MICROCRUNCH: An Ultra-fast Arithmetic Computing System

Part 1

Extremely fast floating point processing can be attained by coupling an INTEL 8231 arithmetic processing unit to the OSI Superboard II and using a partial compiler to generate machine code representations of mathematical equations and loops written in BASIC.

John E. Hart
Dept. of Astrogeophysics
University of Colorado
Boulder, Colorado 80309

An editorial in *BYTE* magazine (*BYTE*, vol. 5, number 10, Oct. 1980) quoted a survey that indicated that 40% of the readers of that microcomputer magazine were scientists or engineers. Obviously a very large number of small system users got into microcomputing because they hoped to use their machines for mathematical problems occurring in these fields. Although many applications of 6502 processors have been in tasks that do not require sophisticated mathematical manipulation (like graphics, games, word processing, etc.) there is certainly a host of interesting and/or practical problems that can be approached via numerical analysis on a microcomputer. These problems span the entire spectrum of mathematical modeling, from ecosystems to weather systems, from circuit analysis to support calculations in data analysis.

Such applications are only limited by the product of the floating point throughput (or speed) of the microprocessor and associated software, and the patience of the operator to wait around for the answer. It is often most profitable and convenient to approach mathematical problems in an interactive mode, where, for example, a problem depending on a certain parameter is iterated to an end point. The result is then inspected by the operator, the parameter varied, and the solution repeated, until the desired answer is obtained. Such a scheme would be fruitful if the iteration time is fairly short. If you have to wait half an hour between answers it can be very frustrating. The iteration time is, of course, proportional to the length of the mathematical problem, in terms of the total number of floating point operations per iteration, divided by the effective computing speed of the machine being used. Unfortunately when it comes to floating point number crunching, microcomputers can be annoyingly slow. The purpose of this series of two articles is to describe a 6502-based system called MICROCRUNCH that is extremely fast at floating point mathematical number crunching.

The system consists of an OSI Superboard II with the 610 board memory expansion, interfaced to an INTEL 8231 math chip, which will be discussed later, in detail. This article describes the hardware necessary to accomplish this interface.

True number crunching speed is only possible if such a math chip is treated as a co-processor in the sense that floating point operations executed by the 8231 are done asynchronously as the 6502 is preparing for the next operation. Thus a special BASIC compiler that converts higher order statements into optimal 6502 machine code is a must if the potential for fast execution inherent in the 8231 is to be realized. Part 2 of this series will describe the software necessary to do this. We start by indicating what kind of speeds can be attained with the MICROCRUNCH system.

Computing speed for mathematical applications is usually measured in terms of megaflops (Mflops); or millions of floating point operations (+, −, *, /) that a computer, plus associated support software can execute per second. Obviously no one expects a micro to compete with a 32-bit mainframe designed specifically to do scientific computing, but it is interesting to compare a few typical systems in this regard and to note how well a little 8-bit micro can perform. Computing speed can be crudely estimated by running the following simple benchmark program on several machines.

```
A=1.00013
X=1
FOR I=1TO40000
X=X * A
NEXT I
PRINT X
STOP
```

From this, one gets a pretty good idea of the Mflop capability of a machine, since usually, the overhead for the FOR loop part of this little program is small compared to the time it takes to look up the variables X and A, and to perform the multiplication. I have tried this little loop on a variety of computers, some of which used a FORTRAN version. The results are shown in table 1.

There are several conclusions that can be made from this table, such as:

1. Traditional 6502 or Z-80 machines with BASIC interpreters are quite slow, doing about 100 to 200 flops per second. A calculation with 10,000 flops would take a couple of minutes, which is too slow for comfortable interactive computing.

2. The use of a compiler (Pascal or FORTRAN) on the straight 6502 machines only helps by a factor of 2 or so in speed. Although for a compiler the variable fetch and line decode times go way down, the time for actual addition, division, etc., in floating point stays the same.

3. Increasing the computer clock helps in direct proportion to the clock increase. At most, this might gain a factor of 4 if the typical 6502 micro can be made to run at 4 MHz.

4. Floating point chips without compilers are almost useless.

5. The optimal 8-bit system described here outperforms many standard minicomputers, at a fraction of their cost.

6. If you want personal number crunching in excess of around .01 Mflops (10^4 floating point operations per second), be prepared to spend a large amount of money.

Assuming the reader is interested in attaining floating point throughput in excess of 50 times the typical micro performance, we proceed to outline the MICROCRUNCH hardware, including circuits, a layout, and a parts list.

The Hardware

The physical system is shown in figure 1. The basic computer is the OSI Superboard II. It has been connected to a fully populated OSI 610 memory board. Thus the starting element is essentially a 6502 computer with 32K of RAM. The 610 board has an expansion plug that contains buffered data, address, phase two, read/write, and interrupt lines. This is connected to the arithmetic processor board (APB) whose circuit is given below. This APB board could be connected to any 6502 system that has available the same buffered lines as on the OSI 610. These are given in more detail in table 2.

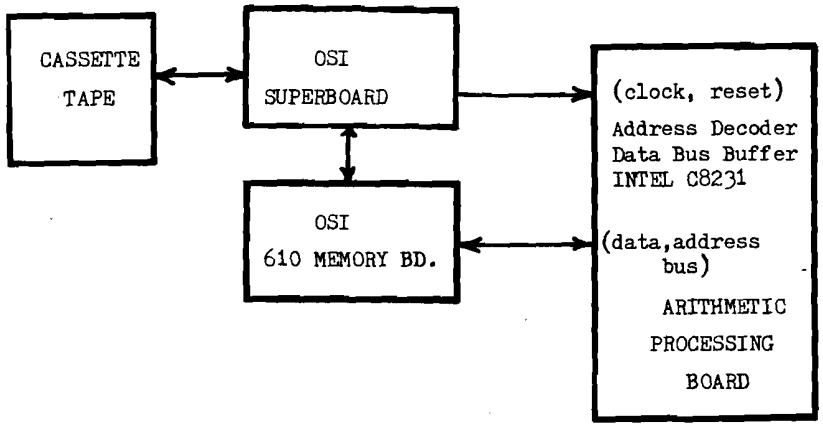
Thus, in principal, the APB circuit can be used on a variety of machines (AIM, Apple, etc.) provided the address assigned to the arithmetic processor does not conflict with the memory map of the host computer. Because the compiler described in part 2 of this article uses up 20K of memory, and the upper 12K of this system is needed for source and object code storage, there is not much room left for a disk operating system. So, I use magnetic tape as a bulk storage medium. This would not be necessary if a machine with 48K of RAM were employed. However, the tape storage system I use is almost as fast as disk, so there is little performance loss here (see "An Ultra-Fast Tape Storage System," J.E. Hart, MICRO, November 1980, 30:11).

In addition I have jumped the fundamental clock on my Superboard up to 2 MHz as described by J.R. Swindell ("The Great Superboard Speedup," MICRO, February 1980, 21:30). The timing for the MICROCRUNCH system in table 1 was with a 2 MHz clock. For 1 MHz, the Mflop rate is .007. The tape

***Table 1:* Approximate Megaflop Rates for Several Computing Systems**

Computer	Language	Mflop (million flops/sec)
TRS80 model I (Z-80)	BASIC interpreter	.00012
TRS80 model II (Z-80)	" "	.00026
INTERCOLOR (Z-80)	" "	.00014
APPLE II (6502, 1 MHz)	" "	.00019
APPLE II	Pascal compiler	.00034
APPLE II w/AMD9511 floating point board (Calif. Digital)	APPLEFAST interp.	.00026
OSI Superboard II (1 MHz)	BASIC interpreter	.00022
OSI Superboard II (2 MHz)	" "	.00044
PDP1103 w/Hdw. floating point board (DEC)	FORTRAN	.004
*MICROCRUNCH (OSI 2 MHz + INTEL 8231)	BASIC compiler	.011
PDP 1134	FORTRAN	.04 approx.
VAX 11/750 (DEC)		.4 "
CDC 7600	"	4-6 "
CRAY I	"	60 "

***Figure 1:* MICROCRUNCH Hardware**



baud rate and clock modifications are not necessary for successful operation of the APB, but they are useful changes that increase performance and convenience.

The APB part of the system consists of an address decoder, a data bus buffer, a read/write/command/data decoder and the INTEL 8231 arithmetic processing unit. In order to understand the circuits that follow it is necessary to give a brief description of the 8231.

Anyone getting into this project should obtain the 8231 manual from a local INTEL representative, since only a brief sketch of the processor can be given here. When ordering this part, be sure to get the C8231, since this will run at 4 MHz and the regular 8231 will not. The 8231 has the following features of interest:

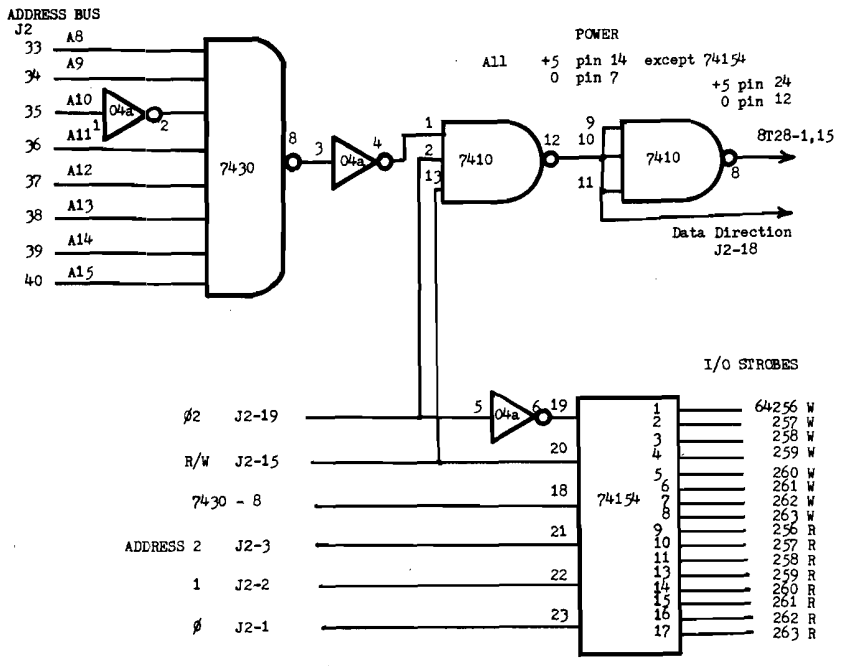
1. An operand stack that stores 4 floating point numbers with 6½ decimal digit precision and a range of about $10^{\pm 20}$. Each floating point number is represented by 4 bytes: 1 for the exponent and 3 for the mantissa. The floating point format will be discussed in part 2. It is, unfortunately, not the same as that used by Microsoft BASIC.

2. A 1-byte status register that can be read into the 6502. This status register contains a busy bit that in-

Table 2: Connector J2a on Arithmetic Board

Pin	Function
1	buffered address bus bit 0
2	" " " " 1
3	" " " " 2
4	" " " " 3
10	buffered data bus bit 0
	" " " " 1
11	" " " " 2
12	" " " " 3
13	
15	buffered read/write (read to 6502 if high)
18	data direction (enable read to 6502 if low)
19	buffered phase 2 clock
28	buffered data bus bit 4
	5
29	" " " " 6
30	" " " " 7
31	
33	buffered address bus bit 8
	9
34	" " " " 10
35	" " " " 11
36	" " " " 12
37	" " " " 13
38	" " " " 14
39	" " " " 15
40	" " " "

***Figure 2:* Address Decoder Circuit**



dicates whether a previously initiated floating point command is still in progress, and an error field that indicates if the previously completed command resulted in an error (overflow, underflow, divide by zero, improper function argument like square root of a negative number, etc.).

3. A 1-byte command register that is written into by the 6502. This initiates a floating point operation on the operand(s) that are stored on the stack in the 8231. These operations include +, −, *, / and a host of transcendental functions like SIN, COS, ARCTAN, etc. (See the manual for a complete description of these.) Suffice it to say that just about any problem you could have done with Microsoft BASIC you can do within the 8231, only much faster. The result of a calculation or operation appears on the top of the stack and can hence be read as a four-byte block transfer back into memory, under control of the 6502. These manipulations and some quirks of the floating stack are discussed in part 2, since they have more to do with software than hardware.

The scenario that emerges is as follows: A mathematical program written in BASIC is compiled by the 6502. There the object code, so generated, causes appropriate 4-byte transfers in and out of the APB, of floating point variables appearing in the mathematical expressions that were compiled. The 6502 also sends operation commands at the appropriate times and checks for errors after an operation is completed. Thus the main task of the hardware is to allow the 6502 to transfer data in and out of the 8231 stack, command, and status registers. Thus, we are really concerned with a fast I/O problem.

Readers of the 8231 manual will note that it also does fixed point arithmetic (16- or 32-bit). None of these functions are used in the MICROCRUNCH system, but software could be written to use these if needed.

Circuit Description

Described below is the circuit for the APB and its interconnections to the 610 board. The components for this board, all bought retail, cost about $340, with $270 going for the INTEL C8231. In addition, the 8231 uses 12V DC so a regulated supply of some sort (low current, 100 mA is fine) is needed. It should be mentioned here that the 8231 is identical in architecture and pin-outs to the older AMD 9511. The latter chips are a little cheaper ($195), but are designed to

run at 2 MHz instead of 4 MHz. I went with the INTEL because the speed increase seemed worth the extra money.

The main interface with the 610 board is via its connector J2. This 40-pin connector is linked to a similar 40-pin IC socket-type connector on the APB with a ribbon cable. Table 2 shows the lines available on J2 that are used on the APB.

In addition to this interface, an additional connector J3 must be used to supply the following signals from the Superboard itself. In my unit this is a 14-pin IC socket connected by a ribbon cable to a similar socket set in one of the prototype holes in the Superboard.

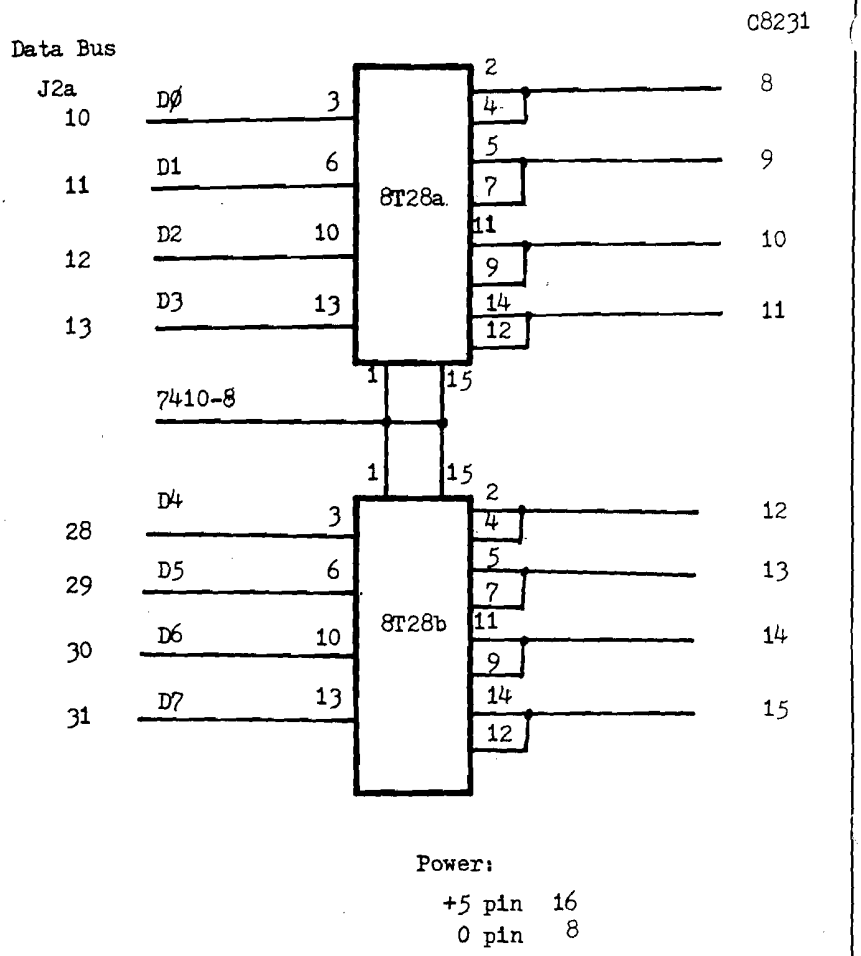
J3 - 1	4 MHz clock (SBD II, U30 - 2)
J3 - 2	Ground
J3 - 3	BRK line (low = reset) (SBD U8 - 40)

The APB circuit will work with any 6502 computer that supplies the I/O connections as described above.

Figure 2 shows the address decoder circuit. Address lines 8 through 15 are fed into an 8 input nand gate, and line 10 is inverted. The output of this gate will go low whenever the address high byte goes to $FB. This is the basic block address for the APB. The output of this gate is fed to one enable input of a 74154 4-to-16-line demultiplexer, and to a set of inverters and gates whose purpose is to generate a data direction pulse in phase with the 02 clock pulse. The outputs of the 74154 are a set of strobes that go low in phase with 02 whenever address FB is selected. Only one strobe is fired, depending, as well, on the R/W, A0, A1, and A2 lines. These strobes can be used to select various I/O devices, 16 in all. For the APB we shall use only 5 of these lines, so the others can be used for future expansion (A-D, D-A, etc.). The data direction pulse does two things. It informs the data buffers on the 610 board when data is going to be fed back to the 6502 (J2-18, low = read) and after inversion, chip 7410-8 does the same for the data buffers just ahead of the 8231.

Figure 3 shows the interconnections for the two on-board 8T28 tri-state buffers needed to drive the cable connecting the APB to the 610 board.

***Figure 3:* Bus Buffer**



Finally, figure 4 shows the interconnections between the strobe lines from the address decoder and the 8231. During a write operation pin 1 of the 7402 NOR gate will go low. This signal is inverted and fed through another part of the 7402 quad NOR gate to give a low CHIP-SELECT pulse. The 8231 timing requirements indicate that the active low WRITE pulse must be shorter than the CHIP-SELECT input so the WRITE strobe is shortened by feeding into a 74123 one-shot. If an operand is being written onto the 8231 floating stack, pin 21 must go low. This is accomplished by sending the inverted WRITE OPERAND strobe to 7402-8. The resulting inverted OR pulse then becomes the appropriate C/$\overline{D}$ line.

A read of either the operand stack or the status register is preceded by a READ INITIATION strobe. For example, a READ STATUS START strobe (e.g. LDA ABS FB00) sets flip-flop 74LS76A. The output of this flip-flop goes high and causes the CHIP-SELECT line (APU-18) and the READ line (APU-20) both to go low. The 8231 then proceeds to send the status register contents to its internal data bus buffer. This takes several clock cycles (like an EPROM), so data is not entered into the 6502 accumulator until a READ ENTER strobe is fired. That is, flip-flop A stays set until an LDA-ABS FB06 instruction is executed. Then strobe line 74154-16 goes low terminating the read by resetting the flip-flop on its rising edge.

Typically, then, two consecutive LDA's are used to read from the 8231. Data is read by LDA-ABS FB01, LDA-ABS FB06. The only difference between this and a status read is that flip-flop B sets the C/$\overline{D}$ line low (via 7402-10) in addition to pulling the CHIP-SELECT and READ lines low. The double LDA read cycle required by this circuit is slightly (20%) less efficient in time than

using the 6502 ready line in a pause circuit. Unfortunately, in the Superboard this line is tied to ground. However, during long mathematical manipulations one is almost always writing data and commands into the APU, reading only at the end of a string of operations. Therefore, this lost time becomes insignificant.

The 4 MHz clock and the reset pulses are connected as indicated.

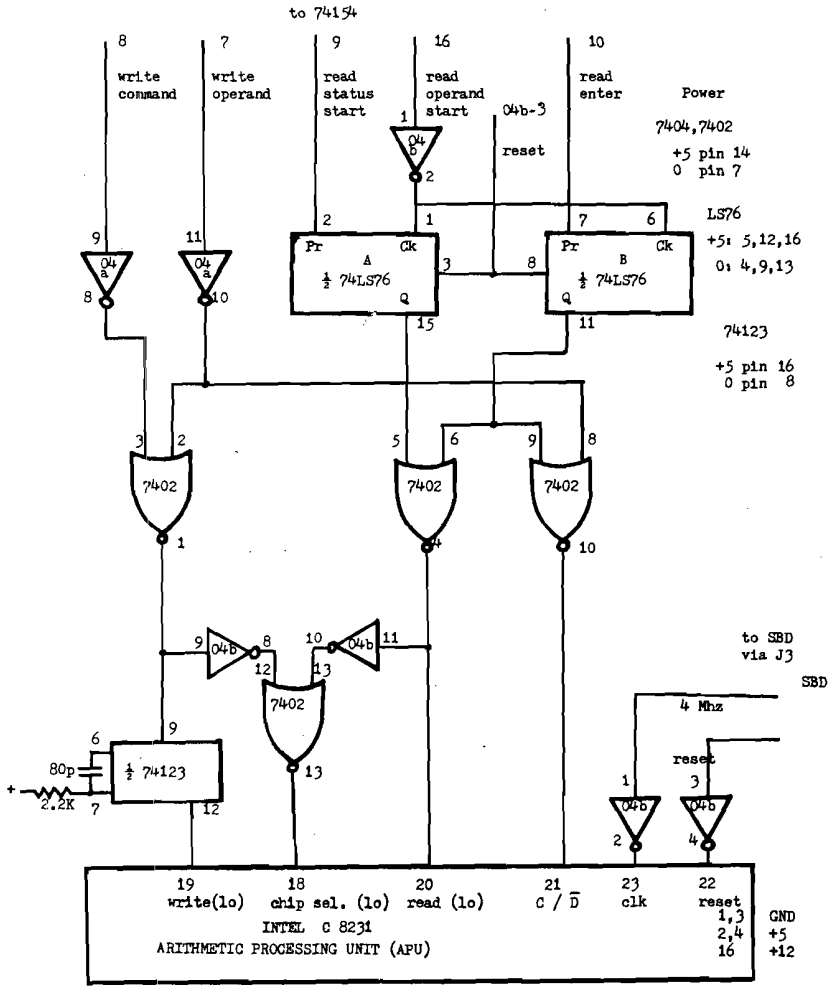
Table 3 gives the APB addresses and typical commands used to communicate with it. For machines other than OSI, these addresses may fall in already assigned areas of the memory map. If so, the base address FB can easily be changed by altering the inputs to the 7430 address gate. For example, if the inverter on line 10 is not used, the high part of the APB address will be $FF. If this is done, however, some straightforward address changes will need to be made in the software presented here and in part 2.

Figure 5 gives a typical layout for the APB. One first installs the wire-wrap sockets (assuming the board will be wire-wrapped, not soldered), and routes the power lines. Install .01 mfd bypass capacitors on each chip between the +5 volt line and ground. After wrapping the preceding circuits, the board should be tested using some simple programs presented below. The basic questions are, can you get operands in and out of the unit, and can you command it to execute operations?

***Table 3:* Arithmetic Board Addresses and Machine Code Access Statements**

Address		Function	Machine Code	
64256	FB00	APU READ STATUS start	LDA-ABS	FB00
64257	FB01	APU OPERAND READ start	LDA-ABS	FB01
64262	FB06	APU WRITE OPERAND	STA-ABS	FB06
64262	FB06	APU READ DATA (status or operand, as determined by previous start pulse)	LDA-ABS	FB06
64263	FB07	APU WRITE command to initiate operation	STA-ABS	FB07

***Figure 4:* Arithmetic Processing Unit Control**



Testing

The first program listed in the appendix asks for an operation code. Among some useful ones for testing are: 26=push constant pi onto top of operand stack, 16=floating add, 17=floating subtract, 18=floating multiply, 19=floating division, 2=SIN, 3=COS, 25=exchange top operand with next lower operand. At the first request for an operation code, enter 26. The program then reads the stack, and assuming all is well, the top four bytes should represent the constant pi in the APU format. The arithmetic processor representations of several useful numbers are (most significant byte first):

pi=	2,201,15,218
1.0=	1,128,0,0
−1.0=	129,128,0,0
2.0=	2,128,0,0
0=	0,0,0,0

Thus the sequence of operations 26,26,3,25,3,17 should result in a zero on the top of the stack. Or 26,26,3,25,3,18 should result in a 1 there. The status register is also read and displayed.

The second program, when run, asks for a number between zero and 255. It writes this onto the top byte of the 8231 stack and then reads it. If what went in equals what comes back, the program asks for another number, otherwise an error message is printed out. With these two programs enough simple testing can be done to insure that the circuit is working correctly. With this hurdle completed we will be ready to look at

the software aspects of the system as described in part two of this article, which will be presented next month.

Appendix

Error codes, Parts list, BASIC test programs, and APU op codes.

INTEL 8231 Error Codes (decimal values of status register)

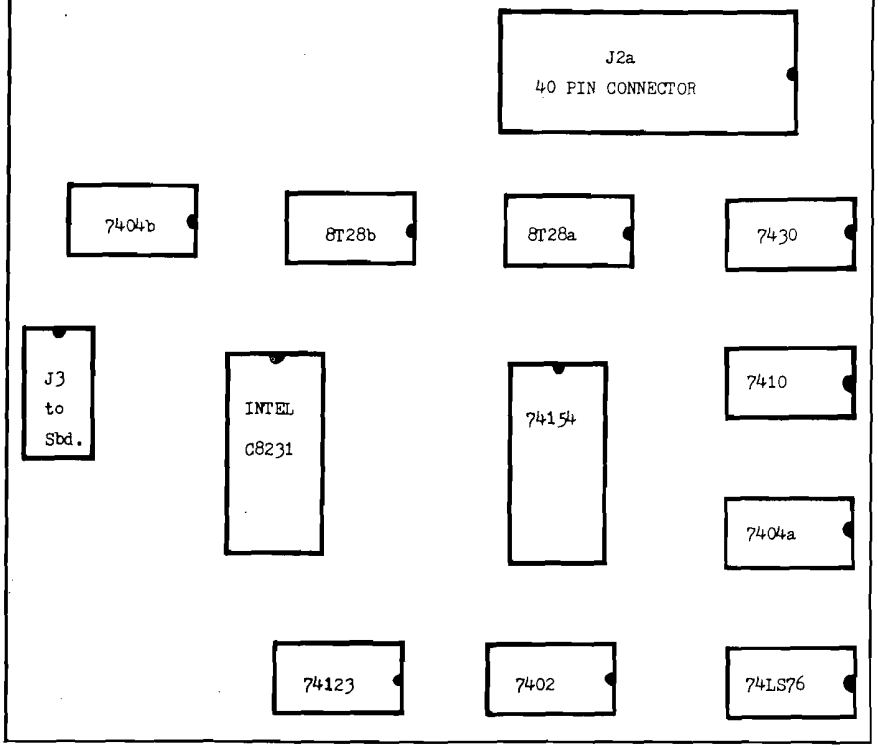
128 or greater	busy, operation not completed
64	top-of-stack negative, no error
32	top-of-stack zero, no error
16	divide by zero
8	negative argument of function not allowed (e.g. square root)
24	argument of function too big (e.g. Arc Sine, Arc Cosine, exponential)
4	underflow, number $< 2.7 \times 10^{-20}$
2	overflow, number $> 9.2 \times 10^{18}$
0	non-negative, non-zero result, no errors

Parts List

1 Vector board (at least 6" × 6")
1 40-pin wire-wrap socket
2 24-pin sockets
7 14-pin sockets (including 1 for connection to Superboard)
3 16-pin sockets
11 .01 disk capacitors (bypass)
1 80pf capacitor
1 2.2K resistor
2 7404 hex inverters
1 7402 quad NOR gate
1 7410 tri, three input NAND gate
1 7430 8 input NAND gate
1 74LS76 edge trigger flip-flop
1 74LS123 re-triggerable one shot
1 74154 4- to 16-line demultiplexer
2 8T28 tri-state buffers
1 INTELC8231 arithmetic processing unit
Ribbon cable and connectors (40 and 14 wire)

MICRO

***Figure 5:* Layout — Wirewrap Side of Board**



Listing 1

```
1  REM  APU TEST 1
2  REM  ENTER OPERATION COMMAND NUMBER
3  REM  STACK IS PRINTED FROM TOP DOWN.
 STACK HOLDS 4,4-BYTE FLT NMBRS.
9  INPUT "COMMAND";Y: POKE 64263,Y
10 A = 64257:B = 64262: PRINT : PRINT
11  PRINT "FOR COMMAND CODE=";Y
17 X =  PEEK (A - 1): PRINT "STATUS=";
PEEK (B)
20  FOR J = 1 TO 16:X =  PEEK (A): PRIN
T  PEEK (B)
25  NEXT J
27  GOTO 9
```

Listing 2

```
1  REM  APU TEST 2
2  REM  ENTER INPUT BYTE BETWEEN ZERO A
ND 255
3  REM  POKE TO APU, THEN READ. IF EQUA
L, OK.
10  INPUT "X=";X
12  POKE 64262,X: REM  WRITE OPERAND ON
 TOP OF APU STACK
15 Y =  PEEK (64257): REM  OPERAND READ
 START
16 Y =  PEEK (64262): REM  READ DATA
20  IF Y <  > X THEN  PRINT "APU R/W ER
ROR": PRINT "X=";X;"Y=";Y
22  IF Y = X THEN  PRINT "R/W OK"
25  GOTO 10
```