Notes on the SWTPC MP-N Calculator Interface and the Calc-1 Program

prepared for

Dr. Daniel W. Scott

CSCI 490.001

by

Daniel Paul Long

May 8, 1979

Notes on the SWTPC MP-N Calculator Interface and the Calc-1 Program

I assembled an SWTPC MP-N Calculator Interface and implemented it using the Calc-1 Program supplied with the kit. The following are my corrections to the documentation and my observations about the interface board and software.

This interface was bought to perform floating-point arithmetic and for its function capabilities such as SIN, COS, and e^x. My application required an integer truncation function that is not performed by this calculator, so I wrote a small assembly language subroutine to do it. A potentially irritating problem is that the calculator chip does not automatically convert to scientific notation if the numbers become too big to display in floating point. The control program must keep track of the display mode.

My application requires fast numerical processing, so, after observing the calculator's computational speed using the Calc-1 program, I decided that its trigonometric functions were too slow. I developed another approach to the problem using pure assembly language rather than partially relying on peripheral hardware processing.

Though I found no major flaws with the Calc-1 software, I did find some mistakes in its documentation. In Table V, the ASCII to Calculator Instruction Lookup Table, there are three errors: the hex value in the table where MSB is 0 and LSB is D should be 21, where MSB is 1 and LSB is 8 should be 2F, and where MSB is 2 and LSB is E, the hex value should be 0A. I added a one page summary of the Calc-1 instruction set to the documentation because the supplied documentation is somewhat confusing.

The Calc-1 program uses a part of RAM also used by the monitor. This provides problems when Calc-1 is halted to record it on cassette tape and resume after it has been recorded. This can be alleviated by changing the "LDX PARADR" instructions in lines 250, 850, 1660, and 1980 to "LDX #$800C," or, in hex, "FE A002" to "CE 800C." However, the interface must now always be plugged into I/O port 3. The data at hex address 028A is output to the terminal to clear the screen. The ADM terminals we have use a different character to clear the screen, so I changed the data at that address to hex 1A. Calc-1 also uses address modification within itself so that, as a result, it cannot be implemented in ROM.

I tested the MP-N interface with the Calc-1 program by making various operand and operator entries and checking the results against the results I obtained with my own pocket scientific calculator. I noticed no major discrepancies between the two sets of results.

S W T P C

MP-N Calculator Interface

Documentation

Assembly Instructions MP-N Calculator Interface Kit

The Southwest Technical Products MP-N Calculator Interface interfaces the SWTPC 6800 Computer System thru a peripheral interface Adaptor (PIA) to the National Semiconductor MM57109 Number Oriented Processor. This "processor" is a Reverse Polish Notation (RPN) calculator chip without the internal keypad interfacing circuitry which has made interfacing to calculator chips so difficult in the past. This chip allows data and instruction entry in conventional binary form and speeds entry with the elimination of the debounce circuitry built into conventional calculator chips. It is called a processor because it has instructions and control lines which allow it to operate in conjunction with ROM and RAM as a stand alone numerical processor. It may however be operated as a computer peripheral for numerical calculation and this is the configuration in which the chip has been implemented.

All interfacing from the 6800 Computer System to the calculator chip has been done thru a 6820 PIA. Both the PIA and calculator chip reside on a 3 ½" X 5 ¼ double sided, plated thru hole circuit board plugged onto one of the seven available interface card positions on the mother board of the 6800 Computer. All data and instructions fed to and all results received from the calculator chip are handled by your own assembler or machine language program. The calculator features reverse Polish notation, floating point or scientific notation, up to an eight digit mantissa and two digit exponent, trig functions, base 10 and natural logarithms, and overflow indicator.

PC Board Assembly

NOTE: Since all of the holes on the PC board have been plated thru, it is only necessary to solder the components from the bottom side of the board. The plating provides the electrical connection from the "BOTTOM" to the "TOP" foil of each hole. Unless otherwise noted it is important that none of the connections be soldered until all of the components of each group have been installed on the board. This makes it much easier to interchange components if a mistake is made during assembly. Be sure to use a low wattage iron (not a gun) with a small tip. Do not use acid core solder or any type of paste flux. We will not guarantee or repair any kit on which either product has been used. Use only the solder supplied with the kit or a 60/40 alloy resin core equivalent. Remember all of the connections are soldered on the bottom side of the board only. The plated-thru holes provide the electrical connection to the top foil.

( ) Before installing any parts on the circuit board, check both sides of the board over carefully for incomplete etching and foil "bridges" or "breaks". It is unlikely that you will find any, but should there be one, especially on the "TOP" side of the board, it will be very hard to locate and correct after all of the components have been installed on the board.

( ) Starting from one end of the circuit board install each of the three, 10 pin Molex female edge connectors along the lower edge of board. These connectors must be inserted from the "TOP" side of the board and must be pressed down firmly against the circuit board, so that each pin extends completely into the holes on the circuit board. Not being careful here will cause the board to either wobble and/or be crooked when plugging it onto the mother board. It is either wobble and/or be crooked when plugging it onto the mother board. It is suggested that you solder only the two end pins of each of the three connectors until all have been installed at which time if everything looks straight and rigid you should solder the as yet unsoldered pins.

( ) Insert the small nylon indexing plug into the lower edge connector pin indicated by the small triangular arrow on the "BOTTOM" side of the circuit board. This prevents the board from being accidently plugged on incorrectly.

( ) Attach all of the resistors to the board. As with all other components unless noted, use the parts list and component layout drawing to locate each part and install from the "TOP" side of the board bending the leads along the "BOTTOM" side of the board and trimming so that 1/16" to 1/8" of wire remains. Solder.

( ) Install the capacitors on the circuit board. Be sure to orient electrolytic capacitor C4 so its polarity matches with that shown on the component layout drawing. Solder.

( ) Install the transistor and diode. These components must be oriented to match the component layout drawing. Solder.

( ) Install integrated circuit IC2 on the circuit board. This component must be oriented so its metal face is facing the circuit board and is secured to the circuit board with a #4 - 40 X 1/4" screw, lockwasher and nut. A heatsink is not used. The three leads of the integrated circuit must be bent down into each of their respective holds. Solder.

NOTE: MOS integrated circuits are susceptible to damage by static electricity. Although some degree of protection is provided internally within the integrated circuits, their cost demands the utmost in care . Before opening and/or installing any MOS integrated circuits you should ground your body and all metallic tools coming into contact with the leads, thru a 1 M ohm 1/4 watt resistor (supplied with the kit). The ground must be an "earth" ground such as a water pipe, and not the circuit board ground. As for the connection to your body, attach a clip lead to your watch or metal ID bracelet. Make absolutely sure you have the 1 Meg ohm resistor connected between you and the "earth" ground, otherwise you will be creating a dangerous shock hazard. Avoid touching the leads of the integrated circuits any more than necessary when installing them, even if you are grounded. On those MOS IC's being soldered in place, the tip of the soldering iron should be grounded as well(separately from your body ground) either with or without a 1 Meg ohm resistor. Most soldering irons having a three prong line cord plug already have a grounded tip. Static electricity should be an important consideration in cold, dry environments. It is less of a problem when it is warm and humid.

( ) Install MOS integrated circuits IC1, IC3, IC4 and IC5 following the precautions given in the preceding section. As they are installed, make sure they are down firmly against the board before soldering all of their leads. <u>Do not</u> bend the leads on the back side of the board. Doing so makes it very difficult to remove the integrated circuit should replacement ever be necessary. The "dot" or "notch" on the end of the package is used for orientation purposes and <u>must</u> match with that shown on the component layout drawing for the IC. Solder.

( ) Working from the "TOP" side of the circuit board, fill in all of the feed-thru's with molten solder. The feed-thru's are those unused holes on the board whose internal plating connects the "TOP" and "BOTTOM" circuit connections. Filling these feed-thru's with molten solder guarantees the integrity of the connections and increases the current handling capability.

( ) Now that all of the components have been installed on the board, double check to make sure all have been installed correctly in their proper location.

( ) Check very carefully to make sure that all connections have been soldered. It is very easy to miss some connections when soldering which can really cause some hard to find problems later during checkout. Also look for solder "bridges" and "cold" solder joints which are another common problem.

Since the MP-N circuit board now contains MOS devices, it is susceptible to damage from severe static electrical sources. One should avoid handling the board any more than necessary and when you must, avoid touching or allowing anything to come into contact with any of the conductors on the board.

Using the Calculator Interface

Table I gives a complete list and description of the calculator chip's instruction set. Remember that some of the instructions are for stand alone processing systems and are not used on this interface. All numerical entry is in Reverse Polish Notation (RPN) and anyone familiar with Hewlett Packard calculators should have no problem with the data entry sequence. For those not familiar with RPN, the following should be helpful:

To add 7 + 8, enter the following
7 enter 8 + (4 entries)
The answer is now stored in the X accumulator within the calculator chip
The OUT instruction may be used to output the answer

To find the inverse sine of 0.5, enter the following:
0.5 INV SIN (5 entries)
The answer is now stored in the X accumulator within the calculator chip.
The OUT instruction may be used to output the answer.

In order to simplify the interfacing between your program and the calculator interface, you will probably want to incorporate the following subroutines into your program.

INITAL SUBROUTINE

The INITAL or initialize subroutine configures the PIA interfacing to the calculator chip. This subroutine need only be used once; and is best placed somewhere at the beginning of your program. It is responsible for initializing the data direction registers and control registers of the PIA. The subroutine requires that the index register be loaded with the "lowest" address of the PIA interfacing to the calculator chip prior to execution.

This "lowest" address depends upon which interface port position the MP-N calculator card is plugged. The table below give the "lowest" address of each interface card position.

Address Assignments

PORTØ	8ØØØ
PORT1	8ØØ4
PORT2	8ØØ8
PORT3	8ØØC
PORT4	8Ø1Ø
PORT5	8Ø14
PORT6	8Ø18
PORT7	8Ø1C

```
86 7F    INITAL LDA A   #$7F     INIT  A SIDE OF PIA
A7 00           STA A   0,X
86 36           LDA A   #$36     HIGH HOLD-POS READY
A7 01           STA A   1,X
86 00           LDA A   #$00     INIT  B SIDE OF PIA
A7 02           STA A   2,X
86 34           LDA A   #$34     NEG R/W
A7 03           STA A   3,X
A6 02           LDA A   2,X      CLEAR R/W FLAG
39              RTS
```

OUTINS SUBROUTINE

The OUTINS or out instruction subroutine is used to get program data and instructions into the calculator. To send a digit or instruction to the calculator chip, use Table II to find the OP code of the instruction you wish to send. Load this OP code into the A accumulator and jump or branch to the OUTINS subroutine. If you have a string of data you wish to send, just recycle thru this subroutine as many times as necessary. The subroutine takes care of all of the READY and HOLD signals to the calculator chip so there is no worry of sending data faster than the calculator chip can accept it. The subroutine destroys the contents of the B accumulator during execution while the contents of the A accumulator and index register are not destroyed.

```
E6 01    OUTINS LDA B   1,X      WAIT FOR READY
2A FC           BPL     OUTINS
A7 00           STA A   0,X      FORWARD INSTRUCTION TO CALC
E6 00           LDA B   0,X      CLEAR FLAG BIT
C6 3C           LDA B   #$3C     LOW HOLD-NEG READY
E7 01           STA B   1,X      BRING HOLD LINE LOW
E6 01    WAIT10 LDA B   1,X
2A FC           BPL     WAIT10   LOOK FOR READY LOW
E6 00           LDA B   0,X      CLEAR FLAG BIT
C6 36           LDA B   #$36     HIGH HOLD-POS READY
E7 01           STA B   1,X      RETURN HOLD LINE HIGH
39              RTS
```

SETMEM SUBROUTINE

The SETMEM or set memory subroutine initializes the memory locations to which the calculator's output data will be stored. This subroutine must be executed immediately before the OUTANS subroutine is used. Although it can be changed, memory locations ØØ2Ø thru ØØ2B have been designated the temporary storage locations for the calculator's output data. The subroutine sets memroy location ØØ2Ø to a ØØ while locations 21 thru 2B are set to 20 (ASCII spaces). This subroutine destroys the contents of the index register and B accumulator. The contents of thre A accumulator are not destroyed.

```
7F 0020 SETMEM CLR      $20       CLEAR $0020
CE 0020        LDX      #$20      BOTTOM OF BUFFER
C6 20          LDA B    #$20
08      LOOP1  INX
E7 00          STA B    0,X       STORE A SPACE
8C 002B        CPX      #$2B      CHECK FOR TOP OF BUFFER
26 F8          BNE      LOOP1
39             RTS
```

OUTANS SUBROUTINE

The OUTANS or output answer subroutine outputs the contents of the X register within the calculator chip in BCD to memory locations 0020 thru 002B. Since the mantissa digit count of the calculator is variable, the previous SETMEM subroutine blanks out any digit location not filled by the OUTANS subroutine. It is very important that the SETMEM subroutine be used each time before executing the OUTANS subroutine. The OUTANS subroutine outputs data in two different formats depending upon whether the calculator chip is in the floating point or scientific mode. The calculator initially starts out in the floating point mode where it will remain until changed by the TOGM (22_{16}) instruction. This calculator does not automatically convert to scientific notation if the numbers become too big to handle in floating point as many do. An MCLR ($2F_{16}$) instruction will always reset the calculator chip to the floating point mode regardless of what mode it was in originally. Since the calculator chip does not tell you what mode it is in when it is outputting data, your program must know so you can process the data accordingly. Table IV shows the format in which the data is stored. At the end of the OUTANS subroutine, the N bit of the condition code register is set if an error has transpired since the last execution of the OUTANS subroutine. You may use a BMI instruction to catch and branch to an error routine to note the error. You should then send an ECLR ($2B_{16}$) instruction to the calculator chip to reset the calculator chip's error flag. Disregarding the error flag on the calculator chip will cause no problems. The chip will continue to function regardless of the state of the flag. The subroutine requires that the index register be loaded with the "lowest" address of the PIA interfacing to the calculator chip prior to execution. Since the SETMEM subroutine usually run prior to this destroys the contents of the index register, don't forget to reload the index register before branching to the OUTANS subroutine. The OUTANS subroutine destroys the contents of both the A and B accumulators during execution while the contents of the index register is not changed.

```
E6 01    OUTANS LDA B  1,X
2A FC           BPL    OUTANS
A6 00           LDA A  0,X       CLEAR FLAG BIT
86 16           LDA A  #$16      SEND AN OUT
A7 00           STA A  0,X
C6 3E           LDA B  #$3E      LOW HOLD-POS READY
E7 01           STA B  1,X       BRING HOLD LINE LOW
E6 01    WAIT30 LDA B  1,X       WAIT FOR SECOND READY
2A FC           BPL    WAIT30
E6 00           LDA B  0,X       CLEAR FLAG BIT
86 0F           LDA A  #$0F
A7 00           STA A  0,X       SEND A NOP
E6 03    WAIT3  LDA B  3,X       LOOK FOR R/W STROBE
2B 06           BMI    OUTDIG    TRANSFER CALC DATA TO MEMORY
E6 01           LDA B  1,X       LOOK FOR READY STROBE
2B 16           BMI    CONFLG    PRINT MEMORY CONTENTS
20 F6           BRA    WAIT3
A6 02    OUTDIG LDA A  2,X       LOAD OUT DATA INTO A
16              TAB
84 0F           AND A  #$0F      ELIMINATE UPPER 4 BITS
8A 30           ORA A  #$30      CONVERT TO ASCII DATA
54              LSR B
54              LSR B
54              LSR B
54              LSR B
CA 20           ORA B  #$20      INCREMENT ADDRESSES BY $20
F7 01C6         STA B  POINT2+1  STORE OUT DATA SEQUENTIALLY
97 00    POINT2 STA A  $0
20 E2           BRA    WAIT3
86 36    CONFLG LDA A  #$36      HIGH HOLD-POS READY
A7 01           STA A  1,X       BRING HOLD LINE HIGH
A6 00           LDA A  0,X       CLEAR FLAG BIT
39              RTS
```

Number Entry Rules

When a digit, decimal point, or π is entered with an 0-9, DP, or PI instruction, the stack is first pushed and the X register cleared: Z→T, Y→Z, X→Y, 0→X. This process is referred to as "initiation of number entry." Following this, the digit and future digits are entered into the X mantissa. Subsequent entry of digits or DP, EE, or CS instructions do not cause initiation of number entry. Digits following the eighth mantissa digit are ignored. This number entry mode is terminated by any insfruciton except 0-9, DP, EE, CS, PI, or HALT. Termination of number entry means two things. First, the number is normalized by adjusting the exponent and decimal point position so that the decimal point is to the right of the first mantissa digit. Second, the next digit, decimal point, or π entered will cause initiation of number entry, as already described. There is one exception to the number entry initiation rule. The stack is <u>not</u> pushed if the instruction prior to the entered digit was an ENTER. However, the X register is still cleared and the entered digit put in X.

The ENTER key itself terminates number entry and pushes the stack. The OUT instruction terminates number entry and prepares the stack for pushing upon the next entry of data. This means that if you use the ENTER and OUT instrucitons consecutively, the stack gets pushed twice which is not what you want. If you wish to ENTER data and immediately OUT the result, use only the OUT instruction. The OUT performs the entry. If you do not wish to OUT the ENTER'ed data, just use the ENTER instruction by itself.

The AIN and IN instructions should <u>not</u> be used for number entry. Provisions have not been made for their use on this interface.

How It Works

Peripheral Interface Adaptor (PIA) IC1 interfaces the MM57109 calculator chip, IC3, to the SWTPC 6800 buss. The first six bits of the A side of the PIA are used to feed instructions to the calculator chip while the eighth is used as an input to monitor the ERROR output of the calculator. Control line CA1 outputs HOLD signals to, while control line CA2 inputs READY signals from the calculator chip. The first four bits of the B side of the PIA are used to input BCD digit data while the last four bits input digit addresses. The CB1 line inputs READ/WRITE signals while the CB2 control line is not used. Hex inverter/buffer, IC4, is used primarily as the 320 to 400 Khz single phase oscillator required by the calculator chip. One section is used to invert the HOLD signal going to the calculator. Shift register IC5 generates the POR signal required for proper startup and initialization. +5 VDC power required by the board is supplied by voltage regulator IC2 while -4 VDC voltage is supplied by transistor Q1 and its associated components. Figure I shows a block diagram for the internal construction of the calculator chip.

Resistors

R1	47K ohm ¼ watt resistor
R2	1K " " "
R3	10K " " "
R4	10K " " "
R5	10K " " "
R6	10K " " "
R7	10K " " "
R8	22K " " "
R9	22K " " "
R10	22K " " "
R11	22K " " "
R12	12K " " "
R13	27 " " "
R14	3.3K " " "
R15	10K " " "
R16	47K " " "
R17	10K " " "

Capacitors

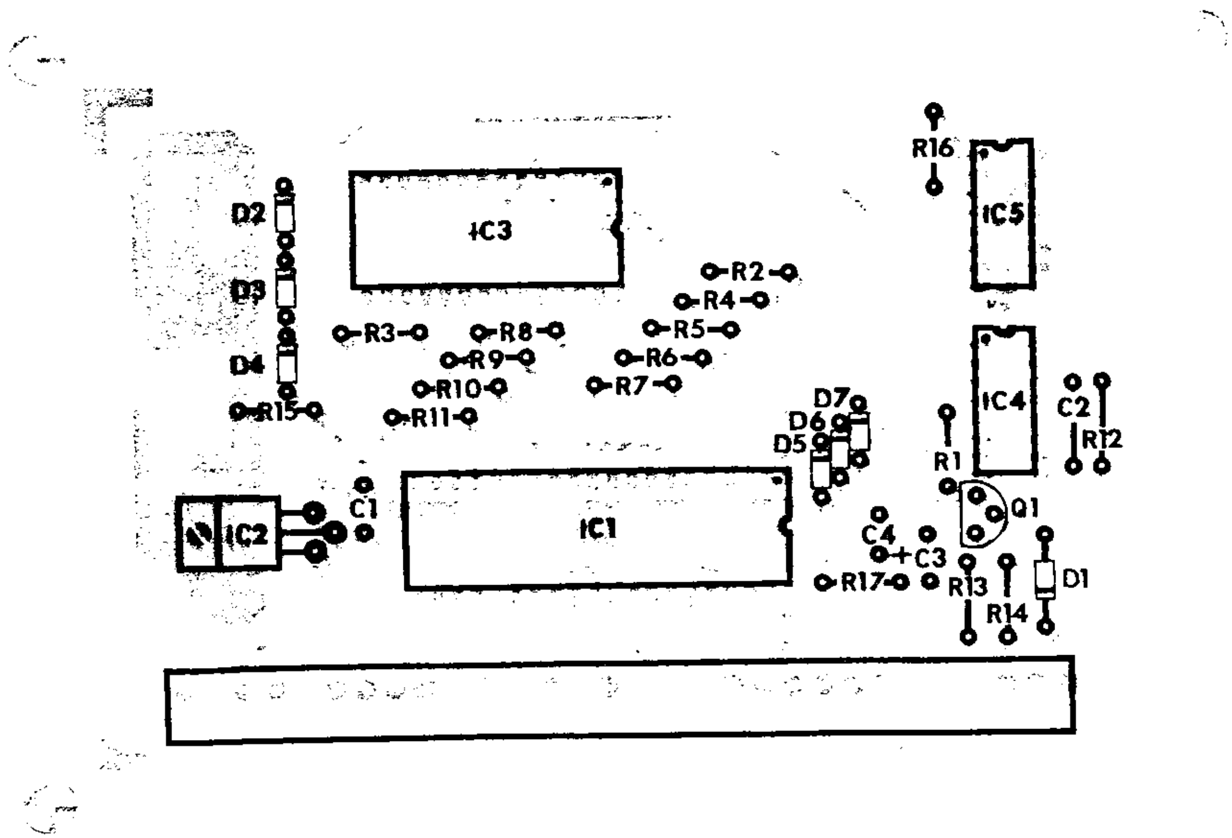
C1	0.1 mfd capacitor
C2	100 pfd capacitor
C3	0.1 mfd capacitor
C4*	10 mfd@ 15 VDC electrolytic

Diodes and Transistors

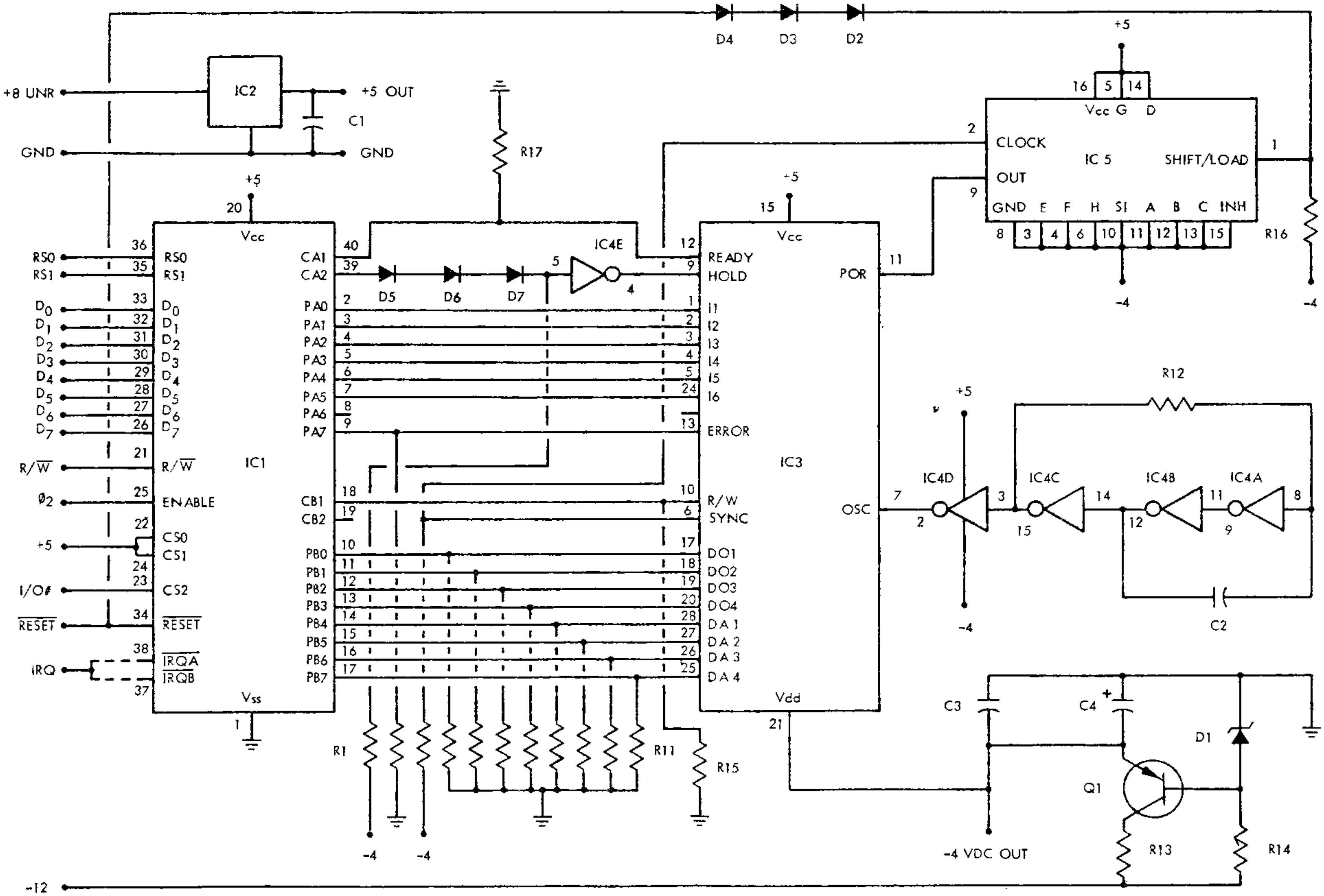
D1*	4.7 volt 400 mw zener diode 1N5230 or 1N4732
D2*	1N4148 silicon diode
D3*	" " "
D4*	" " "
D5*	" " "
D6*	" " "
D7*	" " "
Q1*	2N5087 transistor

Integrated Circuits

IC1*	6820 MOS peripheral interface adaptor 6821
IC2*	7805 voltage regulator
IC3*	MM57109 FAN MOS calculator chip
IC4*	4009 or 14009 MOS hex inverter
IC5*	74C165 MOS shift register


R16
IC5
IC3
D2
D3
D4
R2
R4
R5
R6
R7
R3
R8
R9
R10
R11
R15
IC4
C2
R12
R1
D7
D6
D5
C1
IC2
IC1
Q1
C4
C3
R17
R13
R14
D1

MP-N CALCULATOR INTERFACE



CALC-1 Program

In order to see how the calculator chip is used and how to incorporate these subroutines into a program, the CALC-1 program listing is given. CALC-1 allows the operator to use the calculator chip just as you would a standard RPN desk calculator with the same features. All communication to the chip is done thru the terminal's keyboard with all results displayed on the terminal's display. Since the terminal's keyboard just has standard ASCII characters rather than the labeling found on calculator keys; selected ASCII characters have been substituted for normal calculator function keys. It is the job of the CALC-1 program to accept all data and instruction commands from the terminal's keyboard, send them to the calculator chip and display all results on the terminal's display. The program resides from memory locations 0020 thru 02C0 which is approximately 700 bytes of code. Since most of the lower 256 bytes are used for the ASCII character lookup table and some of the upper is used for terminal interfacing, you should be able to incorporate the package into your program using somewhat less memory than was used here.

The program starts at line 50 by storing the ASCII lookup table from memory locations ØØ8Ø thru ØØFF. This table covers the entire 128 character ASCII set. Whenever an ASCII character is received from the keyboard it is OR'ed with 8Ø, and the resulting address contains the selected command or instruction for the calculator chip. Line 210 ORG's the program at memroy location Ø1ØØ where the terminal's screen is cleared and titled. Line 250 loads the index register extended with the contents of memory locations AØØ2 and AØØ3 with 8ØØC, the starting address of Port 3. If you wish to plug the calculator board onto an I/O port other than PORT 3. Use the table below to find the address to be loaded into memory locations AØØ2 and AØØ3 prior to executing the program.

PORTO	8ØØØ
PORT1	8ØØ4(Serial control interface only)
PORT2	8ØØ8
PORT3	8ØØC
PORT4	8Ø1Ø
PORT5	8Ø14
PORT6	8Ø18
PORT7	8Ø1C

Lines 280 thru 370 contain the INITAL subroutine described in detail earlier. lines 380 thru 410 accept entered keyboard commands, lookup the selected calculator instructions and deposit the data or instruction in the A accumulator. Lines 440 thru 550 contain the OUTINS subroutine described in detail earlier. Lines 550 thru 740 check to see what instruction or data has been entered so the result may be output if appropiate. Line 710 looks for the TOGM instruction so the program knows which display mode to use when outputting data. Lines 770 thru 840 contain the SETMEM subroutine described in detail earlier. Since the SETMEM subroutine destroys the contents of the index register, line 850 reloads it before proceeding to the OUTANS subroutine contained in lines 880 thru 1200. Line 1210 checks to see of the ERROR flag was set during the last output sequence. If so, program control is transferred to lines 1220 thru 1350 where an error message is output and the error flag cleared by sending an ECLR instruction to the calculator chip. Line 1380 tests to see if the calculator is in the floating point or scientific mode. If floating point, control is transferred to lines 1400 thru 1670. If scientific, control is transferred to lines 1680 thru 1990. In both modes the data is output to the display in the selected mode and program control is transferred back to line 380 where new commands or data may be entered.

```
00010                       NAM     CALC-1
00020                       OPT     O
00030                       OPT     S
00040 0080                  ORG     $0080
00050 0080 0F               FCB     $0F,$0F,$0F,$0F,$0F,$0F,$0F,$0F
      0081 0F
      0082 0F
      0083 0F
      0084 0F
      0085 0F
      0086 0F
      0087 0F
00060 0088 0F               FCB     $0F,$0F,$0F,$0F,$0F,$21,$0F,$0F
      0089 0F
      008A 0F
      008B 0F
      008C 0F
      008D 21
      008E 0F
      008F 0F
00070 0090 0F               FCB     $0F,$0F,$0F,$0F,$0F,$0F,$0F,$0F
      0091 0F
      0092 0F
      0093 0F
      0094 0F
      0095 0F
      0096 0F
      0097 0F
00080 0098 2F               FCB     $2F,$0F,$0F,$0F,$0F,$0F,$0F,$0F
      0099 0F
      009A 0F
      009B 0F
      009C 0F
      009D 0F
      009E 0F
      009F 0F
00090 00A0 21               FCB     $21,$0F,$0F,$0F.$0F,$0F,$0F,$0F
      00A1 0F
      00A2 0F
      00A3 0F
      00A4 0F
      00A5 0F
      00A6 0F
      00A7 0F
00100 00A8 0F               FCB     $0F,$0F,$3B,$39,$0F,$3A,$0A,$3C
      00A9 0F
      00AA 3B
      00AB 39
      00AC 0F
      00AD 3A
      00AE 0A
      00AF 3C
00110 00B0 00               FCB     $00,$01,$02,$03,$04,$05,$06,$07
      00B1 01
```

```
      00B2 02
      00B3 03
      00B4 04
      00B5 05
      00B6 06
      00B7 07
      00B8 08              FCB     $08,$09,$0F,$0F,$0F,$0F,$22,$0F
      00B9 09
      00BA 0F
      00BB 0F
      00BC 0F
      00BD 0F
      00BE 22
      00BF 0F
      00C0 0F              FCB     $0F,$1B,$36,$25,$2D,$0B,$2C,$1C
      00C1 1B
      00C2 36
      00C3 25
      00C4 2D
      00C5 0B
      00C6 2C
      00C7 1C
      00C8 1D              FCB     $1D,$20,$0F,$0F,$0F,$18,$35,$23
      00C9 20
      00CA 0F
      00CB 0F
      00CC 0F
      00CD 18
      00CE 35
      00CF 23
      00D0 0D              FCB     $0D,$33,$37,$24,$26,$32,$34,$31
      00D1 33
      00D2 37
      00D3 24
      00D4 26
      00D5 32
      00D6 34
      00D7 31
      00D8 30              FCB     $30,$2B,$0C,$0F,$0F,$0F,$38,$0F
      00D9 2B
      00DA 0C
      00DB 0F
      00DC 0F
      00DD 0F
      00DE 38
      00DF 0F
      00E0 0F              FCB     $0F,$0F,$36,$25,$2D,$0B,$2C,$1C
      00E1 0F
      00E2 36
      00E3 25
      00E4 2D
      00E5 0B
      00E6 2C
      00E7 1C
```

```
00E8 1D                FCB     $1D,$20,$0F,$0F,$0F,$18,$35,$23
00E9 20
00EA 0F
00EB 0F
00EC 0F
00ED 18
00EE 35
00EF 23
00F0 0D                FCB     $0D,$33,$37,$24,$26,$32,$34,$31
00F1 33
00F2 37
00F3 24
00F4 26
00F5 32
00F6 34
00F7 31
00F8 30                FCB     $30,$2B,$0C,$0F,$0F,$0F,$0F,$0F
00F9 2B
00FA 0C
00FB 0F
00FC 0F
00FD 0F
00FE 0F
00FF 0F
0100                   ORG     $0100
0100 8E A047 START     LDS     #$A047    DECREMENT STACK
0103 CE 0287           LDX     #CLRSCN
0106 BD E07E           JSR     PDATA1    CLEAR AND TITLE TERM.
0109 FE A002           LDX     PARADR
010C 8D 02             BSR     INITAL
010E 20 13             BRA     COMAND
0110 86 7F   INITAL    LDA A   #$7F      INIT. A SIDE OF PIA
0112 A7 00             STA A   0,X
0114 86 36             LDA A   #$36      HIGH HOLD-POS READY
0116 A7 01             STA A   1,X
0118 86 00             LDA A   #$00      INIT. B SIDE OF PIA
011A A7 02             STA A   2,X
011C 86 34             LDA A   #$34      NEG R/W
011E A7 03             STA A   3,X
0120 A6 02             LDA A   2,X       CLEAR R/W FLAG
0122 39                RTS
0123 BD E1AC COMAND    JSR     INEEE     GET OPERATOR DATA
0126 8A 80             ORA A   #$80      POSITION TO THE TOP OF TABLE
0128 B7 012C           STA A   POINT+1
012B 96 00   POINT     LDA A   $00
012D 81 21             CMP A   #$21
012F 27 43             BEQ     ZERMEM
0131 8D 02             BSR     OUTINS
0133 20 17             BRA     CHRCHK
0135 E6 01   OUTINS    LDA B   1,X       WAIT FOR READY
0137 2A FC             BPL     OUTINS
0139 A7 00             STA A   0,X       FORWARD INSTRUCTION TO CALC.
013B E6 00             LDA B   0,X       CLEAR FLAG BIT
013D C6 3C             LDA B   #$3C      LOW HOLD-NEG READY
```

```
013F E7 01             STA B  1,X      BRING HOLD LINE LOW
0141 E6 01     WAIT10  LDA B  1,X
0143 2A FC             BPL    WAIT10   LOOK FOR READY LOW
0145 E6 00             LDA B  0,X      CLEAR FLAG BIT
0147 C6 36             LDA B  #$36     HIGH HOLD-POS READY
0149 E7 01             STA B  1,X      RETURN HOLD LINE HIGH
014B 39                RTS
014C 81 2F     CHRCHK  CMP A  #$2F
014E 26 03             BNE    SKIP75
0150 7F 02AE           CLR    FORMAT
0153 7D 02AF   SKIP75  TST    SMDC     CHECK FOR PREVIOUS SMDC INSTR
0156 26 1C             BNE    ZERMEM
0158 81 0F     CONT50  CMP A  #$0F
015A 27 C7             BEQ    COMAND   GET MORE DATA IF NOP
015C 81 18             CMP A  #$18
015E 26 05             BNE    SKIP25
0160 73 02AF           COM    SMDC
0163 20 BE             BRA    COMAND   GET MORE DATA IF SMDC
0165 81 20     SKIP25  CMP A  #$20
0167 27 BA             BEQ    COMAND   GET MORE DATA IF INV
0169 81 0B             CMP A  #$0B
016B 23 B6             BLS    COMAND   GET MORE DATA IF NUMBERS
016D 81 22             CMP A  #$22     LOOK FOR TOGM
016F 26 03             BNE    ZERMEM
0171 73 02AE           COM    FORMAT
0174 7F 02AF   ZERMEM  CLR    SMDC     ZERO SMDC
0177 8D 02             BSR    SETMEM
0179 20 11             BRA    LODADR
017B 7F 0020   SETMEM  CLR    $20      CLEAR $0020
017E CE 0020           LDX    #$20     BOTTOM OF BUFFER
0181 C6 20             LDA B  #$20
0183 08        LOOP1   INX
0184 E7 00             STA B  0,X      STORE A SPACE
0186 8C 002B           CPX    #$2B     CHECK FOR TOP OF BUFFER
0189 26 F8             BNE    LOOP1
018B 39                RTS
018C FE A002   LODADR  LDX    PARADR
018F 8D 02             BSR    OUTANS
0191 20 3D             BRA    OUTCHR
0193 E6 01     OUTANS  LDA B  1,X
0195 2A FC             BPL    OUTANS
0197 A6 00             LDA A  0,X      CLEAR FLAG BIT
0199 86 16             LDA A  #$16     SEND AN OUT
019B A7 00             STA A  0,X
019D C6 3E             LDA B  #$3E     LOW HOLD-POS READY
019F E7 01             STA B  1,X      BRING HOLD LINE LOW
01A1 E6 01     WAIT30  LDA B  1,X      WAIT FOR SECOND READY
01A3 2A FC             BPL    WAIT30
01A5 E6 00             LDA B  0,X      CLEAR FLAG BIT
01A7 86 0F             LDA A  #$0F
01A9 A7 00             STA A  0,X      SEND A NOP
01AB E6 03     WAIT3   LDA B  3,X      LOOK FOR R/W STROBE
01AD 2B 06             BMI    OUTDIG   TRANSFER CALC DATA TO MEMORY
01AF E6 01             LDA B  1,X      LOOK FOR READY STROBE
```

```
01B1 2B 16          BMI     CONFLG   PRINT MEMORY CONTENTS
01B3 20 F6          BRA     WAIT3
01B5 A6 02   OUTDIG LDA A   2,X      LOAD OUT DATA INTO A
01B7 16             TAB
01B8 84 0F          AND A   #$0F     ELIMINATE UPPER 4 BITS
01BA 8A 30          ORA A   #$30     CONVERT TO ASCII DATA
01BC 54             LSR B
01BD 54             LSR B
01BE 54             LSR B
01BF 54             LSR B
01C0 CA 20          ORA B   #$20     INCREMENT ADDRESSES BY $20
01C2 F7 01C6        STA B   POINT2+1 STORE OUT DATA SEQUENTIALLY
01C5 97 00   POINT2 STA A   $0
01C7 20 E2          BRA     WAIT3
01C9 86 36   CONFLG LDA A   #$36     HIGH HOLD-POS READY
01CB A7 01          STA A   1,X      BRING HOLD LINE HIGH
01CD A6 00          LDA A   0,X      CLEAR FLAG BIT
01CF 39             RTS
01D0 2A 1E   OUTCHR BPL     CONT1    SKIP IF NO ERROR
01D2 E6 01   WAIT70 LDA B   1,X      WAIT FOR READY
01D4 2A FC          BPL     WAIT70
01D6 86 2B          LDA A   #$2B     ERROR CLEAR INSTRUCTION
01D8 A7 00          STA A   0,X
01DA E6 00          LDA B   0,X      CLEAR FLAG BIT
01DC C6 3C          LDA B   #$3C     LOW HOLD-NEG READY
01DE E7 01          STA B   1,X      BRING HOLD LOW
01E0 E6 01   WAIT71 LDA B   1,X
01E2 2A FC          BPL     WAIT71
01E4 E6 00          LDA B   0,X      CLEAR FLAG BIT
01E6 C6 36          LDA B   #$36     HIGH HOLD-POS READY
01E8 E7 01          STA B   1,X      RETURN HOLD HIGH
01EA CE 02B0        LDX     #ERRMSG
01ED BD E07E        JSR     PDATA1
01F0 CE 02A8 CONT1  LDX     #CRLF
01F3 BD E07E        JSR     PDATA1
01F6 7D 02AE        TST     FORMAT
01F9 2B 3F          BMI     SCINOT
01FB CE 0022 FLOPNT LDX     #$22     FLOATING POINT NOTATION
01FE A6 00          LDA A   0,X      INPUT MANTISSA SIGN DATA
0200 84 08          AND A   #$08     MASK BIT 4
0202 26 04          BNE     MINPNT
0204 86 20          LDA A   #$20     LOAD A SPACE
0206 20 02          BRA     PRINT1
0208 86 2D   MINPNT LDA A   #$2D     LOAD A MINUS
020A BD E1D1 PRINT1 JSR     OUTEEE   PRINT CHARACTER
020D 08      DPIND  INX
020E E6 00          LDA B   0,X
0210 C4 0F          AND B   #$0F
0212 E7 00          STA B   0,X
0214 C6 2F          LDA B   #$2F
0216 E0 00          SUB B   0,X
0218 D7 21          STA B   $21      STORE DEC. PT  POSITION IND.
021A 08      DIGLOP INX
021B A6 00          LDA A   0,X
```

```
0210 BD E1D1           JSR    OUTEEE    OUTPUT ASCII NUMBER
0220 9C 20             CPX    $20       TIME FOR DEC. PT.?
0222 26 05             BNE    ENDCH1
0224 86 2E             LDA A  #$2E
0226 BD E1D1           JSR    OUTEEE
0229 8C 002B ENDCH1    CPX    #$2B      CHECK FOR LAST DIGIT
022C 26 EC             BNE    DIGLOP    GET NEXT DIGIT
022E CE 02A8           LDX    #CRLF
0231 BD E07E           JSR    PDATA1    PRINT CR/LF
0234 FE A002           LDX    PARADR
0237 7E 0123           JMP    COMAND
023A 96 22   SCINOT    LDA A  $22       SCIENTIFIC NOTATION
023C 84 08             AND A  #$08      LOOK FOR NEGATIVE MANTISSA
023E 26 04             BNE    NEGPNT
0240 86 20             LDA A  #$20      SPACE IF NOT
0242 20 02             BRA    PRINT2
0244 86 2D   NEGPNT    LDA A  #$2D
0246 BD E1D1 PRINT2    JSR    OUTEEE    PRINT SIGN
0249 CE 0023           LDX    #$23
024C 08      NUMLOP    INX
024D A6 00             LDA A  0,X
024F BD E1D1           JSR    OUTEEE
0252 8C 0024           CPX    #$24      LOOK FOR DEC. PT. DIGIT
0255 26 05             BNE    SKIPDP
0257 86 2E             LDA A  #$2E
0259 BD E1D1           JSR    OUTEEE    PRINT DEC. PT.
025C 8C 002B SKIPDP    CPX    #$2B      CHECK FOR LAST DIGIT
025F 26 EB             BNE    NUMLOP
0261 86 45             LDA A  #$45
0263 BD E1D1           JSR    OUTEEE    PRINT AN E
0266 96 22             LDA A  $22       LOAD SIGN BYTE
0268 84 01             AND A  #$01
026A 27 05             BEQ    SKPSGN
026C 86 2D             LDA A  #$2D
026E BD E1D1           JSR    OUTEEE    PRINT A -
0271 96 20   SKPSGN    LDA A  $20
0273 BD E1D1           JSR    OUTEEE    PRINT EXPONENT MSD
0276 96 21             LDA A  $21
0278 BD E1D1           JSR    OUTEEE    PRINT EXPONENT LSD
027B CE 02A8           LDX    #CRLF
027E BD E07E           JSR    PDATA1    PRINT CR/LF
0281 FE A002           LDX    PARADR
0284 7E 0123           JMP    COMAND
0287 0D      CLRSCN    FCB    $0D,$0A,$10,$1A,$00
0288 0A
0289 10
028A 1A
028B 00
028C 53                FCC    'SWTPC 6800 CALC-1 CALCULATOR'
028D 57
028E 54
028F 50
0290 43
0291 20
```

```
0292 36
0293 38
0294 30
0295 30
0296 20
0297 43
0298 41
0299 4C
029A 43
029B 2D
029C 31
029D 20
029E 43
029F 41
02A0 4C
02A1 43
02A2 55
02A3 4C
02A4 41
02A5 54
02A6 4F
02A7 52
02A8 0D       CRLF   FCB    $0D,$0A,$00,$00,$00,$04
02A9 0A
02AA 00
02AB 00
02AC 00
02AD 04
02AE 00       FORMAT FCB    $00
02AF 00       SMDC   FCB    $00
02B0 0D       ERRMSG FCB    $0D,$0A,$00,$00
02B1 0A
02B2 00
02B3 00
02B4 45              FCC     ERROR
02B5 52
02B6 52
02B7 4F
02B8 52
02B9 04              FCB    $04
     E07E     PDATA1 EQU    $E07E
     A002     PARADR EQU    $A002
     E1AC     INEEE  EQU    $E1AC
     E1D1     OUTEEE EQU    $E1D1
A048                 ORG    $A048
A048 0100            FDB    $0100
A002                 ORG    $A002
A002 800C            FDB    $800C
                     END
START  0100
INITAL 0110
COMAND 0123
POINT  012B
OUTINS 0135
```

```
WAIT10 0141
CHRCHK 014C
SKIP75 0153
CONT50 0158
SKIP25 0165
ZERMEM 0174
SETMEM 017B
LOOP1  0183
LODADR 018C
OUTANS 0193
WAIT30 01A1
WAIT3  01AB
OUTDIG 01B5
POINT2 01C5
CONFLG 01C9
OUTCHR 01D0
WAIT70 01D2
WAIT71 01E0
CONT1  01F0
FLOPNT 01FB
MINPNT 0208
PRINT1 020A
DPIND  020D
DIGLOP 021A
ENDCH1 0229
SCINOT 023A
NEGPNT 0244
PRINT2 0246
NUMLOP 024C
SKIPDP 025C
SKPSGN 0271
CLRSCN 0287
CRLF   02AB
FORMAT 02AE
SMDC   02AF
ERRMSG 02B0
PDATA1 E07E
PARADR A002
INEEE  E1AC
OUTEEE E1D1

TOTAL ERRORS 00000
```

RPN-the only language that lets you "speak" with confidence and consistency to a pocket-sized computer calculator.

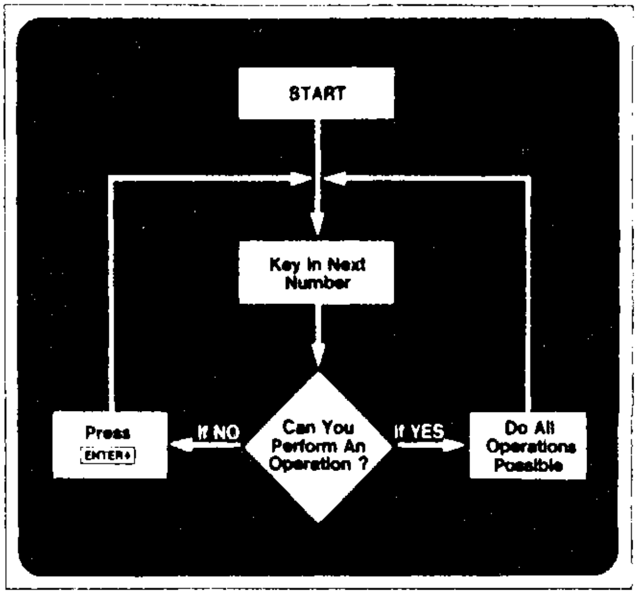
In 1967, Hewlett-Packard embarked on a major new development effort: to design a family of advanced computer calculators powerful enough to solve complex engineering/scientific problems yet simple enough to be used by anyone who works with numbers.

As part of this effort, HP carefully evaluated the strengths and weaknesses of the various languages which an operator might use to communicate with an electronic calculating device. Among those studied were:

- computer languages such as BASIC and FORTRAN,
- various forms of algebraic notation, and
- RPN (*Reverse Polish Notation*), a parenthesis-free but unambiguous language derived from that developed by the Polish mathematician, Jan Lukasiewicz.

As might be expected, each of these languages was found to excel in a particular application. For its biggest programmable desktop calculators, HP selected BASIC. For its other powerful desktop calculators, with less extensive storage capacity, HP chose algebraic notation.

But, given the design constraints of a pocket-sized scientific computer calculator, RPN proved the simplest, most efficient, most consistent way to solve complex mathematical problems.



The RPN method consists of four, easy-to-remember steps. Once learned, it can be applied to almost any mathematical expression.

Only RPN offers these powerful advantages

Compared to alternative logic systems, Hewlett-Packard believes that only RPN—in combination with a 4-register operational memory stack—gives you these powerful advantages.

1. You can always enter your data the same way, i.e., from left to right—the same way you read an equation. Yet, there is no need for a parenthesis key; nor for a complicated "operational hierarchy."
2. You can always proceed through your problem the same way. Once you've entered a number, you ask: "Can I perform an operation?" If yes, you do it. If no, you press [ENTER↑] and key in the next number.
3. You always see all intermediate answers—as they are calculated—so that you can check the progress of your calculation as you go. As important, you can review all numbers stored in the calculator at any time by pressing a few keys. There is no "hidden" data.
4. You don't have to think your problem all the way through beforehand unless the problem is so complex that it may require simultaneous storage of three or more intermediate answers.
5. You can easily recover from errors since all operations are performed sequentially, immediately after pressing the appropriate key.
6. You don't have to write down and re-enter intermediate answers, a real time-saver when working with numbers of eight or nine digits each.
7. You can communicate with your calculator confidently, consistently because you can always proceed the same way.

If all this sounds too good to be true, bear with us—you'll soon get the chance to see for yourself. But first, we need to describe how RPN and the 4-register operational stack operate.

The RPN method—it takes a few minutes to learn but can save years of frustration.

Yes, the RPN method does take some getting used to. But, once you've learned it, you can use the RPN method to solve almost any mathematical expression—confidently, consistently.

There are only four easy-to-follow steps:

1. Starting at the left side of the problem, key in the first or next number.
2. Determine if any operations can be performed. If so, do all operations possible.
3. If not, press [ENTER↑] to save the number for future use.
4. Repeat steps 1 through 3 until your calculation is completed.

A diagram of the RPN method is shown above.

Simple arithmetic, the RPN way.

Just to show how it works, let's try the RPN method on two simple problems (we'll use them again in the comparisons that begin on the next page).

Problem: $3 \times 4 = 12$

RPN solution:

Step	Press	See Displayed
1. Key in first number.	[3]	3
2. Since only one number has been keyed in, no operations are possible. Press [ENTER↑].	[ENTER↑]	3
3. Key in next number.	[4]	4
4. Since both numbers are now in calculator, multiplication can be performed.	[×]	12

Problem: $(3 \times 4) + (5 \times 6) = 42$

RPN solution:

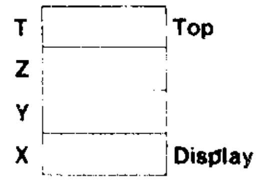
Step	Press	See Displayed
1. Key in first number.	[3]	3
2. No operations possible. Press [ENTER↑].	[ENTER↑]	3
3. Key in second number.	[4]	4
4. Since both numbers are in calculator, first multiplication is possible.	[×]	12
5. Key in next number. (*First intermediate answer will be automatically stored for future use.*)	[5]	5
6. No operations possible. Press [ENTER↑].	[ENTER↑]	5
7. Key in next number.	[6]	6
8. Second multiplication is possible since both numbers are in calculator.	[×]	30
9. Addition is possible since both intermediate answers have been calculated and are stored in 4-register operational stack.	[+]	42

If you've followed us this far, you've noticed two important facts:

1. Both of these problems were solved in the same, consistent manner, using the same simple set of rules.
2. All intermediate answers were displayed as they were calculated, and stored and retrieved as needed to complete the calculation. With RPN and a 4-register operational memory stack, there is almost never a need to write down intermediate answers.

How the operational stack works.

The four registers of HP's exclusive operational stack can be represented by the following diagram.



When a number is keyed in, it goes into the X register for display. Pressing the [ENTER↑] key duplicates the contents of the X register into the Y register and moves all other numbers in the stack up one position.

When an operation key ([+], [−], [×], [÷], [x^y]) is pressed the operation is performed on the numbers in the X and Y registers, and the answer appears in the X register for display. Numbers in the other registers automatically drop one position.

To demonstrate these points, we'll show what happens to the stack as we solve the problem: $(3 \times 4) + (5 \times 6) = 42$.

T									
Z						12	12		
Y		3	3		12	5	5	12	
X	3	3	4	12	5	5	6	30	42
	[3]	[ENTER↑]	[4]	[×]	[5]	[ENTER↑]	[6]	[×]	[+]

As you can see, all numbers are automatically positioned in the stack on a last-in-first-out basis, in the proper order for subsequent use.

Now that we've described how RPN logic operates, we can proceed with our problem-by-problem comparison of this system versus two others used in today's scientific pocket calculators.

We think you will find it interesting.

Calc-1 Instuction Set

Full Name	ASCII Character
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
Decimal point	.
Enter exponent	E
Change sign	Z
Constant PI	P
Set mantissa digit count	M
X exchange M	A
Memory store	G
Memory recall	H
Inverse mode	I
Enter	sp or cr
Toggle mode	>
Roll stack	O
Sine X	S
Cosine X	C
Tangent X	T
Error clear	Y
Radians to degrees	F
Degrees to radians	D
Master clear	cntrl X
X exchange Y	X
E to X	W
Ten to X	U
Square	Q
Square root	V
Natural log of X	N
Base 10 log of X	B
One divided by X	R
Y to X	^
Plus	+
Minus	-
Times	*
Divide	/

Table I

MM57109 Instruction Description Table (Continued) (* Indicates 2-word instruction)

CLASS	SUBCLASS	MNEMONIC*	OCTAL OP CODE	FULL NAME	DESCRIPTION
Branch	Count	IBNZ	31	Increment memory and branch if M ≠ 0	M + 1 → M. If M = 0, skip second instruction word. Otherwise, branch to address specified by second instruction word.
		DBNZ	32	Decrement memory and branch if M ≠ 0	M − 1 → M. If M = 0, skip second instruction word. Otherwise, branch to address specified by second instruction word.
I/O	Multi-digit	IN*	27	Multidigit input to X	The processor supplies a 4-bit digit address (DA4–DA1) accompanied by a digit address strobe ($\overline{DAS}$) for each digit to be input. The high order address for the number to be input would typically come from the second instruction word. The digit is input on D4–D1, using ISEL = 0 to select digit data instead of instructions. The number of digits to be input depends on the calculation mode (scientific notation or floating point) and the mantissa digit count (See Data Formats and Instruction Timing). Data to be input is stored in X and the stack is pushed (X → Y → Z → T). At the conclusion of the input, DA4–DA1 = 0.
		OUT*	26	Multidigit output from X	Addressing and number of digits is identical to IN instruction. Each time a new digit address is supplied, the processor places the digit to be output on DO4–DO1 and pulses the R/$\overline{W}$ line active low. At the conclusion of output, DO4–DO1 = 0 and DA4–DA1 = 0.
I/O	Single-digit	AIN	16	Asynchronous Input	A single digit is read into the processor on D4–D1. ISEL = 0 is used by external hardware to select the digit instead of instruction. It will not read the digit until $\overline{ADR}$ = 0 (ISEL = 0 selects $\overline{ADR}$ instead of I_5), indicating data valid. F2 is pulsed active low to acknowledge data just read.
I/O	Flags	SF1	47	Set Flag 1	Set F1 high, i.e. F1 = 1.
		PF1	50	Pulse Flag 1	F1 is pulsed active high. If F1 is already high, this results in it being set low.
		SF2	51	Set Flag 2	Set F2 high, i.e. F2 = 1.
		PF2	52	Pulse Flag 2	F2 is pulsed active high. If F2 is already high, this results in it being set low.
		PRW1	75	Pulse R/$\overline{W}$ 1	Generates R/$\overline{W}$ active low pulse which may be used as a strobe or to clock extra instruction bits into a flip-flop or register.
		PRW2	76	Pulse R/$\overline{W}$ 2	Identical to PRW1 instruction. Advantage may be taken of the fact that the last 2 bits of the PRW1 op code are 10 and the last 2 bits of the PRW2 op code are 01. Either of these bits can be clocked into a flip-flop using the R/$\overline{W}$ pulse.
Mode Control		TOGM	42	Toggle Mode	Change mode from floating point to scientific notation or vice-versa, depending on present mode. The mode affects only the IN and OUT instructions. Internal calculations are always in 8-digit scientific notation.
		SMDC*	30	Set Mantissa Digit Count	Mantissa digit count is set to the contents of the second instruction word (=1 to 8).
		INV	40	Inverse Mode	Set inverse mode for trig or memory function instruction that will immediately follow. Inverse mode is for next instruction only.

Table I

MM57109 Instruction Description Table (* Indicates 2-word instruction)

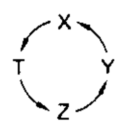
CLASS	SUBCLASS	MNEMONIC*	OCTAL OP CODE	FULL NAME	DESCRIPTION
Digit Entry		[redacted]	00 01 02 03 04 05 06 07 10 11	[redacted]	Mantissa or exponent digits. On first digit (d) the following occurs: $Z \rightarrow T$ $Y \rightarrow Z$ $X \rightarrow Y$ $d \rightarrow X$ See description of number entry on page 11.
			12	[redacted]	Digits that follow will be mantissa fraction.
			13		Digits that follow will be exponent.
			14		Change sign of exponent or mantissa. Xm = X mantissa Xe = X exponent CS causes $-Xm \rightarrow Xm$ or $-Xe \rightarrow Xe$ depending on whether or not an EE instruction was executed after last number entry initiation.
		[redacted]	15	[redacted]	$3.1415927 \rightarrow X$, stack not pushed.
			41		Terminates digit entry and pushes the stack. The argument entered will be in X and Y. $Z \rightarrow T$ $Y \rightarrow Z$ $X \rightarrow Y$
		NOP	77	No Operation	Do nothing instruction that will terminate digit entry.
		HALT	17	Halt	External hardware detects HALT op code and generates HOLD = 1. Processor waits for HOLD = 0 before continuing. HALT acts as a NOP and may be inserted between digit entry instructions since it does not terminate digit entry.
Move		ROLL	43	[redacted]	Roll Stack. 
		POP	56	Pop	Pop Stack. $Y \rightarrow X$ $Z \rightarrow Y$ $T \rightarrow Z$ $0 \rightarrow T$
		XEY	60	[redacted]	Exchange X and Y. $X \leftrightarrow Y$
		XEM	33	[redacted]	Exchange X with memory. $X \leftrightarrow M$
		MS	34	[redacted]	Store X in Memory. $X \rightarrow M$
		MR	35	[redacted]	Recall Memory into X. $M \rightarrow X$
		LSH	36	Left Shift Xm	X mantissa is left shifted while leaving decimal point in same position. Former most significant digit is saved in link digit. Least significant digit is zero.
		RSH	37	Right Shift Xm	X mantissa is right shifted while leaving decimal point in same position. Link digit, which is normally zero except after a left shift, is shifted into the most significant digit. Least significant digit is lost.

Table I

MM57109 Instruction Description Table (Continued) (* Indicates 2-word instruction)

CLASS	SUBCLASS	MNEMONIC*	OCTAL OP CODE	FULL NAME	DESCRIPTION
Math	F (X,Y)	+	71	[redacted]	Add X to Y. X + Y → X. On +, −, x, / and YX instructions, stack is popped as follows: Z → Y T → Z 0 → T Former X, Y are lost.
		−	72	[redacted]	Subtract X from Y. Y − X → X
		x	73	[redacted]	Multiply X times Y. Y x X → X
		/	74	[redacted]	Divide X into Y. Y ÷ X → X
		YX	70	[redacted]	Raise Y to X power. Y^X → X
	F (X,M)	INV +*	40, 71	Memory Plus	Add X to memory. M + X → M On INV +, −, x and / instructions, X, Y, Z, and T are unchanged.
		INV −*	40, 72	Memory Minus	Subtract X from memory. M − X → M
		INV x*	40, 73	Memory Times	Multiply X times memory. M x X → M
		INV /*	40, 74	Memory Divide	Divide X into memory. M ÷ X → M
	F (X) Math	1/X	67	[redacted]	1 ÷ X → X. On all F (X) math instructions Y, Z, T and M are unchanged and previous X is lost.
		SQRT	64	[redacted]	$\sqrt{X}$ → X
		SQ	63	[redacted]	X^2 → X
		10X	62	[redacted]	10^X → X
		EX	61	[redacted]	e^X → X
		LN	65	[redacted]	ln X → X
		LOG	66	[redacted]	log X → X
	F (X) Trig	SIN	44	[redacted]	SIN(X) → X. On all F(X) trig functions, Y, Z, T, and M are unchanged and the previous X is lost.
		COS	45	[redacted]	COS(X) → X
		TAN	46	[redacted]	TAN(X) → X
		INV SIN*	40, 44	Inverse sine X	$SIN^{-1}(X)$ → X
		INV COS*	40, 45	Inverse cosine X	$COS^{-1}(X)$ → X
		INV TAN*	40, 46	Inverse tan X	$TAN^{-1}(X)$ → X
		DTR	55	[redacted]	Convert X from degrees to radians.
		RTD	54	[redacted]	Convert X from radians to degrees.
Clear		MCLR	57	[redacted]	Clear all internal registers and memory; initialize I/O control signals, MDC = 8, MODE = floating point. (See initialization.)
		ECLR	53	[redacted]	0 → Error flag
Branch	Test	JMP*	25	Jump	Unconditional branch to address specified by second instruction word. On all branch instructions, second word contains branch address to be loaded into external PC.
		TJC*	20	Test jump condition	Branch to address specified by second instruction word if JC (I_6) is true (=1). Otherwise, skip over second word.
		TERR*	24	Test error	Branch to address specified by second instruction word if error flag is true (= 1). Otherwise, skip over second word. May be used for detecting specific errors as opposed to using the automatic error recovery scheme dealt with in the section on Error Control.
		TX = 0*	21	Test X = 0	Branch to address specified by second instruction word if X = 0. Otherwise, skip over second word.
		TXF*	23	Test |X| < 1	Branch to address specified by second instruction word if |X| < 1. Otherwise, skip over second word. (i.e. branch if X is a fraction.)
		TXLT0*	22	Test X < 0	Branch to address specified by second instruction word if X < 0. Otherwise, skip over second word.

Table II

MM57109 Instruction Summary Table (* = 2-word instruction)

$I_4 - I_1$	$I_6 I_5$			
	0	1	2	3
0	[redacted]	TJC*	[redacted]	[redacted]
1	[redacted]	TX=0*	[redacted]	[redacted]
2	[redacted]	TXLT0*	[redacted]	[redacted]
3	[redacted]	TXF*	[redacted]	[redacted]
4	[redacted]	TERR*	[redacted]	[redacted]
5	[redacted]	JMP*	[redacted]	[redacted]
6	[redacted]	OUT*	[redacted]	[redacted]
7	[redacted]	IN*	SF1	[redacted]
8	[redacted]	[redacted]	PF1	[redacted]
9	[redacted]	IBNZ*	SF2	[redacted] (M+)
A	[redacted]	DBNZ*	PF2	[redacted] (M−)
B	[redacted]	[redacted]	[redacted]	[redacted] (M×)
C	[redacted]	[redacted]	[redacted]	[redacted] (M÷)
D	[redacted]	[redacted]	[redacted]	PRW1
E	AIN	LSH	POP	PRW2
F	HALT	RSH	[redacted]	NOP

SET memory locations A002-A003 to 800C! prior to running program.

Table III - CALC-1 Instruction to ASCII Character Lookup Table

FULL NAME	HEX OP CODE	MNEMONIC	ASCII CHARACTER
[redacted]	00	∅∅	∅
[redacted]	01	∅1	1
[redacted]	02	∅2	2
[redacted]	03	∅3	3
[redacted]	04	∅4	4
[redacted]	05	∅5	5
[redacted]	06	∅6	6
[redacted]	07	∅7	7
[redacted]	08	∅8	8
[redacted]	09	∅9	9
[redacted]	0A	DP	.
[redacted]	0B	EE	E
[redacted]	0C	CS	Z
[redacted]	0D	PI	P
Asynchronous Input	0E	AIN	
Halt	0F	HALT	
Test Jump	1∅	TJC	
Test X=∅	11	TX=0	
Test X<∅	12	TXLTO	
Test l X l<l	13	TXF	
Test Error	14	TERR	
Jump	15	JMP	
Multidigit Out	16	OUT	
Multidigit In	17	IN	
[redacted]	18	SMDC	M
Inc & Branch if M≠∅	19	IBNZ	
Dec & Branch if M=∅	1A	DBNZ	
[redacted]	1B	XEM	A
[redacted]	1C	MS	G
[redacted]	1D	MR	H
Left shift Xm	1E	LSH	
Right shift Xm	1F	RSH	
[redacted]	2∅	INV	I
[redacted]	21	EN	space or Carriage Return
[redacted]	22	TOGM	>
[redacted]	23	ROLL	O
[redacted]	24	SIN	S
[redacted]	25	COS	C
[redacted]	26	TAN	T
Set Flag 1	27	SF1	
Pulse Flag 1	28	PF1	
Set Flag 2	29	SF2	
Pulse Flag 2	2A	PF2	
[redacted]	2B	ECLR	Y
[redacted]	2C	RTD	F
[redacted]	2D	DTR	D
Pop	2E	POP	
[redacted]	2F	MCLR	Cntrl X

Table III - CALC-1 Instruction to ASCII Character Lookup Table

NAME	HEX OP CODE	MNEMONIC	ASCII CHARACTER
[redacted]	3Ø	XEY	X
[redacted]	31	EX	W
[redacted]	32	1QX	U
[redacted]	33	SQ	Q
[redacted]	34	SQRT	V
[redacted]	35	LN	N
[redacted]	36	LOG	B
[redacted]	37	1/X	R
[redacted]	38	YX	^
[redacted]	39	+	+
[redacted]	3A	-	-
[redacted]	3B	X	*
[redacted]	3C	/	/
Pulse R/W 1	3D	PRW1	
Pulse R/W 2	3E	PRW2	
No Operation	3F	NOP	

Table IV - Floating Point Mode OUT data storage

Memory Location	DP POS	D7	D6	D5	D4	D3	D2	D1	D0
2Ø		Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
21		Ø	Ø	1	Ø	Ø	Ø	Ø	Ø
22		Ø	Ø	1	1	Sm	Ø	Ø	Ø
23		Ø	Ø	1	1	Dp	POS		
24	ØB	Ø	Ø	1	1	BCD	digit(left most)		
25	ØA	Ø	Ø	1	1	BCD	digit		
26	Ø9	Ø	Ø	1	1	BCD	digit		
27	Ø8	Ø	Ø	1	1	BCD	digit		
28	Ø7	Ø	Ø	1	1	BCD	digit		
29	Ø6	Ø	Ø	1	1	BCD	digit		
2A	Ø5	Ø	Ø	1	1	BCD	digit		
2B	Ø4	Ø	Ø	1	1	BCD	digit(right most)		

Table IV - Scientific Mode OUT data storage

Memory Location	D7	D6	D5	D4	D3	D2	D1	D0
20	Ø	Ø	1	1	Most signif. exp. digit			
21	Ø	Ø	1	1	Least signif. exp. digit			
22	Ø	Ø	1	1	Sm	Ø	Ø	Se
23	NOT USED							
24	Ø	Ø	1	1	BCD	digit (left most)		
25	Ø	Ø	1	1	BCD	digit		
26	Ø	Ø	1	1	BCD	digit		
27	Ø	Ø	1	1	BCD	digit		
28	Ø	Ø	1	1	BCD	digit		
29	Ø	Ø	1	1	BCD	digit		
2A	Ø	Ø	1	1	BCD	digit		
2B	Ø	Ø	1	1	BCD	digit (left most)		

Notes:

1) If the Mantissa Digit Count (set by SMDC instruction, initially 8) is less than 8, the unused digit memory locations will be filled with ASCII spaces ($2Ø_{16}$)

2) Sm is the sign of the mantissa. Ø = positive 1= negative

3) Se is the sign of the exponent Ø = positive 1= negative

4) DP POS is the decimal point position. The decimal point should follow the digit whose address is stored in memory location 24 when in the Scientific mode. In the Floating Point mode AND the data in memory location 23 with ØF and subtract the result from 2F and OR this with 2Ø. The decimal point should follow the digit whose address is given by the result.

Table V - ASCII to CALCULATOR INSTRUCTION LOOKUP TABLE

LSB	MSB	0	1	2	3	4	5	6	7
0		OF	OF	21	00	OF	OD	OF	OD
1		OF	OF	OF	01	1B	33	OF	33
2		OF	OF	OF	02	36	37	36	37
3		OF	OF	OF	03	25	24	25	24
4		OF	OF	OF	04	2D	26	2D	26
5		OF	OF	OF	05	OB	32	OB	32
6		OF	OF	OF	06	2C	34	2C	34
7		OF	OF	OF	07	1C	31	1C	31
8		OF	2F	OF	08	1D	30	1D	30
9		OF	OF	OF	09	20	2B	20	2B
A		OF	OF	3B	OF	OF	OC	OF	OC
B		OF	OF	39	OF	OF	OF	OF	OF
C		OF	OF	OF	OF	OF	OF	OF	OF
D		21	OF	3A	OF	18	OF	18	OF
E		OF	OF	OA	22	35	38	35	OF
F		OF	OF	3C	OF	23	OF	23	OF

Example: An ASCII P is a hex 50 which points in the table to a OD which is the constant PI instruction for the calculator chip

TABLE VI- ERROR CONDITIONS

The ERROR flag on the calculator chip is set when:

1) LN X when X ≤ 0 LOG X when X ≤ 0

2) Any result $< 10^{-99}$ Any result $\geq 10^{99}$

3) TAN 90°, 270°, 450°, etc.

4) SIN X, Cos X, TAN X when $|X| \geq 9000^\circ$

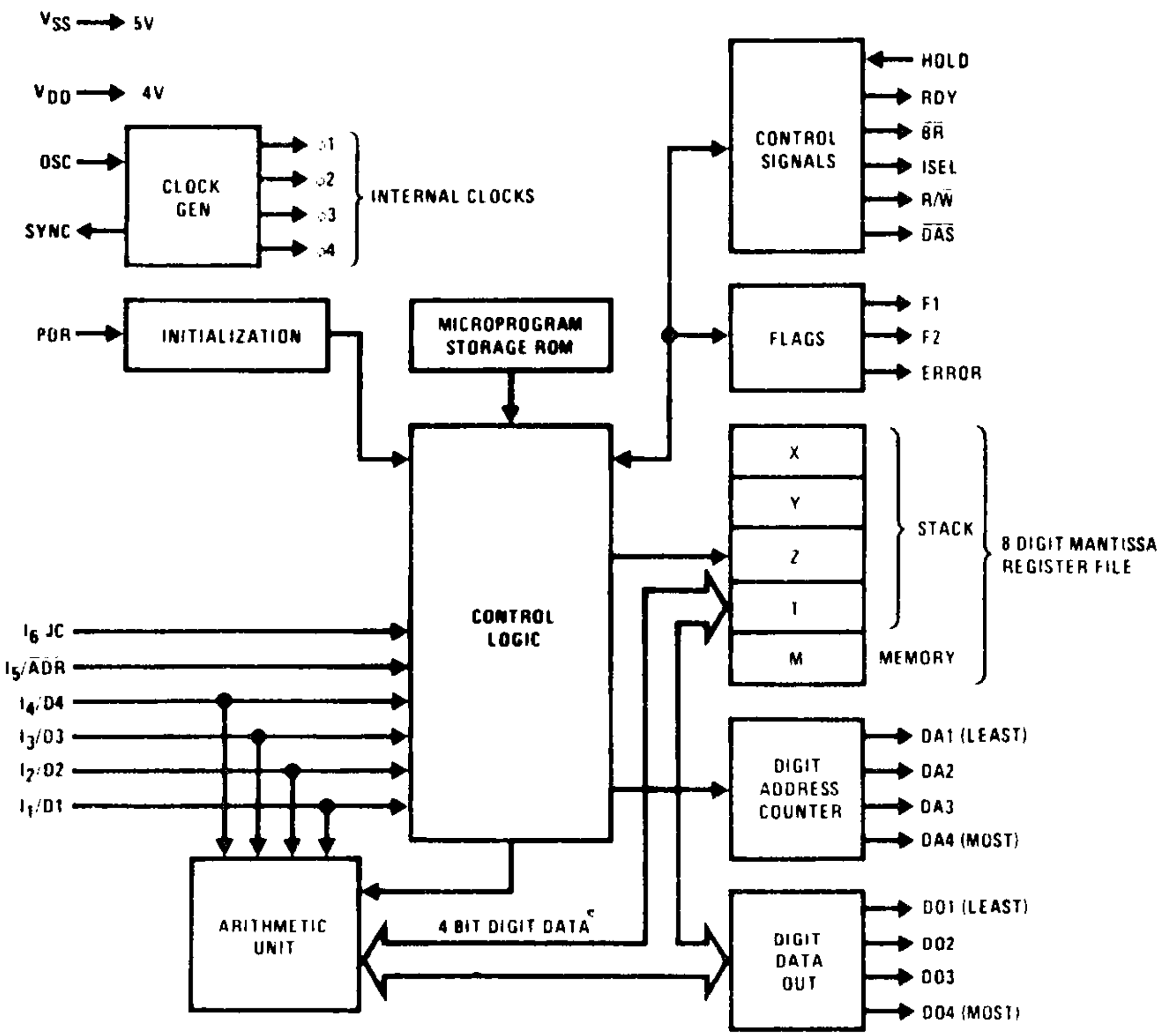
5) SIN^{-1} X, COS^{-1} X when $|X| > 1$ or $|X| \leq 10^{-50}$

6) SQRT X when $X < 0$

7) dividing by 0

8) Outputting a number in floating point mode if the number of mantissa digits to the left of the decimal point is greater than the mantissa digit count.

Figure I



VSS 5V
VDD 4V
OSC
SYNC
CLOCK GEN
INTERNAL CLOCKS
POR
INITIALIZATION
MICROPROGRAM STORAGE ROM
CONTROL SIGNALS
HOLD
RDY
BR
ISEL
R/W
DAS
FLAGS
F1
F2
ERROR
X
Y
Z
T
M
STACK
MEMORY
8 DIGIT MANTISSA REGISTER FILE
CONTROL LOGIC
I6 JC
I5/ADR
I4/D4
I3/D3
I2/D2
I1/D1
DIGIT ADDRESS COUNTER
DA1 (LEAST)
DA2
DA3
DA4 (MOST)
ARITHMETIC UNIT
4 BIT DIGIT DATA
DIGIT DATA OUT
DO1 (LEAST)
DO2
DO3
DO4 (MOST)

ASCII to Hexadecimal Conversion Table

LSB \ MSB	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	END	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	–	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL