# ND-500 Single Prec. Array Proc. Func.

ND-805013.3 EN

Norsk Data

# *ND-500 Single Prec. Array Proc. Func.*

*ND-805013.3 EN*

*Send all documentation requests to:*

*Norsk Data A.S*
*Publication Department*
*P.O. Box 25 – Bogerud*
*N-0621 Oslo 6*
*NORWAY*

## Preface:

### THE PRODUCT

This manual describes the mathematical functions in the ND-500 APF library.

The APF library routines may be called from FORTRAN. The routines are designed to speed up the execution of operations on single precision real arrays.

The APF library utilizes a special microprogram for the ND-500 CPU. This microprogram is an extension of the standard microprogram.

The ND-numbers for this product on the different ND-500 models are:

| Computer | ND-number | Microprogram version |
|---|---|---|
| ND-500/1 series | ND-10338 | 104xx |
| ----- " ----- | ND-10412 | 106xx |
| ND-500/2 series | ND-10701 | 152xx |
| ----- " ----- | ND-10786 | 152xx |
| ND-570/2 serie | ND-10700 | 150xx |

NOTE The ND-530/2 can not use this product.

### THE MANUAL

This manual provides a functional description of the APF library, and thereby the special array processing functions. A listing of each routine is used to describe the routines. Listing and parameter description is mainly in FORTRAN. Two of the routines are described in ND-500 assembler, although they are callable from FORTRAN.

### CHANGES FROM PREVIOUS VERSION

The description of the routines is revisted to make this manual consistent with the related manual ND-500/2 Double Precision Array Processing Functions, ND-05.018. Small changes is done in the structure. Some documentation errors are corrected.

### THE READER

This manual is written for people creating and running programs using array processing functions.

## PREREQUISITE KNOWLEDGE

The reader is assumed to be familiar with FORTRAN and ought to have some knowledge of the ND-500 assembler. It is also assumed that the reader is familiar with using ND-500 computers, and knows how to generate, load and run programs on such systems.

## RELATED MANUALS

Documentation further describing the use of ND-500 computers is found in these manuals:

| | |
|---|---|
| ND-500 Loader/Monitor | ND.60.136 |
| ND FORTRAN Reference Manual | ND.60.145 |
| ND-500 Reference manual | ND.05.009 |
| ND-500/2 Double Precision Array Processing Functions | ND.05.018 |

## TABLE OF CONTENTS

# 1 BASIC CONCEPTS

## 1.1 PREREQUISITE SOFTWARE FOR USING THE APF LIBRARY

Together with the microprogram, the APF library must be installed. The actual file name is: ND-500-APF-LIB:NRF.

These are the only modifications to be performed on the system.

The APF library utilizes a special microprogram for the ND-500 CPU. This microprogram contains the array processing functions. It is an extension of the standard ND-500 instruction set.

The array processing functions are floating point operations performed in 32 bits floating point format by the ND-500 floating point arithmetic.

## 1.2 THE SINGLE PRECISION FLOATING POINT FORMAT

The number range for the single precision floating point format is:

$$8.6 * 10^{-78} \le |N| \le 5.8 * 10^{76}$$

The accuracy corresponds approximately to 7 decimal digits.

## 1.3 HARDWARE CONCEPTS

The ND-500 CPU gives the possibility of parallel processing. This means that indexing, memory access, floating point arithmetic, integer arithmetic and loop control may be run in parallel. This is done as far as possible to obtain high speed operations. Temporary results to be used in later calculations, are kept in registers in the ND-500 CPU, accessed directly by both the floating point arithmetic and integer arithmetic.

Arrays involved in an operation are accessed through the ND-500 memory management system. This system will cause automatic allocation of memory, and reservation of continuous memory space for array processing is not required. The result of an array processing function is present in the output array when returning from the function.

The ND-500 array processing functions are fully interruptable to maintain the ND-500 CPU resources being shared by the different processes currently running on the system.

## 1.4 ARRAY PROCESSING DEFINITIONS

An array contains a group of numbers that are related to each other in some way, and the array may be multi-dimensional. An array is termed a matrix in mathematical terminology.

An array is used to represent equations of different kinds, for example, linear equations. Each row in the array represents one particular equation, thus the array represents a system of equations of the same kind.

The entries in the array are the coefficients of the equations. They are called elements in this manual.

Each row of elements is named a vector in this manual, regardless of what kind of equation it mathematically represents. In those array processing functions which are closely related to mathematical vectors, the row of elements is referred to as a complex vector.

Most of the array processing functions are performed on one-dimensional vectors.

Three parameters are necessary to specify a vector:

| | |
|---|---|
| V - | The logical name of the vector. A single precision real. |
| INC - | The step value (increment) for the index in the array. An integer. |
| NN - | Element count. Number of elements in the array. An integer. |

NOTE It is assumed that the lower limit for the array index is 1!

## Example of Use of a Function

```
PROGRAM CLEAR

REAL VA(2000)
INTEGER INCA,NN;

<Other program statements>

INCA = 2
NN = 1000

CALL VCLRXXX(VA,INCA,NN)

<Other program statements>

END
```

This function causes each second element of the vector VA to be set to zero (increment is 2).

For vectors where the elements are stored in consecutive locations, the index increment is equal to 1. The flexibility to specify index increments is present to most of the functions.

For complex vectors, each complex equation is represented by two consecutive elements. This corresponds to the real and the imaginary part of the complex vector. The real element is immediately followed by the imaginary element, as the complex vector is represented in the rectangular coordinate system.

This means that for each index in the array there are one real and one imaginary element.

Example of Use of a Complex Function:

```
PROGRAM CVMULIPLY

COMPLEX VA(1:100)
COMPLEX VB(1:100)
COMPLEX VC(1:100)

<Other program statements>

CALL CVMULXX(VA,2,VB,2,VC,1,50,1)

<Other program statements>

END
```

This function causes each second complex vector in array VA to be multiplied with each second vector from VB. The results are stored in consecutive locations in VC. This function corresponds to mathematical multiplication of complex numbers.

# 2 USING THE ARRAY PROCESSING FUNCTIONS

The array processing functions can be called from FORTRAN. The array processing library is used to transfer the parameters from the call to the array processing instructions. Thus the array processing functions are linked to the main program at load time as a part of the program.

Writing a FORTRAN program for array processing with ND-500 array processing functions is much the same as using the FORTRAN equivalent for the array processing function. Before starting an array processing function, input and output arrays for the operations must be defined. Initialization of input arrays is also required. This means that data for processing must be placed in the input arrays for the actual array processing function. Then the array processing function may be called. The result of the operation is present in the output array when returning from an array processing function.

Calling the function APMOVE is a bit different from the other array processing functions, because a function, named LOCARG, must be called before APMOVE. See also page 80.

An example of calling array processing functions is given below. Detailed layout of the different array processing functions is given in chapter 5.

<u>Example of Creating a FORTRAN Program Using Array Processing Functions</u>

<u>Source Program in FORTRAN:</u>

```
         PROGRAM DOKKT
C        Set name and size of arrays to be used.
         DIMENSION VA(100),VB(100),VC(100)
         INTEGER*2 I2(100)
         INTEGER*4 I4(100)
C        Initiate source array.
         I4(1) = 1
         I4(2) = 10
         I4(3) = 100
         I4(4) = 100
C        Get pointer to array I4.
         IADDA = LOCARG(I4)
C        Get pointer to array VC.
         IADDD = LOCARG(VC)
C        Set type of conversion with APMOVE  and element count.
         IFTM  = 0
         NN    = 4
C        Convert 32 bit integer to single floating point and move.
            CALL APMOVE(IADDA,IADDD,IFTM,NN,*100)
100      CONTINUE
```

<u>The program continues on next page</u>....

....Continued from previous page

```
C         Set index increment and element count.
          INCC  = 1
          NN    = 100
C         Clear the result array to be used in the next operation.
             CALL VCLRXXX(VA,INCC,NN,*101)
101       CONTINUE
          NN    = 4
C         Expand array VC with result in array VA.NC is returned.
             CALL VXPNDXX(VC,VA,NN,NC,*102)
102       CONTINUE

C         Set scalar value,index increments and element count.
          B     = 200.0
          INCA  = 1
          INCC  = 1
          NN    = NC
C         200.0 divided with each element of VA.Result in VB.
             CALL VDIVSXX(VA,INCA,B,VB,INCC,NN,*103)
103       CONTINUE
C         Set index increments and element count.
          INCA = 2
          INCB = 4
          INCC = 1
          NN   = 25
C         Add each second of VA to each fourth of VB.Result in VC.
             CALL VADDXXX(VA,INCA,VB,INCB,VC,INCC,NN,*104)
104       CONTINUE
          IADDD = LOCARG(I2)
          IADDA = LOCARG(VC)
          IFTM  = 4
C         Convert array VC to 16 bit integer.Result in I2.
             CALL APMOVE(IADDA,IADDD,IFTM,NN,*105)
105       CONTINUE
          IADDD = LOCARG(I4)
          IFTM  = 3
C         Convert array VC to 32 bit integer.Result in I4.
             CALL APMOVE(IADDA,IADDD,IFTM,NN,*106)
106       CONTINUE
C         Result of the operations in arrays VC,I4 and I2.
          WRITE (1,1000)
          DO FOR I=1,25
             WRITE (1,1001)VC(I),I4(I),I2(I)
          ENDDO
1000      FORMAT (' RESULT OF OPERATION IS :',/,
        + '....... REAL ........ INTEGER*4  INTEGER*2 .')
1001      FORMAT (3X,G12.7,5X,I11,5X,I6)
          END
```

Compiling the Program:

```
@FORTRAN-500 ↵

ND-500 ANSI 77 FORTRAN COMPILER - 203054H
FTN: COMPILE ↵

SOURCE-FILE DOKKT ↵

LIST-FILE ↵

OBJECT-FILE "DOKKT" ↵

- CPU TIME USED: 1.0 SECONDS.  67 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=401 DATA SIZE=2188 COMMON SIZE=0

FTN: EXIT ↵
```

Loading the Program:

```
@LINKAGE-LOADER ↵

ND-Linkage-Loader -  C        22. January    1982 Time: 15:5

N11: SET-DOMAIN DOKKT ↵

N11: LOAD DOKKT,ND-500-APF-LIB ↵
PROGRAM:.........555 P     DATA:...........4144 D
ND-500-APF-LIB
PROGRAM:........1004 P     DATA:...........4500 D
NLL: CLOSE Y ↵
Segment no........30       is linked

_____________________________________________________

22. January    1982 Time: 15:5

Unsatisfied references :

None!
_____________________________________________________

Defined symbols :

DOKKT...........4 P01 VADDXXX.......555 P01 VDIVSXX.......616 P01
VCLRXXX.......653 P01 VXPNDXX.......704 P01 APMOVE........732 P01
LOCARG........774 P01

Program:........1004 P     Data:...........6500 D

N11: EXIT ↵
```

Executing the Program:

```
N500: DOKKT ↵

 RESULT OF OPERATION IS :
...... REAL ........ INTEGER*4  INTEGER*2 .
  30.00000                30         30
  26.48485                26         26
  25.21531                25         25
  25.01976                25         25
  25.42088                25         25
  26.18768                26         26
  27.19480                27         27
  28.36829                28         28
  29.66173                29         29
  31.04448                31         31
  32.49554                32         32
  34.00000                34         34
  35.54700                35         35
  37.12843                37         37
  38.73813                38         38
  40.37133                40         40
  42.02425                42         42
  43.69392                43         43
  45.37788                45         45
  47.07420                47         47
  48.78122                48         48
  50.49760                50         50
  52.22221                52         52
  53.95409                53         53
  55.69241                55         55

N500: EXIT ↵
```

# 3 ND-500 ARRAY PROCESSING FUNCTIONS PERFORMANCE

## 3.1 ND-560/1 PROCESSING PERFORMANCE

| NAME | OPERATION | Number of elements | Improvement ratio to FTN function [1]) | Typical execution time pr. loop (microsec.) |
|---|---|---|---|---|
| CFFTXXX | COMPLEX FAST FOURIER TRANSFORM | 1024 | 3.80 | 62.25 |
| CONVXXX | CONVOLUTION (CORRELATION) | 150000 | 8.84 | 1.29 |
| CVMULXX | COMPLEX VECTOR MULTIPLY | 1500 | 3.20 | 8.55 |
| DOTPRXX | DOT PRODUCT | 1500 | 3.73 | 2.57 |
| IBMFPCV | IBM TO ND FLOATING POINT CONVERT | 1500 | 8.00 | 4.86 |
| IMGBLD | IMAGE BUILD | 1500 | 4.53 | 2.25 |
| MAXMGVX | MAX. MAGNITUDE ELEMENT IN VECTOR | 1500 | 5.43 | 1.54 |
| MAXMINX | MAX. AND MIN. VALUE IN VECTOR | 1500 | 5.50 | 2.38 |
| MAXVXXX | MAX. VALUE IN VECTOR | 1500 | 6.30 | 1.34 |
| MINMGVX | MIN. MAGNITUDE ELEMENT IN VECTOR | 1500 | 5.47 | 1.61 |
| MINVXXX | MIN. VALUE IN VECTOR | 1500 | 5.41 | 1.56 |
| MXMNMGX | MAX. AND MIN. MAG. ELEMENT IN VECTOR | 1500 | 4.54 | 2.62 |
| NDFPCV | ND TO IBM FLOATING POINT CONVERT | 1500 | 14.00 | 3.00 |
| PREDICT | PREDICT | 1500 | 4.82 | 2.85 |
| RFFTXXX | REAL FAST FOURIER TRANSFORM | 1024 | 2.40 | 58.60 |
| SVEMGXX | SUM OF VECTOR ELEMENTS MAGNITUDE | 1500 | 5.70 | 1.24 |
| SVESQXX | SUM OF VECTOR ELEMENTS SQUARE | 1500 | 4.20 | 1.93 |
| SVEXXXX | SUM OF VECTOR ELEMENTS | 1500 | 5.40 | 1.24 |
| SVSXXXX | SUM OF VECTOR ELEMENTS SIGNED SQUARE | 1500 | 5.65 | 1.93 |
| VABSXXX | VECTOR ABSOLUTE VALUE | 1500 | 6.48 | 1.27 |
| VADDXXX | VECTOR ADD | 1500 | 6.00 | 1.93 |
| VAVGABS | VECTOR AVERAGE ABSOLUTE VALUE | 1500 | 4.00 | 1.65 |
| VCLRXXX | VECTOR CLEAR | 1500 | 6.34 | 0.80 |
| VCOSXXX | VECTOR COSINE | 1500 | 1.40 | 15.80 |
| VDIVSXX | VECTOR SCALAR DIVIDE | 1500 | 3.77 | 2.90 |
| VDIVXXX | VECTOR DIVIDE | 1500 | 4.09 | 3.19 |
| VFLNZXX | VECTOR FIRST AND LAST NON-ZERO VALUE | 1500 | 8.47 | 0.61 |
| VGENXXX | VECTOR GENERATE | 1500 | 7.44 | 1.29 |
| VMAXMGX | VECTOR MAXIMUM MAGNITUDE | 1500 | 6.00 | 2.28 |
| VMAXXXX | VECTOR MAXIMUM | 1500 | 6.32 | 2.07 |
| VMINMGX | VECTOR MINIMUM MAGNITUDE | 1500 | 6.00 | 2.28 |
| VMINXXX | VECTOR MINIMUM | 1500 | 6.32 | 2.07 |
| VMOVXXX | VECTOR MOVE | 1500 | 6.39 | 1.16 |
| VMULXXX | VECTOR MULTIPLY | 1500 | 5.93 | 1.92 |
| VNEGXXX | VECTOR NEGATIVE | 1500 | 7.40 | 1.16 |
| VNMOSXX | VECTOR SINC INTERPOLATION | 1500 | 3.88 | 21.68 |

[1]) (Time used by machine code) / (Time used by microcode).

[2]) ASM - ND-500 assembler.

| NAME | OPERATION | Number of elements | Improvement ratio to FTN function 1) | Typical execution time pr. loop (microsec.) |
|---|---|---|---|---|
| VNMOXXX | VECTOR LINEAR INTERPOLATION | 1500 | 3.12 | 8.16 |
| VSADDXX | VECTOR SCALAR ADD | 1500 | 5.67 | 1.64 |
| VSINXXX | VECTOR SINE | 1500 | 1.40 | 15.09 |
| VSMADDX | VECTOR SCALAR MULTIPLY AND ADD | 1500 | 5.30 | 2.21 |
| VSMULXX | VECTOR SCALAR MULTIPLY | 1500 | 5.68 | 1.64 |
| VSQRTXX | VECTOR SQUARE ROOT | 1500 | 1.78 | 8.85 |
| VSQXXXX | VECTOR SQUARE | 1500 | 5.10 | 1.93 |
| VSSQXXX | VECTOR SIGNED SQUARE | 1500 | 6.20 | 1.93 |
| VSUBXXX | VECTOR SUBTRACT | 1500 | 6.00 | 1.92 |
| VSWAPXX | VECTOR SWAP | 1500 | 5.20 | 2.00 |
| VTAPERX | VECTOR TAPER | 1500 | 3.00 | 2.07 |
| VXPNDXX | VECTOR EXPAND | 1500 | 4.07 | 1.77 |
| WIENERX | WIENER FILTER | 81 | 3.03 | 365.40 |
| XBTMUX | DISPLAY PROCESSOR FUNCTION | 1500 | 5.45 | 50.00 |

| NAME | OPERATION | Number of elements | Improvement ratio to ASM function 2) | Typical execution time pr. loop (microsec.) |
|---|---|---|---|---|
| APMOVE | CONVERT AND MOVE FORMAT 0 | 1500 | 2.68 | 1.38 |
| APMOVE | CONVERT AND MOVE FORMAT 1 | 1500 | 2.38 | 1.53 |
| APMOVE | CONVERT AND MOVE FORMAT 2 & 5 | 1500 | 1.17 | 0.89 |
| APMOVE | CONVERT AND MOVE FORMAT 3 | 1500 | 2.57 | 1.38 |
| APMOVE | CONVERT AND MOVE FORMAT 4 | 1500 | 2.86 | 1.53 |
| DMXB | DEMULTIPLEX SEGMENT B | 1500 | 2.66 | 3.01 |

The figures have been taken from a ND-560/1 with 128 k byte cache memory.

1) (Time used by machine code) / (Time used by microcode).

2) ASM - ND-500 assembler.

## 3.2 ND-570/2 PROCESSING PERFORMANCE

| NAME | OPERATION | Number of elements | Improvement ratio to FTN function [1] | Typical execution time pr. loop (microsec.) |
|---|---|---|---|---|
| CFFTXXX | COMPLEX FAST FOURIER TRANSFORM | 1024 | 3.35 | 41.50 |
| CONVXXX | CONVOLUTION (CORRELATION) | 150000 | 6.50 | 0.93 |
| CVMULXX | COMPLEX VECTOR MULTIPLY | 1500 | 2.60 | 6.40 |
| DOTPRXX | DOT PRODUCT | 1500 | 3.40 | 1.72 |
| IBMFPCV | IBM TO ND FLOATING POINT CONVERT | 1500 | 7.15 | 3.29 |
| IMGBLD | IMAGE BUILD | 1500 | 3.50 | 1.85 |
| MAXMGVX | MAX. MAGNITUDE ELEMENT IN VECTOR | 1500 | 5.56 | 1.00 |
| MAXMINX | MAX. AND MIN. VALUE IN VECTOR | 1500 | 3.98 | 1.61 |
| MAXVXXX | MAX. VALUE IN VECTOR | 1500 | 4.64 | 0.90 |
| MINMGVX | MIN. MAGNITUDE ELEMENT IN VECTOR | 1500 | 5.62 | 1.05 |
| MINVXXX | MIN. VALUE IN VECTOR | 1500 | 4.15 | 1.01 |
| MXMNMGX | MAX. AND MIN. MAG. ELEMENT IN VECTOR | 1500 | 4.31 | 1.76 |
| NDFPCV | ND TO IBM FLOATING POINT CONVERT | 1500 | 9.95 | 2.50 |
| PREDICT | PREDICT | 1500 | 4.78 | 1.61 |
| RFFTXXX | REAL FAST FOURIER TRANSFORM | 1024 | 2.08 | 37.50 |
| SVEMGXX | SUM OF VECTOR ELEMENTS MAGNITUDE | 1500 | 5.25 | 0.74 |
| SVESQXX | SUM OF VECTOR ELEMENTS SQUARE | 1500 | 3.70 | 1.20 |
| SVEXXXX | SUM OF VECTOR ELEMENTS | 1500 | 4.85 | 0.74 |
| SVSXXXX | SUM OF VECTOR ELEMENTS SIGNED SQUARE | 1500 | 5.25 | 1.20 |
| VABSXXX | VECTOR ABSOLUTE VALUE | 1500 | 5.35 | 0.83 |
| VADDXXX | VECTOR ADD | 1500 | 3.70 | 1.60 |
| VAVGABS | VECTOR AVERAGE ABSOLUTE VALUE | 1500 | 4.40 | 0.89 |
| VCLRXXX | VECTOR CLEAR | 1500 | 5.60 | 0.48 |
| VCOSXXX | VECTOR COSINE | 1500 | 1.47 | 10.67 |
| VDIVSXX | VECTOR SCALAR DIVIDE | 1500 | 2.45 | 2.40 |
| VDIVXXX | VECTOR DIVIDE | 1500 | 2.82 | 2.52 |
| VFLNZXX | VECTOR FIRST AND LAST NON-ZERO VALUE | 1500 | 7.62 | 0.39 |
| VGENXXX | VECTOR GENERATE | 1500 | 5.25 | 0.92 |
| VMAXMGX | VECTOR MAXIMUM MAGNITUDE | 1500 | 3.68 | 1.97 |
| VMAXXXX | VECTOR MAXIMUM | 1500 | 3.75 | 1.85 |
| VMINMGX | VECTOR MINIMUM MAGNITUDE | 1500 | 3.68 | 1.97 |
| VMINXXX | VECTOR MINIMUM | 1500 | 3.75 | 1.85 |
| VMOVXXX | VECTOR MOVE | 1500 | 5.50 | 0.80 |
| VMULXXX | VECTOR MULTIPLY | 1500 | 3.70 | 1.60 |
| VNEGXXX | VECTOR NEGATIVE | 1500 | 5.65 | 0.82 |
| VNMOSXX | VECTOR SINC INTERPOLATION | 150 | 3.00 | 14.90 |

[1] (Time used by machine code) / (Time used by microcode).

[2] ASM - ND-500 assembler.

| NAME | OPERATION | Number of elements | Improvement ratio to FTN function [1] | Typical execution time pr. loop (microsec.) |
|---|---|---|---|---|
| VNMOXXX | VECTOR LINEAR INTERPOLATION | 1500 | 2.38 | 5.52 |
| VSADDXX | VECTOR SCALAR ADD | 1500 | 3.25 | 1.53 |
| VSINXXX | VECTOR SINE | 1500 | 1.47 | 10.60 |
| VSMADDX | VECTOR SCALAR MULTIPLY AND ADD | 1500 | 3.40 | 1.90 |
| VSMULXX | VECTOR SCALAR MULTIPLY | 1500 | 3.27 | 1.50 |
| VSQRTXX | VECTOR SQUARE ROOT | 1500 | 1.55 | 6.02 |
| VSQXXXX | VECTOR SQUARE | 1500 | 4.20 | 1.18 |
| VSSQXXX | VECTOR SIGNED SQUARE | 1500 | 5.70 | 1.18 |
| VSUBXXX | VECTOR SUBTRACT | 1500 | 3.70 | 1.60 |
| VSWAPXX | VECTOR SWAP | 1500 | 2.80 | 2.10 |
| VTAPERX | VECTOR TAPER | 1500 | 2.53 | 1.37 |
| VXPNDXX | VECTOR EXPAND | 1500 | 3.30 | 1.13 |
| WIENERX | WIENER FILTER | 81 | 1.90 | 251.85 |
| XBTMUX | DISPLAY PROCESSOR FUNCTION | 1500 | 4.45 | 32.10 |

| NAME | OPERATION | Number of elements | Improvement ratio to ASM function [2] | Typical execution time pr. loop (microsec.) |
|---|---|---|---|---|
| APMOVE | CONVERT AND MOVE FORMAT 0 | 1500 | 1.83 | 1.24 |
| APMOVE | CONVERT AND MOVE FORMAT 1 | 1500 | 2.40 | 1.00 |
| APMOVE | CONVERT AND MOVE FORMAT 2 & 5 | 1500 | 1.73 | 0.49 |
| APMOVE | CONVERT AND MOVE FORMAT 3 | 1500 | 1.90 | 1.25 |
| APMOVE | CONVERT AND MOVE FORMAT 4 | 1500 | 3.68 | 0.75 |
| DMXB | DEMULTIPLEX SEGMENT B | 1500 | 2.60 | 1.94 |
| VTAPERX | VECTOR TAPER | 1500 | 2.53 | 1.37 |

The figures have been taken from a ND-570/2 with 32k bytes of cache memory.

[1]) (Time used by machine code) / (Time used by microcode).

[2]) ASM - ND-500 assembler.

# 4 ARRAY PROCESSING FUNCTIONS

## 4.1 INTRODUCTION

This chapter contains a listing of each array processing function. For each routine, the required parameter list for calling the array processing function is included together with definitions.

The ND-500 array processing functions are implemented as one machine instruction, except the functions DMXB, WIENERX, CFFTXXX and RFFTXXX, which are partly microcoded. These are implemented as different instructions to be executed consecutively.

The function DMXB uses two instruction codes. Functions XBTMUX and IMGBLD use a third instruction code. All of the other functions use a fourth instruction code. The contents of the record register are the only difference between the functions and are used to distinguish between them.

For each routine, an identification number is given as a cross reference between the object code and the array processing function. This identification number is given as two octal numbers :
'Ident (R:I) : xxx:nnnnnnB'. 'xxx' are the contents of the record register. 'nnnnnn' is the instruction code used for the processing function.

The library for the ND-500 array processing functions consists of one routine for each of the array processing functions. Each routine builds a data stack used by the array processing function to find addresses of input and output arrays, scalar values or addresses, index increments and element counts.

An address is a pointer to the logical memory for both input and output arrays.

Scalars to be used in an operation are located in the data stack as 32 bit floating point numbers.

Scalars to be returned from an operation are returned to the address given in the data stack.

Index increments and element counts are given in the data stack as 32 bit integers.

It is not necessary to use the alternative return argument when calling a routine.

## 4.2 VECTOR ADD (VADDXXX)

### Format

VADDXXX(VA,INCA,VB,INCB,VC,INCC,NN,*)        Ident (R:I) : 001:177517B

### Explanation

Add the corresponding elements of two vectors. VCn = VAn + VBn, 'n' is the element index.

### Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

### Listing

```
      SUBROUTINE VADDXXX(VA,INCA,VB,INCB,VC,INCC,NN,*)
      DIMENSION VA(1),VB(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = VB(IB) + VA(IA)
         IA = IA + INCA
         IB = IB + INCB
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.3 VECTOR SUBTRACT (VSUBXXX)

Format

VSUBXXX(VA,INCA,VB,INCB,VC,INCC,NN,*) Ident (R:I) : 002:177517B

Explanation

Subtract the corresponding elements of two vectors. VCn = VBn - VAn, 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VSUBXXX(VA,INCA,VB,INCB,VC,INCC,NN,*)
      DIMENSION VA(1),VB(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = VB(IB) - VA(IA)
         IA = IA + INCA
         IB = IB + INCB
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.4 VECTOR MULTIPLY (VMULXXX)

Format

```
VMULXXX(VA,INCA,VB,INCB,VC,INCC,NN,*)        Ident (R:I) : 003:177517B
```

Explanation

Multiply the corresponding elements of two vectors. VCn = VBn * VAn, 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VMULXXX(VA,INCA,VB,INCB,VC,INCC,NN,*)
      DIMENSION VA(1),VB(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = VB(IB) * VA(IA)
         IA = IA + INCA
         IB = IB + INCB
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.5 VECTOR DIVIDE (VDIVXXX)

Format

VDIVXXX(VA,INCA,VB,INCB,VC,INCC,NN,*) Ident (R:I) : 004:177517B

Explanation

Divide the corresponding elements of two vectors. VCn = VBn/VAn, 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VDIVXXX(VA,INCA,VB,INCB,VC,INCC,NN,*)
      DIMENSION VA(1),VB(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = VB(IB) / VA(IA)
         IA = IA + INCA
         IB = IB + INCB
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.6 VECTOR MAXIMUM (VMAXXXX)

### Format

VMAXXXX(VA,INCA,VB,INCB,VC,INCC,NN,*)        Ident (R:I) : 005:177517B

### Explanation

Form a vector from the maximum value of each corresponding pair of elements of two vectors. VCn = VAn if VAn > VBn, else VCn = VBn. 'n' is the element index.

### Parameters

|  |  |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

### Listing

```
      SUBROUTINE VMAXXXX(VA,INCA,VB,INCB,VC,INCC,NN,*)
      DIMENSION VA(1),VB(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = AMAX1(VA(IA),VB(IB))
         IA = IA + INCA
         IB = IB + INCB
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.7 VECTOR MINIMUM (VMINXXX)

Format

VMINXXX(VA,INCA,VB,INCB,VC,INCC,NN,*) Ident (R:I) : 006:177517B

Explanation

Form a vector from the minimum value of each corresponding pair of elements of two vectors. VCn = VAn if VAn < VBn, else VCn = VBn. 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VMINXXX(VA,INCA,VB,INCB,VC,INCC,NN,*)
      DIMENSION VA(1),VB(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = AMIN1(VA(IA),VB(IB))
         IA = IA + INCA
         IB = IB + INCB
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.8 VECTOR MAXIMUM MAGNITUDE (VMAXMGX)

Format

VMAXMGX(VA,INCA,VB,INCB,VC,INCC,NN,*)          Ident (R:I) : 007:177517B

Explanation

Form a vector from the maximum absolute value of each corresponding pair of elements of two vectors. VCn = |VAn| if |VAn| > |VBn|, else VCn = |VBn|. 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VMAXMGX(VA,INCA,VB,INCB,VC,INCC,NN,*)
      DIMENSION VA(1),VB(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = AMAX1(ABS(VA(IA)),ABS(VB(IB)))
         IA = IA + INCA
         IB = IB + INCB
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.9 VECTOR MINIMUM MAGNITUDE (VMINMGX)

Format

VMINMGX(VA,INCA,VB,INCB,VC,INCC,NN,*)        Ident (R:I) : 010:177517B

Explanation

Form a vector from the minimum absolute value of each corresponding pair of elements of two vectors. VCn = |VAn| if |VAn| < |VBn|, else VCn = |VBn|. 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VMINMGX(VA,INCA,VB,INCB,VC,INCC,NN,*)
      DIMENSION VA(1),VB(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = AMIN1(ABS(VA(IA)),ABS(VB(IB)))
         IA = IA + INCA
         IB = IB + INCB
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.10 VECTOR SQUARE (VSQXXXX)

Format

VSQXXXX(VA,INCA,VC,INCC,NN,*)                    Ident (R:I) : 042:177517B

Explanation

Square the elements of a vector. VCn = $(\text{VAn})^2$. 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VSQXXXX(VA,INCA,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) =  VA(IA)**2
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## *4.11 VECTOR SIGNED SQUARE (VSSQXXX)*

Format

VSSQXXX(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 011:177517B

Explanation

Multiply each element of a vector with the absolute value of itself. VCn = VAn * |VAn|. 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VSSQXXX(VA,INCA,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = SIGN(VA(IA)**2,VA(IA))
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## *4.12 VECTOR ABSOLUTE VALUE (VABSXXX)*

<u>Format</u>

VABSXXX(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 012:177517B

<u>Explanation</u>

Form a vector from the absolute values of the elements in a vector.
VCn = |VAn|. 'n' is the element index.

<u>Parameters</u>

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

<u>Listing</u>

```
      SUBROUTINE VABSXXX(VA,INCA,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = ABS(VA(IA))
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.13 VECTOR SQUARE ROOT (VSQRTXX)

Format

VSQRTXX(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 013:177517B

Explanation

Take the square roots of the elements in a vector. VCn = √VAn. 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VSQRTXX(VA,INCA,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = SQRT(VA(IA))
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## *4.14 VECTOR SINE (VSINXXX)*

Format

VSINXXX(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 014:177517B

Explanation

Compute the sine of the elements of a vector. VCn = sin(VAn). 'n' is the element index. The arguments in VA must be in radians.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VSINXXX(VA,INCA,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = SIN(VA(IA))
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.15 VECTOR COSINE (VCOSXXX)

Format

VCOSXXX(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 015:177517B

Explanation

Compute the cosine of the elements of a vector. VCn = cos(VAn). 'n' is the element index. The arguments in VA must be in radians.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VCOSXXX(VA,INCA,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = COS(VA(IA))
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## *4.16 VECTOR MOVE (VMOVXXX)*

Format

VMOVXXX(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 016:177517B

Explanation

Move the elements from one vector into another. VCn = VAn, 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VMOVXXX(VA,INCA,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = VA(IA)
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## *4.17 VECTOR SWAP (VSWAPXX)*

Format

VSWAPXX(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 063:177517B

Explanation

Swap the elements between two vectors. VAn → VBn and VBn → VAn, 'n' is the element index.

Parameters

|  |  |
|---|---|
| VA | : Name of input and output vector VA. |
| INCA | : VA index increment. |
| VC | : Name of input and output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VSWAPXX(VA,INCA,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      REAL HOLD
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         HOLD   = VC(IC)
         VC(IC) = VA(IA)
         VA(IA) = HOLD
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.18 VECTOR NEGATIVE (VNEGXXX)

Format

VNEGXXX(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 064:177517B

Explanation

Form a vector from the elements of another vector multiplied with -1.
VCn = -VAn, 'n' is the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VNEGXXX(VA,INCA,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = - VA(IA)
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.19 ND TO IBM FLOATING POINT CONVERT (NDFPCV)

Format

NDFPCV(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 017:177517B

Explanation

Convert the elements of a vector into ND floating point format to IBM floating point format. Symbolically this can be represented by the formula: VCn = IBMFP(VAn), 'n' denotes the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE NDFPCV(VA,INCA,VC,INCC,NN,*)
      REAL VA(1),VC(1),TEMP
      INTEGER MANT,CHAR,HIDBIT,HEXCHR,RSHFT,ITEMP
      INTEGER ROUND(3)
      EQUIVALENCE (TEMP,ITEMP)
      DATA MASK1   /00017777777B/
      DATA HIDBIT  /00020000000B/
      DATA MASK2   /17777777777B/
      DATA MASK3   /20000000000B/
      DATA ROUND   /1,2,4/
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         TEMP = VA(IA)
         IF (ITEMP .EQ. 0) THEN
            VC(IC) = 0.0
            GO TO 100
         ENDIF
         MANT = (IAND(ITEMP,MASK1)  + HIDBIT)*2
         CHAR = IAND(ITEMP,MASK2)
         CHAR = ISHFT(CHAR,-22)
         HEXCHR = CHAR/4
         RSHFT = 4 - MOD(CHAR,4)
         IF (RSHFT .NE. 4) THEN
            MANT = MANT + ROUND(RSHFT)
            MANT = ISHFT(MANT,-RSHFT)
            HEXCHR = HEXCHR + 1
         ENDIF
         IF (HEXCHR .GT. 127) THEN
```

```
            HEXCHR = 127
            MANT = 00077777777B
         ENDIF
         ITEMP = ISHFT(HEXCHR,24) + MANT
         IF (VA(IA) .LT. 0.) ITEMP = IOR(ITEMP,MASK3)
         VC(IC) = TEMP
 100  CONTINUE
      IA = IA + INCA
      IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## *4.20 IBM TO ND FLOATING POINT CONVERT (IBMFPCV)*

Format

IBMFPCV(VA,INCA,VC,INCC,NN,*) Ident (R:I) : 020:177517B

Explanation

To convert the elements of a vector in IBM floating point format into ND floating point format. Symbolically this can be represented by the formula: VCn = NDFP(VAn), 'n' denotes the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE IBMFPCV(VA,INCA,VC,INCC,NN,*)
      REAL VA(1),VC(1),TEMP
      INTEGER MANTSFT,CHAR,RSHFT,ITEMP,SHFTCNT
      EQUIVALENCE (TEMP,ITEMP)
      DATA MASK1 /17700000000B/
      DATA MASK2 /00077777777B/
      DATA MASK3 /20000000000B/
      DATA MASK4 /00074000000B/
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         SHFTCNT = -1
         TEMP = VA(IA)
         IF (ITEMP .EQ. 0) THEN
            VC(IC) = 0.0
            GO TO 100
         ENDIF
         CHAR = IAND(ITEMP,MASK1)
         CHAR = ISHFT(CHAR,-24)*4
         MANTSFT = IAND(ITEMP,MASK4)
         MANTSFT = ISHFT(MANTSFT,-20)
         IF (MANTSFT.GT.0) SHFTCNT=4
         IF (MANTSFT.GT.1) SHFTCNT=3
         IF (MANTSFT.GT.3) SHFTCNT=2
         IF (MANTSFT.GT.7) SHFTCNT=1
         CHAR = CHAR-SHFTCNT+1
         ITEMP = ISHFT(ITEMP,SHFTCNT)
         ITEMP = IAND (ITEMP,MASK2)
         ITEMP = ISHFT(ITEMP,-2) + ISHFT(CHAR,22)
         IF ( VA(IA) .LT. O. ) ITEMP = IOR (ITEMP,MASK3)
         VC(IC) = TEMP
         IF(SHFTCNT.EQ.-1)VC(IC)=17777777777B
 100     CONTINUE
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.21 SUM OF VECTOR ELEMENTS (SVEXXXX)

Format

SVEXXXX(VA,INCA,VC,NN,*) Ident (R:I) : 021:177517B

Explanation

Add the elements of a vector. $VC = VA_1 + VA_2 + \ldots + VAnn$, 'nn' is the element count.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output scalar VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE SVEXXXX(VA,INCA,VC,NN,*)
      DIMENSION VA(1)
      IA  = 1
      SUM = 0.0
      DO FOR M = 1,NN
         SUM = SUM + VA(IA)
         IA  = IA + INCA
      ENDDO
      VC = SUM
      RETURN 1
      END
```

## *4.22 SUM OF VECTOR ELEMENTS MAGNITUDE (SVEMGXX)*

Format

SVEMGXX(VA,INCA,VC,NN,*) Ident (R:I) : 065:177517B

Explanation

Form the sum of the absolute values of the elements of a vector.
$VC = |VA_1| + |VA_2| + ....+ |VAnn|$ , 'nn' is the element count.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output scalar VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE SVEMGXX(VA,INCA,VC,NN,*)
      DIMENSION VA(1)
      IA  = 1
      SUM = 0.0
      DO FOR M = 1,NN
         SUM = SUM + ABS(VA(IA))
         IA  = IA + INCA
      ENDDO
      VC = SUM
      RETURN 1
      END
```

## 4.23 SUM OF VECTOR ELEMENTS SQUARE (SVESQXX)

Format

SVESQXX(VA,INCA,VC,NN,*) Ident (R:I) : 066:177517B

Explanation

Form the sum of the squared elements of a vector.
$VC = (VA_1)^2 + (VA_2)^2 + \ldots + (VAnn)^2$, 'nn' is the element count.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output scalar VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE SVESQXX(VA,INCA,VC,NN,*)
      DIMENSION VA(1)
      IA = 1
      SUM= 0.0
      DO FOR M = 1,NN
         SUM=SUM + VA(IA)**2
         IA = IA + INCA
      ENDDO
      VC=SUM
      RETURN 1
      END
```

## *4.24 SUM OF VECTOR ELEMENTS SIGNED SQUARE (SVSXXXX)*

Format

SVSXXXX(VA,INCA,VC,NN,*) Ident (R:I) : 022:177517B

Explanation

Form the sum of the elements of a vector, where each element at first is multiplied with the absolute value of itself.
$VC = VA_1 * |VA_1| + VA_2 * |VA_2| + \ldots + VAnn * |VAnn|$, 'nn' is the element count.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output scalar VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE SVSXXXX(VA,INCA,VC,NN,*)
      DIMENSION VA(1)
      IA = 1
      SUM= 0.0
      DO FOR M = 1,NN
         SUM=SUM + SIGN(VA(IA)*VA(IA),VA(IA))
         IA = IA + INCA
      ENDDO
      VC=SUM
      RETURN 1
      END
```

## 4.25 VECTOR AVERAGE ABSOLUTE VALUE (VAVGABS)

Format

VAVGABS(VA,INCA,VC,NN,*) Ident (R:I) : 023:177517B

Explanation

Form the mean value of the absolute values of the elements of a vector. VC = ( $|VA_1|$ + $|VA_2|$ + ....+ $|VAnn|$) / nn , 'nn' is the element count.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output scalar VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VAVGABS(VA,INCA,VC,NN,*)
      DIMENSION VA(1)
      IA = 1
      SUMABS = 0.0
      DO FOR I = 1,NN
         SUMABS = SUMABS + ABS(VA(IA))
         IA = IA + INCA
      ENDDO
      VC = SUMABS/NN
      RETURN 1
      END
```

## *4.26 MAXIMUM VALUE IN VECTOR (MAXVXXX)*

Format

MAXVXXX(VA,INCA,VC,NN,*) Ident (R:I) : 024:177517B

Explanation

Scan a vector for its element with maximum value and return this ($VC_1$) together with the corresponding index ($VC_2$).

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE MAXVXXX(VA,INCA,VC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      VC(1) = VA(IA)
      VC(2) = IA
      DO FOR M = 2,NN
         IA = IA + INCA
         IF (VA(IA) .GT. VC(1)) THEN
            VC(1) = VA(IA)
            VC(2) = IA
         ENDIF
      ENDDO
      RETURN 1
      END
```

## 4.27 MINIMUM VALUE IN VECTOR (MINVXXX)

Format

MINVXXX(VA,INCA,VC,NN,*) Ident (R:I) : 025:177517B

Explanation

Scan a vector for its element with minimum value and return this ($VC_1$) together with the corresponding index ($VC_2$).

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE MINVXXX(VA,INCA,VC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      VC(1) = VA(IA)
      VC(2) = IA
      DO FOR M = 2,NN
         IA = IA + INCA
         IF (VA(IA) .LT. VC(1)) THEN
            VC(1) = VA(IA)
            VC(2) = IA
         ENDIF
      ENDDO
      RETURN 1
      END
```

## 4.28 MAXIMUM MAGNITUDE ELEMENT IN VECTOR (MAXMGVX)

Format

MAXMGVX(VA,INCA,VC,NN,*) Ident (R:I) : 026:177517B

Explanation

Scan a vector for its element with maximum absolute value, and return this ($VC_1$) together with the corresponding index ($VC_2$).

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE MAXMGVX(VA,INCA,VC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      VC(1) = ABS(VA(IA))
      VC(2) = IA
      DO FOR M = 2,NN
         IA = IA + INCA
         VAABS = ABS(VA(IA))
         IF (VAABS .GT. VC(1)) THEN
            VC(1) = VAABS
            VC(2) = IA
         ENDIF
      ENDDO
      RETURN 1
      END
```

## 4.29 MINIMUM MAGNITUDE ELEMENT IN VECTOR (MINMGVX)

Format

MINMGVX(VA,INCA,VC,NN,*) Ident (R:I) : 027:177517B

Explanation

Scan a vector for its element with minimum absolute value, and return this ($VC_1$) together with the corresponding index ($VC_2$).

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE MINMGVX(VA,INCA,VC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      VC(1) = ABS(VA(IA))
      VC(2) = IA
      DO FOR M = 2,NN
         IA = IA + INCA
         VAABS = ABS(VA(IA))
         IF (VAABS .LT. VC(1)) THEN
            VC(1) = VAABS
            VC(2) = IA
         ENDIF
      ENDDO
      RETURN 1
      END
```

## *4.30 MAXIMUM AND MINIMUM VALUE IN VECTOR (MAXMINX)*

Format

MAXMINX(VA,INCA,VC,NN,*) Ident (R:I) : 030:177517B

Explanation

Scan a vector for its element with maximum value and its element with minimum value. The maximum value is returned in $VC_1$, and with index for VA in $VC_3$. The minimum value is returned in $VC_2$, and with index for VA in $VC_4$.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE MAXMINX(VA,INCA,VC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      VC(1) = VA(IA)
      VC(2) = VA(IA)
      VC(3) = IA
      VC(4) = IA
      DO FOR M = 2,NN
         IA = IA + INCA
         IF (VA(IA) .GT. VC(1)) THEN
            VC(1) = VA(IA)
            VC(3) = IA
         ELSEIF (VA(IA) .LT. VC(3)) THEN
            VC(2) = VA(IA)
            VC(4) = IA
         ENDIF
      ENDDO
      RETURN 1
      END
```

## 4.31 MAXIMUM AND MINIMUM MAGNITUDE ELEMENT IN VECTOR (MXMNMGX)

Format

MXMNMGX(VA,INCA,VC,NN,*) Ident (R:I) : 031:177517B

Explanation

Scan a vector for its element with absolute maximum value and its element with minimum absolute value. The maximum value is returned in $VC_1$ , and with index for VA in $VC_3$. The minimum value is returned in $VC_2$ , and with index for VA in $VC_4$.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VC | : Name of output vector VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE MXMNMGX(VA,INCA,VC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      VC(1) = ABS(VA(IA))
      VC(2) = ABS(VA(IA))
      VC(3) = IA
      VC(4) = IA
      DO FOR M = 2,NN
         IA = IA + INCA
         VAABS = ABS(VA(IA))
         IF (VAABS .GT. VC(1)) THEN
            VC(1) = VAABS
            VC(3) = IA
         ELSEIF (VAABS .LT. VC(3)) THEN
            VC(2) = VAABS
            VC(4) = IA
         ENDIF
      ENDDO
      RETURN 1
      END
```

## 4.32 VECTOR SCALAR ADD (VSADDXX)

Format

VSADDXX(VA,INCA,B,VC,INCC,NN,*)    Ident (R:I) : 032:177517B

Explanation

Add the elements of a vector together with a scalar value.
VCn = VAn + b, where 'b' denotes the scalar, and 'n' denotes the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| B | : Scalar B. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VSADDXX(VA,INCA,B,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = B + VA(IA)
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.33 VECTOR SCALAR MULTIPLY (VSMULXX)

Format

VSMULXX(VA,INCA,B,VC,INCC,NN,*) Ident (R:I) : 033:177517B

Explanation

Multiply the elements of a vector with a scalar value.
VCn = VAn * b, where 'b' denotes the scalar, and 'n' denotes the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| B | : Scalar B. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VSMULXX(VA,INCA,B,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = B * VA(IA)
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.34 VECTOR SCALAR DIVIDE (VDIVSXX)

Format

VDIVSXX(VA,INCA,B,VC,INCC,NN,*) Ident (R:I) : 034:177517B

Explanation

Form a vector from a scalar value divided with the elements of another vector. VCn = b/VAn, 'b' denotes the scalar and 'n' denotes the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| B | : Scalar B. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VDIVSXX(VA,INCA,B,VC,INCC,NN,*)
      DIMENSION VA(1),VC(1)
      IA = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = B / VA(IA)
         IA = IA + INCA
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## 4.35 DOT PRODUCT (DOTPRXX)

Format

DOTPRXX(VA,INCA,VB,INCB,VC,NN,*) Ident (R:I) : 035:177517B

Explanation

Add the product of the corresponding elements of two vectors. This function corresponds to the mathematical dot product, also called scalar product, of two vectors.
$VC = VA_1 * VB_1 + VA_2 * VB_2 + \ldots. + VAnn * VBnn$ , 'nn' is the element count.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output scalar VC. |
| NN | : Element count. |

Listing

```
      SUBROUTINE DOTPRXX(VA,INCA,VB,INCB,VC,NN,*)
      DIMENSION VA(1),VB(1)
      IA = 1
      IB = 1
      SUM = 0.0
      DO FOR M = 1,NN
         SUM = SUM + VA(IA) * VB(IB)
         IA = IA + INCA
         IB = IB + INCB
      ENDDO
      VC=SUM
      RETURN 1
      END
```

## *4.36 VECTOR CLEAR (VCLRXXX)*

Format

VCLRXXX(VC,INCC,NN,*) Ident (R:I) : 036:177517B

Explanation

Set the elements of a vector to all zeros.

Parameters

|  |  |
|---|---|
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VCLRXXX(VC,INCC,NN,*)
      DIMENSION VC(1)
      IC = 1
      DO FOR M = 1,NN
         VC(IC) = 0.0
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## *4.37 CONVOLUTION (CONVXXX)*

Format

```
CONVXXX(VA,INCA,VB,INCB,VC,INCC,NC,NB,*)      Ident (R:I) : 037:177517B
                                              Ident (R:I) : 067:177517B
```

Explanation

Perform a convolution or correlation operation on two vectors. The general equation for the output coefficients in vector VC is:

$$VC_{ic} = \sum_{ib=0}^{NB-1} (VA_{ic+ib} * VB_{ib})$$

'ib' and 'ic' denote indices for VB and VC.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA, operand. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB, operator. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NC | : Element count for vector VC. |
| NB | : Element count for vector VB. |

NOTE The element count for vector VA must be: NB+NC-1.

Listing

```
      SUBROUTINE CONVXXX(VA,INCA,VB,INCB,VC,INCC,NC,NB,*)
      DIMENSION VA(1),VB(1),VC(1)
      DO FOR N = 0,NC-1
         IC  = N*INCC+1
         SUM = 0.0
         DO FOR M = 0,NB-1
            IA = (N+M) * INCA + 1
            IB =     M * INCB + 1
            SUM = SUM + VA(IA)*VB(IB)
         ENDDO
         VC(IC) = SUM
      ENDDO
      RETURN 1
      END
```

## Performance

The execution time for the function depends on the element counts of NB and NC. Approximate execution time formulas (ftime) for the function are as follows.

ND-560/1:

ftime [microsec.] = NC * (1.35 * (NB-2) + 5.7) , NB $\leq$ 1000.

ftime [microsec.] = NC * (2.30 * NB + 4.0) , NB > 1000.

ND-570/2:

ftime [microsec.] = NC * (0.88 * (NB-2) + 3.0) , NB $\leq$ 1000.

ftime [microsec.] = NC * (1.26 * NB + 2.0) , NB > 1000.

## *4.38 COMPLEX VECTOR MULTIPLY (CVMULXX)*

Format

CVMULXX(VA,INCA,VB,INCB,VC,INCC,NN,NF,*)     Ident (R:I) : 040:177517B

Explanation

Multiply two complex vectors. This function corresponds to mathematical multiplication of complex numbers. An own flag selects whether the result should be conjugated or not. VA = VAr + VAi, VB = VBr + VBi.

If the conjugate flag $\geq$ 0 then:

VC = (VAr * VBr - VAi*VBi)r + (VAr*VBi + VAi*VBr)i, else:

VC = (VAr * VBr - VAi*VBi)r - (VAr*VBi + VAi*VBr)i.

'r' and 'i' denotes real and imaginary elements.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |
| NF | : Conjugate flag. |

| | |
|---|---|
| NF = +1 | : Normal complex multiply. |
| NF = -1 | : Multiply with conjugate of VA. |

Listing

```
      SUBROUTINE CVMULXX(VA,INCA,VB,INCB,VC,INCC,NN,NF,*)
      DIMENSION VA(1),VB(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR M = 1,NN
         VC(IC)   = VA(IA)*VB(IB)-VA(IA+1)*VB(IB+1)*NF
         VC(IC+1) = VA(IA)*VB(IB+1)+VA(IA+1)*VB(IB)*NF
         IA = IA + INCA
         IB = IB + INCB
         IC = IC + INCC
      ENDDO
      RETURN 1
      END
```

## *4.39 COMPLEX FAST FOURIER TRANSFORM (CFFTXXX)*

Format

| | |
|---|---|
| CFFTXXX(C,N,LF,*) | Ident (R:I) : 054:177517B |
| | Ident (R:I) : 055:177517B |
| | Ident (R:I) : 056:177517B |
| | Ident (R:I) : 057:177517B |
| | Ident (R:I) : xxx:177516B |

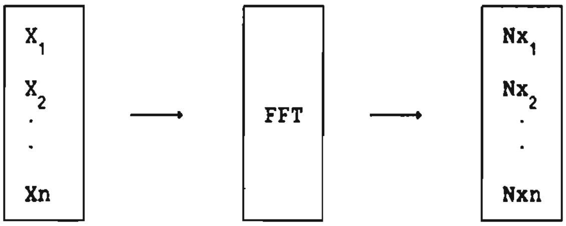
Parameters

| | | |
|---|---|---|
| C | : | Name of complex input and output vector VC. |
| N | : | Complex element count, in power of 2. |
| LF | : | Direction flag. |
| LF = +1 | : | Forward FFT of vector VC. |
| LF = -1 | : | Reverse FFT of vector VC. |

Explanation

To perform an in-place complex forward, or an inverse Fast Fourier Transform (FFT).

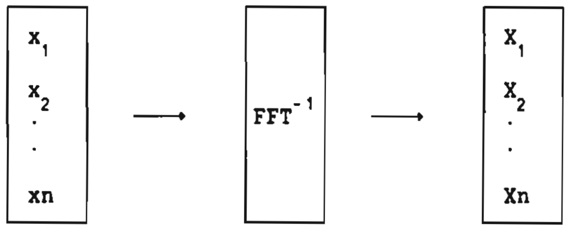
Symbolically the forward FFT can be represented by the block diagram:



The 'X' denotes the complex input coefficients to the vector VC, and 'Nx' denotes the complex output coefficients in vector VC. 'n' is the element count.

NOTE The output coefficients should be multiplied with 1/N for properly scaling.

Symbolically the inverse FFT can be represented by the block diagram:



The 'x' denotes the complex input coefficients to the vector VC, and 'X' denotes the complex output coefficients in vector VC. 'n' is the element count. The output coefficients are properly scaled.

A series of radix 2 passes is used to obtain the coefficients.

A sine table is used to find sine and cosine, instead of calculating them. In this way, the routine is improved with respect to execution time. This is done for element count up to 65536 ($2^{16}$). The sine table covers angles from 0 to $\pi/2$.

Listing

```
      SUBROUTINE CFFTXXX(C,N,LF,*)
      DIMENSION M(20)
      COMPLEX C(N)
      COMPLEX WK,HOLD,Q,CFN
      FN = FLOAT(N)
      F  = FLOAT(LF)
      X  = ALOG2(FN)
      N2 = NINT(X)
C                             MAX ELEMENT COUNT CFFT.
      IF (N2 .GT. 20) STOP
      DO 10 I = 1,N2
10       M(I) = 2**(N2-I)
      FPX  = F * 6.283185308 / FN
      DO 40 L = 1,N2
         NBLOCK = 2**(L-1)
         LBLOCK = N / NBLOCK
         LBHALF = LBLOCK / 2
         K      = 0
         DO 40 IBLOCK = 1,NBLOCK
            FK     = K
            V      = FPX * FK
            COSV   = COS(V)
            SINV   = SIN(V)
            WK     = CMPLX (COSV,SINV)
            ISTART = LBLOCK * (IBLOCK-1)
            DO 20 I = 1,LBHALF
               J     = ISTART + I
               JH    = J + LBHALF
               Q     = C(JH) * WK
               C(JH) = C(J) - Q
               C(J)  = C(J) + Q
20          CONTINUE
            DO 30 I = 2,N2
               II = I
               IF (K .LT. M(I)) GO TO 40
30          K = K - M(I)
40    K = K + M(II)
```

```
C       REORDERING THE TRANSFORM:

        K = 0
        DO 70 J = 1,N
           IF (K .LT. J) GO TO 50
           HOLD   = C(J)
           C(J)   = C(K+1)
           C(K+1) = HOLD
50         DO 60 I = 1,N2
              II = I
              IF (K .LT. M(I)) GO TO 70
60         K = K - M(I)
70      K = K + M(II)
        IF (F .LT. 0.0) RETURN 1
C        INVERSE TRANSFORM

        CFN = CMPLX (FN,0.0)
        DO 80 I = 1,N
80         C(I) = C(I) / CFN
        RETURN 1
        END
```

Performance

This table for CFFT function provides version C or newer of the APF library.

| Number of elements | Improvement ratio to FORTRAN | | Typical execution time pr. loop (millisec.) | |
|---|---|---|---|---|
| | ND-560/1 | ND-570/2 | ND-560/1 | ND-570/2 |
| 32 | 4.35 | 3.66 | 1.26 | 0.83 |
| 64 | 4.25 | 3.57 | 2.80 | 1.84 |
| 128 | 4.15 | 3.55 | 6.10 | 4.00 |
| 256 | 4.00 | 3.50 | 13.55 | 8.77 |
| 512 | 3.95 | 3.38 | 29.25 | 19.25 |
| 1024 | 3.80 | 3.30 | 63.75 | 41.88 |
| 2048 | 3.75 | 2.80 | 138.00 | 91.00 |
| 4096 | 3.65 | 2.73 | 297.50 | 240.00 |
| 8192 | 3.55 | 2.70 | 640.00 | 520.00 |
| 16384 | 3.35 | 2.70 | 1460.00 | 1100.00 |
| 32768 | 3.30 | 2.68 | 3140.00 | 2360.00 |
| 65536 | 3.25 | 2.68 | 6700.00 | 4960.00 |

## References

Among many references on FFT are :

G. D. Bergland: "A Guided Tour of the Fast Fourier Transform"
IEEE Spectrum, July 1969.

E. O. Brigham: "The Fast Fourier Transform"
Prentice-Hall, Englewood Cliffs, 1974.

## *4.40 REAL FAST FOURIER TRANSFORM (RFFTXXX)*

Format

RFFTXXX(C,N,LF,*)

Ident (R:I) : 061:177517B
Ident (R:I) : 062:177517B

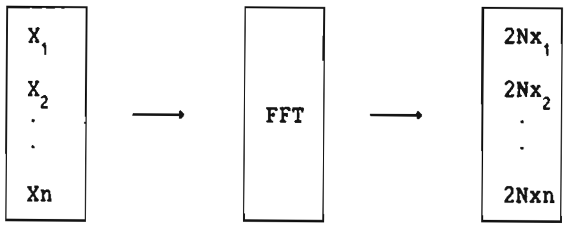
Parameters

| | | |
|---|---|---|
| C | : | Name of complex input and output vector VC. |
| N | : | Complex element count, in power of 2. |
| LF | : | Direction flag. |
| LF = +1 | : | Forward FFT of vector VC. |
| LF = -1 | : | Reverse FFT of vector VC. |

Explanation

To perform an in-place real to complex forward, or complex to real inverse Fast Fourier Transform (FFT).

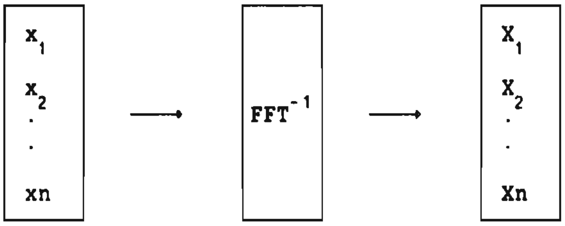
Symbolically the forward FFT can be represented by the block diagram:



The 'X' denotes the real input coefficients to the vector VC, and '2Nx' denotes the complex output coefficients in vector VC. 'n' is the element count.

NOTE The output coefficients should be multiplied with 1/2N for properly scaling.

Symbolically the inverse FFT can be represented by the block diagram:



The 'x' denotes the complex input coefficients to the vector VC, and 'X' denotes the real output coefficients in vector VC. 'n' is the complex element count. The output coefficients are properly scaled.

The RFFTXXX routine utilizes the CFFTXXX routine, refer to section 4.39.

Listing

```
      SUBROUTINE RFFTXXX(C,N,LF,*)
      COMPLEX C(1)
      COMPLEX UC,VC,UC2,VC2,WK,WK2,CFN
      F     = FLOAT(LF)
      NHALF = N / 2
      NFOUR = N / 4
      VK    = F*3.141592654 / NHALF
C     Using the original CFFT routine.
         CFFTXXX(C,NHALF,LF,*910)
910   CONTINUE

      IF (F .LT. 0.0) THEN
         FDIV1 = 1.0
         FDIV2 = 2.0
      ELSE
         FDIV1 = 2.0
         FDIV2 = 4.0
      ENDIF
      URE  =   REAL(C(1))
      VRE  =  AIMAG(C(1))
      CRE  =  (URE+VRE) / FDIV1
      CIM  =  (URE-VRE) / FDIV1
      C(1) =  CMPLX (CRE,CIM)
      URE  =    (REAL(C(NFOUR+1))) / FDIV1
      VRE  = (F*AIMAG(C(NFOUR+1))) / FDIV1
      C(NFOUR+1) = CMPLX (URE,VRE)
      IRX  = NHALF + 2
      DO 200 IR = 2 , NFOUR
         IR2  = IRX - IR
         FIR  = IR  - 1
         URE  = (REAL(C(IR))  +  REAL(C(IR2))) / FDIV2
         UIM  = (AIMAG(C(IR)) - AIMAG(C(IR2))) / FDIV2
         VRE  = (AIMAG(C(IR)) + AIMAG(C(IR2))) / FDIV2
         VIM  = (REAL(C(IR2)) -  REAL(C(IR)))  / FDIV2
         UC   = CMPLX (URE,UIM)
         VC   = CMPLX (VRE,VIM)
         UC2  = CONJG (UC)
         VC2  = CONJG (VC)
         V    = VK * FIR
         COSV = COS(V)
         SINV = SIN(V)
         WK   = CMPLX (COSV,SINV)
         WK2  = CMPLX (-COSV,SINV)
         C(IR)  = UC  + VC  * WK
         C(IR2) = UC2 + VC2 * WK2
200   CONTINUE
      RETURN 1
      END
```

## Performance

This table for RFFT function provides version C or newer of the APF library.

| Number of elements | Improvement ratio to FORTRAN | | Typical execution time pr. loop (millisec.) | |
|---|---|---|---|---|
| | ND-560/1 | ND-570/2 | ND-560/1 | ND-570/2 |
| 32 | 2.20 | 2.10 | 1.40 | 0.82 |
| 64 | 2.20 | 2.10 | 3.00 | 1.77 |
| 128 | 2.25 | 2.10 | 6.30 | 3.76 |
| 256 | 2.30 | 2.10 | 13.20 | 8.05 |
| 512 | 2.30 | 2.10 | 27.80 | 17.00 |
| 1024 | 2.30 | 2.15 | 58.00 | 35.80 |
| 2048 | 2.30 | 2.15 | 121.00 | 74.50 |
| 4096 | 2.30 | 2.15 | 252.50 | 156.25 |
| 8192 | 2.30 | 2.05 | 530.00 | 375.00 |
| 16384 | 2.35 | 2.05 | 1090.00 | 780.00 |
| 32768 | 2.30 | 2.05 | 2360.00 | 1640.00 |
| 65536 | 2.30 | 2.05 | 4940.00 | 3420.00 |

For references on FFT, see page 60.

## *4.41 VECTOR TAPER (VTAPERX)*

Format

VTAPERX(VA,VC,NN,IFLAG,*) Ident (R:I) : 041:177517B

Explanation

Multiply each element of a vector with an increasing or decreasing factor. An own flag selects either the decreasing or the increasing factor. The factor is a function of the element count.

If flag > 0 then:

$VC_1 = VA_1 * (1/nn)$ , $VC_2 = VA_2 * (2/nn)$ .. $VCnn = VAnn * (1)$.

So the general element equation is: VCn = VAn * (n/nn).

If flag $\leq$ 0 then:

$VC_1 = VA_1 * (1 - 1/nn)$, $VC_2 = VA_2 * (1 - 2/nn)$ .. $VCnn = VAnn * (0)$.

So the general element equation is: VCn = VAn * (1 - n/nn).

'nn' denotes the element count and 'n' the element index.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| VC | : Name of output vector VC. |
| NN | : Element count. |
| IFLAG | : Flag. |

Listing

```
      SUBROUTINE VTAPERX(VA,VC,NN,IFLAG,*)
      DIMENSION VA(1),VC(1)
      REAL MULT,MINC
      IF (IFLAG .LE. 0) GO TO 10
      MULT = 1.0/NN
      MINC = MULT
      GO TO 20
 10   CONTINUE
      MULT = (NN-1)*1.0/NN
      MINC = -1.0/NN
 20   CONTINUE
      DO 30 I = 1,NN
         VC(I) = VA(I) * MULT
         MULT  = MULT  + MINC
 30   CONTINUE
      RETURN 1
      END
```

## *4.42 WIENER FILTER (WIENERX)*

Format

WIENERX(LR,R,G,F,A,ISW,WNF,LEC,*)

Ident (R:I) : 047:177517B
Ident (R:I) : 050:177517B

Explanation

To find the so called single solution channel normal equations, by using the Toeplitz recursive algorithm.

Further description of the algorithm is given in Silva & Robinson : Deconvolution of geophysical time series in the exploration for oil and natural gas. 1979.

Parameters

| | |
|---|---|
| LR | : Length of filter. Element count. |
| R | : Auto correlation coefficients: R(1),R(2),......,R(LR). |
| G | : Right-hand side coefficients: G(1),G(2),......,G(LR). |
| F | : Filter coefficients : F(1),F(2),......,F(LR). |
| A | : Prediction error operators : A(1),A(2),......,A(LR). |
| ISW | : Flag. |
| WNF | : White noise factor (not used). |
| LEC | : Loop on error count. Output parameter. |

ISW = +1: General algorithm.
ISW = 0 : Only prediction error operators as results.

Listing

```
      SUBROUTINE WIENERX(LR,R,G,F,A,ISW,WNF,LEC,*)
      DIMENSION R(LR),G(LR),F(LR),A(LR)
      IFLAG = 0
      V     = R(1)
      D     = R(2)
      A(1)  = 1.0
      F(1)  = G(1)/V
      Q     = F(1)*R(2)
      DO 600 L = 2,LR
        A(L)    = -D/V
        AL      = A(L)
        IF (V .LE. 0.0) THEN
           LEC = L
           RETURN 1
        ENDIF
        IF (ISW .EQ. 0) F(L)=V
        V       = V + AL * D
        D       = R(L+1) + AL * R(2)
        L2      = L/2
        IF (L .LE. 3) GO TO 150
```

```
         DO 100 J = 2,L2
           K    = L - J + 1
           HOLD = A(J)
           A(J) = A(J) + AL * A(K)
           D    = D + A(J) * R(K+1)
           A(K) = A(K) + AL * HOLD
100        D    = D + A(K) * R(J+1)
150      IF (2*L2 .EQ. L) GO TO 200
         LH    = L2 + 1
         A(LH) = A(LH) + AL * A(LH)
         D     = D + A(LH) * R(LH+1)
200      IF (ISW .EQ. 0) GO TO 600
         F(L)  = (G(L) - Q)/V
         FL    = F(L)
         L1    = L - 1
         Q     = FL * R(2)
         DO 300 J = 1,L1
           K    = L - J + 1
           F(J) = F(J) + FL * A(K)
300        Q    = Q + F(J) * R(K+1)
600     CONTINUE
        RETURN 1
        END
```

## *4.43 VECTOR GENERATE (VGENXXX)*

Format

VGENXXX(SCALAR,SCINC,VC,INCC,NN,*)          Ident (R:I) : 043:177517B

Explanation

Form a vector as a ramp function with a start value and a slope as input parameters.

$VC_1$ = sc + scinc, $VC_2$ = sc + 2 * scinc .. VCnn = sc + nn * scinc.

So the general element expression is: VCn = sc + n * scinc.

'nn' denotes the element count, 'n' the element index, 'sc' start value, and 'scinc' slope.

Parameters

| | |
|---|---|
| SCALAR | : Scalar for start value. |
| SCINC | : Scalar for increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VGENXXX(SCALAR,SCINC,VC,INCC,NN,*)
      DIMENSION V(1)
      IC = 1
      SC = 0.0
      DO FOR I = 1,NN
         VC(IC) = SCALAR + SC
         IC = IC + INCC
         SC = I  * SCINC
      ENDDO
      RETURN 1
      END
```

## 4.44 VECTOR LINEAR INTERPOLATION (VNMOXXX)

Format

VNMOXXX(VA,VB,VR,LA,LB,*) Ident (R:I) : 044:177517B

Explanation

Perform linear interpolation between samples.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| VB | : Name of input vector VB. |
| VR | : Name of output vector VR. |
| LA | : Element count vector VA. |
| LB | : Element count vector VB. |

Listing

```
       SUBROUTINE VNMOXXX(VA,VB,VR,LA,LB,*)
       DIMENSION VA(1),VB(1),VR(1)
       IFLAG = 0
       LA1   = LA + 1
       DO 20 M = 1,LB
          IF (VB(M) .GT. 0.0) GO TO 21
 20       VR(M) = 0.0
 21    CONTINUE
       IF (M .GT. 1) GO TO 40
       DO 30 M = 1,LB
          IF ((VB(M+1)-VB(M)) .NE. 0.0) GO TO 40
 30       VR(M) = 0.0
 40    CONTINUE
       L = M
       DO 100 M = L,LB
           IF(VB(M).GT.LA1)THEN
              VR(M) = 0.0
              GO TO 100
           ENDIF
           IF (IFLAG .EQ.  1) GO TO 70
           IF (M     .EQ. LB) GO TO 70
           IF (VB(M+1) .LT. VB(M)) THEN
              DO 60 I = 1,M
 60              VR(I) = 0.0
              IFLAG = 1
              GO TO 100
           ENDIF
 70        CONTINUE
           IB = INT(VB(M))
           FB = VB(M) - IB
           IF (IB .LE. 0) THEN
              VR(M) = 0.0
           ELSE
              VR(M) = (VA(IB+1) - VA(IB))*FB + VA(IB)
           ENDIF
 100   CONTINUE
       RETURN 1
       END
```

## 4.45 VECTOR SINC INTERPOLATION (VNMOSXX)

Format

VNMOSXX(VA,VB,VR,LA,LB,*) Ident (R:I) : 045:177517B

Explanation

Perform sinc interpolation between samples.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| VB | : Name of input vector VB. |
| VR | : Name of vector VR. |
| LA | : Element count vector VA. |
| LB | : Element count vector VB. |

Listing

```
      SUBROUTINE VNMOSXX(VA,VB,VR,LA,LB,*)
      DIMENSION VA(1),VB(1),VR(1)
      DIMENSION FILTER(8,7)
      DATA FILTER/
C
     +  -0.00442400 ,  0.02585229 , -0.08848375 ,  0.97214937 ,
     +   0.12341321 , -0.03566697 ,  0.00774525 , -0.00007569 ,
C
     +  -0.00583580 ,  0.0402596   , -0.14005053 ,  0.89166689 ,
     +   0.27481657 , -0.07685405 ,  0.01818759 , -0.00057665 ,
C
     +  -0.00514438 ,  0.04393956 , -0.15734833 ,  0.76733017 ,
     +   0.44274765 , -0.11675274 ,  0.02960985 , -0.00174663 ,
C
     +  -0.00346141 ,  0.03929961 , -0.14670730 ,  0.61239052 ,
     +   0.61239052 , -0.14670724 ,  0.03929964 , -0.00346142 ,
C
     +  -0.001746630,  0.02960985 , -0.11675292 ,  0.44274879 ,
     +   0.76732922 , -0.15734845 ,  0.04393965 , -0.00514441 ,
C
     +  -0.00057664 ,  0.01818759 , -0.07685423 ,  0.27481735 ,
     +   0.89166594 , -0.14005071 ,  0.04025969 , -0.00583582 ,
C
     +  -0.00007568 ,  0.00774526 , -0.03566715 ,  0.12341386 ,
     +   0.97214890 , -0.08848411 ,  0.02585241 , -0.00442402 /

      DATA NFPTS /8/

      IFLAG = 0
      LA1   = LA + 1
      DO 20 M = 1,LB
```

```
         IF (VB(M) .GT. 0.0) GO TO 21
 20      VR(M) = 0.0
 21   CONTINUE
      IF (M .GT. 1) GO TO 40
      DO 30 M = 1,LB
         IF ((VB(M+1)-VB(M)) .NE. 0.0) GO TO 40
 30      VR(M) = 0.0
 40   CONTINUE
      L = M
      DO 100 M = L,LB
          IF(VB(M).GT.LA1)THEN
             VR(M) = 0.0
             GO TO 100
          ENDIF
          IF (IFLAG .EQ.  1) GO TO 70
          IF (M     .EQ. LB) GO TO 70
          IF (VB(M+1) .LT. VB(M)) THEN
             DO 60 I = 1,M
 60             VR(I) = 0.0
             IFLAG = 1
             GO TO 100
          ENDIF
 70       CONTINUE
          IB = INT(VB(M))
          FB = VB(M) - IB
          IF((IB-3) .GT. 0 .AND. (IB+4) .LE. LA) THEN
             NRFILT= .5 + FB/.125
             GO TO (1,2,2,2,2,2,2,2,3),NRFILT+1
 1           VR(M) = VA(IB)
             GO TO 4
 3           VR(M) = VA(IB+1)
             GO TO 4
 2           VR(M) = FDOTPR(VA(IB-3),FILTER(1,NRFILT),NFPTS)
 4           CONTINUE
          ELSEIF (IB .LE. 0) THEN
             VR(M) = 0.0
          ELSEIF (M .EQ. LB) THEN
             VR(M) = VA(IB)
          ELSE
             VR(M) = (VA(IB+1) - VA(IB))*FB + VA(IB)
          ENDIF
 100  CONTINUE
      RETURN 1
      END
      FUNCTION FDOTPR(VA,VB,N)
      DIMENSION VA(1),VB(1)
      FDOTPR = 0.0
      DO FOR I = 1,N
         FDOTPR = FDOTPR + VA(I)*VB(I)
      ENDDO
      RETURN
      END
```

## 4.46 VECTOR EXPAND (VXPNDXX)

<u>Format</u>

VXPNDXX(VA,VC,NN,NC,*) Ident (R:I) : 052:177517B

<u>Explanation</u>

Expand input vector VA into output vector VC.

Vector VA must be organized in this way:

$VA_1$ , $VA_3$ , $VA_5$ , ..... , $VA_{nn-1}$ contain the arguments for the function in increasing order. $VA_2$ , $VA_4$, $VA_6$ , .... , $VA_{nn}$ contain the corresponding function values. The argument in $VA_1$ must be greater or equal to 1.0.

This is also expressed as: $VA_{2i} = f(VA_{2i-1})$ , for $i \leq nn/2$.

'nn' denotes the element count.

After execution, VC contains the new function values approximated for the arguments 1,2,3,4, .... ,$VC_{nn-1}$. The approximation method is based on linear interpolation. Remember that VC must be large enough to contain the number of elements specified by $VC_{nn-1}$.

<u>Parameters</u>

| | |
|---|---|
| VA | : Name of input vector. |
| VC | : Name of result vector. |
| NN | : Element count in input vector. |
| NC | : Element count in result vector. |

<u>Listing</u>

```
      SUBROUTINE VXPNDXX(VA,VC,NN,NC,*)
      DIMENSION VA(1),VC(1)
      IFIRST = 1
      LOOP   = NN - 3
      DO 100 I = 1,LOOP,2
         SLOPE = (VA(I+3)-VA(I+1))/(VA(I+2)-VA(I))
         ILAST = VA(I+2)
         RINC  = SLOPE*(IFIRST-VA(I))
         DO 50 J = IFIRST,ILAST
            VC(J) = VA(I+1)+RINC
            RINC    = RINC+SLOPE
 50      CONTINUE
         IFIRST = ILAST+1
 100  CONTINUE
      NC = VA(NN-1)
      RETURN 1
      END
```

## 4.47 VECTOR FIRST AND LAST NON-ZERO VALUE (VFLNZXX)

Format

```
VFLNZXX(VA,XINDF,XINDL,NN,*)                  Ident (R:I) : 051:177517B
```

Explanation

Find the indices of the first and last non-zero elements in a vector.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| XINDF | : Name for first non-zero index. |
| XINDL | : Name for last non-zero index. |
| NN | : Element count. |

Listing

```
      SUBROUTINE VFLNZXX(VA,XINDF,XINDL,NN,*)
      DIMENSION VA(1)
      DO 100 I = 1,NN
         IF (VA(I) .NE. 0.0) THEN
            XINDF = I
            GO TO 105
         ENDIF
 100  CONTINUE
      XINDF = NN
      XINDL = 1
      GO TO 205
 105  CONTINUE
      DO 200 I = NN,1,-1
         IF (VA(I) .NE. 0.0) THEN
            XINDL = I
            GO TO 205
         ENDIF
 200  CONTINUE
 205  RETURN 1
      END
```

## 4.48 VECTOR SCALAR MULTIPLY AND ADD (VSMADDX)

### Format

VSMADDX(VA,INCA,SC,VB,INCB,VC,INCC,NN,*)     Ident (R:I) : 053:177517B

### Explanation

Add the corresponding elements from two vectors, where the elements of one of the vectors are multiplied with a scalar value.
VCn = VAn * sc + VBn, where 'sc' denotes the scalar, and 'n' denotes the element index.

### Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| INCA | : VA index increment. |
| SC | : Scalar value. |
| VB | : Name of input vector VB. |
| INCB | : VB index increment. |
| VC | : Name of output vector VC. |
| INCC | : VC index increment. |
| NN | : Element count. |

### Listing

```
      SUBROUTINE VSMADDX(VA,INCA,SC,VB,INCB,VC,INCC,NN,*)
      DIMENSION VA(1),VB,(1),VC(1)
      IA = 1
      IB = 1
      IC = 1
      DO FOR I = 1,NN
         VC(IC)= VA(IA) * SC + VB(IB)
         IA    = IA + INCA
         IB    = IB + INCB
         IC    = IC + INCC
      ENDDO
      RETURN 1
      END
```

## *4.49 PREDICT (PREDICT)*

Format

PREDICT(VA,VB,N,L,*) Ident (R:I) : 046:177517B

Explanation

Form the vector VB with $VB_1$ equal to 1.0, the next elements $VB_2$ , $VB_3$ , .... , $VB_L$ equal to 0.0, and the elements $VB_{L+1}$ , $VB_{L+2}$ , .... , $VB_{L+N}$ elements equal to - $VA_1$ , - $VA_2$ , .... , - $VA_N$ .

Note that the elements in VA also are multiplied with -1.

Parameters

| | |
|---|---|
| VA | : Name of input vector VA. |
| VB | : Name of output vector VB. |
| N | : Element count for number of negative elements. |
| L | : Element count for number of zero elements.<br>Element count of VB must be at least : N + L . |

Listing

```
      SUBROUTINE PREDICT(VA,VB,N,L,*)
      DIMENSION VA(1),VB(1)
         DO 100 I = 1,L
 100        VA(I) = (-1.0)*VA(I)
         VB(1) = 1.0
         DO 200 I = 2,N
 200        VB(I) = 0.0
         DO 300 I = 1,L
 300        VB(I+N) = VA(I)
      RETURN 1
      END
```

## *4.50 DISPLAY PROCESSOR FUNCTION (XBTMUX)*

Format

**XBTMUX(IBSWTH,ISCANS,NDOTS,*)** Ident (R:I) : 001:177507B

Explanation

Put on proper raster bits for 32 scan lines for a display processor. A swath is converted to a scan.

Parameters

| | |
|---|---|
| IBSWTH | : Name of input vector IBSWTH. |
| ISCANS | : Name of output vector ISCANS. |
| NDOTS | : Element count. |

Listing

```
       SUBROUTINE XBTMUX (IBSWTH,ISCANS,NDOTS,*)
       INTEGER*4 IBSWTH(1)
       INTEGER*4 ISCANS(1)
       INTEGER*4 NDOTS
       INTEGER*4 IMASK(32)
      DATA IMASK/

     - 20000000000B,10000000000B,04000000000B,02000000000B,
     - 01000000000B,00400000000B,00200000000B,00100000000B,
     - 00040000000B,00020000000B,00010000000B,00004000000B,
     - 00002000000B,00001000000B,00000400000B,00000200000B,
     - 00000100000B,00000040000B,00000020000B,00000010000B,
     - 00000004000B,00000002000B,00000001000B,00000000400B,
     - 00000000200B,00000000100B,00000000040B,00000000020B,
     - 00000000010B,00000000004B,00000000002B,00000000001B/

       ISCAN = 1
       IND = 1
 10    CONTINUE
       MASK = IMASK(ISCAN)
       NBIT = 1
       ISWD = 0
       DO 100 N=1,NDOTS
          IBIT = IAND(IBSWTH(N),MASK)
          IF (IBIT.NE.0) ISWD=IOR(ISWD,IMASK(NBIT))
          NBIT = NBIT + 1
          IF (NBIT.LE.32) GOTO 100
          ISCANS(IND) = ISWD
          IND  = IND + 1
          NBIT = 1
          ISWD = 0
 100   CONTINUE
       ISCAN = ISCAN + 1
       IF (ISCAN.LE.32) GOTO 10
       RETURN 1
       END
```

## 4.51 IMAGE BUILD (IMGBLD)

Format

IMGBLD(IBPOS,IBSET,NN,ISWTH,*) Ident (R:I) : 002:177507B

Explanation

Raster 32 scans input.

Parameters

| | |
|---|---|
| IBPOS | : Name of input/output vector IBPOS. |
| IBSET | : Name of input/output vector IBSET. |
| NN | : Element count. |
| ISWTH | : Name of output vector ISWTH. |

Listing

```
      SUBROUTINE IMGBLD(IBPOS,IBSET,NN,ISWTH,*)
      INTEGER*4 IBPOS(1)
      INTEGER*4 IBSET(1)
      INTEGER*4 NWDS
      INTEGER*4 ISWTH(1)
      INTEGER*4 IBIT(33)
     DATA IBIT/
    - 00000000000B,
    - 20000000000B,30000000000B,34000000000B,36000000000B,
    - 37000000000B,37400000000B,37600000000B,37700000000B,
    - 37740000000B,37760000000B,37770000000B,37774000000B,
    - 37776000000B,37777000000B,37777400000B,37777600000B,
    - 37777700000B,37777740000B,37777760000B,37777770000B,
    - 37777774000B,37777776000B,37777777000B,37777777400B,
    - 37777777600B,37777777700B,37777777740B,37777777760B,
    - 37777777770B,37777777774B,37777777776B,37777777777B/

      DO 100 N=1,NN
         NBITS = IBSET(N)
         IF (NBITS.LE.0) GOTO 100
         NBEG = IBPOS(N)
         IF (NBEG.LT.32) GOTO 50
         IBPOS(N) = NBEG - 32
      GOTO 100
 50   CONTINUE
         NEND = NBEG+NBITS
         NSET = NBITS
         IF (NEND.LE.32) GOTO 60
         NSET = 32 - NBEG
         NEND = 32
 60    CONTINUE
          NP = NBEG + 1
          NS = NEND+1
          IPAT = IEOR(IBIT(NP),IBIT(NS))
          ISWTH(N) = IOR(IPAT,ISWTH(N))
          IBSET(N) = NBITS-NSET
          IBPOS(N) = 0
 100   CONTINUE
       RETURN 1
       END
```

## *4.52 CONVERT AND MOVE (APMOVE)*

Format

APMOVE(ADDR1,ADDR2,IFTM,NN,*) Ident (R:I) : 060:177517B

Explanation

Convert input vector (pointed to by ADDR1) according to the format descriptor IFTM, and move the result to output vector (pointed to by ADDR2).

To get the pointer to a vector, the routine LOCARG must be called. It has the call format:

ADDR = LOCARG(V) ,

which means integer ADDR points at vector V after execution of the call. See also page 5, and the listing in the end of this section.

Format conversion:

IFTM = 0: Integer*4 (32 bits) into single floating point.
IFTM = 1: Integer*2 (16 bits) into single floating point.
IFTM = 2: 32 bits move operation.
IFTM = 3: Single floating point into integer*4 (32 bits).
IFTM = 4: Single floating point into integer*2 (16 bits).
IFTM $\geq$ 5: 32 bits move operation.

Parameters

| | |
|---|---|
| ADDR1 | : Pointer to address of input vector. |
| ADDR2 | : Pointer to address of output vector. |
| IFTM | : Format descriptor. |
| NN | : Element count. |

Listing

```
      ROUTINE  APMOVE
 DSTK: STACK FIXED
 APAR1: W BLOCK 1
 APAR2: W BLOCK 1
 APAR3: W BLOCK 1
 APAR4: W BLOCK 1
 COUNT: W BLOCK 1
      ENDSTACK

 APMOVE: ENTF DSTK
        W1 := IND(B.APAR1)    %.. Get input address.
        W2 := IND(B.APAR2)    %.. Get output address.
        W4 := IND(B.APAR4)    %.. Get element count.
        W3 := IND(B.APAR3)    %.. Get format description.
        IF -Z GO FMT1         %.. If IFTM = 0 : next.
 %       Integer*4 to single float convert and move.
         W SET1 W3
 LOOP0:  W FCONV W1.0,W2.0; W1+4; W2+4; W LOOPI W3,W4,LOOP0
         GO END

 FMT1:   W DECR W3              %.. IFTM = IFTM - 1.
         IF -Z GO FMT2          %.. If IFTM = 0 : next.

 %       Integer*2 to single float convert and move.
         W SET1 W3
 LOOP1:  H FCONV W1.0,W2.0; W1+2; W2+4; W LOOPI W3,W4,LOOP1
         GO END

 FMT2:   W DECR W3              %.. IFTM = IFTM - 1.
         IF -Z GO FMT3          %.. If IFTM = 0 : next.
 %       Single float move .
 FMT5:   W BMOVE W1.0,W2.0,W4
         GO END

 FMT3:   W DECR W3              %.. IFTM = IFTM -1.
         IF -Z GO FMT4          %.. If IFTM = 0 : next.

 %       Single float to integer*4 convert and move.
         W SET1 W3
 LOOP3:  F WCONV W1.0,W2.0; W1+4; W2+4; W LOOPI W3,W4,LOOP3
         GO END

 FMT4:   W DECR W3              %.. IFTM = IFTM - 1.
         IF -Z GO FMT5          %.. If IFTM = 0 : next, else
                                %..    IFTM = 5 is assumed.

 %       Single float to integer*2 convert and move.
         W SET1 W3
 LOOP4:  F HCONV W1.0,W2.0; W1+4; W2+2; W LOOPI W3,W4,LOOP4
         GO END

 END:    R := B.PREVB
         W MOVE 1,R.AUX
         RET
         ENDROUTINE

         ROUTINE  LOCARG
 DSTK:   STACK FIXED
 INARG:  W BLOCK 1
         ENDSTACK

LOCARG:  ENTF DSTACK
         W1 := B.INARG
         RET

         ENDROUTINE
```

## 4.53 DEMULTIPLEX (DMXB)

Format

DMXB(INDAT,INGAIN,OUTPUT,IFIXGN,GNLOC,NBYSCN,NSAMP)

Ident (R:I) : xxx:177505B
Ident (R:I) : xxx:177506B

Explanation

Demultiplex of field tape.

Parameters

| | |
|---|---|
| INDAT | : Name of input vector. |
| INGAIN | : Gain code buffer address. |
| OUTPUT | : Name of output vector. |
| IFIXGN | : Initial / early gain. |
| GNLOC | : Gain location 4 bit index. |
| NBYSCN | : Number of bytes in each scan. |
| NSAMP | : Element count. |

GNLOC is an odd number  : High order 4 bit group byte.
GNLOC is an even number : Low order 4 bit group byte.

Listing

```
         ROUTINE DMXB
 PARMS:   STACK FIXED
 INDAT:   W BLOCK 1
 INGAIN:  W BLOCK 1
 OUTPUT:  W BLOCK 1
 IFIXGN:  W BLOCK 1
 GNLOC:   W BLOCK 1
 NBYSCN:  W BLOCK 1
 NSAMP:   W BLOCK 1

 %        Scaler for any given gain is 2**(GAIN*(-1)).

 %        Note that the least significant bit is not used
 %        and bit 1 corresponds to 0.5 millivolt.

 GNTAB:   W DATA 07760000000B      % Scaler for gain =  0
          W DATA 07740000000B      % Scaler for gain =  1
          W DATA 07720000000B      % Scaler for gain =  2
          W DATA 07700000000B      % Scaler for gain =  3
          W DATA 07660000000B      % Scaler for gain =  4
          W DATA 07640000000B      % Scaler for gain =  5
          W DATA 07620000000B      % Scaler for gain =  6
          W DATA 07600000000B      % Scaler for gain =  7
          W DATA 07560000000B      % Scaler for gain =  8
          W DATA 07540000000B       % Scaler for gain =  9
          W DATA 07520000000B       % Scaler for gain = 10
          W DATA 07500000000B       % Scaler for gain = 11
          W DATA 07460000000B       % Scaler for gain = 12
          W DATA 07440000000B       % Scaler for gain = 13
          W DATA 07420000000B       % Scaler for gain = 14
          W DATA 07400000000B       % Scaler for gain = 15
          W DATA 07360000000B       % Scaler for gain = 16
          W DATA 07340000000B       % Scaler for gain = 17
          W DATA 07320000000B       % Scaler for gain = 18
          W DATA 07300000000B       % Scaler for gain = 19
          W DATA 07260000000B       % Scaler for gain = 20
          W DATA 07240000000B       % Scaler for gain = 21
          W DATA 07220000000B       % Scaler for gain = 22
          W DATA 07200000000B       % Scaler for gain = 23
          W DATA 07160000000B       % Scaler for gain = 24
          W DATA 07140000000B       % Scaler for gain = 25
          W DATA 07120000000B       % Scaler for gain = 26
          W DATA 07100000000B       % Scaler for gain = 27
          W DATA 07060000000B       % Scaler for gain = 28
          W DATA 07040000000B       % Scaler for gain = 29
          W DATA 07020000000B       % Scaler for gain = 30
          W DATA 07000000000B       % Scaler for gain = 31
LIMIT:    W BLOCK 1
NUMSAM:   W BLOCK 1
IGAIN:    W BLOCK 1
MASKI:    W BLOCK 1
          ENDSTACK

%         Gain type 2 - SEG B compatible gain configuration.

DMXB:     ENTF PARMS
SEGB:     W2 := B.INGAIN            % Address of gain values.
          W3 := B.INDAT             % Address of input.
          R := B.OUTPUT             % Address of output.
          W1 := IND(B.NSAMP)        % Element count.
          W1 =: B.NUMSAM            % Save as loop counter.
          W SET1 B.LIMIT            % Set lower limit for loop.
          W1 := IND(B.IFIXGN)       % Fixed gain.
          W1 LADDR B.GNTAB(W1)      % Address of gain table.
          W1 =: B.IGAIN
          W4 := IND(B.GNLOC)        % 4 bit gain index.
          BI1 := BI4                % High or low order 4 bit.
          W4 - 1                    % Adjust for 0 based indexing.
          W SHL W4,-1               % Divide by 2 to get byte index.
          W2 + W4                   % Point to proper gain byte.
          W4 := IND(B.NBYSCN)       % Number of bytes in each scan.
          W1 COMP 1

          IF = GO LOOPH             % Odd : Gain is in high order 4 bits.
```

```
%         Gain is in low order 4 bits of gain byte.

LOOPL:    BY1 := W2.0                 % Gain byte.
          BY1 AND 15                  % With proper gain bits.
          H FCONV W3.0,F2             % Convert I2 to R4.
          W2 + W4 ; W3 + W4           % Next gain byte and input data.
          F MUL3 IND(B.IGAIN)(W1),F2,R.0
          W RLADDR R.4                % Next output position.
          W LOOPD B.NUMSAM,B.LIMIT,LOOPL ; RET

%         Gain is in high order 4 bits of gain byte.

LOOPH:    BY1 := W2.0                 % Gain byte.
          BY SHL BY1,-4               % With proper gain bits.
          H FCONV W3.0,F2             % Convert I2 to R4.
          W2 + W4 ; W3 + W4           % Next gain byte and input data.
          F MUL3 IND(B.IGAIN)(W1),F2,R.0
          W RLADDR R.4                % Next output position.
          W LOOPD B.NUMSAM,B.LIMIT,LOOPH ; RET

          ENDROUTINE
```

## 5 FAST BYTE MOVE

Please refer to the ND-500 Reference Manual, ND-05.009, for a description of the notation etc.

An instruction for fast move of data is implemented. The speed for this instruction depends on the size of the data cache system. The data cache system is either a 128 Kbyte data cache (dc = 4), a 64 Kbyte data cache (dc = 2), a 32 Kbyte data cache (dc = 1), or no data cache at all (dc = 0).

**Format** : BY SSMOV <source/r/BY>,<destination/w/BY>,<m/r/W>

| Assembly notation | Name | Hex code | Octal code |
|---|---|---|---|
| BY SSMOV | byte move | OFE77H | 177167B |

**Operation** :

```
i -> 0
while i < m do
   S(i..i+dc*4) -> D(i..i+dc*4) ; i + dc*4 -> i
enddo
```

**Description** :

<m> bytes are moved from <source> to <destination> operand. The source and destination operands are pointers to the start of the data blocks. Overlap is not taken care of. Constants and registers are illegal as source or destination operands and will cause an illegal operand specifier (IOS) trap condition. The source and destination addresses and number of bytes also have to be a multiplum of dc*4 (either 16, 8 or 4), else an IOS trap condition will occur.

**Trap conditions** : Illegal operand specifier, addressing traps.

**Termination conditions** : Data status bits are reset. K = 0.

**Example**:

Copy a number of bytes from one location to another on a system with 128 Kbyte data cache (dc = 4):

```
BY SSMOV IND(B.24B),R.0,B.30B
```

The move will only be done as long as the following is true :

```
   contents of B.24B         :  ppppppppppx0B
   contents of R register    :  ppppppppppy0B
   contents of B.30B         :  pppppppppz0B
```

where "p" means any value, and the legal values for x, y and z are 0, 2, 4 and 6.

The speed of the instruction depends on the system configuration. The maximum speed of the instruction is obtained on a system containing 128 Kbyte data cache. The speed is approximately four times the speed of the instruction BMOVE. With dc=1, the instruction runs at approximately the same speed as the instruction BMOVE.

Microseconds per byte moved for the instruction BY SSMOV:

| Number of bytes | | ND-560/1 | | | ND-570/2 | |
|---|---|---|---|---|---|---|
| | | Data cache system: | | | Data cache system: | |
| Octal | Decimal | dc = 4 | dc = 2 | dc = 1 | dc = 2 | dc = 1 |
| 100B | 64 | 0.127 | 0.192 | 0.290 | 0.187 | 0.265 |
| 200B | 128 | 0.096 | 0.160 | 0.258 | 0.140 | 0.210 |
| 400B | 256 | 0.082 | 0.145 | 0.243 | 0.113 | 0.187 |
| 1000B | 512 | 0.073 | 0.137 | 0.234 | 0.105 | 0.175 |
| 2000B | 1024 | 0.069 | 0.132 | 0.230 | 0.095 | 0.166 |
| 4000B | 2048 | 0.068 | 0.130 | 0.228 | 0.092 | 0.163 |
| 10000B | 4096 | 0.067 | 0.130 | 0.228 | 0.090 | 0.162 |
| 20000B | 8192 | 0.067 | 0.129 | 0.226 | 0.089 | 0.161 |
| 40000B | 16384 | 0.066 | 0.129 | 0.225 | 0.089 | 0.160 |
| 100000B | 32768 | 0.066 | 0.129 | 0.225 | 0.089 | 0.160 |

## Index