User's Manual

PROGRAM

Relocatable Math Library File

TAPES

Library Binary: 099-000001

093-000041-03

ABSTRACT

This document provides a brief description of all the routines available using Data General's math library tape 099-000001. These descriptions are in alphabetical order according to function. All the information necessary to CALL these routines is provided in this document.

Ordering No. 093-000041

Printed in the United States of America
Rev. 03, May 1973

Original Release - January 1970
First Revision - April 1972
Second Revision - December 1972
Third Revision - May 1973

This revision of the Relocatable Math Library File manual, 093-000041-03, supersedes manual number 093-000041-02. For a description of the changes made in this revision, see the list of changes at the end of the manual.

INTRODUCTION

The following is a brief description of all the relocatable math library subroutines available on tape number 099-000001. These subroutines appear in alphabetical order according to function. Each description will elaborate on the items explained below.

PURPOSE: Explains the function performed by the routine.

TITLE: Gives the name of the routine (necessary for editing the library tape).

ENTRY: Gives the name by which a routine is referenced in an .EXTN statement. This name is identical to the JSR entry point unless specified otherwise.

INPUT: Describes necessary input format.

OUTPUT: Describes results of the routine.

CALLING SEQUENCE AND ENTRY POINTS:
The relocatable routines contained in the math library are called by declaring the appropriate entry point as a normal external within the user program. For example, to call double precision absolute value:

```
                .EXTN          .DABS
                  .
                  .
                  .
                 JSR           @DUMMY
                  .
                  .
                  .
DUMMY:          .DABS
```

The names of the entry point(s) are given for all routines, and unless otherwise noted all user calls must use the above method.

ERROR CONDITIONS:
Explains or cautions about the idiosyncrasies of the routine.

CARRY AND REGISTERS:

Gives states of active registers upon exit.

LENGTH AND TIME:

Gives the number of words used by this routine and an approximation of execution time. Unless otherwise noted, execution times are calculated for the Nova. To obtain approximate execution times for other Nova-family machines, use the following conversions:

.2 * Nova execution time ≈ Supernova or Nova 800 execution times

.35 * Nova execution time ≈ Nova 1200 execution time.

ALGORITHM:

Describes the method used to produce the desired result.

REFERENCE:

Cites literature that may be of use in obtaining further information.

PROGRAM LISTING:

An assembly listing of the routine is given.

In most instances there will be no ENTRY, ALGORITHM, or PROGRAM LISTING entries given. The ALGORITHM and PROGRAM LISTING entries will be included in future editions of this manual. ENTRY information is given only when the entry is different from the JSR entry point.

TABLE OF CONTENTS

PURPOSE:

This routine computes the absolute value of a fixed point, single precision, two's complement number.

TITLE:

The title is .ABS.

INPUT:

The input is a single precision number in AC0.

OUTPUT:

The absolute value of the input is returned in AC0.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .ABS with normal return to the instruction following the call.

ERROR CONDITIONS:

The absolute value of -2**15 cannot be represented and will be returned unchanged.

CARRY AND REGISTERS:

Carry and AC0 may be destroyed; AC1, AC2 and AC3 are unchanged.

LENGTH AND TIME:

This routine consists of 3 words and is normally relocatable.
For $X \geq 0$, execution time is 8.2 μs.
For $X < 0$, execution time is 13.8 μs.

PURPOSE:

This routine computes the absolute value of a double precision, fixed point, two's complement number.

TITLE:

The title is .DABS.

INPUT:

A number in AC0 (high order), AC1 (low order).

OUTPUT:

The absolute value of the input returned in AC0, AC1 - high order in AC0, low order in AC1.

CALLING SEQUENCE AND ENTRY POINT:

Indirect at .DABS, with normal return to the instruction following the call.

ERROR CONDITIONS:

Caution: The absolute value of -2**31 cannot be represented and is returned unchanged.

CARRY AND REGISTERS:

AC0, AC1, and Carry are destroyed; AC2 and AC3 remain unchanged.

LENGTH AND TIME:

This routine consists of 6 instructions and is normally relocatable.
For $X \geq 0$, execution is 8.2. µs
For $X < 0$, execution is 19.4 µs

PURPOSE:

This routine computes the sum of two double precision, two's complement integers.

TITLE:

The title is .DADD.

INPUT:

The first operand must be in AC0, AC1 (high order, low order). The second operand must be in storage, higher order word followed by lower order word. The address of the higher order word of the second operand must be given after the JSR @DUMMY.

OUTPUT:

The double precision sum will be returned in AC0, AC1 (high order, low order).

CALLING SEQUENCE AND ENTRY POINT:

Indirect at .DADD, then address of second operand with return to the instruction following the second operand address.

ERROR CONDITIONS:

Caution: No check is made for overflow.

CARRY AND REGISTERS:

AC0, AC1, AC3 and Carry are destroyed; AC2 remains unchanged.

LENGTH AND TIME:

This routine consists of 15 (octal) instructions and is normally relocatable.
Execution time is 54.9 µs.

ARCTANGENT

(Single Precision)

PURPOSE:

To calculate the fixed point arctangent of the quotient of two input arguments.

TITLE:

ATANX

ENTRY:

.ATANX

INPUT:

Argument dividend in AC0, argument divisor in AC1. Both arguments are expressed in radians in the following format:

sign	integer	fraction
bit 0	bits 1 and 2	bits 3 through 15

The sign bit is set to a 1 only if the argument is negative.

OUTPUT:

The result x, expressed in radians, falls in the range $-\pi \leq x \leq \pi$. The result, in AC2, is given in the same format as that described for input arguments.

CALLING SEQUENCE:

JSR indirect through page zero entry .ATANX . Return is to the next sequential location following the call.

ERROR CONDITIONS:

None; all input arguments will be interpreted in the format illustrated above.

CARRY AND REGISTERS:

AC0 and AC1 are saved; AC3 and Carry are destroyed.

LENGTH AND TIME:

One ZREL and 100 octal NREL locations. Average execution time on the NOVA 1200 is 1.3 ms.

ALGORITHM:

The quotient of the input arguments, x, is found by means of a call to the unsigned integer divide routine, DVD. For the range $0 \leq x \leq 1$, the arctangent of x is calculated to be equal to x * P (x**2). Calls to the unsigned integers multiply routine (.MPYU) and Polynomial expansion function (.POLY) are made. A sixth order polynomial is computed, $P(x^2) = P_0 + P_1 x + P_2 x^2 \ldots + P_6 x^6$, with the following coefficients:

$P_0 =$ .99999
$P_1 =$ - .33326
$P_2 =$ .19881
$P_3 =$ - .13487
$P_4 =$ .83871 * 10^{-1}
$P_5 =$ .37012 * 10^{-1}
$P_6 =$ .78633 * 10^{-2}

For other values of x, one of the following quadrant adjustments is made, where m and n represent the original arguments input in AC0 and AC1 respectively.

$$\text{ARCTAN}\left(\frac{+m}{+n}\right) = \pi/2 - \text{ARCTAN}\left(\frac{n}{m}\right)$$

$$\text{ARCTAN}\left(\frac{-m}{+n}\right) = -\text{ARCTAN}\left(\frac{m}{n}\right)$$

$$\text{ARCTAN}\left(\frac{+m}{-n}\right) = \pi - \text{ARCTAN}\left(\frac{m}{n}\right)$$

$$\text{ARCTAN}\left(\frac{-m}{-n}\right) = -\left(\pi - \text{ARCTAN}\left(\frac{m}{n}\right)\right)$$

REFERENCE:

John F. Hart, "Computer Approximations" New York: John Wiley & Sons, Inc., 1968; pages 128 - 129, INDEX 4990

PROGRAM LISTING:

```
                                ;          FIXED POINT ARCTAN(OF TWO ARGUMENTS)

                                .TITLE     ATANX
                                .ENT       .ATANX
                                .EXTD      .POLY, MPYU
                                .EXTN      DVD

                                .ZREL
00000-000007' .ATANX:           ATANX
                                .NREL

00000'000000  RTURN:  0
00001'000000  SAV0:   0
00002'000000  SAV1:   0
00003'000000  SAV2:   0
00004'000000  SIGN:   0
00005'000000  COMPL:  0
00006'000000  SUPPL:  0

00007'054771  ATANX:  STA     3,RTURN       ;SAVE RETURN ADDRESS
00010'040771          STA     0,SAV0        ;SAVE AC0,1
00011'044771          STA     1,SAV1
00012'101120          MOVZL   0,0           ;GET SIGN
00013'152560          SUBCL   2,2           ;IN AC2
00014'050770          STA     2,SIGN        ;SAVE IT
00015'125120          MOVZL   1,1           ;SAVE SIGN OF AC1
00016'152560          SUBCL   2,2           ;AS SUPPLIMENT FLAG
00017'050767          STA     2,SUPPL
00020'176400          SUB     3,3           ;SET COMPLEMENT FLAG
00021'106432          SUBZ#   0,1,SEC       ;IF AC0>AC1
00022'000404          JMP     .+4           ;AND SWAP ARGUMENTS
00023'111000          MOV     0,2
00024'175400          INC     3,3
00025'121001          MOV     1,0,SKP
00026'131000          MOV     1,2           ;AC0<=AC1
00027'054756          STA     3,COMPL
00030'112415          SUB#    0,2,SNR       ;CANNOT BE EQUAL
00031'151400          INC     2,2           ;AC0/AC2 < 1
00032'126400          SUB     1,1
00033'177777          DVD                   ;GET AC0/AC2
00034'131000          MOV     1,2
00035'006002S         JSR     @ MPYU        ;GET X**2
00036'050745          STA     2,SAV2
00037'111000          MOV     0,2
00040'020430          LDA     0,ATNCF       ;POINT ARCTAN COEFF.S
00041'024426          LDA     1,ATCFCT      ;ARCTAN COEFF COUNT
00042'006001S         JSR     @.POLY        ;EVALUATE POLYNOMIAL
```

PROGRAM LISTING:

```
00043'030740          LDA     2,SAV2
00044'006002S         JSR     @ MPYU          ;GET X*P
00045'014740          DSZ     COMPL           ;COMPLEMENT ANGLE?
00046'000404          JMP     .+4             ;DO NOT COMPLEMENT
00047'024417          LDA     1,PI.2          ;PI/2
00050'106400          SUB     0,1             ;PI.2-X*P
00051'121000          MOV     1,0
00052'101220          MOVZR   0,0
00053'014733          DSZ     SUPPL           ;SUPPLIMENT?
00054'000404          JMP     .+4             ;NO
00055'105000          MOV     0,1
00056'020410          LDA     0,PI.2
00057'122400          SUB     1,0             ;PI-ANGLE
00060'024724          LDA     1,SIGN
00061'125220          MOVZR   1,1             ;GET SIGN
00062'111200          MOVR    0,2             ;BIT IN
00063'020716          LDA     0,SAV0          ;RESTORE AC0,1
00064'024716          LDA     1,SAV1
00065'002713          JMP     @RTURN

00066'144417  PI.2:   144417  ;PI/2 1.44417 OCTAL
00067'000006  ATCFCT: 6
00070'000071' ATNCF:  .+1
00071'000402          402     ;.78633 7627 -2
00072'175502          -2276   ;-.37012 99998 -1
00073'005275          5275    ;.83871 18962 -1
00074'167274          -10504  ;-.13487 19133
00075'014563          14563   ;.19881 48243 4
00076'152527          -25251  ;-.33326 51491 7
00077'077777          77777   ;.99999 93478 2
```

```
                      .END    ;END OF ARC TAN
```

BCD to BINARY
(Single Precision)

PURPOSE:

This routine converts a single precision number in BCD to binary.

TITLE:

The title is .BCDB.

INPUT:

A BCD integer in AC1 (maximum value 9999 decimal).

OUTPUT:

Binary equivalent of BCD integer is returned in AC1.

CALLING SEQUENCE AND ENTRY POINT:

Indirect at .BCDB with normal return to the instruction following the call.

ERROR CONDITIONS:

If a digit greater than binary 1001 is encountered in the input, Carry will be set, AC1 will be unchanged, and AC0 will contain the bad digit. Otherwise, Carry will be zero on return.

CARRY AND REGISTERS:

AC0, AC1, AC3, and Carry are destroyed; AC2 is unchanged.

LENGTH AND TIME:

This routine consists of 53 (octal) words and is normally relocatable.
Execution time is 1.034 m.s.

PURPOSE:

This routine converts a double precision number in BCD to binary.

TITLE:

The title is .DBCB.

INPUT:

A double precision integer is passed in AC0, AC1 (high order, low order) of maximum value of 99999999 decimal.

OUTPUT:

The binary equivalent of the input is returned in AC0, AC1 (high order, low order).

CALLING SEQUENCE AND ENTRY POINT:

Indirect at .DBCB with normal return to the instruction following the call.

ERROR CONDITIONS:

If a digit greater than 9 is encountered in the input, Carry will be set and AC0 will contain the bad digit. Otherwise, Carry will be zero.

CARRY AND REGISTERS:

AC0, AC1, AC3, and Carry are destroyed; AC2 is unchanged.

LENGTH AND TIME:

This routine consists of 76 (octal) words and is normally relocatable.
Execution time is 2.174 ms.

BINARY TO BCD
(Single Precision)

PURPOSE:

This routine converts a binary number to its BCD equivalent.

TITLE:

The title is .BBCD.

INPUT:

An unsigned binary number in AC1.

OUTPUT:

The BCD equivalent in AC1.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .BBCD with normal return to the instruction following the call.

ERROR CONDITIONS:

If a number greater than 9999 is input for conversion, no conversion will take place and Carry will be set. Otherwise, Carry will be zero.

CARRY AND REGISTERS:

AC1, AC3 and Carry are destroyed; AC0, and AC2 are unchanged.

LENGTH AND TIME:

This routine is 41 (octal) words and is normally relocatable. Execution time is 273.8 + N * 14.1 μs where N is the sum of the digits of the result.

PURPOSE:

This routine converts a double precision binary number to a BCD number.

TITLE:

The title is .DBBC.

INPUT:

A positive, double precision binary number in AC0, AC1 (high order, low order).

OUTPUT:

The BCD equivalent is in AC0, AC1 (high order, low order).

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .DBBC with normal return to the instruction following the call.

ERROR CONDITIONS:

If AC0, AC1 contains a number greater than 99999999, no conversion will take place and Carry will be set. Otherwise, Carry will be reset.

CARRY AND REGISTERS:

AC0, AC1, AC3, and Carry are destroyed; AC2 is unchanged.

LENGTH AND TIME:

This routine consists of 57 (octal) words and is normally relocatable.

PURPOSE:

This routine converts a single precision two's complement number to an ASCII character string.

TITLE:

The title is .BIND.

INPUT:

A single precision, two's complement integer is passed in AC1.

OUTPUT:

An ASCII character string terminated by a null word. Characters are passed right adjusted in AC0 to the routine whose address must be in ZREL location .PTCH. The string is of the form:

+DDDDD(NULL)
or
-DDDDD(NULL)

CALLING SEQUENCE AND ENTRY POINT:

Indirect at .BIND with normal return to the instruction following the call.

CARRY AND REGISTERS:

AC1, AC3 and Carry are destroyed; AC 0 and AC2 remain unchanged.

LENGTH AND TIME:

This routine consists of 51 (octal) words and is normally relocatable.
Execution time is (378.3 + N * 14.1) μs where N is the sum of the digits of the result.

PURPOSE:

This routine converts a double precision two's complement number to an ASCII decimal character string.

TITLE:

The title is .DBD.

INPUT:

A double precision, two's complement integer is passed in AC1, AC2 (high order, low order).

OUTPUT:

ASCII character string of the form:

+DDDDDDDDDD(NULL)
or
-DDDDDDDDDD(NULL)

is outputted. Characters are passed right adjusted, bit 8 = 0, in AC0 to a user routine whose address must be stored in ZREL location .PTCH.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .DBD with normal return to the instruction following the call.

CARRY AND REGISTERS:

AC1, AC2, AC3, and Carry are destroyed; AC0 remains unchanged.

LENGTH AND TIME:

This routine consists of 112 (octal) words and is normally relocatable.

Execution time is 1.061 + N * .047 ms where N is the sum of the digits of the result.

PURPOSE:

This routine computes the Gray Code equivalent of a 16-bit binary word.

TITLE:

The title is .BGRY.

INPUT:

A binary word in AC0.

OUTPUT:

Gray Code equivalent in AC0.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .BGRY with normal return to the instruction following the call.

CARRY AND REGISTERS:

AC0, AC3, and Carry are destroyed; AC1, AC2 are unchanged.

LENGTH AND TIME:

This routine consists of 13 (octal) words and is normally relocatable.
Execution time is 50.3 μs.

PURPOSE:

This routine converts a 16-bit binary word to an octal ASCII character string.

TITLE:

The title is .BINO.

INPUT:

A 16-bit binary number is passed in AC1.

OUTPUT:

An ASCII character string terminated by a null character. Characters are passed right adjusted in AC0 to the user routine whose address must be stored in ZREL location .PTCH. The string is of the form:

OOOOOO(NULL)

where "O" represents octal digits.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .BINO with normal return to the instruction following the call.

CARRY AND REGISTERS:

AC0, AC1, AC3, and Carry are destroyed; AC2 remains unchanged.

LENGTH AND TIME:

This routine consists of 27 (octal) words and is normally relocatable.

Execution time is 367.6 + N * 20.0 μs, where N is the sum of the digits of the result (the sum expressed in decimal).

DECIMAL TO BINARY
(Single Precision)

PURPOSE:

This routine converts ASCII characters to a single precision binary number.

TITLE:

The title is .DBIN.

INPUT:

Input characters will be requested by calling a user "get a character" routine whose address must be stored in ZREL location .GTCH. This user routine must be provided. ASCII characters should be returned, right adjusted in AC0 with bit 8 = 0. This routine need not save any registers or Carry. Input should be in the form:

S D D . . . D D (break)

where "S" represents the sign ("-" or optionally "+"), D represents an ASCII decimal digit, and "break" is any ASCII character other than a digit.

OUTPUT:

Upon exit, AC0 will contain the ASCII break character and AC1 will contain the single precision, two's complement binary equivalent of the input.

CALLING SEQUENCE AND ENTRY POINTS:

Indirect to .DBIN with normal return to the instruction following the call.

If it is desired to output a signal character, the calling sequence is indirect to .DBNI. An ASCII "S" followed by a null character will be transmitted via AC0 to a "put a character" routine whose address must be in ZREL location .PTCH.

ERROR CONDITIONS:

Caution: The absolute value of the result is N MOD 2**15.
For example: +96741 converts to +31205.
-2**15 converts to 0.

CARRY AND REGISTERS:

AC0, AC1, and Carry are destroyed; AC2 and AC3 are unchanged.

LENGTH AND TIME:

This routine consists of 65 (octal) words and is normally relocatable. Execution time is approximately 110 + I * 82.2 μs where I is the number of digits in the input.

PURPOSE:

This routine converts ASCII characters to a double precision binary number.

TITLE:

The title is .DDB .

INPUT:

Input characters will be requested by calling a user "get a character" routine whose address must be stored in ZREL location .GTCH. This user routine must be provided. ASCII characters should be returned, right adjusted in AC0 with bit 8 = 0. This routine need not save any registers or Carry. Input should be in the form:

S D D . . . D D (break)

where "S" represents the sign ("-" or optionally "+"), D represents an ASCII decimal digit, and "break" is any ASCII character other than a digit.

OUTPUT:

Upon exit, AC0 will contain the ASCII break character. AC1 and AC2 contain the double precision two's complement equivalent of the input.

CALLING SEQUENCE AND ENTRY POINTS:

Indirect to .DDB with normal return to the instruction following the call.

To output a signal character, the call is indirect to .DDBI . This will cause an ASCII "D" followed by a null to be sent via AC0 to a "put a character" routine whose address must be stored in ZREL location .PTCH . This routine must accept ASCII characters in the same format as .GTCH .

ERROR CONDITIONS:

Caution: The absolute value of the result is N MOD 2**31. (see .DBIN).

CARRY AND REGISTERS:

All accumulators and Carry are destroyed.

LENGTH AND TIME:

This routine consists of 77 (octal) words and is normally relocatable. Execution time is approximately $69.90 + I * 43.35$ μs on the Nova 1200, where I is the number of digits input.

Signed DIVIDE
(Single Precision)

PURPOSE:

Divides two fixed point , two's complement numbers.

TITLE:

The title is .DIV.

INPUT:

Dividend in AC0 (high order) and AC1 (low order).
Divisor in AC2.

OUTPUT:

Quotient in AC1. Remainder in AC0 (same sign as dividend).

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .DIV with normal return to the instruction following the call.

ERROR CONDITIONS:

If the magnitude of the quotient exceeds 2**15-1, Carry is set and the dividend remains unchanged. Otherwise, Carry will be zero.

CARRY AND REGISTERS:

AC0, AC1, AC3, and Carry are destroyed; AC2 remains unchanged.

LENGTH AND TIME:

This routine consists of 45 (octal) words and is normally relocatable.
Total average execution time is 605 μs.

PURPOSE:

This routine calculates the quotient of two signed double precision numbers.

TITLE:

The title is DDIV.

INPUT:

The double precision divisor must be in AC0, AC1 (high order, low order). The quadruple precision dividend should be stored in four consecutive words, highest order to lowest order. AC2 must contain the address of the highest order word of the dividend.

OUTPUT:

The double precision quotient is returned in AC0, AC1 (high order, low order). Its sign is determined by the algebraic rules for signed division. The double precision remainder is stored in two consecutive memory words with the high order word first. AC2 will contain the address of the higher order remainder word. The sign of the remainder is the same as the sign of the dividend.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .DDIV with normal return to the instruction following the call.

ERROR CONDITIONS:

If the magnitude of the quotient would exceed 2**31-1 or |dividend| >|divisor|, an error condition exists, Carry is set, and return is made with unpredictable results.

CARRY AND REGISTERS:

All accumulators and Carry are destroyed.

LENGTH AND TIME:

This routine occupies 140 (octal) locations. Execution time is approximately 2.98 milliseconds.

REFERENCE:

"How to Use the Nova Computers", section 2.2.

Unsigned DIVIDE
(Single Precision)

PURPOSE:

To calculate the quotient of two unsigned integers.

TITLE:

DVD

ENTRIES:

DVD, .DIVI, .DIVU

INPUT:

If DVD or .DIVU entry, the dividend is input in AC0 (high order portion) and AC1 (low order portion). If .DIVI entry, the dividend is input in AC1 alone. In either case, the divisor is input in AC2.

OUTPUT:

The remainder is output in AC0, the quotient in AC1.

CALLING SEQUENCES:

The calling sequences consist of indirect calls through page zero entries .DIVI or .DIVU with return to the next sequential location following the call. DVD is equivalent to JSR @ .DIVU.

ERROR CONDITIONS:

Any inputs which would yield a quotient larger than $2^{16}-1$ (i.e., AC0 $\geq$ AC2) causes Carry to be set and return to be made to the caller. Otherwise Carry is reset.

CARRY AND REGISTERS:

AC2 is unchanged; AC0, AC1 and AC3 are destroyed under normal operation. AC1 is also left unchanged under error conditions, and Carry is always set or reset as discussed above.

LENGTH AND TIME:

Two ZREL locations and 21 octal NREL locations are required. Average execution times for .DIVI are 103 μs on the Supernova and 545 μs on the Nova. Average execution times for DVD are 102 μs on the Supernova and 539 μs on the Nova.

ALGORITHM:

. DIVI creates an all-zero high order dividend upon entry and then enters the DVD common logic.

DVD initially compares the divisor with the high order portion of the dividend. If that is greater than or equal to the divisor, the result would exceed $2^{16}-1$ and could not be represented in 16 bits using conventional two's complement notation. In this case Carry is set and return is made.

Otherwise, the less significant (LS) half of the dividend is shifted left, and 16_{10} iterations of the following logic are performed. First the more significant (MS) half of the dividend (AC0) is shifted to the left and is compared to the divisor (AC2). If the divisor is equal to or less than the MS dividend, the divisor is subtracted from the MS dividend and the LS dividend (AC1) is shifted left. Otherwise, no subtraction is performed and only the LS dividend left shift is performed. In both cases, Carry contains the latest quotient bit and is shifted in behind the MS portion of the dividend when the iteration is repeated. Upon completion, the 16-bit quotient is entirely assembled in AC1, and the final adjusted dividend in AC0 is the remainder. The routine yields an exact answer.

REFERENCE:

"How to Use the Nova Computers", section 2.2 .

```
                        .TITLE  DVD
                        .ENT    DVD,.DIVI,.DIVU

                        .ZREL
00000-000001'.DIVU:     DIVU
00001-000000'.DIVI:     DIVI

                        .NREL
     006000-            DVD = JSR          @.DIVU


00000'102400 DIVI:      SUB 0,0            ; INTEGER DIVIDE, CLEAR AC0
00001'054416 DIVU:      STA 3,SAV3         ; SAVE AC3
00002'142432            SUBZ# 2,0,SZC      ; TEST FOR OVERFLOW
00003'000412            JMP DIV1           ; SET CARRY AND RETURN
00004'034414            LDA 3,M20          ; 16 ITERATIONS
00005'125120            MOVZL 1,1          ; SHIFT LOW DIVIDEND
00006'101100 DVD0:      MOVL 0,0           ; SHIFT HIGH DIVIDEND
00007'142412            SUB# 2,0,SZC       ; DOES DIVISOR GO IN?
00010'142400            SUB 2,0            ; YES
00011'125100            MOVL 1,1           ; SHIFT LOW DIVIDEND
00012'175404            INC 3,3,SZR        ; CHECK COUNT
00013'000773            JMP DVD0           ; NOT DONE
00014'176441            SUBO 3,3,SKP       ; DONE , CLEAR CARRY
00015'176420 DIV1:      SUBZ 3,3           ; SET CARRY
00016'002401            JMP @SAV3          ; RETURN


00017'000000 SAV3:      0                  ; SAVE AC3

00020'177760 M20:       -20                ; - 16 DECIMAL

                        .END     ;END OF INTEGER DIVIDE
```

PURPOSE:

This routine computes the binary equivalent of a 16-bit Gray Code word.

TITLE:

The title is .GRYB.

INPUT:

Input is a Gray Code word in AC0.

OUTPUT:

The binary equivalent is returned in AC0.

CALLING SEQUENCE AND ENTRY POINTS:

Indirect to .GRYB with normal return to the instruction following the call.

CARRY AND REGISTERS:

AC0, AC3 and Carry are destroyed; AC1, AC2 are unchanged.

LENGTH AND TIME:

This routine consists of 22 (octal) words and is normally relocatable.
Execution time is 536.4 μs.

PURPOSE:

This routine computes the exclusive OR of two unsigned numbers.

TITLE:

The title is .XOR.

INPUT:

One 16-bit quantity is passed in AC0, the second in AC1.

OUTPUT:

The exclusive OR of the two quantities is returned in AC0.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .XOR with normal return to the instruction following the call.

CARRY AND REGISTERS:

AC0, AC3, and Carry are destroyed; AC1 and AC2 are unchanged.

LENGTH AND TIME:

This routine is 7 words and is normally relocatable.
Execution time is 34.0 μs.

PURPOSE:

This routine computes the logical inclusive OR of two unsigned numbers.

TITLE:

The title is .OR .

INPUT:

One 16-bit quantity is passed in AC0, the second in AC1.

OUTPUT:

The inclusive OR of the two quantities is returned in AC0.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .OR with normal return to the instruction following the call.

CARRY AND REGISTERS:

AC0 is destroyed; AC1, AC2, AC3, and Carry are unchanged.

LENGTH AND TIME:

This routine consists of 5 words and is normally relocatable. Execution time is 25.6 µs.

(Single Precision)

PURPOSE:

This routine multiplies two, fixed point, single precision, two's complement numbers.

TITLE:

The title is .MPY.

INPUT:

One fixed point, single precision operand is passed in AC1, the second in AC2.

OUTPUT:

The double precision result is returned in AC0 (high order) and AC1 (low order).

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .MPY with normal return to the instruction following the call.

CARRY AND REGISTERS:

AC0, AC1, AC3, and Carry are destroyed, AC2 remains unchanged.

LENGTH AND TIME:

This routine consists of 16 (octal) words and is normally relocatable.
For execution time in addition to unsigned multiply, 56.4 μs.
For execution time in unsigned multiply, 340 μs.
Total average execution time is 396.4 μs.

PURPOSE:

To calculate the product of two unsigned integers or the sum of that product and a third unsigned integer (MPY).

TITLE:

MPY

ENTRIES:

MPY, .MPYU, .MPYA

INPUT:

The multiplier and multiplicand are input in AC1 and AC2 (MPY). If a third integer is to be added to the product, the integer is input in AC0.

OUTPUT:

The result is output with the high order portion in AC0 and the low order portion in AC1.

CALLING SEQUENCES:

The calling sequences consist of indirect calls through page zero entries .MPYU and .MPYA, with return to the next sequential location following the call. MPY is equivalent to JSR @ .MPYA. .MPYU is used if no third integer is to be added to the product.

ERROR CONDITIONS:

None.

CARRY AND REGISTERS:

AC2 and Carry are restored; AC0, AC1, and AC3 are destroyed.

LENGTH AND TIME:

Two ZREL locations and 14 octal NREL locations are required. Average execution times for .MPYU are 87 µs on the Supernova and 441 µs on the Nova. Average execution times for MPY are 86 µs on the Supernova and 435 µs on the Nova.

ALGORITHM:

.MPYU creates an all-zero addendum upon entry, and then enters the MPY common logic. The multiply-and-add function is performed by iteratively examining the least significant bit (LSB) of the multiplicand and then shifting the multiplicand to the right. If the LSB was a one, then the multiplier is added to an accumulating partial product

ALGORITHM (cont'd)

and that sum is shifted to the right. If the LSB was a zero, the accumulating partial product is simply shifted rightwards. Initially the partial product is zero in .MPYU, and is equal to the addendum in MPY. The process is carried out for 16_{10} iterations and yields the exact answer.

REFERENCE:

"How to Use the Nova Computers", section 2.2 .

```
                              .TITLE   MPY
                              .ENT     MPY,.MPYU,.MPYA

                              .ZREL

00000-000001'.MPYA:           MPYA
00001-000000'.MPYU:           MPYU

                              .NREL

      006000-                 MPY = JSR            @.MPYA

00000'102460 MPYU:            SUBC 0,0             ; CLEAR AC0, DON'T DISTURB
                                                   ; CARRY
00001'054411 MPYA:            STA 3,SAV3           ; SAVE AC3
00002'034411                  LDA 3,M20            ; 16 TIMES THRU LOOP
00003'125203 MPY1:            MOVR 1,1,SNC         ; CHECK NEXT MULTIPLIER BIT
00004'101201                  MOVR 0,0,SKP         ; 0, JUST SHIFT
00005'143220                  ADDZR 2,0            ; 1, ADD MULTIPLICAND AND SHIFT
00006'175404                  INC 3,3,SZR          ; CHECK FOR 16TH TIME THRU
00007'000774                  JMP MPY1             ; NO, CONTINUE
00010'125260                  MOVCR 1,1            ; YES, SHIFT LAST LOW BIT
                                                   ; (NOTE IT WAS COMPLEMENTED BY
                                                   ; FINAL INC)
00011'002401                  JMP @SAV3            ; RETURN

00012'000000 SAV3:            0                    ; RETURN ADDRESS
00013'177760 M20:             -20                  ; -16 DECIMAL

                              .END     ;END OF UNSIGNED MULTIPLY
```

Signed MULTIPLY (Double Precision)

PURPOSE:

This routine calculates the product of two signed double precision numbers.

TITLE:

The title is DMPY.

INPUT:

The double precision multiplier must be in AC0, ACl (high order, low order). The double precision multiplicand should be stored in two consecutive words, higher order first. The location following the calling instruction (JSR) should contain the address of the high order word of the multiplicand.

OUTPUT:

The quadruple precision product is stored in four consecutive locations within the DMPY subroutine, highest order word first. AC2 contains the address of the highest order word. The sign of the product is determined by the algebraic rules for signed multiplication.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .DMPY with return to the second location after the call since the location after the call contains the address of the multiplicand.

CARRY AND REGISTERS:

All accumulators and Carry are destroyed.

LENGTH AND TIME:

This routine occupies 103 (octal) locations. Execution time is approximately 1.62 milliseconds.

REFERENCE:

"How to use the Nova Computers", section 2.2.

PURPOSE:

This routine computes -D where D is a double precision two's complement integer.

TITLE:

The title is .DNEG.

INPUT:

A double precision, two's complement number in AC0, AC1 (high order, low order).

OUTPUT:

The negative of the input is returned in AC0 (high order), AC1 (low order).

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .DNEG with normal return to the instruction following the call.

ERROR CONDITIONS:

Caution: The negative of -2**31 cannot be represented and is returned unchanged.

CARRY AND REGISTERS:

AC0, AC1, AC3, and Carry are destroyed; AC2 remains unchanged.

LENGTH AND TIME:

This routine consists of 4 words and is normally relocatable. Execution time is 13.8 μs.

OCTAL to BINARY
(Single Precision)

PURPOSE:

This routine converts an ASCII octal character string to a binary number.

TITLE:

The title is .OBIN .

INPUT:

Input characters will be requested by calling a user-written "get a character" routine whose address must be stored in ZREL location .GTCH . This user routine must be provided. Upon return from the call, this routine should return an ASCII character, right adjusted in AC0 with bit 8 = 0. Input should be of the form:

OO...OO(break)

where "O" represents octal digits.

OUTPUT:

AC0 contains the break character, and AC1 contains the binary number (MOD 200000 octal).

CALLING SEQUENCE AND ENTRY POINTS:

Indirect to .OBIN .

It is desired to output a signal character, the calling sequence is indirect to .OBNI .
An ASCII "0" followed by a null character will be transmitted via AC0 to a user-written "put character" routine whose address must be stored in ZREL location .PTCH . In both cases, return is to the first word after the call.

ERROR CONDITIONS:

Caution: Result is N MOD 200000 (octal) e.g., 576452* Converts to 176452.

CARRY AND REGISTERS:

AC0, AC1, AC3, and Carry are destroyed; AC2 is unchanged.

LENGTH AND TIME:

This routine consists of 42 (octal) words and is normally relocatable.
Execution time for .OBIN is 63.0 + I * 70.2 μs
where I represents the number of digits in the input.

PURPOSE:

This routine computes the even parity bit over a 16-bit number and returns the bit in Carry.

TITLE:

The title is .PRTY.

INPUT:

A 16-bit number is passed in AC0.

OUTPUT:

The even parity bit over the contents of AC0 will be returned in Carry.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .PRTY with return to the word following the call.

CARRY AND REGISTERS:

AC3 and Carry are destroyed; AC0, AC1, AC2 remain unchanged.

LENGTH AND TIME:

This routine consists of 16 (octal) words and is normally relocatable.
Average execution time is 215.4 μs.

POLYNOMIAL EXPANSION
(Single Precision)

PURPOSE:

To calculate an integer Polynomial expansion series of the form $P(x) = P_0 + P_1x^1 + P_2x^2 + \dots + P_nx^n$

TITLE:

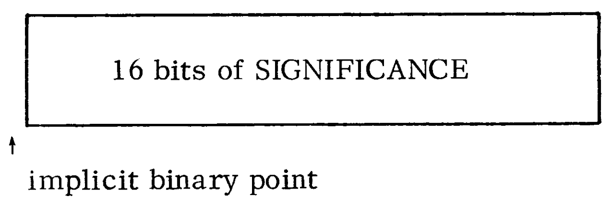
POLYN

ENTRY:

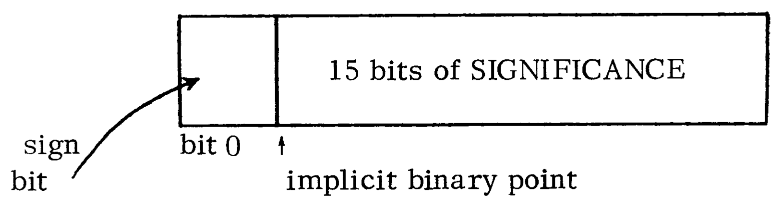
.POLY

INPUT:

The order of the polynomial is input as an integer in AC1. The argument x is a positive number input in AC1 with its binary point normally to the left of bit 0, the most significant bit. (There is no sign bit reserved since all inputs must be positive.) Any shifting of the binary point is understood to shift the implicit point of the result in like fashion.



A pointer to the highest order coefficient is in AC0 where all coefficients are two's complement numbers, usually in the following format:



Any shifting of the binary point is understood to shift the implicit binary point of the result in like fashion. The coefficients are placed in a list with the following structure:

P_n
P_{n-1}
⋮
P_1
P_0

↓ increasing addresses

OUTPUT:

The result, output in AC1, is in the same format as the coefficients, with shifting of the implicit binary point as required.

CALLING SEQUENCE:

JSR indirect through page zero entry
.POLY

ERROR CONDITION:

None.

CARRY AND REGISTERS:

AC2 is saved; AC0, AC1, AC3 and Carry are destroyed.

LENGTH AND TIME:

One ZREL location and 22 octal NREL locations. Execution time depends directly upon the order of the polynomial expansion.

ALGORITHM:

A cumulative partial product is formed by successively adding the next highest order coefficient not processed to a partial product formed by x and the highest order coefficient not processed, and then by multiplying this partial sum by x. Only the most significant word of this product is retained. The process is performed iteratively, until all coefficients through P_1 have been processed. P_0 is added to the final result.

In equation form,

$$P(x) + ((((P_n x) + P_{n-1})x)\dots) + P_0$$

PROGRAM LISTING:

```
---
                                 ;FIXED POINT POLYNOMIAL

                                 ;INPUT:  IN AC2
                                 ;        AS A ONE WORD INTEGER
                                 ;        COEFFICIENT POINTER IN AC0
                                 ;        COEFFICIENTS IN DESCENDING POWER
                                 ;        IN TWO'S COMPLEMENT FORM
                                 ;        ORDER OF POLYNOMIAL IN AC1

                                 ;OUTPUT:          IN TWO'S COMPLEMENT
                                 ;        FORM IN AC1


                                 .TITLE  POLYN
                                 .ENT    .POLY
                                 .EXTD    MPYU

                                 .ZREL
00000-000000'  .POLY:            POLYN
                                 .NREL

00000'054417   POLYN:   STA      3,PLRTN          ;SAVE RETURN
00001'040417            STA      0,COEFF          ;SAVE COEFF. POINTER
00002'044417            STA      1,COUNT          ; AND COUNT
00003'026415            LDA      1,@COEFF         ;HIGHEST COEFF
00004'121122   LOOP:    MOVZL    1,0,SZC          ;NEGATIVE?
00005'124440            NEGO     1,1              ;YES,MAKE IT +VE,SET CARRY
00006'006001$           JSR      @ MPYU           ;AC0,1 = AC1*AC2
00007'105002            MOV      0,1,SZC          ;NEGATIVE SIGN?
00010'124400            NEG      1,1              ;YES,NEGATE RESULT
00011'010407            ISZ      COEFF            ;RAISE COEFF. POINTER
00012'022406            LDA      0,@COEFF         ;ADD NEXT COEFF
00013'107000            ADD      0,1              ;TO RUNNING SUM
00014'014405            DSZ      COUNT            ;DONE?
00015'000767            JMP      LOOP             ;GO TO NEXT COEFF
00016'002401            JMP      @PLRTN           ;RETURN


00017'000000   PLRTN:   0
00020'000000   COEFF:   0
00021'000000   COUNT:   0

                        .END     ;END OF POLYNOMIAL
```

PURPOSE:

This routine generates a (pseudo) random sequence of integers in the range 0 to 2**16-1.

TITLE:

The title is .RAND.

INPUT:

The address of the previous random value(or initially a starting value) must be provided in the word after the call to .RAND.

OUTPUT:

The new 16-bit random result will be returned in AC0 and will also replace the previous value in memory.

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .RAND followed by the address of the old value. Return will be to the instruction after the address parameter.

ERROR CONDITIONS:

Caution: If a K-bit number ($1 \leqslant K \leqslant 16$) is needed, use the most significant K bits (the least significant K bits are not as random). For example, to obtain random N MOD 2, use the sign bit of the result.

CARRY AND REGISTERS:

AC0, AC3, and Carry are destroyed; AC1 and AC2 are unchanged.

LENGTH AND TIME:

This routine consists of 36 (octal) words and is normally relocatable.
Execution time is 244.7 µs.

PURPOSE:

To calculate the sine or cosine of an angle expressed in radians.

TITLE:

SINX

ENTRIES:

.SINX, .COSX

INPUT:

The argument m is input in AC0, within the range $-4 < n < 4$, in the following format:

sign	integer	fraction
bit 0	bits 1 and 2	bits 3 through 15

The sign bit is set to a 1 only if the argument is negative.

OUTPUT:

The result is output in AC1, in the same format as the input.

CALLING SEQUENCES:

JSR indirect through either page zero entry .SINX or .COSX as appropriate.
Return is to the next sequential location following the call.

ERROR CONDITIONS:

None.

CARRY AND REGISTERS:

AC0 is saved; AC2, AC3, and Carry are destroyed.

LENGTH AND TIME:

Two ZREL and 65 octal NREL locations.
Average execution time on the NOVA 1200 is .9 ms.

ALGORITHM:

Upon entry to .COSX, m is subtracted from $\pi/2$, and the sine logic is performed on this difference. If the original argument input to .SINX is negative, the result will be equal to -SIN(m); negative arguments input to .COSX will be treated as positive arguments, since COS(-m)=COS (m).

Upon entry to .SINX the product of m and $2/\pi$ is calculated by means of a call to .MPYU.

The heart of the SINE logic consists of the following expansion:

$$\mathrm{SIN}\left(\frac{m\pi}{2}\right) = m\,(P_0 + m\,(P_1 + m\,(P_2 + m\,P_3)))$$

with the following polynomial coefficients:

$P_0 = .15707 * 10$
$P_1 = -.64589$
$P_2 = +.79434 * 10^{-1}$
$P_3 = -.43330 * 10^{-2}$

REFERENCE:

John F. Hart, "Computer Approximations", New York: John Wiley & Sons, Inc., 1968; pages 116 and 117, INDEX 3300.

(Single Precision)

PROGRAM LISTING:

```
                                  ;FIXED POINT SIN/COS

                                  .TITLE  SINX
                                  .ENT    .SINX,.COSX
                                  .EXTD   .POLY, MPYU

                                  .ZREL
      00000-000001' .SINX:        SINX
      00001-000000' .COSX:        COSX
                                  .NREL

      00000'126401  COSX:         SUB     1,1,SKP          ;SET SIN/COS FLAG
      00001'126520  SINX:         SUBZL   1,1
      00002'044456                STA     1,SNFLG
      00003'040457                STA     0,SAV0           ;SAVE INPUT
      00004'054455                STA     3,RTURN          ;SAVE RETURN
      00005'111120                MOVZL   0,2              ;GET SIGN
      00006'125005                MOV     1,1,SNR          ;MAKE IT +VE IF COS ENTRY
      00007'101020                MOVZ    0,0
      00010'126560                SUBCL   1,1
      00011'044453                STA     1,SIGN           ;SAVE SIGN
      00012'024437                LDA     1,.20VPI         ;GET 2/PI
      00013'006002$               JSR     @ MPYU           ;AC0,1 = AC1*AC2
      00014'125120                MOVZL   1,1              ;PUSH OUT INTEGER PART
      00015'101100                MOVL    0,0
      00016'152560                SUBCL   2,2
      00017'125120                MOVZL   1,1
      00020'101100                MOVL    0,0
      00021'151100                MOVL    2,2              ;AC2 IS INTEGER PART
      00022'014436                DSZ     SNFLG            ;SINE?
      00023'151400                INC     2,2              ;NO,INCREMENT INTEGER
      00024'151222                MOVZR   2,2,SZC          ;ODD INTEGER PART?
      00025'100000                COM     0,0              ;YES, COMPLIMENT FRACTION
      00026'151232                MOVZR#  2,2,SZC          ;ODD INTEGER/2?
      00027'010435                ISZ     SIGN             ;YES CHANGE SIGN
      00030'111000                MOV     0,2              ;FRACTION IN AC2
      00031'050432                STA     2,SAV2           ;SAVE X
      00032'105000                MOV     0,1
      00033'006002$               JSR     @ MPYU           ;X**2
```

SIN/COS PROGRAM LISTING (cont'd):

```
00034'111000          MOV     0,2
00035'020416          LDA     0,SNCOF     ;AC0 POINTS TO COEFF.S
00036'024414          LDA     1,CFCNT     ;ORDER OF POLYNOMIAL
00037'006001$         JSR     @.POLY      ;EVALUATE POLYNOMIAL
00040'030423          LDA     2,SAV2
00041'006002$         JSR     @.MPYU
00042'151220          MOVZR   2,2
00043'143220          ADDZR   2,0
00044'024420          LDA     1,SIGN
00045'125220          MOVZR   1,1         ;GET SIGN IN CARRY
00046'105200          MOVR    0,1         ;PUSH INTO RESULT
00047'020413          LDA     0,SAV0      ;RESTORE INPUT ARGUMENT
00050'002411          JMP     @RTURN      ;RETURN TO CALL+1

00051'121377  .20VPI: 121377  ;2/PI(.505746  OCTAL,TWEAKED)
00052'000003  CFCNT:  3
00053'000054' SNCOF:  .+1
00054'177562          -216    ;-.0043331
00055'005053          5053    ;.0794343
00056'126524          -51254  ;-.6458928
00057'044420          44420   ;1.570791(ADD 1 LATER)

00060'000000  SNFLG:  0
00061'000000  RTURN:  0
00062'000000  SAV0:   0
00063'000000  SAV2:   0
00064'000000  SIGN:   0

---
              .END    ;END OF SINX,COSX
```

PURPOSE:

This routine sorts a table of pairs of words so that the first words of the pairs are in ascending order.

TITLE:

The title is RSORT.

INPUT:

A table containing pairs of words; the first word of each pair, the key word, must be an unsigned integer. AC1 must contain the starting address of the table; AC2 must contain the ending address of the table.

OUTPUT:

A table of pairs of words, sorted so that the key words are in ascending order. The key word and its accompanying data word are unchanged by the sort.

CALLING SEQUENCE AND ENTRY POINT:

Direct to RSORT with normal return to the instruction following the call.

CARRY AND REGISTERS:

AC0, AC3, and Carry are destroyed; AC1 and AC2 are unchanged.

LENGTH AND TIME:

This routine consists of 127 (octal) words and is normally relocatable. Execution time is approximately .3 seconds for a table containing 1000 pairs of words.

PROGRAM LISTING:

```
0001   RSORT
                              ;ROUTINE TO SHUFFLE A TABLE OF PAIRS OF WORDS
                              ;SO THE FIRST WORDS OF THE PAIRS(KEY WORDS)
                              ;ARE IN ASCENDING ORDER.
                              ;KEY WORD-DATA WORD PAIRING LEFT UNCHANGED.
                              ;KEY WORDS UNSIGNED INTEGERS.

                              ;CALLING SEQUENCE:

                              ;(LOAD IN AC1 STARTING ADDRESS OF TABLE
                              ;AND ADDRESS OF THE LAST WORD OF TABLE IN AC2)
                              ;        JSR    RSORT
                              ;        RETURN

                              ;TYPICAL TIME: .3 SEC FOR 1000 PAIRS


                              .TITLE  RSORT
                              .ENT    RSORT
                              .NREL

00000'020460 RSORT:  LDA     0,.STAK         ;SET UP STACK POINTER
00001'040460         STA     0,STAK
00002'102620         SUBZR   0,0             ;SET BIT 0
00003'010456 SORT1:  ISZ     STAK
00004'056455         STA     3,@STAK         ;SAVE RETURN
00005'010454         ISZ     STAK
00006'052453         STA     2,@STAK         ;SAVE HIGH POINTER
00007'135000         MOV     1,3             ;LOW POINTER
00010'024452         LDA     1,C2
00011'167000         ADD     3,1
00012'132112         ADCL#   1,2,SZC         ;SCAN COMPLETE?
00013'000440         JMP     SORT6           ;YES
00014'054447         STA     3,TEMP          ;SAVE LOW POINTER
00015'025400 SORT2:  LDA     1,0,3
00016'123415         AND#    1,0,SNR         ;LOW KEY BIT SET?
00017'000420         JMP     SORT3           ;NO
00020'024442         LDA     1,C2
00021'132400         SUB     1,2             ;DROP HIGH POINTER
00022'025000         LDA     1,0,2
00023'123414         AND#    1,0,SZR         ;HIGH KEY BIT SET?
00024'000415         JMP     SORT4           ;YES,NO SWAP
00025'040437         STA     0,BIT           ;SAVE COMPARISON BIT
00026'021400         LDA     0,0,3         ;SWAP KEY WORD-DATA WORD PAIRS
00027'025000         LDA     1,0,2
```

PROGRAM LISTING (cont'd):

```
00030'041000          STA     0,0,2
00031'045400          STA     1,0,3
00032'021401          LDA     0,1,3
00033'025001          LDA     1,1,2
00034'041001          STA     0,1,2
00035'045401          STA     1,1,3
00036'020426          LDA     0,BIT           ;RECOVER COMPARISON BIT
00037'024423 SORT3:   LDA     1,C2
00040'137000          ADD     1,3             ;BUMP LOW POINTER
00041'156414 SORT4:   SUB#    2,3,SZR         ;SCAN OVER?
00042'000753          JMP     SORT2           ;NO,CONTINUE
00043'101222          MOVZR   0,0,SZC         ;ALL BIT POSITIONS COVERED?
00044'000406          JMP     SORT5           ;YES
00045'024416          LDA     1,TEMP          ;NO,NEXT BIT
00046'004735          JSR     SORT1
00047'032412          LDA     2,@STAK
00050'004733          JSR     SORT1
00051'101121          MOVZL   0,0,SKP
00052'101100 SORT5:   MOVL    0,0
00053'026406 SORT6:   LDA     1,@STAK
00054'014405          DSZ     STAK
00055'036404          LDA     3,@STAK
00056'014403          DSZ     STAK
00057'001400          JMP     0,3

00060'000065'.STAK:   STAK0
00061'000000 STAK:    0
00062'000002 C2:      2
00063'000000 TEMP:    0
00064'000000 BIT:     0

      000042 STAK0:   .BLK    42

                              .END     ;END OF RADIX EXCHANGE SORT
```

PURPOSE:

This routine calculates the square root of an unsigned single precision integer.

TITLE:

ISQRT

ENTRY:

.ISQR

INPUT:

An unsigned fixed point number is input in AC0.

OUTPUT:

The square root of the input, truncated to the nearest integer, is output as an integer in AC1. Accuracy is to within one binary digit.

CALLING SEQUENCE AND ENTRY POINT:

Indirect through page zero entry .ISQR with return to the next sequential instruction following the call.

ERROR CONDITIONS:

There are no error conditions. All input values are acceptable as unsigned integers.

CARRY AND REGISTERS:

AC0, AC2, AC3, and Carry are destroyed.

LENGTH AND TIME:

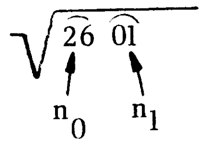
This routine consists of 1 ZREL and 24 octal NREL locations. Average execution time is 82.4 μs on both the Supernova and Nova 800, and 153 μs on the Nova 1200. Average execution time on the Supernova SC is 60.5 μs.

ALGORITHM:

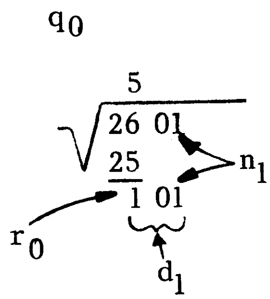
One of three different algorithms is usually selected when extracting a square root: the sum of odd integers method, Newton's iterative method of successive approximations, and the longhand partial quotient procedure. The partial quotient procedure was selected for this subroutine.

The partial quotient method of square root extraction is the same as the pencil-and-paper procedure commonly used to extract square roots of decimal numbers. In this method the number $\underline{n}$ whose root is to be extracted, is first paired off in groupings of two digits each (n_0 n_1, etc.)

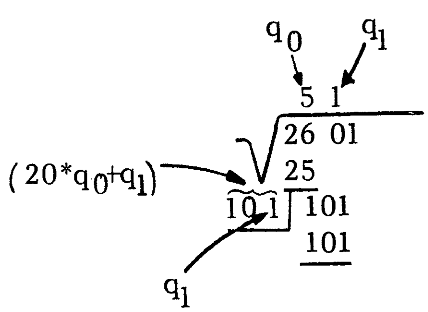
ALGORITHM (cont'd) :



The largest integer root less than or equal to the exact root of n_0 is selected, q_0, and becomes the most significant digit of the extracted root. q_0 is squared and is subtracted from the first pair, yielding remainder r_0.



r_0 is multiplied by 100 and is added to n_1, forming a new partial dividend, d_1 (the first partial dividend, d_0 equaled n_0, the first pair of digits). d_1 is divided by $20 * q_0 + q_1$, where q_1 is the largest possible integer such that $q_1 * (20*q_0 + q_1) \leq d_1$. The process is repeated with each partial dividend until all digits in n have been processed.



The subroutine algorithm is similar to that described above, but is modified to process 16-bit unsigned binary numbers instead of real decimal numbers. The subroutine considers n as 8 pairs of binary digits and repeats its procedure 8 times.

ALGORITHM (cont'd):

This algorithm was selected since its use results in the most efficient coding and fastest execution time for the complete range of permissible input values for all Nova family machines.

The sum of odd integers method given in "How to Use the Nova Computers" yields more efficient coding and faster execution times when input arguments are no greater than 1000_{10}. Over the whole range of permissible inputs however, its average execution time is 412 µs on the Supernova.

Newton's method requires either the use of hardware divide or the software divide subroutines, and its average execution time over the entire permissible range of inputs is 114 µs on the Supernova (with hardware divide).

REFERENCE:

Section 2.2 of "How to Use the Nova Computers" contains a discussion of the sum of odd integers method of square root extraction.

PROGRAM LISTING:

```
---
 0001   ISQRT
                                        ;SQUARE ROOT OF AN UNSIGNED INTEGER
                                        ;INTEGER ARGUMENT IN AC0
                                        ;OUTPUT: SQUARE ROOT TRUNCATED TO NEAREST
                                        ;INTEGER RETURNED IN AC1
                                        ;REGISTERS AND CARRY DESTROYED

                                        ;CALLING SEQUENCE:
                                        ;          JSR      @.ISQRT
                                        ;          RETURN LOCATION

                                                   .TITLE   ISQRT
                                                   .ENT     .ISQR
                                        .ZREL
   00000-000002'.ISQR:                  ISQRT
                                        .NREL
   00000'000000 RTRN:                   0          ;RETURN ADDRESS
   00001'000000 COUNT:                  0          ;COUNTING LOCATION
   00002'054776 ISQRT:                  STA        3,RTRN    ;SAVE RETURN ADDRESS
   00003'024420                         LDA        1,C8      ;SET UP COUNT 8
   00004'044775                         STA        1,COUNT
   00005'126400                         SUB        1,1       ;CLEAR PARTIAL QUOTIENT
   00006'152400                         SUB        2,2       ;AND REMAINDER

   00007'101120 ISQ1:                   MOVZL      0,0       ;ADD (NEXT) HIGHEST
   00010'151100                         MOVL       2,2       ;TWO BITS OF ARG. TO
   00011'101120                         MOVZL      0,0       ;FOUR TIMES PARTIAL REMAINDER
   00012'151100                         MOVL       2,2       ;TO GET NEW PARTIAL DIVIDEND
   00013'135120                         MOVZL      1,3       ;NEW QUOTIENT IS TWICE THE OLD ONE
   00014'175140                         MOVOL      3,3       ;IF TWICE NEW QUOTIENT+1 (DIVISOR)
   00015'172432                         SUBZ#      3,2,SZC   ;IS EQUAL TO OR LESS THAN
   00016'172420                         SUBZ       3,2       ;THE DIVIDEND, INCREMENT
   00017'125100                         MOVL       1,1       ;QUOTIENT AND SUBTRACT
   00020'014761                         DSZ        COUNT     ;DIVISOR FROM DIVIDEND
   00021'000766                         JMP        ISQ1      ;TO GET NEW REMAINDER
   00022'002756                         JMP        @RTRN     ;AND REPEAT 7 MORE TIMES

   00023'000010 C8:                     10         ;EIGHT

                                        .END       ;END OF SQRT(INTEGER)
---
 0002   ISQRT

C8        000023'        1/19      1/37
COUNT     000001'        1/17      1/20      1/33
ISQ1      000007'        1/24      1/34
ISQRT     000002'        1/14      1/18
RTRN      000000'        1/16      1/18      1/35
.ISQR     000000-        1/14
```

PURPOSE:

This routine calculates the square root of an unsigned double precision integer, and expresses the result as a truncated single precision integer.

TITLE:

DISQR

INPUT:

An unsigned double precision fixed point number is input in AC0 and ACl (more significant half in AC0).

OUTPUT:

The square root of the input, truncated to the nearest integer, is output as an integer in AC2. Accuracy is to within one binary digit.

CALLING SEQUENCE AND ENTRY POINT:

Indirect through page zero entry .DISQ with return to the next sequential instruction following the call.

ERROR CONDITIONS:

There are no error conditions. All input values are acceptable as unsigned integers.

CARRY AND REGISTERS:

AC0, ACl, AC3, and Carry are destroyed.

LENGTH AND TIME:

This routine consists of 1 ZREL and 16 octal NREL locations. Average execution time on the Nova 1200 with software divide is 6 ms.

ALGORITHM:

Initially a trial square root of 177777 octal is assumed. An integer division of the input argument by this trial square root is then performed, and the arithmetic mean of this quotient and the trial root is calculated, yielding a new trial root. The process of dividing the new trial root by the input argument continues until the new root differs by less than two from the next most recent trial root.

PROGRAM LISTING:

```
---
0001    DISQR
                              ;SQUARE ROOT OF AN UNSIGNED TWO WORD INTEGER
                              ;INPUT INTEGER IN AC0,1
                              ;RESULT (TRUNCATED TO NEAREST )INTEGER IN AC2
                              ;ACCUMULATORS AND CARRY LOST

                              ;CALLING SEQUENCE:
                              ;         ---
                              ;         ------
                              ;         -------(LOAD INTEGER IN AC0 AND AC1)
                              ;         JSR       @.DISQ
                              ;         RESULT LOCATION

                              ;RANGE OF ARGUMENT 0 TO (2**16-1)**2
                              ;NO ERROR MESSAGES
                              ;SUPPORTING ROUTINE  DVD
                              ;(UNSIGNED SINGLE PRECISION  INTEGER DIVIDE)

                                        .TITLE   DISQR
                                        .ENT     .DISQ
                                        .EXTN    DVD
                              .ZREL
  00000-000003'.DISQ:         DISQR
                              .NREL
  00000'000000 INTG0:         0
  00001'000000 INTG1:         0
  00002'000000 RTRN:          0

  00003'054777 DISQR:         STA       3,RTRN          ;SAVE RETURN ADDRESS
  00004'040774                STA       0,INTG0         ;SAVE HIGH WORD
  00005'044774                STA       1,INTG1         ;AND LOW WORD
  00006'152000                ADC       2,2             ;TRIAL ROOT SET 177777(OCTAL)

  00007'020771 DISQ1:         LDA       0,INTG0
  00010'024771                LDA       1,INTG1         ;LOAD ARGUMENT
  00011'177777                DVD                       ;DIVIDE IT BY TRIAL ROOT
  00012'133220                ADDZR     1,2             ;TAKE THE MEAN OF QUOTIENT AND
  00013'146654                SUBOR#    2,1,SZR         ;TRIAL ROOT TO IMPROVE ROOT
  00014'000773                JMP       DISQ1           ;IF NEW ROOT DIFFERS BY LESS
  00015'002765                JMP       @RTRN           ;THAN TWO FROM QUOTIENT, RETURN

                              .END      ;END OF SQUARE ROOT OF INTEGER
```

PURPOSE:

This routine computes the difference of two double precision two's complement integers.

TITLE:

The title is .DSUB.

INPUT:

The minuend is passed in AC0, AC1, (high order, low order). The subtrahend must be in two consecutive memory words, higher order followed by lower order. The word following the JSR should contain the address of the higher order word of the subtrahend.

OUTPUT:

The double precision difference is returned in AC0, AC1 (high order, low order).

CALLING SEQUENCE AND ENTRY POINT:

Indirect to .DSUB followed by the address of higher order word of subtrahend. Return will be to the instruction following the address parameter.

ERROR CONDITIONS:

Caution: No check is made for overflow.

CARRY AND REGISTERS:

AC0, AC1, AC3, and Carry are destroyed; AC2 remains unchanged.

LENGTH AND TIME:

This routine is 15 (octal) words and is normally relocatable. Execution time is 54.9 μs.

CHANGES FROM REVISION 2 TO REVISION 3 OF THE RELOCATABLE MATH LIBRARY FILE MANUAL

Substantive changes are described in the following list. Typographical corrections are not included.

Page 19 — Signed double precision division is now performed by the routine DMPY.

Page 30 — Signed double precision multiplication is now performed by the routine DDIV. The descriptions of DDIV and DMPY (page 19) replace that of .DPMD.

Page 42 — A description of RSORT, which sorts pairs of words in ascending order, has been added.

Some character conversion routines now require a "get a character" and/or "put a character" routine whose address must be stored in ZREL locations .GTCH and .PTCH, respectively. This change is reflected in the following pages: 12, 13, 15, 16, and 17.

cut along dotted line

DATA GENERAL CORPORATION
PROGRAMMING DOCUMENTATION
REMARKS FORM

DOCUMENT TITLE ______________________________

DOCUMENT NUMBER (lower righthand corner of title page) ______________

TAPE NUMBER (if applicable) ______________________________

Specific Comments. List specific comments. Reference page numbers when applicable. Label each comment as an addition, deletion, change or error if applicable.

General Comments and Suggestions for Improvement of the Publication.

FROM: Name: ______________________ Date: ____________
Title: ______________________
Company: ______________________
Address: ______________________
____

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary If Mailed In The United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Programming Documentation

FOLD UP SECOND FOLD UP

STAPLE