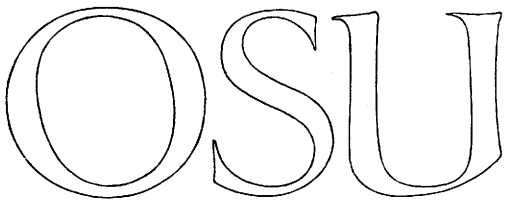
ccm-73-04

FORTRAN REFERENCE MANUAL

August 1973



COMPUTER CENTER
Oregon State University
Corvallis, Oregon 97331

FORTRAN REFERENCE MANUAL

ccm-73-04

Oregon State University

August, 1973

TABLE OF CONTENTS

OS-3 FORTRAN REFERENCE MANUAL

Introduction

FORTRAN is probably the most widely used scientific programming language in the United States. However, both the definition of the language and the method of implementation vary from one installation to another.

This manual describes OS-3 FORTRAN, version 3.1. Since OS-3 is a unique operating system, this manual will attempt to provide a complete technical description of the OS-3 FORTRAN implementation, with special emphasis on those features which are non-ASA standard or are not in agreement with CONTROL DATA's definitions.

The User's Virtual Machine

Each user running on the CDC 3300 under Oregon State's Open Shop Operating System (OS-3) has a "virtual" (or simulated) computer with 65536 24-bit words of core storage. Half of this, or 32K, is accessible to the FORTRAN user. The remaining half may be accessed via subprograms written in assembly language.

The following registers in the user's virtual machine are used by FORTRAN object programs:

A - accumulator
Q - multiplier/quotient
EU - extension (upper)
EL - extension (lower)
and Register Files 40-77.

Each of these is 24 bits long.

In addition, there are three 15-bit index registers designated B1, B2, B3.

Each virtual machine may have a number of logical I/O units (LUNs) "equipped" at any one time. FORTRAN programs may access LUNS 1 through 63.

NOTATION

The following meta-language symbols will be used to describe FORTRAN syntax:

<X>	- a syntactic entity called X as opposed to the letter X itself.
|	- separates elements; means "or," so <X>|Y means either a syntactic entity called X or else the letter Y.
{ }	- means zero or more occurrences of whatever is inside the braces.
$\{\ \}_m^n$	- m to n occurrences of whatever is inside the braces.
$\lceil x \rceil$	- smallest integer $\geq$ x.

Examples: <DIGIT> = 0|1|2|3|4|5|6|7|8|9

<SIGN> = +|-

<INT CONST> = $\{\text{<sign>}\}_0^1\text{<DIGIT>}\{\text{<DIGIT>}\}_0^6$

$\lceil 33/4 \rceil = 9$

The format of each section to follow is:

	TITLE	- Meta-language name for entity being defined.
Source	Description	- An attempt to say in English what the entity is syntactically and what it means semantically.
	Source Examples	- Of only the syntactic entity described above.
	Examples in Context	- May use well-known, but not yet defined, constructions.
	General Syntax	- Meta-language definition of the entity.

Object
- Object Form — A description or depiction of the object (machine-level) form of the entity.
- Examples — The same examples as given above in the 'Source Examples' section now in their object form.

CHARACTER SET <CHARA>

Description

The following table gives the characters of the FORTRAN character set.

	Symbol	6-bit BCD	ASCII
<LTR>	A-I	21-31	301-311
	J-R	41-51	312-322
	S-Z	62-71	323-332
	blank	60	240
	=	13	275
	+	20	253
	-	40	255
	*	54	252
	/	61	257
	(	74	250
	)	34	251
	,	73	254
	.	33	256
	$	53	244
<DIGIT>	0-9	00-11	260-271

SOURCE RECORD FORMAT

Description

If position one of a FORTRAN source record contains the letter C, the record is a comment and is copied onto the list unit (if one exists) and is otherwise ignored by the compiler.

For non-comment records, positions one to five contain the statement number, <STNUM>, if one is used. Only FORMAT and executable statements may be numbered, and only those referenced by other statements should be. Each statement number must be an integer in the range 1 to 32767 and may occur anywhere in positions one to five. (The word CEJECT in positions one to six will begin a new page of listing, if one is being generated.)

Position six or a FORTRAN source record is special. If this position is not blank or zero, the physical record is taken to be a logical continuation of its predecessor.

The body of each FORTRAN statement must occur between positions 7 and 72. Positions 73 through 80 are ignored and may be used for sequence numbering, etc. Records may not be longer than 80 characters. Remember this is true on continuation cards too.

SINGLE-WORD INTEGER CONSTANT <SWIC>

Description

A standard precision (24-bit) integer constant is specified by an optional sign (+ or -) followed by one to seven decimal digits. The number must be less than 8,388,608 in absolute value.

Source Examples

```
1          -1          -1234567
```

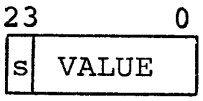
Examples in Context

```
I=1        J=J+1
```

General Syntax

$\{+|-\}_0^1\{\langle DIGIT\rangle\}_1^7$ where <DIGIT> = 0|1|2|3|4|5|6|7|8|9|

Object Form

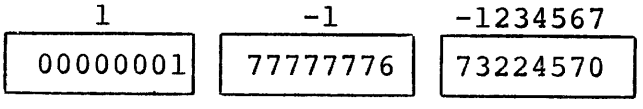


This depicts a single 24-bit word of CDC 3300 core storage. Twenty-three bits are available to represent the value of the integer constant, so the range is

$$-(2^{23}-1) = -8{,}388{,}607 \text{ to } +(2^{23}-1) = 8{,}388{,}607.$$

The sign bit (bit 23) is 0 if the number is positive; one if the number is negative. Negative numbers are represented in one's complement form.

Examples



(Each octal digit represents three bits.)

DOUBLE-WORD INTEGER CONSTANT

Description

A double-word integer constant is specified by an optional sign (+ or -) followed by 1 to 15 decimal digits and the letter D.

Source Examples

```
1D          -1D          -1234567890D
```

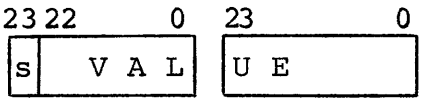
Examples in Context

```
A=1D
```

General Syntax

$\{+|-\}_{0}^{1}\{<DIGIT>\}_{1}^{15}D$

Object Form



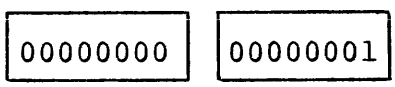
Two consecutive words of core storage are used; 47 bits are available to represent the value of the constant, so the range is

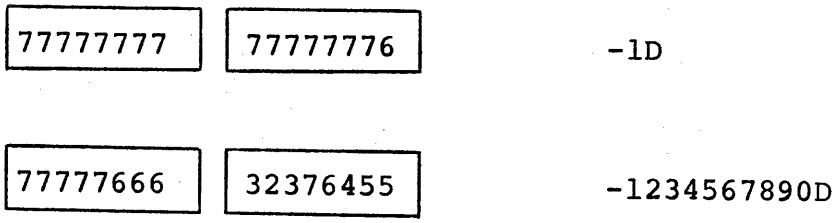
$-(2^{47}-1) = -140,737,488,355,327$ to

$+(2^{47}-1) = 140,737,488,355,327$

NOTE: In ANSI FORTRAN, the "D" suffix means double-precision real (floating point). Beware when converting programs from other installations.

Examples




77777777
77777776
-1D
77777666
32376455
-1234567890D

OCTAL CONSTANT

Description

An octal constant is specified by an optional sign (+ or -) followed by 1 to 16 octal digits and the letter B. From one to eight source digits produce a single-word octal constant in the machine. Nine to sixteen digits produce a double-word octal constant.

Source Examples

```
1B          10B          77B          -11B          -77777777B
1122334455667700B
```

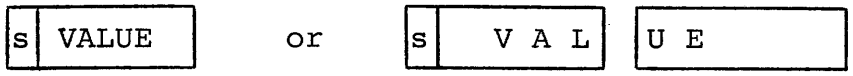
Examples in Context

```
MASK=70707070B
IF (ICHAR.EQ.60B) GO TO 50
```

General Syntax

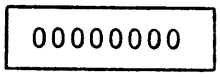
$\{+|0\}_{0}^{1}\{<\text{OCT DIG}>\}_{1}^{16}B$ where <OCT DIG> = 0|1|2|3|4|5|6|7

Object Form

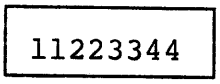


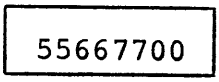
Examples

00000001	1B
00000010	10B
00000077	77B
77777766	-11B



-77777777B





1122334455667700B

REAL OR FLOATING-POINT CONSTANT <FPC>

Description

There are several source forms:

1) An optional sign, followed by a string of at most 11 decimal digits. This string must include a decimal point (period character, "."), or

2) the string of digits may be followed by the letter E, with the decimal point omitted, or

3) the decimal point may be included and the string followed by an "E" and an optionally signed integer exponent in the range -308 to +308.

Source Examples

1E 1.0 1.0E6 1.0E-100

Examples in Context

X=1.0 Y=X+1.0 IF (X+.6.EQ.5.5) Y=1.E-1

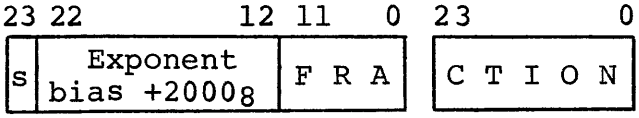
General Syntax

$\{+|-\}_0^1\{\text{<DIGIT>}\}_1^{11}\text{E}$ or

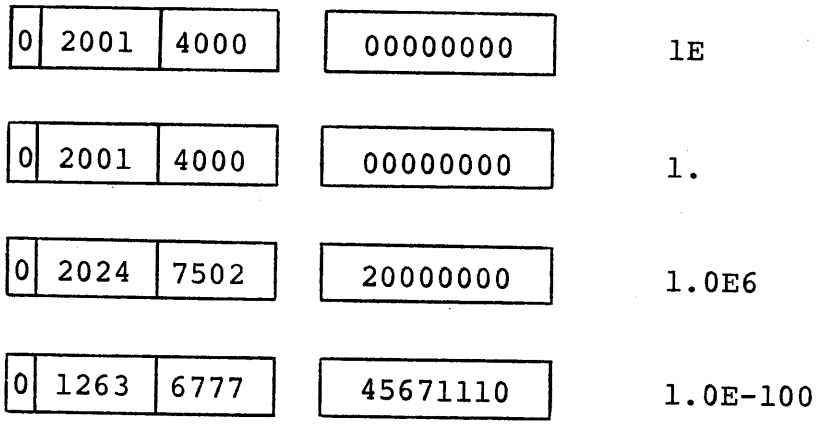
$\{+|-\}_0^1\{\text{<DIGIT>}\}_X^N \cdot \{\text{<DIGIT>}\}_y^{11-N} \quad \{\text{E<SWIC>}\}_0^1$

where X+Y≥1 and the <SWIC> is an optionally signed integer exponent in the range -308 to +308.

Object Form



Examples


0 2001 4000
00000000
1E
0 2001 4000
00000000
1.
0 2024 7502
20000000
1.0E6
0 1263 6777
45671110
1.0E-100

HOLLERITH--LEFT JUSTIFIED

Description

From one to eight BCD characters can be specified as a constant by preceding the string of characters by an integer from one to eight and the letter H. Up to four characters are stored in one computer word; five to eight require two words. Unspecified character positions in the right end of the word are filled with blanks (60B).

Source Examples

```
3HTWO          4HFORE          5HTHREE
```

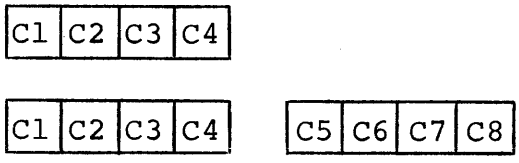
Examples in Context

```
DATE=6HAUG 39    IF (ZNAME.EQ.5HSMITH) GO TO 50    ITEM=3HBAD
```

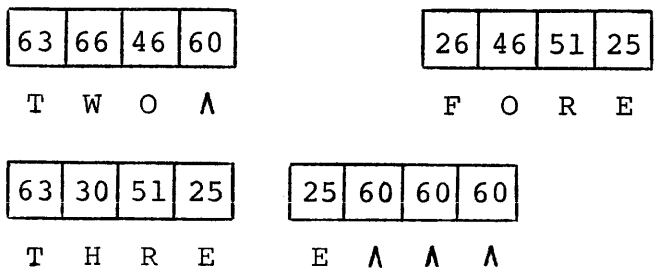
General Syntax

$nH\{<CHARA>\}_{1}^{8}$ where n is an integer 1|2|3|4|5|6|7|8 specifying the length of the character string.

Object Form



Examples



(NOTE: CDC 3300 BCD codes, blank filled on right.)

HOLLERITH--LEFT-JUSTIFIED STRING FORM

Description

Any string of legal FORTRAN characters (see page 3.0) enclosed in single quotes. (Single quotes cannot be used inside the string.)

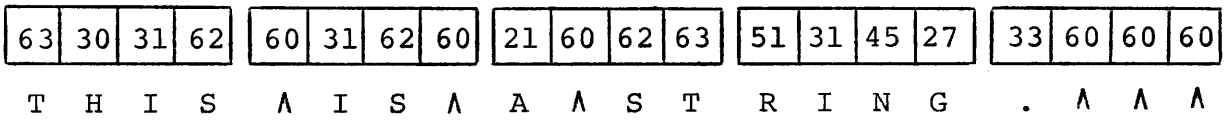
Source Example

'THIS IS A STRING.'

Object Form

The characters are stored in BCD code, four per computer word in ⌈(length of string)/4⌉ consecutive words of core storage.

Example



HOLLERITH--RIGHT JUSTIFIED

Description

From one to eight BCD characters can be specified as a constant by preceding the string of characters by an integer from one to eight and the letter R. Up to four characters are stored in one computer word; five to eight characters require two words. Unspecified character positions in the left end of the word are filled with zeros (00B).

Source Examples

```
3RTWO          4RFORE          5RTHREE
```

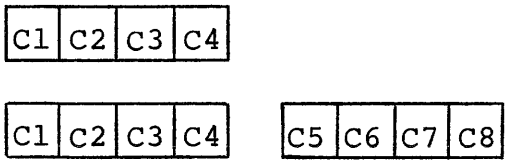
Examples in Context

```
IF (KODE.EQ.1RA) GO TO 50          JX=2R17
```

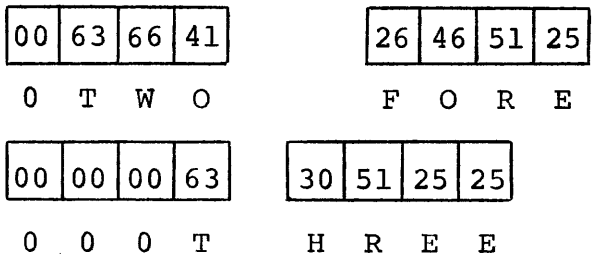
General Syntax

$nR\{<CHARA>\}_{1}^{8}$ where n is an integer 1|2|3|4|5|6|7|8 specifying the length of the character string.

Object Form



Examples



NOTE: The various types of constants defined on pages 5.0 though 11.0 will be referred to as <CONST>'s.

SIMPLE INTEGER VARIABLE <SIV>

Description

A single word of core storage can be given a name and then be used to store either an integer or from one to four BCD characters. The name consists of from one to eight non-blank alphanumeric characters--the first chosen from I,J,K,L,M, or N. (This can be overridden by a TYPE statement--see page 16.0.) The name actually refers to the address of the word of core storage and not to the data there.

Source Examples

```
I          KREAL          MOO
```

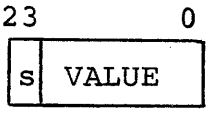
Examples in Context

```
J=I+1          DO 20 K=1,5          NINE=8
```

General Syntax

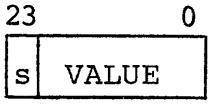
I|J|K|L|M|N{<LTR>|<DIGIT>}$_{0}^{7}$

Object Form

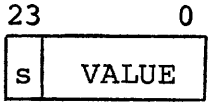


(See page 5.0.)

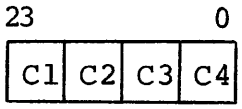
Examples



loc 73077↔I



loc 65002↔KREAL



loc 70633↔MOO, storing characters

(The address values used here were chosen arbitrarily.)

SIMPLE REAL VARIABLE <SRV>

Description

Two consecutive words of core storage can be given a name and then be used to store either a finite representation for a real number or from one to eight BCD characters. The name consists of from one to eight non-blank alphanumeric characters--the first other than I,J,K,L,M, or N. (This can be overriden by a TYPE statement--see page 16.0.) The name actually refers to the address of the first of the two words of core storage, not to the data there.

Source Examples

```
A          OLDVALUE          XBEFORE
```

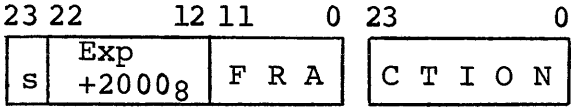
Examples in Context

```
XP1=X+1.          IF (TEST.LE.0.5) GO TO 751
XCOORD=XCOORD+STEP
```

General Syntax

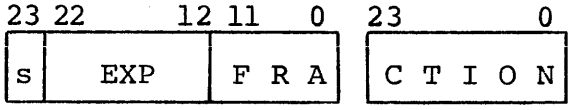
A|B|C|...H|O|P|...Y|Z{<LTR>|<DIGIT>}$_{0}^{7}$

Object Form

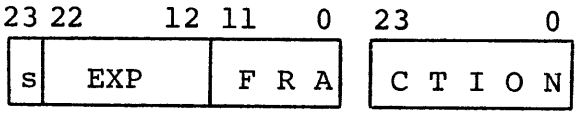


Thus "A" might be a symbolic name for locations 73101 and 73102, "OLDVALUE" for 65264 and 65265.

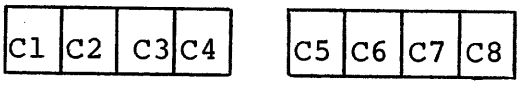
Examples



locs 73077 and 73100⟷A



locs 65002 and 65003⟷OLDVALUE



locs 70633 and 70634⟷XBEFORE,
storing characters

<NAME>

NOTE: Together <SIV> and <SRV> comprise the syntactic class

$$\text{<LTR>}\{\text{<LTR>}\,|\,\text{<DIGIT>}\}_0^7$$

which we will henceforth call <NAME>.

However, elements of <NAME> are used to name entities which are <u>not</u> variables. For example, main programs, functions, subroutines, labeled common blocks, type-other variables, BLOCK DATA and INCLUDE modules all use this syntactic construction. The semantic distinction is clear to FORTRAN from additional context.

SUBSCRIPT EXPRESSION <SSE>

Description

An expression used as a subscript must be either (1) a single-word integer constant, (2) a simple integer variable, perhaps multiplied by an integer constant, or (3) an expression as described in (2), plus or minus another single-word integer constant.

Source Examples

a) 6 b) 6*J c) 7*J+1 d) 6*J-3
e) J+1 f) J-1

Examples in Context

```
A(6)=5.          X(6*J,7*J+1)=SQRT(2.0)
```

General Syntax

<SWIC>|{<SWIC>*}$_0^1$ <SIV>{{+|-}<SWIC>}$_0^1$

NOTE: FORTRAN does not permit negative or zero subscripts. The <SWIC>s in these expressions must be non-negative.

Object Form

1) For subscripts expressed as constants, a "bias" or address adjustment is computed at compile time and added to the base address of the variable being subscripted.

2) For an expression involving a variable, a subroutine may be included within the object program to compute the appropriate address adjustment, given the current value of the variable.

The subroutine for a particular variable used in a subscript expression is re-executed each time the value of the variable

is changed within that program. BEWARE CHANGING INDEX VARIABLES IN SUBPROGRAMS!

Examples

a) For the constant <SSE> of six, an address adjustment value of +5 is computed at compile time. (The sixth element has address = base +5.)

b) For 6*J a subroutine (to be called each time the value of J is changed in the running object program) is included, and the subroutine's output is put in an auxiliary integer variable. This value is loaded into one of the three index registers when reference is made to an element of the array.

c) For 7*J+1 another internal variable is used at run time to store 7*J. A compile-time address adjustment is contributed by the additive constant.

SUBSCRIPTED VARIABLE <SUBVAR>

Description

A single name can have the effect of referring to the locations of more than one value. The name then refers to the first word of a block of contiguous core storage locations; subscript expressions are used to select a particular element of the block. The block or "array" may be thought of as having one, two, or three dimensions.

The total size of the block of core storage to be reserved for the array must be declared before the first executable statement by one or more of the following:

Declaratives

	DIMENSION	(max size = 32767 words)
	COMMON	
	COMMON/<NAME>/	
Type Statements	CHARACTER	(max size = 16384 characters)
	INTEGER	
	REAL	
	INTEGER2	
	TYPE <NAME>	(<SWIC>)

Declarative Source Examples

i)	DIMENSION	M(2,3,5),A(2,3,5)
ii)	COMMON	M(2,3,5),A(2,3,5)
iii)	COMMON/EXPL/	M(2,3,5),A(2,3,5)
iv)	CHARACTER	M(2,3,5),A(2,3,5)
v)	REAL	M(2,3,5),A(2,3,5)
vi)	INTEGER	M(2,3,5),A(2,3,5)
vii)	INTEGER2	M(2,3,5),A(2,3,5)

Declarative General Syntax

<DECL><DSUBVAR> {,<DSUBVAR>}

where <DECL> = DIMENSION|COMMON|COMMON/<NAME>/|CHARACTER|
INTEGER|REAL|INTEGER2|DOUBLE PRECISION|TYPE
<OTHER> (<SWIC>)

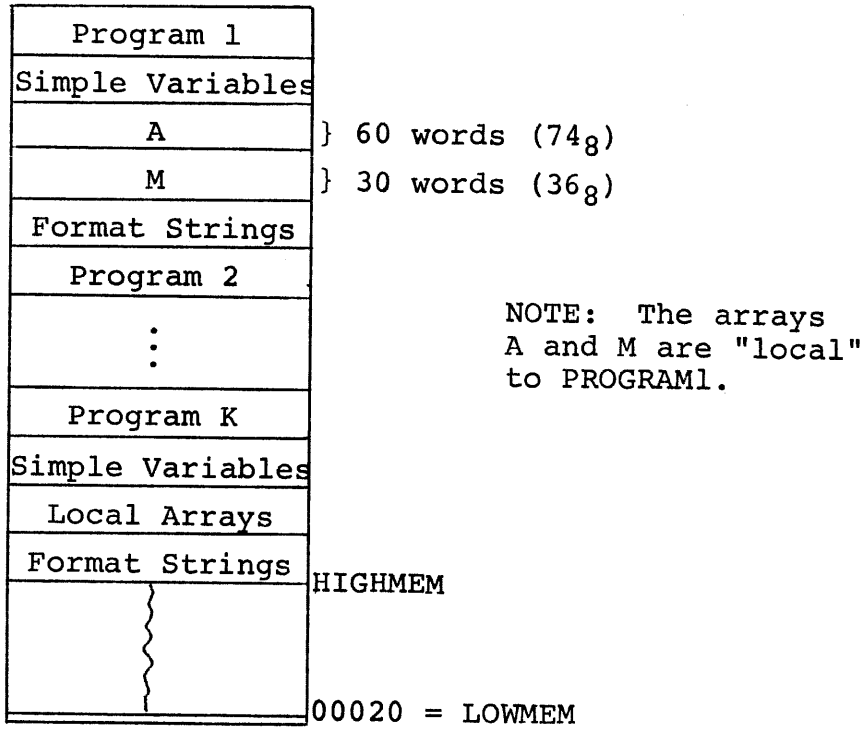
and <DSUBVAR> = <NAME> (<SWIC>{,<SWIC>}$_0^2$)

The integer constants, <SWIC>s, must all be unsigned, and their product times number of words per variable must be at most 32768.

Object Form

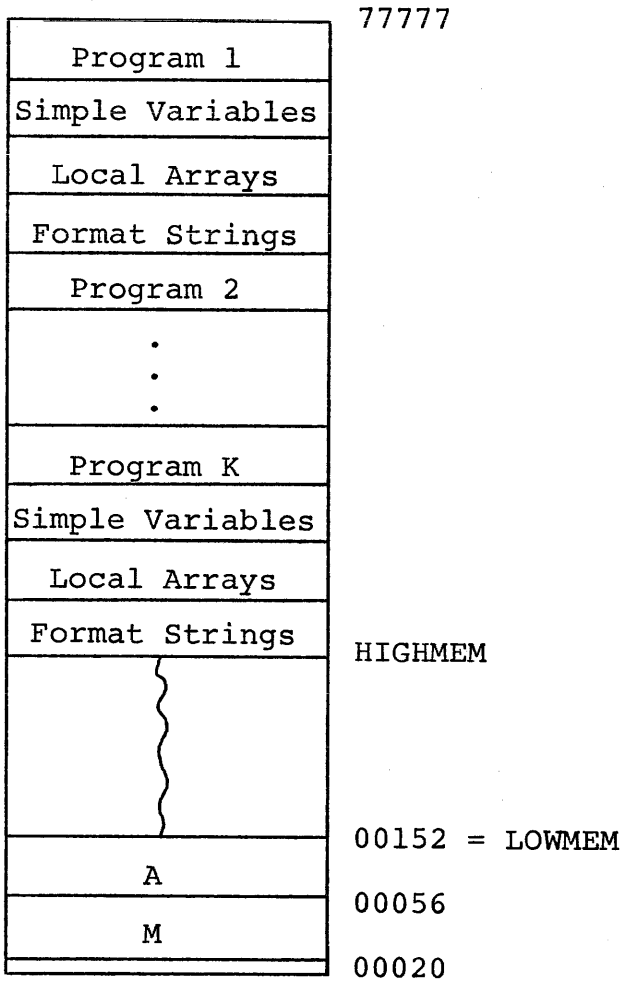
i) DIMENSION. An array which is specified in a DIMENSION statement, and not further specified in a COMMON statement, is local to the program in which it is declared and can be referenced by other subprograms only if the name (location) is passed as a parameter in the calling sequence.

For DIMENSION M(2,3,5),A(2,3,5) the following storage allocation results:

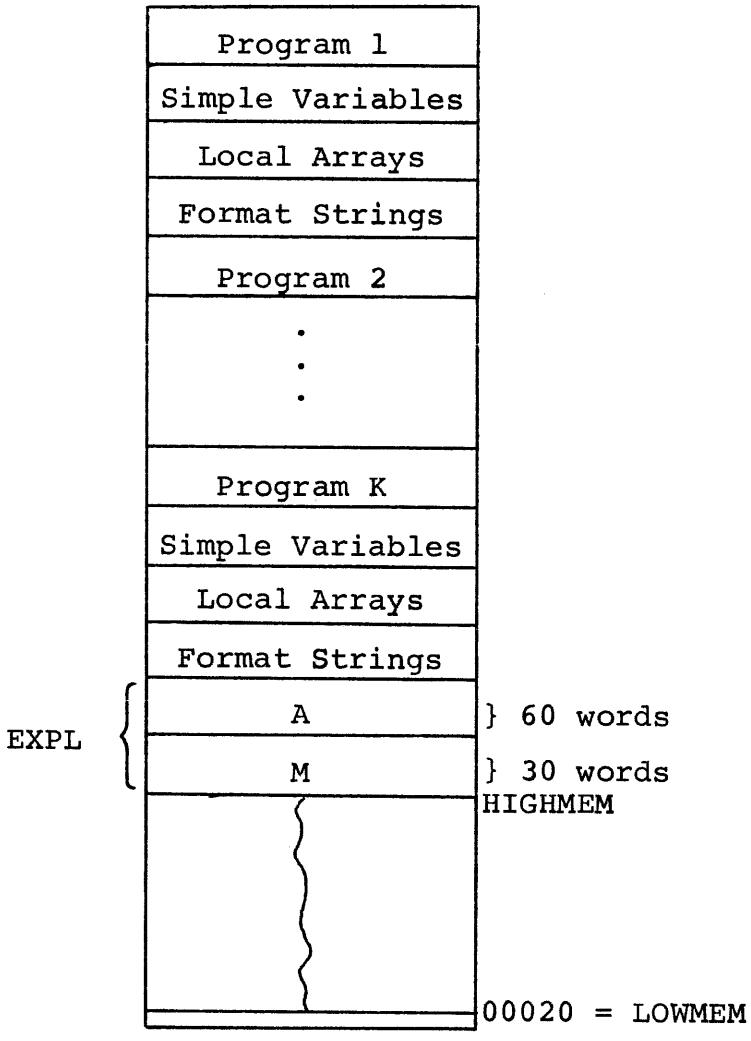


ii) If the name of a variable (subscripted or simple) is mentioned in a COMMON or COMMON/<NAME>/ statement, then that variable is located in numbered or labeled COMMON, respectively. The term "numbered COMMON" means that elements have actual run-time addresses computable from their relative position in the list of names used in the COMMON statement.

In particular, the first location in numbered COMMON has absolute machine address = 20_8.



iii) OS-3 FORTRAN permits the use of labeled common blocks. There may be an arbitrary number of these used by a system of subprograms. However, to avoid confusing the loader, the names of all labeled common blocks and subprograms must be distinct. Labeled common blocks will be preset with data as the subprograms are loaded if appropriate DATA statements are used in the source text. (See page 19.0.)

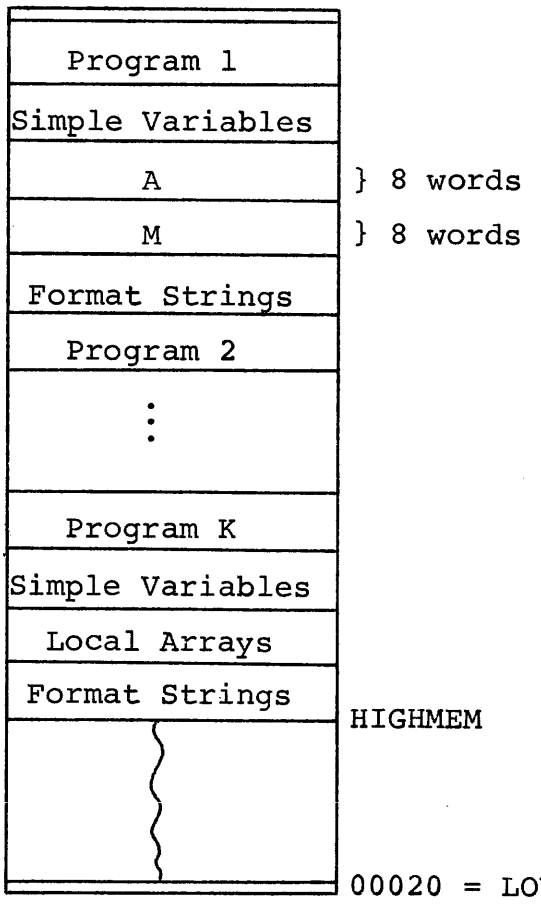


Some rules of thumb regarding labeled common include:

1) Variables appear in core in the order they were listed in the COMMON statement. (This is not generally true for DIMENSIONed variables.)

2) The first subprogram loaded that uses a particular labeled common block determines its length. No subsequently loaded program can change the length of the block.

iv) Each word of a simple or subscripted variable declared in a CHARACTER statement is subdivided into four six-bit fields, each of which is addressable by the FORTRAN program. Thus, M(1,2,3) in the example below is the fifteenth character in the array and occurs as the third field in the fourth word of M.



v) Same as i) except M and A are each 60 words long.

vi) Same as i) except M and A are each 30 words long.

vii) Same as v), and the compiler generates double precision integer arithmetic instructions for calculations involving elements of M or A.

SUBSCRIPTED VARIABLE
(Revisited)

General Syntax

<SUBVAR> = <NAME>(<SSE>{,<SSE>}$_0^2$)

Source Examples

i) M(1), M(2), A(1), A(2)
ii) M(1,1,1), A(1,1,1)
iii) M(1,2,3), A(1,2,3)
iv) M(I+1,J,2*K+3)
v) A(I+1,J,2*K+3)

Object Form

If a <SUBVAR> appears with fewer <SSE>s specified than were used in the <DECL> which established its dimensions, then the missing <SSE>s are assumed to be one. Thus M(1) and M(1,1,1) are equivalent; M(2) is equivalent to M(2,1,1).

For an <SSE> that is a constant, the address is computed at compile time to be

ADDRESS OF FIRST ELEMENT - 1*(WORDS PER ELEMENT) + <SSE>

For an integer or character array, an element referenced by a single <SSE> of the form

<NAME>{{<+|->}<SWIC>}$_0^1$

is accessed at run time by loading the value of the <SIV> into an index register. The + or - <SWIC> portion is handled as an address adjustment, computed at compile time. Since index registers are 15 bits long, the maximum size of an array is 2^{15} = 32768 words. NOTE: Due to the design of the 3300, character arrays should be no longer than 16384.

Reference, by means of a <SSE> involving a variable, to an element in an array, where the single item size is not one

word or one character, requires that the value of the variable be multiplied by the single item size before the index register is loaded.

For an array M with dimensions d_1, d_2, and d_3, reference to M(I,J,K) is accomplished by computing

$$(I-1) + (J-1)*d_1 + (K-1)*d_1*d_2$$

and storing this value in a memory location. The component of this sum for I is changed only when I is changed--similarly for J and K.

i) a. LDA/STA M

b. LDA/STA M+1

c. LDAQ/STAQ A

d. LDAQ/STAQ A+2

NOTE: LACH and SACH instructions are used if M is type character.

ii) Same as i) a and i) c.

iii) Assuming M and A are dimensioned (2,3,5), we have

a. LDA/STA M+14 NOTE: 14=(1-1)+(2-1)*2+(3-1)*2*3

b. LDAQ/STAQ A+28 28=14*SINGLE ITEM SIZE

iv) Three run-time subroutines will have computed $I+J*d_1 + (2*K)*d_1*d_2$ and stored this in a memory location with a name of the form IFN.xx (visible only on assembly language listing--A output).

The compile-time address adjustment would be $(1-1) + (0-1)*d_1 + (3-1)*d_1*d_2 = +10$. Access to the correct element of the array is thus accomplished by

```
LDI      IFN.xx, b
LDA/STA  M+10, b
```

v)
```
        LDI       IFN.yy, b'
        LDAQ      A+20, b'
        STAQ
```

where CONTENTS(IFN.yy)=2*CONTENTS(IFN.xx).

Again note the use of the 15-bit index registers to realize the displacement portion of a subscripted memory reference. Thus, at most 32768 words can be addressed in this way.

<VAR> AND <VARLIST>

NOTE: The syntactic classes <SUBVAR>, <SIV>, and <SRV> together comprise the class <VAR>.

A string of <VAR>s, separated by commas, will be denoted by <VARLIST>. That is

<VAR> := <SIV>|<SRV>|<SUBVAR>

and

<VARLIST> := <VAR>{,<VAR>}

EQUIVALENCE DECLARATIVE

Description

<NAME>s equated in an EQUIVALENCE statement all refer to the same physical core location at run time. A particularly useful application of this technique on the 3300 is to EQUIVALENCE a CHARACTER array to a word array.

Source Examples

```
EQUIVALENCE (ZIP,ZAP)
```

Examples in Context

```
DIMENSION       IWORDS(200), WORDS(100)
CHARACTER       CHARS(800)
COMMON          CHARS
EQUIVALENCE     (IWORDS, WORDS, CHARS)
```

General Syntax

```
EQUIVALENCE (<VARLIST>) {,(<VARLIST>)}
```

Object Form

EQUIVALENCE is a compiler control command, producing nothing overt in the object code. During compilation, all the symbols included between pairs of parentheses are made to refer to the same run-time core address, if possible.

Errors which make this impossible include:

1) EQUIVALENCEing two elements already in COMMON.
2) Extending a COMMON block backwards beyond its beginning. A COMMON block may be extended forward past its present end, however.
3) "Rearranging COMMON."

The following example violates all of these rules:

```
PROGRAM BOMB
COMMON       X(10), JAKEX(20), Q
EQUIVALENCE  (JAKEX(10),X)    violates 1 and 2
EQUIVALENCE  (Q,R), (R,X)     violates 3
      :
```

The diagnostic "EQUIVALENCE RELATION ERROR" would be issued for both statements.

For the "IN CONTEXT" example given above, IWORDS(1)=WORDS(1)= CHARS(1) through CHARS(4)=physical core address 20_8, since CHARS brings all these arrays into numbered COMMON. In general, WORDS(K)=IWORDS(2*K-1)=CHARS(8*K-7), so the same data can be accessed one or two words or one character at a time.

TYPE OTHER ARITHMETIC

Description

Type OTHER arithmetic allows the user to write his own arithmetic package and interface it with FORTRAN.

The declaration is:

TYPE <NAME>(<SWIC>) <VARLIST>

where <NAME> is the name of the type OTHER arithmetic, and the <SWIC> gives the number of words per variable.

Examples

TYPE INTERVAL (4) A,ZOT,BUG(12)
Note that BUG occupies 48 words of storage.

TYPE KLUDGE(8) QQQ,VVV,GROSS
NOTE: Instead of using the AQ and other hardware registers, the compiler requires the user to establish a type OTHER accumulator in core. Operations such as load, store, add, multiply, etc. are compiled as subroutine calls to user-supplied subprograms.

However, conditional statements (IFs) use the Machine-A register for tests; so after each operation (including load and store), the hardware-A register must indicate if the value of the type OTHER accumulator is zero (=0), negative (<0), or positive (>0). Remember that the result of arithmetic operations must never be -0. One way of accomplishing this is to leave 0, -1, or 1 in the A register for testing.

General Syntax

TYPE <NAME>(<SWIC>) <VARLIST>

Object Form

The compiler generates the following code:

```
RTJ     NN.opXY     (user-supplied subroutine)
77      Location of operand (may be index modified
        for array).
RETURN  here with condition code in A register
        (user's responsibility).
```

X is the mode of the operand and Y is the mode of the accumulator. Example: Put a real variable into the type OTHER accumulator (type KLUDGE arithmetic).

```
RTJ     KL.LDRO
77      Address of real value to be loaded.
RETURN  here with value in type OTHER accumulator,
        condition code in hardware-A register.
```

Type specifiers are:

```
R       real
I       integer
J       integer2
X       character
```

The OP portion of the RTJ address field is:

```
LD      for load
LN      for load negative
ST      for store
AD      for add
SB      for subtract
MU      for multiply
DV      for divide
EX      for exponentiate
CM      for complement
```

DATA STATEMENT

Description

Local program variables and variables in labeled common may be preset at load time if a DATA statement is used in the source program. The DATA statement does not generate executable instructions in the object program. Elements which are preset in this way may be changed by executed instructions in the running object program, but a DATA statement cannot be "re-performed." Like all declaratives, DATA statements must occur in the source program before the first executable statement.

NOTE: OS-3 FORTRAN DATA statements are very non-ASA standard. Beware!

Source Examples

```
                    i)  DATA (I=2), (X=3), (Y=4.0), (J=5.0)
                   ii)  DATA (A(1,1)=1.0), (A(2,2)=1.0),
                             (A(1,2)=0.0), (A(2,1)=0.0)
Assume A
dimensioned
   2 x 2           iii) DATA (((A(I,J),I=1,2),J=1,2)=1.0,
                             0.0, 0.0, 1.0)
                    iv)  DATA (A=1.0, 0.0, 0.0, 1.0)
```

Then ii), iii), and iv) are equivalent; each sets A(1,1)= A(2,2)=1.0 and A(1,2)=A(2,1)=0.0, so that A is a 2 x 2 identity matrix.

General Syntax

DATA (<ASGN>) {,(<ASGN>)}

where each <ASGN> is an assignment statement of the form <WAS> = <REPCON>. A replicated constant, <REPCON>, has the form <CONST>|<SWIC>(<CONST>). The word address specifier, <WAS>, has the form <VAR>{<IMPDO>} with the implied DO's

nested at most three deep. (See page 27.0 for a discussion of the implied DO's.) NOTE: Here, the increment for each DO must be left unspecified (to default to one).

Object Form

The <WAS> specifies one or more word addresses. Each <CONST> is converted to internal form, and the compiler generates special records in the relocatable object deck so that when the deck is loaded, the area of core beginning at the specified word address is initially loaded with the converted constants.

If the mode of the <VAR> is integer or character, the compiler increments its storage allocation counter by one word for each constant to be stored. For real variables, the counter is incremented by two.

NOTE: The compiler does not compute the amount of storage to allocate based on the length of the constant to be stored.

Constants are packed according to their mode--not according to the mode of the variable. Thus DATA (A=2) results in the left word of A being loaded with 00000002_8. The storage allocation counter in the compiler is advanced by two, so the second word of A is not changed.

NOTE: Do not initialize variable in numbered (unlabeled) common, because if an overlay is created, the numbered common area is not written out.

Examples

i) Variables I and Y will be properly initialized. X will contain 00000003, 00000000_8, and J will contain the left half of the representation for floating-point five, i.e., 20035000_8.

ii) Works, provided A has been dimensioned at least 2 x 2.

iii) Equivalent to ii).

iv) The compiler does not complain if A is not dimensioned, but only two words of core are allocated instead of the eight that are needed.

ARITHMETIC EXPRESSION <AEX>

Description

The following are arithmetic expressions:

<OPERAND>
1) an arithmetic or right-justified numeric character constant
2) a variable, simple or subscripted
3) a function reference
4) one of these (1-3), followed by an arithmetic operation symbol

{ + - * / ** } = <OPERATOR>

followed by an arithmetic expression
5) an arithmetic expression inside parentheses
6) an arithmetic expression preceded by a unary minus (-)

The precedence of the arithmetic operations is

highest	**	exponentiation
	-	unary minus
	* and /	multiplication and division
	+ and -	addition and subtraction

Parentheses can be used to override these precedence rules.

The value of the expression will be of the same type as the highest operand type occurring. The order is

highest	NON-STANDARD	TYPE <OTHER>
	REAL	
	INTEGER2	
	INTEGER	
lowest	CHARACTER	

Source Examples

```
 i)  1
ii)  2.0
```

iii) X

iv) SQRT (X + Y)

v) A + B - C * C/E ** ABS(F)

vi) -A + B - C * D/E ** ABS(F)

vii) (A + B - C) * D/E ** ABS(F)

viii) -(A + B - C) * D/E ** ABS(F)

Examples in Context

```
DISCRIM = SQRT(B**2 - 4.*A*C)     IF (B*B - 4.*A*C) 10, 20, 30
```

General Syntax

$\{+|-\}_{0}^{1}$<OPERAND>{<OPERATOR><AEX>}|(<AEX>)

Object Form

An arithmetic expression which includes one or more operator symbols will be compiled into a series of executable machine instructions. A single constant or variable is represented as shown on pages 5.0 and 12.0.

Examples

i) An integer constant--see page 5.0.

ii) A real constant--see page 8.0.

iii) A simple real variable--see page 13.0.

iv)
```
LDAQ        X
FAD         Y
STAQ        TEMP
RTJ         SQRT
77          TEMP
```

v)
```
LCAQ        C           -C
FMU         D           TIMES D
STAQ        TEMP1       IN TEMP1
RTJ         ABS
77          F           ABS(F)
STAQ        TEMP2       IN TEMP2
LDAQ        E
RTJ         POWER       E**ABS(F)
77          TEMP2
STAQ        TEMP2       IN TEMP2
```

```
        LDAQ        TEMP1
        FDV         TEMP2       DO THE DIVISION
        FAD         B           ADD B
        FAD         A           ADD A
```

vi) Same as v) except final instruction becomes FSB A.

In both cases note the higher precedence of ** over / and the call upon a library subroutine to perform exponentiation to a real power.

vii)
```
        LDAQ        A
        FAD         B
        FSB         C           A+B-C
        FMU         D           TIMES D
        STAQ        TEMP1       IN TEMP1
        RTJ         ABS
        77          F           ABS(F)
        STAQ        TEMP2       IN TEMP2
        LDAQ        E
        RTJ         POWER
        77          TEMP2       E**ABS(F)
        STAQ        TEMP2       IN TEMP2
        LDAQ        TEMP1
        FDV         TEMP2
```

Note that the parentheses override the precedence conventions, making (A+B-C) a subexpression to be evaluated before multiplication by D.

viii) Same as vii) plus two additional instructions to complement result. (Arithmetic is one's complement, so the negative of a number is simply its bit-wise complement.)

LOGICAL EXPRESSION <LEX>

Description

A logical expression is one of the following:

1) a single arithmetic expression
2) two arithmetic expressions "joined" by one of six relational operators: {.EQ. .NE. .LT. .GT. .LE. .GE.} = <RELOP>
3) the unary logical operator .NOT. followed by a logical expression
4) two logical expressions joined by either of the logical operators .AND. or .OR.

The precedence hierarchy for all FORTRAN operators is:

highest	function reference
	**
	unary minus
	* and /
	+ and -
•	.EQ. .NE. .LT. .GT. .LE. .GE.
	.NOT.
	.AND.
lowest	.OR.

The logical constants .TRUE. and .FALSE. are <u>not</u> recognized in source statements. Use 1 and 0 instead.

Source Examples

i) P.OR..NOT.Q.AND.R
ii) P.LE.5.OR.X-Y.GT.A.AND..NOT.Q
iii) A.EQ.B.OR..NOT.C.NE.D.AND.E.LT.F+G*H**ABS(X-Y)

Examples in Context

```
IF.(I.GT.5.OR.I.LE.0)  STOP
```

Note that I.GT.5.OR..LE.0 is <u>not</u> a <LEX>.

IF (I.EQ.9.OR.J.EQ.9) GO TO 5 IF (I.OR.J.EQ.9) GO TO 5

Note that these two are not equivalent.

General Syntax

<AEX>{<RELOP><AEX>}$_0^1$ | .NOT.<LEX> |

<LEX>.AND.<LEX> | <LEX>.OR.<LEX>

Object Form

The CDC 3300 does not have a distinct representation for logical values. Any single-word operand appearing in a source statement will be handled properly at run time; multiple-word operands will not.

The logical value FALSE is represented internally by zero (all 24 bits = OFF) or -0 (all bits = ON). Anything else represents TRUE.

Examples

i) P.OR..NOT.Q.AND.R

```
        LDAQ            P
        AZJ,NE          SUCCESS
        LDAQ            Q
        AZJ,NE          FAIL
        LDAQ            R
        AZJ,NE          SUCCESS
FAIL ~~~~
```

Note that the .NOT. operates only on Q. All operands must be one word long or the AZJ,NE instructions may not work properly.

ii) P.LE.5.OR.X-Y.GT.A.AND..NOT.Q

```
        LDAQ            P
        RTJ             ADDMIXED
        77              ADDR OF 5
        AZJ,GE          SUCCESS
        LDAQ            X
        FSB             Y
        COMPLEMENT
        FAD             A
```

```
      AZJ,GE          FAIL
      LDAQ            Q
      AZJ,EQ          SUCCESS
FAIL  ~~~~
```

iii) A.EQ.B.OR.NOT.C.NE.D.AND.E.LT.F+G*H**ABS(X-Y)

```
      LDAQ            A
      FSB             B
      AZJ,EQ          SUCCESS
      LDAQ            C
      FSB             D
      AZJ,NE          FAIL
      LDAQ            X
      FSB             Y
      STAQ            TEMP1
      RTJ             ABS
      77              TEMP1
      STAQ            TEMP1
      LDAQ            H
      RTJ             POWER
      77              TEMP1
      FMU             G
      FAD             F
      COMPLEMENT
      FAD             E
      AZJ,LT          SUCCESS
FAIL  ~~~~
```

FUNCTION REFERENCE <FUNREF>

Description

A function subprogram returns a single value (real, integer, or character) to the calling program. Thus the construction

<NAME>(<AP>{,<AP>}$_{0}^{62}$)

really stands for a single value, and so can be used as an operand in an arithmetic or logical expression.

The addresses of the actual parameters, <AP>s, are passed by the calling program to the function subprogram. Thus the subprogram may change the values in these locations. An index variable changed in this way will not be updated in the main program.

An actual parameter may be simple enough to fall within the definition of a subscript expression (see page 15.0). In this case, there is no syntactic difference between a function reference and a subscripted variable. FORTRAN assumes that X(I) occurring in an expression means "transfer control to a subprogram (or entry point) called X, passing the address of I as a parameter," unless X has been declared to be a subscripted variable by one or more of the declaratives listed on page 16.0.

In general, actual parameters may be arithmetic expressions, function references or subprogram names. Logical expressions are not allowed. Subprogram names passed as parameters must be declared "external" to the calling program by a statement of the form

EXTERNAL <NAME>{,<NAME>}

so that the compiler knows they are not local variables.

Source Examples

```
SQRT(B*B-4*A*C)                    SMURD(ANGLE,COS)
```

Examples in Context

```
Z=SQRT(ABS(SMURD(X,SIN)))+1.55
```

Here ABS and SQRT are library subprograms which will be called in order. SMURD is presumably a user-written function subprogram (see page 28.0) which expects its second parameter to be a subprogram name. Since there is no syntactic clue that SIN is not a local variable, the name must be declared EXTERNAL in the calling program, or it will be treated as a local variable.

General Syntax

<NAME>(<AP>{,<AP>}$_{0}^{62}$)

Object Form

Each function reference generates the following code in the main program:

```
RTJ             <NAME>
77              address of actual parameter 1
77              address of actual parameter 2
 :
 :
77              address of last actual parameter ≤ 63
```

See page 28.0 for details at the receiving end of the calling sequence.

Examples

i) SQRT(B*B-4*A*C)

```
LDAQ            B
FMU             B
STAQ            TEMP
ECHA            -4
RTJ             CONVERT
```

```
        FMU             A
        FMU             C
        FAD             TEMP
        STAQ            TEMP
        RTJ             SQRT
        77              TEMP
```

ii) SMURD(ANGLE,COS)

```
        RTJ             SMURD
        77              ANGLE
        77              COS
```

iii) Z = SQRT(ABS(SMURD(X,SIN)))+1.55

```
        RTJ             SMURD
        77              X
        77              SIN
        STAQ            TEMP
        RTJ             ABS
        77              TEMP
        STAQ            TEMP
        RTJ             SQRT
        77              TEMP
        FAD             ADDR OF 1.55
        STAQ            Z
```

Assignment

<ASSIGN>

Description

The value of a variable may be changed within a running object program by a statement of the form

<VAR> = {<VAR> = } = <EXPR>

where <EXPR> is an arithmetic or logical expression. The compiler handles mixed mode situations automatically.

Source Examples

```
 i)  X = Y = Z = 1.0
ii)  DISCRIM = SQRT(B*B-4*A*C)
iii) Z = I+J
 iv) P = Q.AND.R.OR..NOT.S
```

General Syntax

<VAR> = {<VAR> = } <EXPR> with <EXPR> = <AEX>|<LEX>

Object Form

The value of the expression is computed by in-line machine instructions and calls to function subprograms, and then stored in the location(s) named by the variable(s).

Examples

```
i)  LDAQ        Address of floating-point constant 1.0
    STAQ        Z
    STAQ        Y
    STAQ        X
```

ii)

```
LDAQ    B
FMU     B       B*B
STAQ    TEMP    IN TEMP
ECHA    -4
RTJ     FLOAT
FMU     A       -4*A*C
FMU     C
FAD     TEMP    PLUS B*B
STAQ    TEMP    IN TEMP
RTJ     SQRT
77      TEMP    CALL SQRT
STAQ    Z       ASSIGN TO Z
```

iii)

```
LDA     I
ADA     J
RTJ     CONVERT
STAQ    Z
```

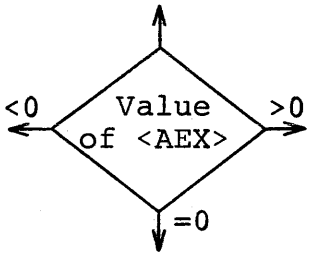
iv)

```
        LDAQ    Q
        AZJ,EQ  TRYS
        LDAQ    R
        AZJ,NE  SUCCEED
TRYS    LDAQ    J
        AZJ,EQ  SUCCEED
        ECHA    0
        UJP     *+2
SUCCEED ECHA    1
        RTJ     CONVERT
        STAQ    P
```

IF STATEMENT

Description

The original FORTRAN II conditional statement is a three-way branch of the form



The source statement for this has the form

```
IF (<AEX>)  <STNUM>,<STNUM>,<STNUM>
              ↑        ↑        ↑
             <0       =0       >0
```

FORTRAN IV relaxed the restrictions on the predicate portion of the IF statement considerably. The so-called "logical" IF statements have the forms

```
      IF (<LEX>)  <STMT>
or    IF (<LEX>)  <STNUM>,<STNUM>
                    ↑        ↑
                  True     False
```

The <STMT> in the first form can be any executable statement except a DO or another IF.

Source Examples

```
 i)  IF (X)  5,10,17
ii)  IF (A.OR.B.AND.C)  40,50
```

General Syntax

```
 i)  IF (<LEX>)  <STMT>     or
ii)  IF (<LEX>)  <STNUM>,<STNUM>
                   ↑        ↑
                 True     False       or
```

iii) IF (<AEX>) <STNUM>,<STNUM>,<STNUM>

↑ ↑ ↑

<0 =0 >0

In case i), the <STMT> cannot be a DO or another IF.

Object Form

For the arithmetic IF, the strategy is to compute the value of the <AEX>, leaving this in the A or AQ registers. Then AZJ instructions are used to test the sign bit of the accumulator.

Thus i) yields

```
LDAQ        X
AZJ,EQ      L.00010
AZJ,LT      L.00005
UJP         L.00017
```

The logical case is as follows:

ii)

```
LDAQ        A
AZJ,NE      L.00040
LDAQ        B
AZJ,EQ      L.00050
LDAQ        C
AZJ,NE      L.00040
UJP         L.00050
```

GO TO STATEMENT

Description

Execution of a FORTRAN program normally proceeds sequentially, statement by statement. A statement of the form

GO TO <STNUM> or
GO TO (<STNUM>{,<STNUM>}), <AEX>

will cause a transfer of control to the statement having the specified number. In the first case, the transfer is to the single statement number given.

In the second case the following occurs:

1) The arithmetic expression is evaluated and truncated to an integer.

2) Control is transferred to the first statement number in the list if the integer is ≤ 1, to the second statement number if it is 2, ..., to the i^{th} statement number if it is i,..., and to the last statement number in the list (say the n^{th} one) if the integer is $\geq n$.

Source Examples

```
GO TO 100          GO TO (100,200,300,400,500),SIN(ALPHA)
```

Note that both of these will always transfer control to statement 100.

General Syntax

GO TO <STNUM> | GO TO (<STNUMLIST>),<AEX>

where <STNUMLIST> := <STNUM>{,<STNUM>}

Object Form

For the simple GO TO, an unconditional jump instruction is compiled. For the "computed" variety, the <AEX> is evaluated

and truncated to an integer, left in A. This is decreased by two, and an AZJ,LT to the first statement number follows, then an AZJ,EQ to the second statement number, then another decrease of two, etc. An unconditional jump to the last statement number completes the code for the computed GO TO.

Examples

```
 i)  UJP          L.00100

ii)  RTJ          SIN
     77           ALPHA
     RTJ          TRUNCATE
     77           TEMP
     INA          -2
     AZJ,LT       L.00100
     AZJ,EQ       L.00200
     INA          -2
     AZJ,LT       L.00300
     AZJ,EQ       L.00400
     UJP          L.00500
```

DO STATEMENT

Description

FORTRAN provides a single statement to accomplish the three steps necessary to establish and control an iterative loop. These are:

1) Initialize a counting variable to some specified value.

2 and 3) Increment the counting variable by another specified value (defaults to one), and test for terminating condition. These may be done in either order.

The terminating condition for a DO loop is simply a check of the current value of the counting variable against another value specified in the source statement. The DO is "satisfied" if the current value is strictly greater than the terminating value. Note that in OS-3 FORTRAN the body of the loop is not performed before the first terminating test.

Source Examples

```
DO 10 KOUNTER=ISTART,IEND,ISTEP
```

Examples in Context

```
    DO 777 INEVER=1,INEVER,INEVER
    PRINT 20, INEVER
 20 FORMAT (X,I12)
777 CONTINUE
```

General Syntax

DO <STNUM> <SIV> = <IQ>,<IQ>{,<IQ>}$_{0}^{1}$

Here <IQ>:= <SWIC>|<SIV> is an integer quantity.

Starting Value

Termination Test Value

Step (Defaults to one)

Object Form

OS-3 FORTRAN generates a very nonstandard code for DO loops. In particular, the termination test is performed before the first execution of the body of the loop; so if the initial value of the counting variable is greater than the terminating value, the loop is completely bypassed. This is not the way IBM FORTRAN compilers handle DO statements. Beware if converting.

In general, the code goes

```
                LOAD        STARTVAL
                JUMP        AROUNDLOOP
LOOP            ------------------------
                  :
                  :
AROUNDLOOP      STA         KOUNTER
                SBA         TERMVAL
                AZJ,LT      LOOP
                AZJ,EQ      LOOP
```

Example

```
                ECHA        1
                UJP         AROUNDLOOP1
LOOP1           RTJ         INIT-FORMATTED-OUTPUT
                77          LUN
                77          FORMAT-NUM
                RTJ         OUTPUT-INTEGER
                77          INEVER
                RTJ         END-FORMATTED-OUTPUT
L.00777         LDA         INEVER
                ADA         INEVER
AROUNDLOOP1     STA         INEVER
                SBA         INEVER
                AZJ,LT      LOOP1
                AZJ,EQ      LOOP
```

Since INEVER-INEVER is always zero, the loop is "infinite." It prints the integral powers of 2 from 1 to 8388607 and then starts over again.

NOTE: Regular DO loops may be nested, at most, ten deep.

IMPLIED DO <IMPDO>

Description

A special form of the DO statement--the implied DO--may be used in I/O and DATA statements to "drive" the statement through a vector or array.

Source Examples

```
WRITE (61,100) (A(I),I=1,10)
READ (60,150) ((VALS(M,N),M=1,5),N=3,17,2)
DATA ((((XMTX(I,J,K),I=1,5),J=3,6),K=4,5)=40(1.0))
```

General Syntax

```
(<VARLIST>,<SIV>=<IQ>,<IQ>{,<IQ>})
              ↑     ↑     ↑     ↑
           Index Start Stop Step
```

Note that the entire construction is enclosed in parentheses and that a comma occurs between the last <VAR> and the <SIV>.

Implied DO's can be nested. That is

```
(<IMPDO>,<SIV>=<IQ>,<IQ>{,<IQ>})
```

is also an <IMPDO>. In I/O statements, nest depth is limited only by statement length limits.

Object Form

See page 26.0.

PROGRAM MODULE TYPES

EXECUTABLE MODULES

Description

Each collection of object program modules loaded for execution must contain exactly one primary transfer address and may contain a secondary transfer address as well. Both of these transfer points are provided by the FORTRAN compiler when a "main program" is compiled. In the source language, a main program is delimited by a PROGRAM <NAME> statement at the beginning and an END statement, which is the final record read by the compiler.

Two other types of FORTRAN source modules generate executable code:

1) The statements FUNCTION <NAME>(<FP>{,<FP>}$_{0}^{62}$) and END are used to delimit a function subprogram. The <NAME> of the function must be assigned a value during execution of the subprogram. This value is returned to the "calling" program as the value of the function. The <NAME> determines the mode of this value unless overridden by a TYPE statement.

2) A subroutine subprogram is delimited by the statements SUBROUTINE <NAME>{(<FP>{,<FP>}$_{0}^{62}$)}$_{0}^{1}$ and END. No value is attached to the <NAME> of a subroutine subprogram.

Either type of subprogram should contain at least one RETURN statement. Functions are invoked implicitly, as described on page 22.0. Subroutines are explicitly CALLed by a statement of the form

CALL <NAME>{(<AP>{,<AP>})}

In both constructions, the <FP>s are formal parameters. These are dummy <NAME>s which are used by the compiler only to determine the mode of each argument to be passed to the subprogram when it is activated by the calling program. Vector and array parameters must be dimensioned locally within the subprogram: The subprogram will compute subscript values based upon these dimensions, not from those given in the calling program. No attempt may be made to make a formal parameter into an actual, core-occupying variable with its own physical address. That is, <FP>s may not be mentioned in COMMON, EQUIVALENCE or DATA statements.

Source Examples

```
PROGRAM TEST
:
END
FUNCTION FRUMP(N,X)
:
RETURN
:
END
SUBROUTINE COEFS(A,B,SEE)
:
RETURN
:
END
```

General Syntax

PROGRAM <NAME>

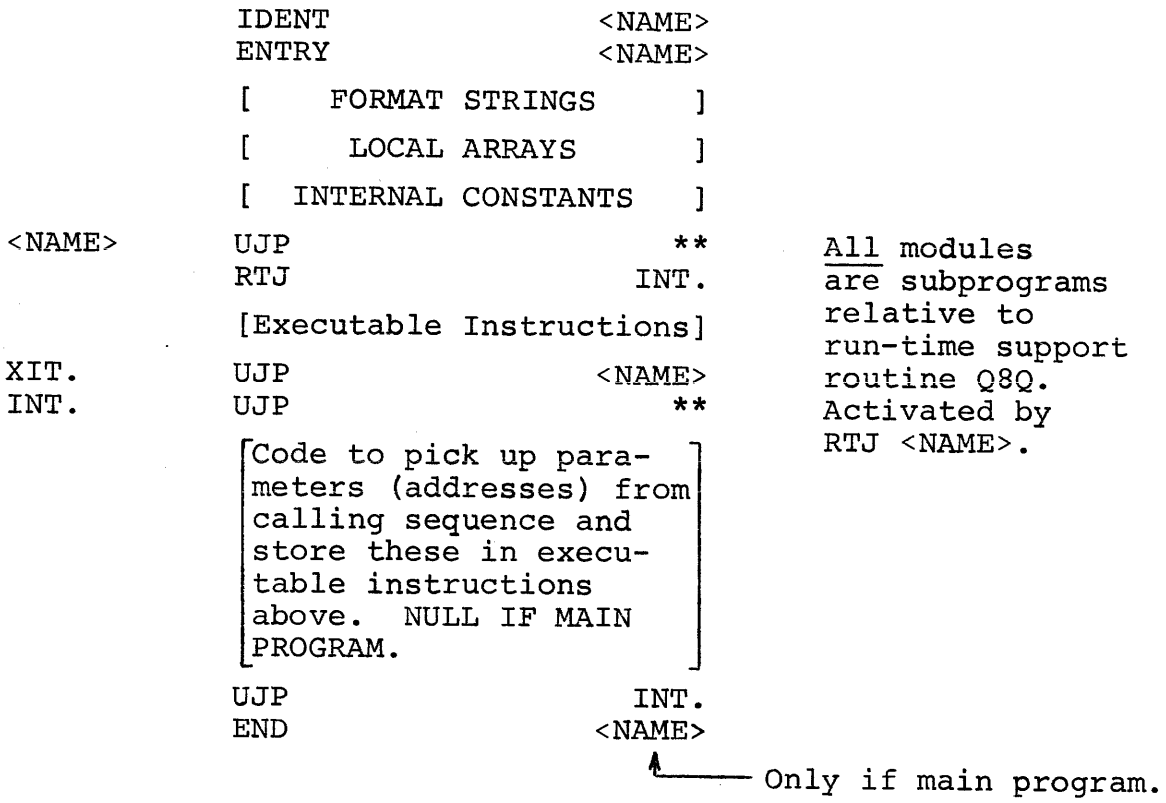
FUNCTION <NAME>(<FP>{,<FP>}$_{0}^{62}$)

SUBROUTINE <NAME>{(<FP>{,<FP>}$_{0}^{62}$)}$_{0}^{1}$

Object Form

Each module delimited by a PROGRAM, FUNCTION, or SUBROUTINE statement is a new task for the compiler, which "forgets" everything it ever knew about previous modules.

The general form of an executable module produced by FORTRAN is as follows:



NOTE: In OS-3 FORTRAN, a subprogram can have more than one entry point. This is specified in the source language by a statement of the form ENTRY <NAME>. The statement is essentially copied straight through to the assembly-level code. No parameter string is specified: The same initialization is performed as is done when the subprogram is activated via its main entry point, so the calling sequences must be identical in terms of number and types of parameters.

Pitfalls of Subprogramming

1. As shown above, FORTRAN-produced subprogram modules do not save and restore index registers.

2. Each reference to a formal parameter within a subprogram requires an SWA or SCHA instruction to set up the actual run-time address within the executable code. Thus, the length of the INT. portion of the subprogram is directly related to the number of <FP>s referenced within the module. If a particular parameter must be referenced more than once, copy it into a local variable within the subprogram, or put it in COMMON.

3. Alteration of a <SIV>, used as part of a <SSE> within this module where the <SIV> is defined, causes re-execution of the code which updates the multiples of the <SIV> that index data elements of length ≠ 1. If one of these variables is changed within a different module, this updating is <u>not</u> performed. Do NOT change the value of an index in a subprogram other than the one where it is defined.

4. The remarks in 3) imply that FORTRAN passes the actual run-time address of each parameter when a subprogram is invoked. Consider this example.

```
    PROGRAM  CHANGE2
    CALL BUMP(2.)
    X=2.
    WRITE (61,100) X
100 FORMAT (XF6.2)
    END

    SUBROUTINE BUMP(Z)
    Z=Z+1.
    RETURN
    END
```

The value printed by the WRITE statement in CHANGE2 will be "3.00." The actual address of the constant 2. in CHANGE2 is passed to BUMP, which changes the 2. to 3.

NON-EXECUTABLE MODULES: BLOCK DATA AND DEFINE

Description

A BLOCK DATA subprogram must be used in ASA-standard FORTRAN to initialize variables in labeled COMMON. OS-3 FORTRAN processes these correctly, but labeled COMMON may also be preset by ordinary DATA statements, so the BLOCK DATA construct presumably would not occur except in an imported program.

OS-3 FORTRAN also has a kind of built-in editor/text inserter in the DEFINE/INCLUDE statement combination. The intended use for this is the case of many subprograms each requiring identical declaratives--COMMON statements in particular. In such a case, the programmer may code the declaratives once, delimit them by the statements DEFINE <NAME> and END, "compile" this "program" ahead of all others in his system, and then have the text of the declarative statements included in-line in any program module by using the single statement INCLUDE <NAME>. The text is stored in core and so exists only during a given call of the compiler. The DEFINE subprogram must be "re-compiled" each time FORTRAN is recalled from the system library.

Limitations

BLOCK DATA subprograms may contain only declaratives: no executable statements are accepted. A DEFINE subprogram may not contain another DEFINE or an INCLUDE. Also note that DEFINE subprograms steal core from the compiler.

Examples

i)
```
BLOCK DATA
COMMON/ZORK/A,B,ISEE
DATA (A='THIS SETS UP A,B AND ISEE')
END
```

ii)
```
DEFINE EGGSWOPL
COMMON A(50),IA(100),CA(200)
EQUIVALENCE (A,IA,CA)
END

PROGRAM SHOWIT2M
INCLUDE EGGSWOPL
   :
END

SUBROUTINE CALLME
INCLUDE EGGSWOPL
   :
RETURN
   :
END

FUNCTION ANYTIME(YOU,LIKE)
INCLUDE EGGSWOPL
DIMENSION ZIP(5)
   :
ANYTIME=0.0
   :
RETURN
   :
END
```

INPUT STATEMENTS

Description

The values of variables can be established or changed within a running object program by reading data into them from peripheral units. This may be done in several ways:

1) Information may be copied, exactly as it appears on the peripheral device, into a contiguous block of core storage set up by the user. The BUFFER IN statement is used to accomplish this.

2) The information may be transferred into a buffer which is invisible to the user and then be moved by run-time support routines into locations named by individual variables. This "re-distribution" may be accomplished with or without the help of a FORMAT statement.

   If no FORMAT is referenced, the information is simply moved word-by-word into the list of locations given in the READ statement. If FORMAT control is specified, each item is converted into the desired internal representation (if possible) as it is moved. This is accomplished at run-time by interpretation of the character string of specifying codes, as given in the FORMAT statement.

BUFFER IN

Description

This statement causes one physical record to be transferred from an external device into a contiguous block of core storage. The result is a bit-by-bit copy of the record in core, with the external device advanced one record.

If the input device is a magnetic tape, the user must correctly specify the actual parity of the tape. For other types of devices, OS-3's input routines automatically read the data with correct parity checking.

Source Examples

```
BUFFER IN (40,0)(IARAY(1),IARAY(1024))
BUFFER IN (13,1)(BTAB(51),BTAB(100))
```

General Syntax

```
BUFFER IN (<IQ>,<IQ>)(<VAR>,<VAR>)
            ↑   ↑
          LUN  0 indicates BCD, even parity and
               1 indicates odd parity binary records.
```

Object Form

All FORTRAN input is performed by calls to run-time library routines.

Examples

For the general case we have

```
ENA         LUN
RTJ         BUFIN
77          PARITY
77          START ADR
77          END ADR
```

NOTE: The user specifies the number of words to be buffered in. If the record is shorter than this, the data transfer

stops at the end of the physical record, and the remainder of the user's buffer is left unchanged. If the record is longer than the number specified, data transmission stops when the requested number of words have been transferred, with the external device moved forward to the next physical record.

SPECIAL CAUTION: Buffering IN a new value into a <SIV> used as part of a <SSE> will <u>not</u> cause proper updating of the multiples of the <SIV> which are used to access data elements of length $\neq 1$.

Description

A READ statement of the form

READ (<IQ>) {<WAS>{,<WAS>}}

or

READ TAPE (<IQ>) {<WAS>{,<WAS>}}

transmits one logical record from an external device into individually named locations in core storage. The information is first read into a buffer, which is invisible to the user, and then moved by run-time support routines into the non-contiguous locations named by the variables in the <LIST>.

No "internalizing" is performed on the data; it is stored in the elements of the <LIST> exactly as it appeared in the input record.

Data to be read in this way should have been written by an unformatted binary WRITE statement (see page 30.02).

Source Examples

```
READ (40) ZIP,IVAL,((ARAY(I,J),I=1,IVAL),J=1,10)
```

Object Form

Again, everything is done by calls to run-time support sub-routines.

Description

A statement of the form

READ (<IQ>,<STNUM>) {<WAS>{,<WAS>}}

↑ ↑

LUN Format Statement Number

causes a physical record to be read from the specified LUN into a buffer invisible to the user. This data is then converted under control of the format codes and moved to the addresses stipulated by the word-address specifiers.

READ <STNUM>, is a synonym for READ (60,<STNUM>).

FORMAT SPECIFIER NOTES

Notation throughout this section on formatted input/output has been standardized. The following are definitions of the symbols used:

- n - How many times to repeat the specifier.
- w - The width of the field in characters.
- d - The number of digits to the right of the decimal point, or the negative power of 10 to multiply the number by ON input (see E specifier).
- s - A string of BCD characters.

Sample format strings look like

1) FORMAT (10X,I5,F13.2,4HTEST,I2)
2) FORMAT (X,2I5,3F10.2)
3) FORMAT (X,2(I3,X,I2))
4) FORMAT (X,3(I3),1X)

If the number of items in the variable list is the same or less than the number of specifiers, then the record is finished when the last variable is processed. However, if the format list is exhausted before the variable list is finished, the record is output or a new record is read and processing starts again at the right-most left parenthesis of the format statement. This process repeats until the variable list is exhausted.

All format specifiers should be separated by commas.

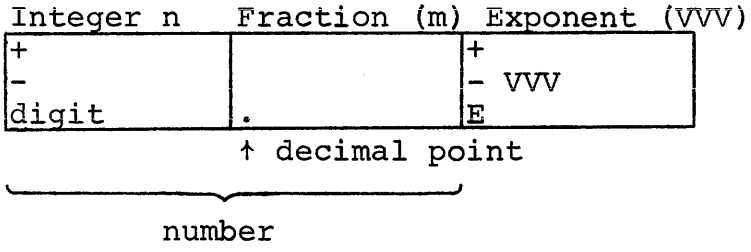
nEw.d INPUT

The Ew.d specifier causes the data to be stored in the corresponding real variables in the input list. Leading blanks are ignored and trailing blanks are treated as zeros.

The field for input may include integer, fraction, and exponent in one of the following forms:

n	.mEVVV
n.m	n.mEVVV
.m	n.m±VVV
EVVV	.m±VVV

Input subfield format is:



The total range of the exponent and number must be between -10^{308} and $+10^{308}$.

If no decimal point is present in the input field, then d specifies a negative power of 10, and the number is multiplied by 10^{-d}. If d is not specified, it defaults to zero.

The number may contain a maximum of 11 digits.

Input Field	Format Specifier	Internal Form
^^^^^6	E6.0	+6
^^^^62	E6.1	+6.1
+67.32	E6.2	+67.32
6.732E01	E8.3	+67.32
-22E300	E7.1	$-2.2*10^{300}$
^9.9+220	E7.0	$+9.9*10^{220}$

The Fw.d specifier causes the data to be stored in the corresponding real variable in the input list. If there is no decimal point in the input data, then a decimal point is inserted d places from the right side of the field. Leading blanks are ignored and trailing blanks are treated as zeros. The input data may have E followed by an optionally signed exponent. (See nEw.d, pages 29.06-29.07.)

Examples:

Input Field	Format Specifier	Internal Value
367.2593	F8.4	367.2593
-4.7366	F7.0	-4.7366
.62543	F6.4	.62543
144.15E-03	F11.2	.14415

The Iw format specifier converts the data from an input field of length w to an integer constant, if possible. Trailing blanks are treated as zeros and leading blanks are ignored. A 24-bit number is stored for integer variables; 48 bits are stored for INTEGER2 variables.

Examples:

Input Field	Format Specifier	Internal Value
^^23^	I5	+230
^^23	I4	+23
-23	I3	-23
-23^	I4	-230

The Ow specifier causes the octal number in the input field to be stored in the corresponding variable in the input list. The field may contain blanks (treated as zeros) and octal digits (0-7). A minus sign may be included to denote the negative (one's complement). If the field length (w) is greater than 8 for integer or 16 for real, only the right 8 or 16 digits are used. The number is stored right-justified in the variable.

Examples:

Input Field	Variable Type	Format Specifier	Internal Form
0124	Integer	O4	+124
7777776	Integer	O8	-1
-1^^	Integer	O4	-100
^^-0	Integer	O4	-0
2001400000000000	Real	O16	+1

nAw INPUT (ALPHANUMERIC INPUT)

The Aw format specifier causes the six-bit BCD characters in the field to be stored in the corresponding variable in the input list. The variable may be either real or integer. If a real variable is used, eight characters are stored; and if an integer variable is used, four characters are stored. Characters stored are left justified and blank filled on the right. If more characters are specified by w than can be stored, only the right-most characters are stored.

Examples:

Input Field	Variable Type	Format Specifier	Internal Value
ABCDEFGHIJKL	Integer	A2	AB^^
ABCDEFGHIJKL	Integer	A4	ABCD
ABCDEFGHIJKL	Integer	A6	CDEF
ABCDEFGHIJKL	Real	A2	AB^^^^^^
ABCDEFGHIJKL	Real	A8	ABCDEFGH
ABCDEFGHIJKL	Real	A10	CDEFGHIJ

nRw INPUT (ALPHANUMERIC RIGHT JUSTIFIED)

The Rw specifier causes the six-bit BCD information in the input field to be stored in the corresponding variable in the input list. Characters are stored right-justified zero filled. This specifier is useful for reading into character variables. The stored length is eight characters for real variables, four characters for integer variables, and one character for character variables. If w is greater than the number allowed for the variable type, only the right-most characters are stored.

Examples:

Input Field	Variable Type	Format Specifier	Internal Form
AB	Character	R2	B
ABCD	Integer	R2	00AC
ABCD	Integer	R4	ABCD
ABCDE	Integer	R5	BCDE
ABCDE	Real	R5	000ABCDE

nX INPUT

The X specifier on input ignores the character whose position it represents in the input record.

Example in context:

```
    READ(60,102)J
102 FORMAT(5X,I2)
```

Input Record: JUNK^22
Result: J=22

/ ON INPUT

The slash will cause the rest of the current record to be ignored and a new record to be read. The scanning starts at column one of the new record.

Example in context:

```
    READ(60,104)J,K
104 FORMAT(I2/I4)
```

Input Records: 23^370
0102
Result: J=23 and K=102

Tp INPUT

The Tp specifier positions the scanning pointer to column p in the input record. This makes it easy to skip over unnecessary data on the record.

Examples in context:

```
    READ(60,100)J,K,L
100 FORMAT(I2,T5,I1,T1,I1)
```

Input Record: 10^^8^^^^

Result: J=10, K=8, and L=1

wHs AND 's' INPUT

The wHs specifier allows text to be read into the format statement to be printed at some later date.

The 's' is the same as wH except it is not necessary to count the number of characters.

Examples:

Input Field	Format Specifier	Format Specifier After Read
ABCD	4HTEXT	4HABCD
ABCDEFG	'GARBAGE'	'ABCDEFG'

OUTPUT STATEMENTS

Description

Data can be written by OS-3 FORTRAN programs in two fundamentally different ways:

1) Information may be copied from a contiguous block of core storage to an output unit--the BUFFER OUT statement.

2) Information may be transferred from individually specified word addresses, with or without benefit of FORMAT interpretation.

See page 29.0 for more details on the implementation of FORMATs.

Description

The contents of a contiguous block of core may be BUFFERed OUT to any peripheral unit capable of accepting output. This statement comes closest to a standard OS-3 assembly language WRITE instruction. Minimum overhead and expense are incurred.

Source Examples

```
BUFFER OUT (40,0)(IARAY,IARAY(1024))
BUFFER OUT (13,1)(BTAB(51),BTAB(100))
```

General Syntax

```
BUFFER OUT (<IQ>,<IQ>)(<VAR>,<VAR>)
             ↑   ↑
            LUN  0 indicates BCD, even parity and
                 1 indicates odd parity, binary records.
```

Object Form

See page 29.01.

Description

A WRITE statement of the form

WRITE (<IQ>) <WAS>{,<WAS>}

or

WRITE TAPE <IQ>, <WAS>{,<WAS>}

produces the following actions:

1) The data from the words of core listed by the word-address specifiers are packed into a 127-word buffer (invisible to the user).

2) If the logical record is less than 127 words long, it and a header word are written, with the first word equal to

0	0	0	1

If the logical record is ≥ 127 words, the first 127 words are written with a leading word of

0	0	0	0

The process continues, with the logical record written as a series of 128-word physical records, each having a leading word of zeros, except the last which has as its header word the count of the number of physical records written.

The <IQ> specifies the LUN. The records are written in binary (odd) parity.

Description

A statement of the form

```
WRITE (<IQ>,<STNUM>) <WAS>{,<WAS>}
        ↑       ↑
       LUN  Format Statement Number
```

causes the data in the words named by the word-address specifiers to be written on the indicated LUN under FORMAT control.

PRINT <STNUM>, is a synonym for WRITE (61,<STNUM>).

PUNCH <STNUM>, is a synonym for WRITE (62,<STNUM>).

The following pages describe format specification codes for output.

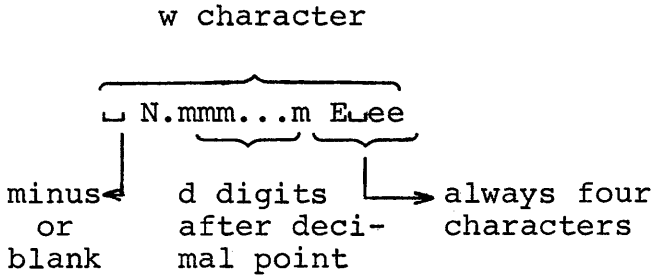
nEw.d OUTPUT

The Ew.d specifier causes the corresponding real variable in the output list to be printed in exponential format. The number is placed right justified and rounded in the field with leading blanks in unused positions. The fractional part may contain a maximum of 11 digits.

The field formats are:

- ␣ a.a...aE␣ee when $|ee| \leq 99$
- ␣ a.a...aEeee when $99 < ee \leq 308$
- ␣ a.a...a-eee when $-308 \leq eee < -99$

The a.a...a are the most significant digits of the number, and ee or eee is the exponent. If d is zero or blank, the decimal point and all digits to the right of the decimal point do not appear. The field w must be wide enough to hold the sign, decimal point, fraction, and exponent. Note that the exponent is always four characters long. There are always d digits to the right of the decimal point as shown below.



So w should be at <u>least</u> d+7 characters long.

Internal Form	Format Specifier	Output Form
+67.32	E8.2	6.73E^01
-67.32	E8.2	6.7*E^01
-67.32	E9.2	-6.73E^01
+67.32	E9.2	^6.73E^01
-67.32	E12.2	^^^-6.73E^01
+67.32	E12.3	^^^6.732E^01

nFw.d OUTPUT

The Fw.d specifier causes the corresponding element in the variable list to be printed. The real number is printed right justified in the field--rounded. The plus sign is omitted for positive numbers. If too few character positions are specified by w, an asterisk (*) is printed in the field to denote a format error. Leading zeros are suppressed.

Examples:

Internal Data	Format Specifier	Output Field
+123.45678	F10.5	^123.45678
-123.45678	F10.5	-123.45678
+123.45678	F8.5	23.4567*
+10.0	F6.2	^10.00

nIw OUTPUT

The Iw format specifier causes the contents of the corresponding variable to be printed as an integer number with leading zeros suppressed. If the number is too large to be printed in the field specified, an asterisk (*) is placed in the field. One position must be reserved for the sign if the number might be negative. The I format is also used when printing INTEGER2 variables.

Examples:

Internal Value	Format Specifier	Output Field
+23	I4	^^23
+23	I2	23
-23	I4	^-23
-23	I3	-23
-23	I2	(error field too small)

The Ow specifier causes the corresponding variable in the output list to be printed in octal. The number is right justified in the field; and if the field is smaller than 8 for integer and 16 for real variables, only the right-most w digits are printed. If the field is larger than 8 for integer and 14 for real, leading blanks are inserted.

Examples:

Internal Data	Variable Type	Format Specifier	Output Field
+144	Integer	O3	144
+144	Integer	O8	00000144
+144	Integer	O11	^^^00000144
+1	Real	O16	2001400000000000

nAw OUTPUT

The Aw format specifier causes the contents of the corresponding variable to be printed as six-bit BCD characters. Integer variables contain four characters and real variables contain eight characters. If w is less than four (for integer) or eight (for real), only the left w characters are printed. If w exceeds four (for integer) or eight (for real), the excess characters are printed as blanks on the left of the field.

Examples:

Internal Form	Variable Type	Format Specifier	Output Field
ABCD	Integer	A2	AB
ABCD	Integer	A4	ABCD
ABCD	Integer	A6	^^ABCD
ABCDEFGH	Real	A2	AB
ABCDEFGH	Real	A8	ABCDEFGH
ABCDEFGH	Real	A10	^^ABCDEFGH

nRw OUTPUT

The Rw specifier causes the six-bit BCD information in the corresponding variable in the output list to be printed. The characters are right justified in the output field. If w exceeds the maximum allowed number of characters for the variable type (one for character, four for integer, eight for real), leading zeros are printed. This format should be used when outputting character variables.

Examples:

Internal Form	Variable Type	Format Specifier	Output Field
A	Character	R1	A
A	Character	R2	0A
ABCD	Integer	R2	CD
ABCD	Integer	R4	ABCD
ABCD	Integer	R6	00ABCD
ABCDEFGH	Real	R5	DEFGH

nX OUTPUT

The X specifier on output causes the character store pointer to skip over the character position it represents in the output string.

Example in context:

```
      WRITE(61,103)J,K
  103 FORMAT(X,I2,4X,I1)
```

Output Records: ^10^^^^8

Result: J=10 and K=8

/ ON OUTPUT

The slash on output causes a new line to be started and the old record to be written out. The position is set to column one of the new record.

Example in context:

```
      WRITE(61,105)J,K
  103 FORMAT(XI2/4XI1)
```

Output Records: ^10

^^^^8

Result: J=10 and K=8

Tp OUTPUT

The Tp specifies the position in which the next character will be placed.

Example in context:

```
    WRITE(61,101)J,K,L          where J=10, K=8, and L=7.
101 FORMAT(T2,I2,T10,I1,T6,I1)
```

Result: ^10^^7^^^8

wHs AND 's' OUTPUT

The wH specifier allows text to be printed in the output. w specifies the number of characters to be printed. The 's' is the same as the wH specifier except it is not necessary to count the number of characters.

Examples:

Format Specifier	Output Field
4HGOOD	GOOD
6HTEST^^	TEST^^
9HTEST1^=^^	TEST1^=^^
'GOOD'	GOOD
'TEST^^'	TEST^^
'TEST1^=^^'	TEST1^=^^

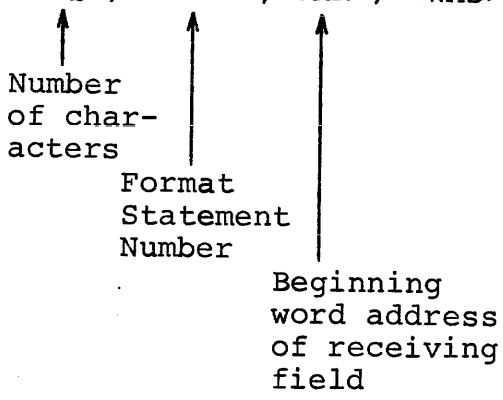
ENCODE

Description

In addition to BUFFER statements, CDC-derived FORTRAN provides two additional non-ASA standard statements, ENCODE and DECODE. The essential idea is to allow use of the FORMAT-controlled data conversion routines for core-to-core transfers, as well as for I/O. ENCODE packs a contiguous area of core from individually named locations under format control.

General Syntax

```
ENCODE (<IQ>,<STNUM>,<VAR>) <WAS>{,<WAS>}
```



Object Form

As in the case of actual I/O, ENCODE is performed by a call to a run-time library routine.

```
ENTERA      100B
RTJ         Q8QINGOT
77          L.<STNUM>
77          address of <IQ>
77          beginning word address
```

The effect is to pack data from the individually named variables into a contiguous block of core, beginning at the specified word address. If the final word is not completely filled, it is padded on the right with blanks.

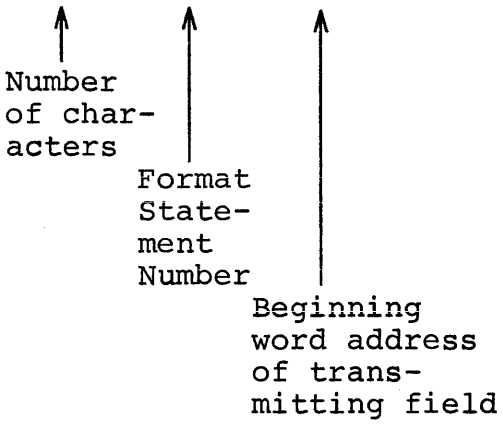
DECODE

Description

DECODE distributes data from a contiguous block of core to individually named locations under FORMAT control.

General Syntax

```
DECODE (<IQ>,<STNUM>,<VAR>) <WAS>{,<WAS>}
```



Object Form

Call to run-time library routine. See previous page.

MISCELLANEOUS I/O COMMANDS

REWIND i

Rewind: Rewinds file or magnetic tape i to the load point. If the file is already rewound, the statement acts as a do-nothing statement.

Before executing a REWIND on an output unit, FORTRAN writes an EOF record.

BACKSPACE i

Backspace: Backspaces file or magnetic tape i one logical record. When the file is already at the load point (rewound), BACKSPACE i acts as a do-nothing statement. Backspace is interpreted as a locate to the beginning of the previous record.

ENDFILE i

Endfile: Writes an end-of-file on file or magnetic tape i.

I/O Status Checking

The FORTRAN library contains functions and subroutines that check status after I/O operations:

Subroutine or Function	Operation Checked	Condition Checked
End-of-file check (EOF) (EOFCK) (EOFCKF)	Previous read/ write I/O request	End-of-file
Parity check (IOCHK) (IOCHKF)	Previous read/ write I/O request	Parity error
Unit status test (UNITST) (UNITSTF)	Last buffer operation	End-of-file, parity errors, completion of operation
Length test (LENGTHF)	Last BUFFER IN	Number of words transferred

Attempting to read past an EOF without checking for EOF causes job termination. For BCD or binary operations, EOFCK or EOFCKF senses the condition; for BUFFER IN, UNITST or UNITSTF senses the condition.

VARIABLE FORMAT

Format lists need not be provided with FORMAT statements; instead, they can be placed in arrays. Placing format lists in arrays and referencing the arrays instead of a FORMAT statement permits the programmer to change, index, and specify formats at the time of execution.

Format arrays are prepared by storing a format list, including left and right parentheses, as it would otherwise appear with a FORMAT statement. Variable specifications can be read in from cards, changed with replacement statements, or preset in labeled common with a DATA statement.

Example:

Prepare an array for format list.

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

```
1   5  7
       DIMENSION IVAR (8)
       READ (K1,1)(IVAR (I), I = 1,8)
    1  FORMAT (8A4)
```

Result: IVAR(1) contains (E12
IVAR(2) contains .2,F
IVAR(3) contains 8.2,
IVAR(4) contains I7,2
IVAR(5) contains E20.
IVAR(6) contains 3,F9
IVAR(7) contains .3,I
IVAR(8) contains 4)^^

When using the specifications, the array can be referenced as follows:

1	5	7
		WRITE (K2, IVAR (1)) A,B,I,C,D,E,J or WRITE (K2, IVAR) A,B,I,C,D,E,J

APPENDIX 1
Compiler Call Parameters

FORTRAN,A=LUN,D=LUN,H=LUN,I=LUN,K=(NUMBER),L=LUN,N=(NUMBER),
P=LUN,R=LUN,S=LUN,X=LUN

This control mode instruction causes OS-3 to load the FORTRAN compiler. For each of the parameters described below, any string of letters may be appended to the single letter shown to the left of the equal sign. NOTE: To force a LUN to be rewound, use /R after its number.

A=LUN — This specifies that an assembly language listing of the program is to be prepared and sent to the logical unit specified. If no LUN is specified, then LUN 61 is assumed unless a LUN is specified in the L parameter.

D=LUN — This specifies that the diagnostic error messages are to be sent to the logical unit specified. If no LUN is specified then the LUN listed with the L parameter is assumed. If L is missing, then LUN 61 is assumed.

H=LUN — This specifies that a FORTRAN source deck is to be sent to the LUN specified. This deck will be in a standard FORTRAN format. Logical unit 62 is assumed if no LUN is specified.

I=LUN — This specifies that the input to the FORTRAN compiler is to come from the logical unit specified. If no LUN is specified, LUN 60 is assumed. The logical unit number in this parameter may be replaced by the name of a saved file. Input logical units are

rewound, if possible, by the FORTRAN compiler before reading. Input units numbered between 50 and 59 inclusive are unequipped at the end of compilation.

K=(NUMBER) This specifies the type of card keypunch used. If (NUMBER) is 029, then the deck was punched in extended BCD code (EBCDIC) used by the IBM 360. If (NUMBER) is 026 or is omitted, then the standard model 026 keypunch is assumed. If (NUMBER) is 027, then the deck is assumed to contain cards punched on both 026 and 029 keypunches. Invalid 026 codes will be translated as 029 code. This option should be used only with special card decks.

L=LUN This specifies that a listing of the source program is to be sent to the logical unit listed. LUN 61 is assumed if no LUN is specified by the user.

N=(NUMBER) This function allows the user to override the normal lines/page set by the system.

P=LUN This functions exactly as X, except that LUN 62 is assumed if no LUN is specified, and no file mark is placed at the end.

R=LUN This specifies that a binary object program should be sent to the LUN specified. This logical unit is released before compilation. After compilation, if no fatal diagnostic occurred, the program will be loaded and run. If no LUN is specified, then logical unit 56 is assumed. This

parameter differs from X only in that a BCD RUN statement is written in place of the file mark at the end of the output, and the file is automatically loaded and executed after compilation.

S=LUN — This instructs the FORTRAN compiler to prepare symbol output on the LUN specified. If no LUN is specified, then the P-LUN is assumed. If P was not specified, then X is assumed.

X=LUN — This specifies that the output from the FORTRAN compiler (binary object program) should be sent to the logical unit specified. If no LUN is specified, then LUN 56 is assumed. A file mark is automatically put at the end of file.

All logical units specified in the FORTRAN control mode instruction must have been previously defined by the user. Any of the parameters A,D,H,I,K,L,N,P,R,S,X may be omitted; the desired ones may be listed in any order.

This control mode instruction will destroy any interrupted program so that it may not be restarted with a GO command.

Examples of FORTRAN control mode instructions are:

```
FORTRAN,I=45,S=33,L=67,P=89,A=91
FORTRAN,L,X
FORTRAN,IN=TEST,RUN=47
FORTRAN,R,INPUT=TEST
FORTRAN,I=45,X,L=18,D=61,K=027,S
FORTRAN,INPUT=33,A=61,X=3,L=47,D=61
FORTRAN,IN=TEST,X,H=37,N=20
```

APPENDIX 2

Deck Structures

BATCH	ON-LINE

Compile only, with listing and diagnostics.

```
7
8JOB                                    (LOGON)

7
8FORTRAN,L                              #FORTRAN,I=<INPT>,L
[                                       [
|  Source Decks                         |  Listing and diagnostics
[                                       [
                                        #LOGOFF
77
88

7
8LOGOFF
```

Compile and run, with listing.

```
7
8JOB                                    (LOGON)

7
8FORTRAN,L,R
[                                       #FORTRAN,I=<INPT>,L,R
|  Source Decks                         [
[                                       |  Listing and diagnostics
77                                      [
88
[                                       Run if no fatal diagnostics
|  Data, if needed                      [
|  77                                   |  Output
|  88                                   [
[                                       END OF FORTRAN EXECUTION
                                        #LOGOFF
7
8LOGOFF
```

BATCH	ON-LINE

Compile, get binary deck.

```
7
8JOB
7
8FORTRAN,X
[
|Source Decks
[
77
88
7
8SAVE,56=<BINDECK>
7
8LOGOFF
```

```
(LOGON)
#FORTRAN,I=<INPT>,X
  NO ERRORS FOR JOB.
#SAVE,56=<BINDECK>
#LOGOFF
```

Load and run binary deck.

```
7
8JOB
7
8LOAD,<BINDECK>
RUN
[
|Data, if needed
|77
|88
[
7
8LOGOFF
```

```
(LOGON)

#LOAD,<BINDECK>
RUN
[
|Output
[
END OF FORTRAN EXECUTION
#LOGOFF
```

APPENDIX 3

Index of Metalanguage Terms

APPENDIX 4

Standard FORTRAN Library Functions and Subroutines

The FORTRAN library contains standard subprograms which check the status of I/O operations, sense machine conditions, perform a FORTRAN dump, or perform various mathematical operations (such as evaluating trigonometric functions or determining square roots).

Some subprograms can be requested either as functions or as subroutines. Both FORTRAN references will be shown. Note that a subroutine requires the use of CALL.

I/O

The FORTRAN library contains functions and subroutines that check status after I/O operations. All parameters must be type integer.

EOFCK and IOCHK are used with read/write I/O statements only. When they reference buffered files, the job terminates abnormally.

Parity Check

Subroutine: IOCHK (k,j)
Function: IOCHKF (i)

IOCHK checks the status on the previous I/O request on logical unit or file i to determine if a parity error occurred.

- i — Identifies the logical unit or file.
- j — Location in which the value is stored. For the function IOCHKF, the value is returned in the A register.

Value	Condition
1	Parity error occurred on previous I/O operation.
2	No parity error occurred.
3	End of allocated area detected during last operation.

Error return: If the logical unit number of file number is not 1-62, the run is terminated with the following message on the standard output unit.

ERROR IN IOCHK CALLED FROM ILLEGAL I/O REQUEST UNIT no.

Sample: Convenient forms for using IOCHKF are:

GO TO (n_1,n_2), IOCHKF (i)

or

IF (IOCHKF (i).EQ.1)n_1,n_2

End-of-File Check

Subroutine: EOFCK (i,j)
Function: EOFCKF (i)

EOFCK checks the status of the previous I/O request on logical unit or file i to determine if an end-of-file was encountered.

i Identifies the logical unit or file.

j Location at which the value is stored. For the function EOFCKF, the value is returned in the A register.

Value	Condition
1	An end-of-file was encountered on the last read operation.
2	No end-of-file was encountered.

On a mass storage file, each of the following is interpreted as an end-of-file condition (value 1 returned in the A register).

Record just read was defined end-of-file record.

Last read attempted to read beyond the highest block written.

Locate for the last read attempted to locate the file limit.

Last read attempted to read beyond the file limit.

Error return: If the logical unit or file number was not 1-62, the run is terminated with the following message on the standard output unit.

ERROR IN EOFCK CALLED FROM xxxxx ILLEGAL I/O REQUEST UNIT no.

Sample: Convenient forms for using EOFCKF are:

GO TO (n_1,n_2), EOFCKF (i)

or

IF (EOFCKF (i).EQ.1)n_1,n_2

Function: EOF (i)
Sample: IF (EOF(2)) GO TO 10

The EOF function returns a true/false value for end-of-file checking. It is the preferred method of checking for end-of-file. The function is true if an end-of-file was encountered.

Unit Status Test

Subroutine: UNITST (i,j)
Function: UNITSTF (i)

UNITST checks the I/O status of the last BUFFER IN or BUFFER OUT operation on logical unit or file i.

i Identifies the logical unit or file.

j Location at which the value is stored. For the function UNITSTF, the value is returned in the A register.

Value	Condition
1	Buffer operation not complete.
2	Buffer operation complete and no errors occurred.
3	Buffer operation complete, but an end-of-file has been sensed.
4	Buffer operation complete, but a parity error has occurred.

Length Test

Function: LENGTHF (i)

The length test determines the number of words transferred during the last BUFFER IN operation on unit i. The number of words is returned in the A register for use by the calling statement.

Example:

```
1  5 7
                :
                :
      J = 1
      BUFFER IN (10,0) (A,Z)        Set flag 1
                                    Initiate buffered
                                    read for logical
                                    unit 10, even
                                    parity. The
                                    first word of
                                    the block is A;
                                    the last Z
      GO TO (5, 6, 7, 8), UNITSTF (10)  Check transmission
                                        status
```

Page missing from
original document

```
  7 KERR = LENGTHF (10)                  KERR will contain
                                         number of words
                                         read
    WRITE (21, 70)                       Error message
 70 FORMAT (12H EOF UNIT 10)             End-of-file error
    GO TO 200
  8 KERR = LENGTHF (10)                  KERR will contain
                                         number of words
                                         read
    WRITE (21, 80)
 70 FORMAT (10X,12HPARITY ERROR)
200 REWIND 10
    STOP
  6 CONTINUE                             BUFFERED trans-
                                         mission complete;
                                         continue program
```

Sense Light Control

Subroutine: SLITE (i)
Function: SLITEF (i)

SLITE turns on sense light i if i = 1-24 or turns off all sense lights if i = 0. The value of i is returned in the A register. i corresponds to a bit in the storage location SENSLIT. i = 1-24 sets the corresponding bit in SENSLIT, and i = 0 clears all bits.

Storage: 104 locations, including SLITET, which is automatically loaded at the same time.

Error return: If i is less than zero or greater than 24, the following message is printed and a 2 is returned in the A register.

ERROR IN SLITE/F CALLED FROM xxxxx I LT 0/GT 24

Sense Light Test

Subroutine: SLITET (i,j)
Function: SLITETF (i)

SLITET tests sense light i, if i = 1-24, a bit in storage location SENSLIT, and clears sense light i if it was set. If i = 0, no bits are tested and a 2 is stored.

j Specifies storage location of result; 1 is stored if light i was on, 2 if it was off. For the function SLITETF, these values are returned in the A register.

Storage: 104 locations, including SLITE, which is automatically loaded at the same time.

Error return: If i is less than zero or greater than 24, a 2 is stored in j (SLITET) or in the A register (SLITETF) and the following message is printed.

ERROR IN SLITET/F CALLED FROM xxxxx I LT 0/GT 24

Exponent Fault Test

Subroutine: EXFLT (j)
Function: EXFLTF (i)

EXFLT uses the internal sense instruction to determine if the bit indicating exponent overflow is set and clears it if it is set.

i Dummy parameter required by EXFLTF but not used.

j Location at which result is stored; a 1 is stored if the fault bit was set, a 2 if it was not. For EXFLTF, these values are returned in the A register.

Storage: 47 locations, including OVERFL and DVCHK.

Overflow Test

Subroutine: OVERFL (j)
Function: OVERFLF (i)

OVERFL uses the internal sense instruction to determine if the bit representing arithmetic overflow is set; and if so, clears this bit.

i Dummy parameter required but not used by OVERFLF.

j Location to which result is returned. A 1 is returned if the overflow bit was set, a 2 if it was not. For OVERFLF, these values are returned in the A register.

Storage: 47 locations, including DVCHK and EXFLT.

Subroutine Library

Some of the commonly used library routines are:

Subroutine	Operating System	Definition
SLITE (i)	all	set sense light i
SLITET (i,j)	all	test sense light i
DVCHK (i)	all	check divide fault
EXFLT (i)	all	check exponent fault
OVERFL (i)	all	check overflow fault
EOFCK (i,j)	all	test for end-of-file on unit i
IOCHK (i,j)	all	test for parity error on unit i
UNITST (i,j)	all	test status on unit i
FORTDUMP (b,e,m,d,c)	all	system dump routine

Function Library

The following FORTRAN library functions are predefined and may be referenced by a program or subprogram. X represents real values; I represents integer values. F is optional as a

final character in most function names. For machine conditions, i designates the component, unit, or file number.

Function	Operating System	Definition
ABS(X);ABSF(X) IABS(I);XABSF(I)	all	absolute value
ALOG(X);LOGF(X)	all	natural log of X
ATAN(X);ATANF(X)	all	arctangent of X radians
COS(X);COSF(X)	all	cosine of X radians
EXP(X);EXPF(X)	all	e to xth power
FLOAT(I);FLOAT(I)	all	integer to real conversion
IFIX(X);XFIXF(X);FIXF(X)	all	real to integer conversion
SIGN(X_1,X_2);SIGNF(X_1,X_2)	all	sign of X_2 times S_1
ISIGN(I_1,I_2);XSIGNF(I_1,I_2)	all	sign of I_2 times I_1
SIN(X);SINF(X)	all	sine of X radians
SQRT(X);SQRTF(X)	all	square root of X
SLITET (i)	all	set sense light i
SLITETF (i)	all	test sense light i
DVCHKF (i)	all	check divide fault
EXFLTF (i)	all	check exponent fault
OVERFLF (i)	all	check overflow fault
EOFCKF (i)	all	test for end-of-file on unit i or file i
IOCHKF (i)	all	test for parity error on unit i or file i
UNITSTF (i)	all	test status of unit i or file i
LENGTHF (i)	all	words in last BUFFER IN on unit i or file i
LOCATEF (i,b)	all	locate to block b of file i

NOT (a) AND (a,b) OR (a,b) FOR (a,b)	all	integer masking functions

The functions XABSF, LOGF, FIXF and XSIGNF must appear in a TYPE declaration to indicate the correct mode of the result.

APPENDIX 5
Printer Carriage Control

Column 1 of information being sent to a line printer specifies control of paper movement during printing. This character will not be printed. Listed below are the control characters and the action that will be performed before and after printing.

Control Character	Action Before Print	Action After Print	Comments
(blank)	space 1	no action	single space, skip over bottom margin
0 (zero)	space 2	no action	double space, skip over bottom margin
- (minus)	space 3	no action	triple space, skip over bottom margin
+	no action	no action	over print
1	eject page	no action	top of page
2	skip to level 12	no action	one inch from bottom of page
3	skip to level 6	no action	level 6 is every 6th line
4	skip to level 5	no action	level 5 is every 5th line
5	skip to level 4	no action	level 4 is every 4th line
6	skip to level 3	no action	level 3 is every 3rd line
7	skip to level 2	no action	level 2 is every 2nd line
8	eject page	no action	same as 1
9*	skip to level 7	no action	
Z*	skip to level 8	no action	

* These codes work on the 512 printer only.

Control Character	Action Before Print	Action After Print	Comments
Y*	skip to level 9	no action	
X*	skip to level 10	no action	
A	space 1	eject	
B	space 1	skip to level 12	level 12 is 1" from bottom of page
C	space 1	skip to level 6	level 6 is every 6th line
D	space 1	skip to level 5	level 5 is every 5th line
E	space 1	skip to level 4	level 4 is every 4th line
F	space 1	skip to level 3	level 3 is every 3rd line
G	space 1	skip to level 2	level 2 is every 2nd line
H*	space 1	space 1	will skip over bottom margin
I*	space 1	skip to level 7	
J*	space 1	skip to level 8	
K*	space 1	skip to level 9	
L*	space 1	skip to level 10	
Q	space 1	clear AUTO PAGE EJECT	print over the crease in the page
R	space 1	set AUTO PAGE EJECT	skip over the crease in the page
S	space 1	print six lines per inch	
T	space 1	print eight lines per inch	
other codes	space 1	no action	same as (blank)

* These codes work on the 512 printer only.