INTERNAL USE ONLY

INTERNATIONAL COMPUTERS AND TABULATORS LIMITED.

Programming Languages Division.

I.C.T. 1900 Series

FORTRAN NOTE 1

Issue 2

25th September, 1967.

1900 Fortran Arithmetic

This note replaces Fortran Note 1 dated 11.3.65. which should be destroyed.

1. The Accumulator

For the prupose of compiling instructions for arithmetic expressions an I.C.T. 1900 series computer is considered to be a one accumulator, one index register, single level store machine. Note is taken, however, that only quantities which might be in upper storage to be indirectly addressed.

The index register used is X3. The single accumulator will be called ACC in this document. The floating point accumulator will be called FPA.

The location of ACC is different according to the mode (INTEGER,REAL, COMPLEX etc.) of the quantity it contains. The locations of ACC for the different modes are given in the following table. The manner in which the quantity is held in ACC is described in Fortran Note 8.

Mode	ACC
Integer	X6
Real	FPA
Double Precision	FPA, X4, X5
Complex	FPA, X4, X5
Logical	X6

If an operation which calls for an operand in ACC has a result which is of a different mode, and the result is to be in ACC, then the accumulator will be the correct one for the new mode.

2. Arithmetic operations

Arithmetic operations in FORTRAN can be divided into 6 main types as follows

1. Binary operations e.g. A + B
2. Unary operations e.g. -A, .NØT. B
3. Array operations e.g. A(I)
4. Function operations e.g. FUN(X,Y)
5. Conversion operations e.g. Integer ⟶ Real
6. Store and Load operations e.g. ACC ⟶ Store

2.1. Binary Operations

This is the set of all definable operations of the form A op B where A and B may be any legitimate quantity of any mode and op is any operator which occurs as an infix with two such operands

viz. + - * / ** .OR. .AND. .EQ. .NE. .LT. .GE. .GT. .LE.

In such an operation the operands will be in the following form at the time the operations is executed.

1. One operand in ACC
   The other operand in Store

or 2. One operand in ACC
   The address of the other operand in X3

The result of a Binary operation will be a quantity in ACC. It is assumed that as the result of a Binary operation the original contents of X3 and ACC are destroyed.

2.2 Unary Operations

These are the operations of the set -A and .NØT. B where A is any legitimate quantity of mode other than LØGICAL and B may only be LØGICAL.

For these operations the operand is in ACC. The result of a Unary operation is a quantity in ACC. No unary operation affects X3.

2.3 Array Operations

These are operations which define the addres of an array element. The operands for the operation depend on the actual method used to generate the address.

The result of an Array operation is an address (of the correct array element) in X3. Nothing may be assumed about the contents of ACC following an array operation.

2.4 Function Operations

These are operations of the form FCN (A,B, ...) where A, B etc. are the arguments of the function.

The result of a function operation is a quantity in ACC where ACC is specified by the mode of the name of the function. The original contents of X3 and ACC are destroyed by a function operation.

2.5 Conversion Operations

These are operations which change the mode of a quantity without changing its value. There are of two types.

1. Those which change the mode of a quantity which is in ACC

2. Those which change the mode of a quantity which has its address in X3.

Type 1. Any Conversion operation which changes the mode of a quantity in ACC will leave the converted quantity in ACC according to the final mode of that quantity.

Type 2. Any conversion operation which changes the mode of a quantity whose address is in X3 will leave the converted quantity in some location and will give as a result in X3, the address of that location.

Operations of type 1 will not destroy the contents of X3

Operations of type 2 will not destroy the contents of ACC. ACC is defined by the final mode of the quantity converted.

2.6 Store and Load Operations

These are basically operations which are required to produce operands of acceptable form for the other types of operations and also to store final results. They are two types

1.	ACC operations	Load ACC from Store
		Store ACC in Store (Assignment)
2.	X3 operations	Load address into X3 from Store
		Store address from X3 in Store

Type 1 operations do not destroy X3

Type 2 operations do not destroy ACC

3. Arithmetic Packages

The operations referred to above are carried out either by compiling open ended subroutines or by compiling a calling sequence to a closed subroutine. Most of the simple operations are dealt with in the former manner, e.g. A = B + C where A, B, and C are of REAL mode results in the following being compiled.

```
LFP     C       Load ACC from store
FAD     B       Binary operation (type 1)
SFP     A       Store ACC in store
```

However had A,B and C been of DOUBLE PRECISION mode then the sequence

```
LFP     C       )
LDX  4  C+2     )Load ACC from store
LDX  5  C+3     )
LDN  3  B       Load X3 with address of B
CALL 1  %FDP    DP addition, binary operation (type 2)
SFP     A       )
STO  4  A+2     )
STO  5  A+3     )Store ACC in store
```

would have been compiled.

In the above, %FDP is the first entry point to a double precision package. There are a number of these arithmetic packages present in the Fortran Library.

%FAP4 - General purpose arithmetic package:-

(i) REAL binary operations* (redundant now)
(ii) Array operations of all types
(iii) INTEGER to INTEGER exponentiation
(iv) REAL to INTEGER exponentiation
(v) REAL/INTEGER conversion operations
(vi) Other special operations

%FEX4 - REAL to REAL exponentiation

%FDP - DOUBLE PRECISION package

(i) D.P. binary operations*
(ii) REAL to DOUBLE PRECISION conversion

%FD2 - DOUBLE PRECISION to INTEGER exponentiation
%FDR - DOUBLE PRECISION to REAL exponentiation
%FDD - DOUBLE PRECISION to DOUBLE PRECISION exponentiation

%FCP - COMPLEX package
(i) COMPLEX binary operations*
(ii) REAL to COMPLEX conversion

%FC2 - COMPLEX to INTEGER exponentiation

*Exponentiation which is also a binary operation is itemised separately since it is a much more complicated operation.

A brief specification of these subroutines is given in the Appendix.

4. Mixed Mode binary operations

Where the two operands are of different mode the operation is normally carried out in two stages.

Stage 1 — One or two conversion operations to make the two operands of the same mode.

Stage 2 — Binary operation

The net effect is as defined in section 1.1. A similar situation exists for mixed mode assignment operations.

As an example consider A = B + C where A is INTEGER, B is DOUBLE PRECISION and C is INTEGER

The code generated is:-

```
LDN     6     C                Load ACC
CALL    1     %FAP4+15         Convert to REAL
LDN     4     0   )            Convert to DOUBLE PRECISION
LDN     5     0   )
LDN     3     B                Load X3 with address of B
CALL    1     %FDP             DP addition
CALL    1     %FAP4+18         Convert to integer
STO     6     A                Store ACC
```

Note that the R.H.S. is always calculated in the 'highest level' mode (i.e. Double Precision is higher than integer) and that the mode of the result is always determined by the mode of the L.H.S.

5. Functions and their Calling Sequence.

There are a number of different types of 'functions'.

1. Basic intrinsic functions
   e.g. ABS, MIN, INT
2. Basic external functions e.g. SIN, SQRT
3. Private Fortran subroutines; those beginning %F,
   e.g. %FAP4, %FINOUT
4. Other external functions e.g. PLOT, ITIME
5. Source language FUNCTION procedures
6. Source language SUBROUTINE procedures
7. Arithmetic statement functions
8. 'Implied DO' functions.

Those of type 1,2,3 or 4 are held in the Fortran Library, the remainder are defined by the user.

Those of type 1,2,5 and 7 return the result in ACC and are as defined in section 1.4

Apart from those of type 3 and certain basic intrinsic functions which have a variable number of arguments the following standard calling sequence is used:

```
CALL   1   SUBR
LDX    3   ARGI   ┐
LDN    3   ARG2   │      One instruction per argument
 .     .    .    │
 .     .    .    │
 .     .    .    │
 .     .    .    │
LDX    3   ARGN   ┘
```

The instructions in the argument list are determined as follows:

1. If the argument is a variable, the argument word must be an instruction which, if obeyed, will place the address of the argument into X3. This will normally be a LDN or a LDX instruction

2. If the argument is a function (to be used as a dummy function) the argument word must be an instruction which, if obeyed, will place the instruction BRN Start of Function. into X3.

3. If the argument is an array, the argument word must be an instruction which, if obeyed, will place the address of the array header (see Fortran Note 25) into X3.

The called routine will return to the first location after the calling sequence. This implies that both the calling and the called routines are independently aware of the number of arguments and of their types.

The 1900 FORTRAN there are certain Basic Intrinsic Functions which have a variable number of arguments. For these functions, the compiler will insert as the first word of the argument list (prior to the first actual argument) a word which contains the 'number of arguments'.

6. Array Addressing

This is fully described in Fortran Note 25.

Appendix.

Brief Specification of the Fortran Arithmetic Subroutines

Notation

X = REAL number held in FPA
Y = REAL number whose address is in X3
Z = REAL number whose address is in X3 but which is temporarily stored in common area %. LIB
I = INTEGER number held in X6
J = INTEGER number whose address is in X3
K = INTEGER number whose address is in X6
N = INTEGER number held in X3 (<4096)
DX = DOUBLE PRECISION number held in FPA, X4, X5
DY = DOUBLE PRECISION number whose address is in X3
CX = COMPLEX number held in FPA, X4, X5
CY = COMPLEX number whose address is in X3

General

The link accumulator used is always X1. The contents of accumulators are not preserved. Entry points to each package are relative to the package name.

1. %. FAP4 - Arithmetic Package

Entry points :

+0	NULL	(used to be X=X+Y)
+1	NULL	(used to be X=X*Y)
+2	NULL	(used to be X=X-Y)
+3	NULL	(used to be X=Y-X)
+4	NULL	(used to be X=-X)
+5	NULL	(used to be X=X/Y)
+6	X3 = Address of 1 or 2 word array element (3 or more dimensions)	
+7	NULL	(used to be X=Y/X)

+8	I=I ** J
+9	I=I ** N
+10	X=X**J
+11	X=X**N
+12	I=J**I
+13	I=N**I
+14	X=Y**I
+15	X=I
+16	Z=J, X unchanged
+17	Z=N, X unchanged
+18	I=X
+19	TRACE initialization
+20	X3=Address of 4 word array element
+21	X=I/J
+22	X=I/N
+23	X=J/I
+24	X=N/I
+25	X3+Address of 1 or 2 word array element (2 dimensions)

2. % FDP - Double Precision Arithmetic Package

Entry points:

+0	DX=DX+DY
+1	DX=DX*DY
+2	DX=DX-DY
+3	DX=DY-DX
+4	DX=-DX
+5	DX=DX/DY
+6	DX=DY/DX
+7	DX=Z

Entry points +8 to +15 also exist, and are used by double precision routines to perform the above operations on non-standard (unpacked) D.P. numbers.

3. % FCP - Complex Arithmetic Package

Entry points:		
	+0	CX=CX+CY
	+1	CX=CX*CY
	+2	CX=CX-CY
	+3	CX=CY-CX
	+4	CX=-CX
	+5	CX=CX/CY
	+6	CX=CY/CX
	+7	CX=X**CY

4. % FEX4 - Real Exponentiation

Entry points:		
	+0	X=X**N
	+1	X=N**X

5. % FD2 - D.P. to Integer Exponentiation

Entry points		
	+0	DX=DX**J
	+1	DX=DX**N
	+2	DX=DY**K

6. % FDR - D.P. to Real Exponentiation

Entry points		
	+0	DX=DX**Y
	+1	DX=DY**X

7. % FDD - D.P. to D.P. Exponentiation

Entry points		
	+0	DX=DX**DY
	+1	DX=DY**DX

8. % FC2 - Complex to Integer Exponentiation

Entry points		
	+0	CX=CX**J
	+1	CX=CX**N
	+2	CX=CY**K