

In a reference to a standard library subroutine, more arguments appeared than are provided for by the subroutine.

LS List

Format error in an input/output list.

MA Misuse of Argument

An argument of the subroutine or function being compiled has been incorrectly used in an EQUIVALENCE statement.

MC Machine Constant

Error in format of a constant in a tag-defining line of coding involving the pseudo operations RES, COM, CON, ABS, SUB, or HOL.

MD Machine-Duplicate-Tag Error

Previously defined tag appears in a tag-defining line of coding involving the pseudo operations RES, COM, CON, ABS, SUB, or HOL.

MF Machine-Format Error

Format error in a tag-defining line of coding involving the pseudo operations RES, COM, CON, ABS, SUB, or HOL.

ML Machine Location Tag

A tag in an additive address field of a machine instruction did not appear in a subsequent location field.

MO Memory Overflow

Compiler field length specified on the job card is too short.

MR Missing Subroutine

Subroutines have been referenced that are not in the standard subroutine library.

MS Missing Statement Number

References have been made to non-existent statement labels.

MT Machine Tag Definition

Format error in a tag-defining line of coding involving the pseudo operations RES, COM, CON, ABS, SUB, or HOL.

MU Machine Undefine Tag

A tag appearing alone in an address field of a machine instruction did not appear in a subsequent location field.

NC Name Conflict

A subroutine or function name conflicts with prior usage of the name.

NM Name

Format error in name (header) card.

OD DIMENSION Statement Order

An array has been referenced prior to being named in a DIMENSION statement.

PN Parentheses

Indicates unpaired parenthesis.

RN RETURN

Format error in a RETURN statement.

SB Subscript

Format error in subscript of an array reference.

SE SENSE

Format error in a SENSE statement.

SF Short Field

Indicates the number of words by which the required field length of the object program or subroutine exceeds the specified length.

SL Subroutine Storage Limit

Compiler field length is exceeded.

SM Statement Number Field

Format error in the statement-label field.

SN System Number

Format error in a position where a statement label should appear.

SY System

Error in the FORTRAN system.

TM Too Many Arguments

A subroutine reference or the program or subroutine being compiled has more than 60 arguments.

TY Type

Format error in a type statement.

UA Unidentified Array

An array has not been previously named in a DIMENSION statement.

UE Unidentified Equipment

A referenced input/output file was not listed in the header card of the main program.

US Unreferenced Binary Subroutine

A subroutine on binary cards following the FORTRAN program has not been referenced.

VC Variable Name Conflict

A variable name conflicts with a prior usage.

VD Variable Dimensioned Array

An array whose dimensions are arguments to the subroutine or function being compiled has been used incorrectly.

VN Variable Name

Variable name is in error.

XF Expression Format

Format error in expression.

XM Expression Mode

Modes are mixed in an expression.

INDEX

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET
6000 Series Chippewa Operating System

FORTRAN Reference Manual

Pub. No. 60132700, Rev. A May, 1966

THIS FORM IS NOT INTENDED TO BE USED AS AN ORDER BLANK. YOUR EVALUATION OF THIS MANUAL WILL BE WELCOMED BY CONTROL DATA CORPORATION. ANY ERRORS. SUGGESTED ADDITIONS OR DELETIONS, OR GENERAL COMMENTS MAY BE MADE BELOW. PLEASE INCLUDE PAGE NUMBER REFERENCE.

FROM NAME: _____

BUSINESS ADDRESS: _____

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

FOLD

FOLD

STAPLE

STAPLE

CONTROL DATA SALES OFFICES

ALAMOGORDO • ALBUQUERQUE • ATLANTA • BILLINGS • BOSTON • CAPE CANAVERAL • CHICAGO • CINCINNATI • CLEVELAND • COLORADO SPRINGS DALLAS • DAYTON • DENVER • DETROIT • DOWNEY, CALIFORNIA • GREENSBORO, NORTH CAROLINA • HONOLULU • HOUSTON • HUNTSVILLE • MIAMI MONTEREY, CALIFORNIA • INDIANAPOLIS • ITHACA • KANSAS CITY, KANSAS LOS ANGELES • MADISON, WISCONSIN • MINNEAPOLIS • NEWARK • NEW ORLEANS • NEW YORK CITY • OAKLAND • OMAHA • PALO ALTO • PHILADELPHIA • PHOENIX • PITTSBURGH • SACRAMENTO • SALT LAKE CITY SAN BERNARDINO • SAN DIEGO • SANTA BARBARA • SAN FRANCISCO SEATTLE • ST. LOUIS • TULSA • WASHINGTON, D. C.

AMSTERDAM • ATHENS • BOMBAY • CANBERRA • DUSSELDORF • FRANKFURT • HAMBURG • JOHANNESBURG • LONDON • LUCERNE • MELBOURNE MEXICO CITY • MILAN • MONTREAL • MUNICH • OSLO • OTTAWA • PARIS TEL AVIV • STOCKHOLM • STUTTGART • SYDNEY • TOKYO (C. ITOH ELECTRONIC COMPUTING SERVICE CO., LTD.) • TORONTO • ZURICH

CONTROL DATA CORPORATION

8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440

Pub. No. 60132700 Rev. A

LITHO IN U.S.A.