DEPARTMENT OF THE INTERIOR

U.S. GEOLOGICAL SURVEY

A Real-Time Seismic Amplitude Measurement System (RSAM)

by

Thomas L. Murray[1]

Elliot T. Endo[1]

[1] Cascades Volcano Observatory
5400 MacArthur Blvd.
Vancouver, Washington, 98661

Open-File Report 89-684

TABLE OF CONTENTS

INTRODUCTION

Although several real-time earthquake detection/recorder systems exist, few address the specific problem of continuous seismic amplitude measurements under conditions where individual events are difficult to recognize, such as those which occur prior to volcanic eruptions. Yet it is during these conditions, when most conventional systems saturate, that seismic information needs to be processed most rapidly. To fill this need, a simple and inexpensive Real-time Seismic Amplitude Measurement System (RSAM) was developed.

Each minute RSAM computes the average amplitude for each of 8 seismic signals for that minute. From this information, seismicitity can continue to be monitored even during periods of intense tremor. Data akin to earthquakes/hour can computed by comparing sucessive two-second amplitudes. If the latest exceeds the earlier by a set ratio (typically 2) it is considered an "RSAM event". Even energy release can be monitored by simply squaring the average amplitude (It is proportional to the electrical energy generated by the geophone. How that relates quantitatively to seismic energy release is still unclear). Though the method almost seems too simple to be effective, RSAM has been a useful tool in predicting the May 1985, May 1986 (figures 1 and 2), and October 1986 dome building eruptions at Mount St. Helens. Both figures show the RSAM data from stations close to the lava dome (Yellow Rock and St. Helens West) beginning to rise above background noise some 48 hours before the time of extrusion of a new lobe. The amplitudes continue increasing and peak near the probable time of extrusion (exact time of extrusion is not known). The spikes and high amplitudes following extrusion are due to surface activity resulting from the emplacement of a new lobe (fig. 1) or the forming of a graben (fig. 2).

Generally, data from RSAM is shown in "RSAM UNITS". "RSAM UNITS" are the direct ouput of the eight-bit analog to digital converter in the system. In a system set up for discriminators with a $\pm$2.5 volt output, **one volt peak-to-peak discriminator output equals roughly 38 RSAM UNITS.** The program in appendix A multiplies the RSAM units by 10 when sending the data to a host computer. **Data transfered to a host computer using the program in appendix A should be divided by 10 to get RSAM UNITS, or 380 to get volts peak-to-peak.** For more precise measurements, each unit should be individually calibrated.

RSAM is not meant to be a replacement for a conventional seismic system. It is to be used as a complement to the conventional system, giving real-time information on tremor/amplitude levels while earthquake locations and magnitudes are being computed by other systems. During times of little or moderate activity, RSAM may be only marginally useful. But during times of tremor or when the earthquake activity is high enough such that the conventional seismic system fails to keep up with activity, RSAM can become the primary monitor of seismicity, simply because the data is continuing to be available in real-time.

Although RSAM can be used as a "stand alone" unit, it is highly recommended that it be configured to periodically transfer its data to at least an IBM XT class computer for data archival and analysis, thus enabling its data to be integrated with the conventional seismic data.

GENERAL DESCRIPTION

The Real-time Seismic Amplitude Measurement System consists of a Tandy (Radio Shack) Model 100 lap computer (or Model 102 - they are essentially the same except for the system bus socket) and an in-house-designed data-acquisition board. The entire unit fits easily in a space 25.4 cm x 33 cm x 12.5 cm. Low power comsumption (90 ma at 12 volts) allows the unit to be powered by a car battery and solar panel if necessary.

The data acquisition board buffers the eight seismic input signals and puts them through a 0.1 hz hi-pass filter to eliminate any DC offsets. The multiplexor selects the desired signal for sampling. The signal is then full-wave rectified to convert any negative component to a positive voltage. The signals are digitized with an 8 bit analog-to-digital converter (A/D). The output of the A/D is considered to be "RSAM units". Communication between the acquisition board and the Model 100/102 is through the Model 100/102's system bus, freeing the Model 100/102's other ports for connection to other peripherals. See figure 3.

The Model 100/102 computes the average signal amplitude once a minute for each input by simply dividing the sum of each inputs' digitized samples by the number of samples. Taking the average over a one minute period allows the cessation of data acquisition for short periods (<15% of the total time) in order to process the data. This greatly simplifies programming as data acquisition and processing do not have to be performed concurrently.

At the beginning of each minute, a call to the data-acquisition sub-program causes the Model 100/102 to digitize 125 samples for each seismic input at a rate of about 50 samples/second/input and return the sums of the digitized values. The returned sums are added to running sums for the entire minute. Another call is then made to the data-acquisition sub-program, and the cycle continues throughout the minute. At the end of the minute, the average amplitudes are computed by dividing the running totals by the number of samples. The process then starts again for the next minute's data (figure 4). Depending on the specific site setup, the averages can be sent to a more powerful computer via an RS-232C link for analysis, stored in memory for later access, or just sent out to a printer. The Model 100/102, though only a 32K, 8085-based computer, still allows for numerous options.

DATA-ACQUISITION BOARD

The RSAM data-acquisition board can be divided into 3 sections: (1) power supply, (2) system bus interface, and (3) signal conditioner/converter. The circuit schematic is shown in figures 5 and 6.

Power Supply

The power supply section (U5 thru U8) converts the 12 volt input (J2) to ±12 volts to power the analog section of the circuit, ±5 volts for the multiplexor, and +6 volts to power the Model 100/102 via J3. This allows the entire unit to be run from a 12 volt battery. Current draw is under 100 milliamps.

System Bus Interface

The system bus interface (U1 thru U4) performs the address decoding for the programmable interface adapter (PIA), U1, and the analog-to-digital converter (A/D), U9. A 40 conductor ribbon cable between J1 and the Model 100/102's system bus socket connects the board to the computer. I/O addresses 0-127 of the Model 100/102 are available for external uses such as this acquisition board. U2 and U3 decode the address for the PIA. Switch S1 selects in which address block (32-63, 64-95, or 96-127) the PIA will reside. This allows for up to three of the boards to be hooked to a single Model 100/102. S2 sets the I/O address for the A/D. Though the switch can be set to any address under 128, it is recommended that it be set to one in the 0-31 block, leaving the higher addresses for the PIAs.

Note that only three of the PIA's 22 digital I/O lines are used to control the multiplexor. The rest are available for circuit enhancements.

The default addresses/switch settings are:

S1 to 32-63 (all positions off except for 2)	for the PIA address
S2 to 5 (all positions off except for 1 and 3)	for the A/D

Signal Conditioner/Converter

The analog seismic signals enter the board via J5. Note that all the signal lows are shorted together. These signals should be tapped from either the outputs of the discriminators or the inputs to the drum recorders.

U10-U17 buffer each of the eight signals and send them through 0.1 hz hi-pass filters to remove the DC offset. The multiplexor (U18) chooses the input to digitize. It is controlled by bits 0-2 of port A of the PIA. Since the A/D will accept only positive voltages, the signal must be full-wave rectified (U19). The signal is then ready for digitizing by the 8-bit A/D (U9).

D1 supplies the reference voltage for the A/D. Full scale for the A/D is twice the reference voltage. For seismic signals with range of ±2.5 volts an LM385-1.22 should be used. For signals with a range of ±5.0 volts an LM336-2.5 should be used. Note that R2 can be used to trim the LM336-2.5 to precisely +2.500 volts but not the LM385-1.22.

FIG. 1 - 15 MINUTE AVERAGES

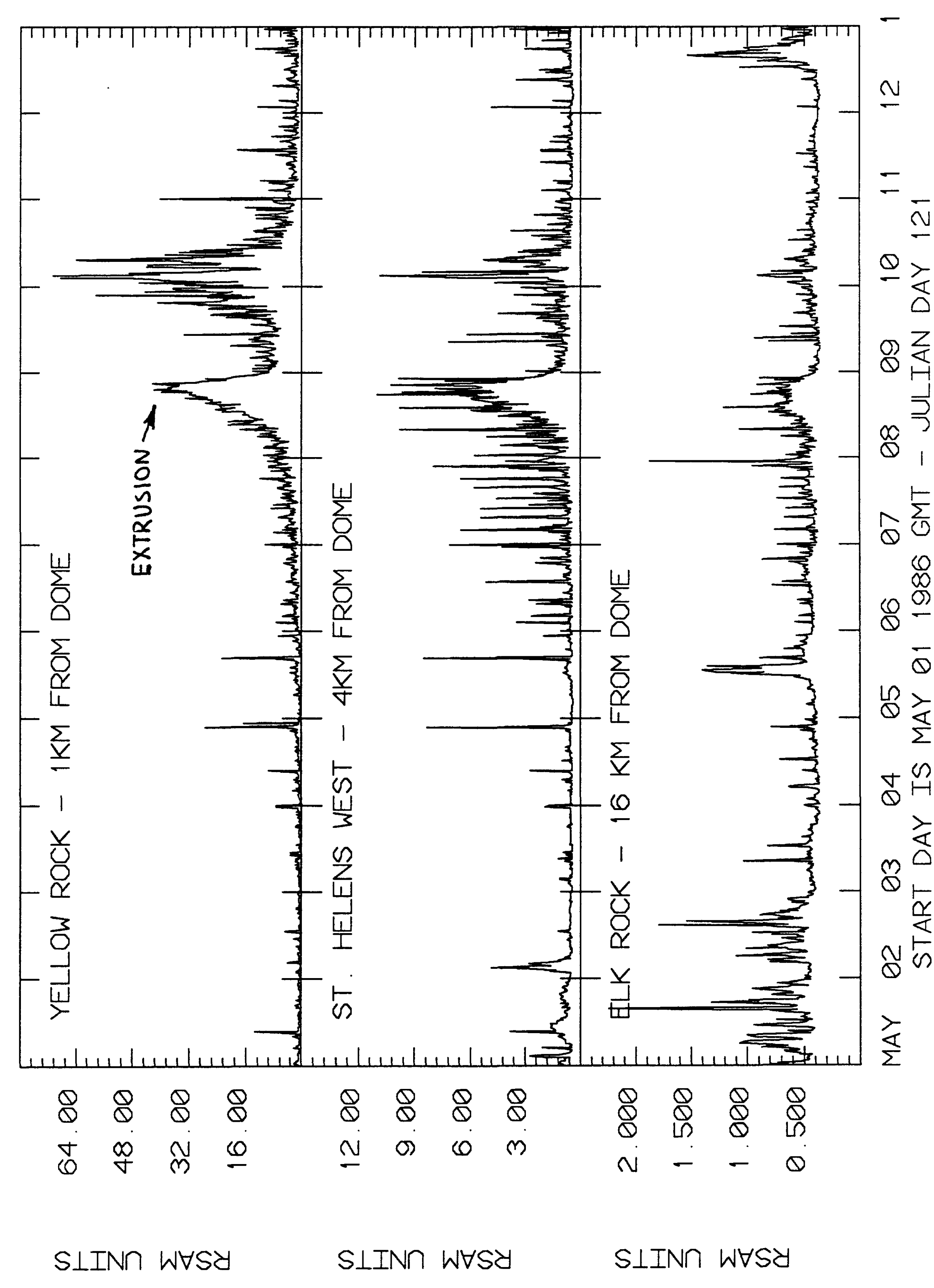


FIG. 2 - HOURLY AVERAGES

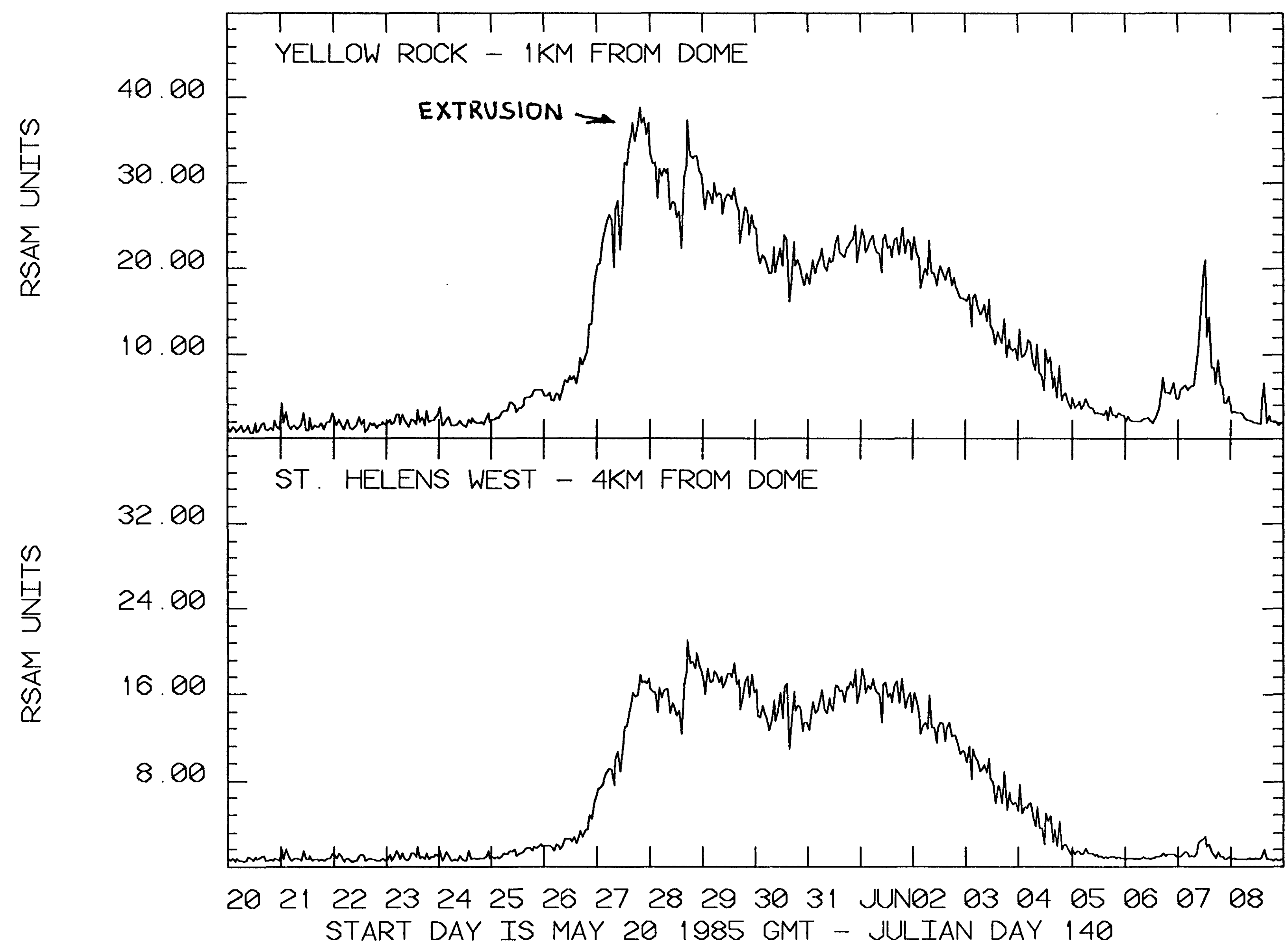


Figure 3
RSAM
BLOCK DIAGRAM

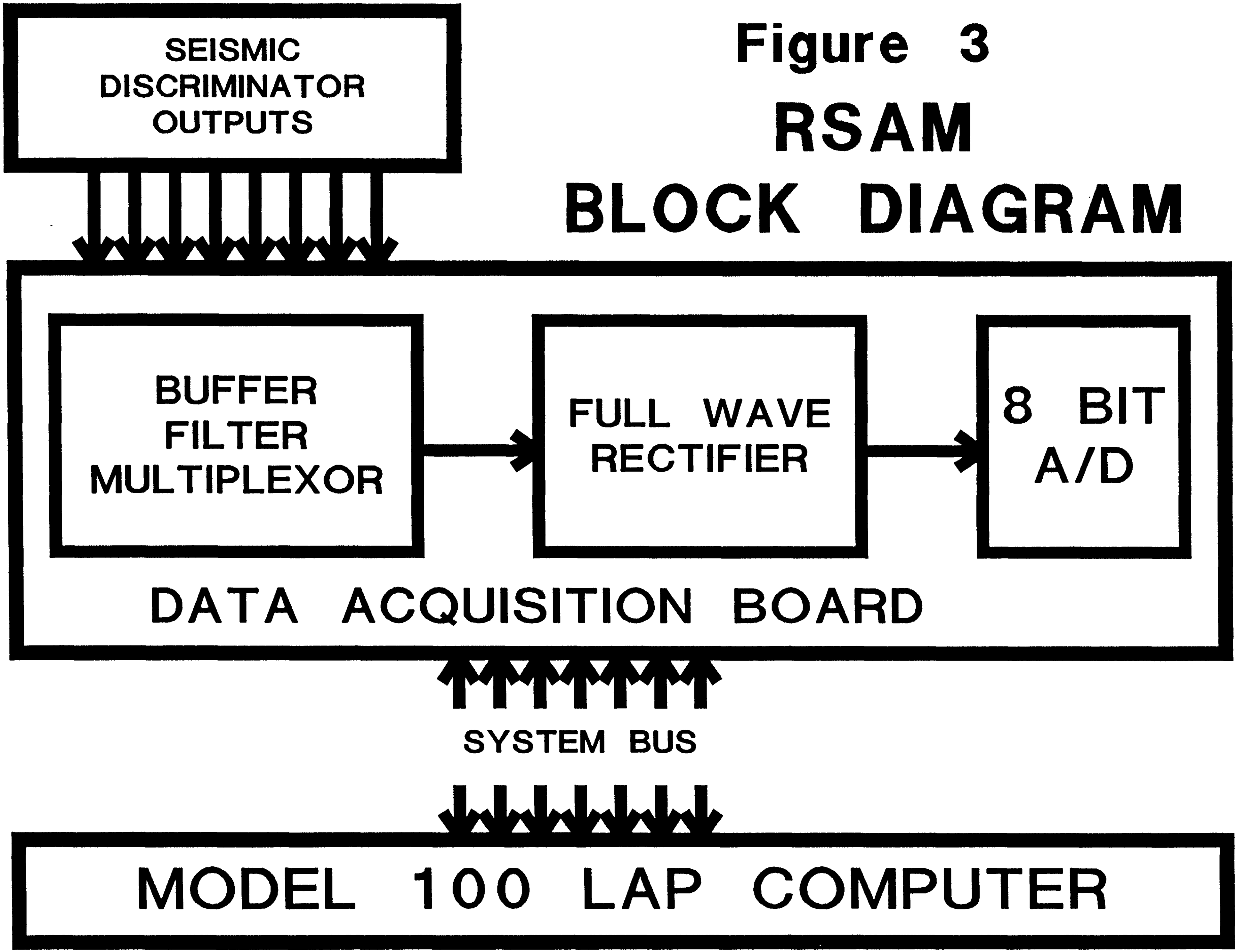


Figure 4 SOFTWARE FLOW CHART

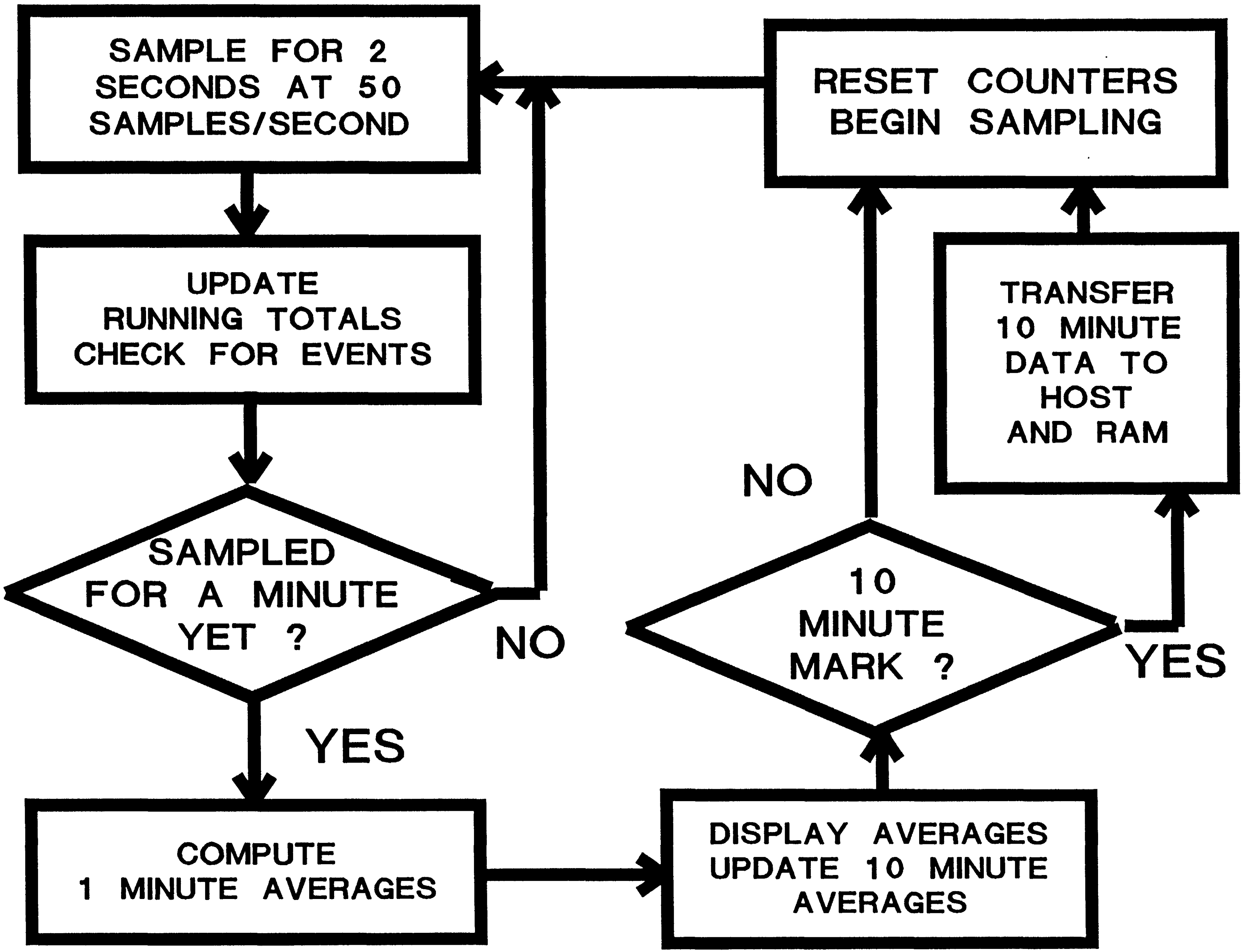


FIG. 5

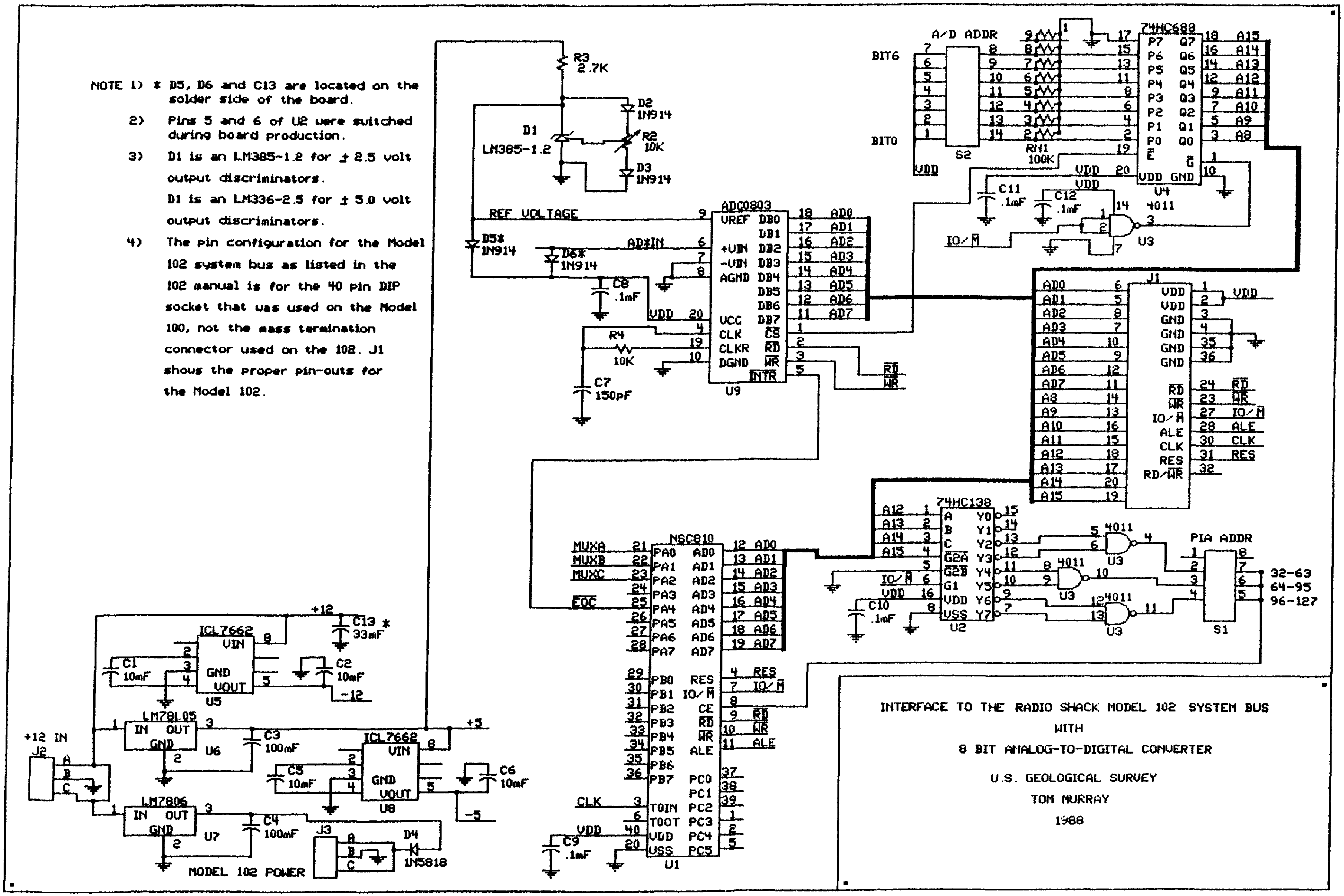
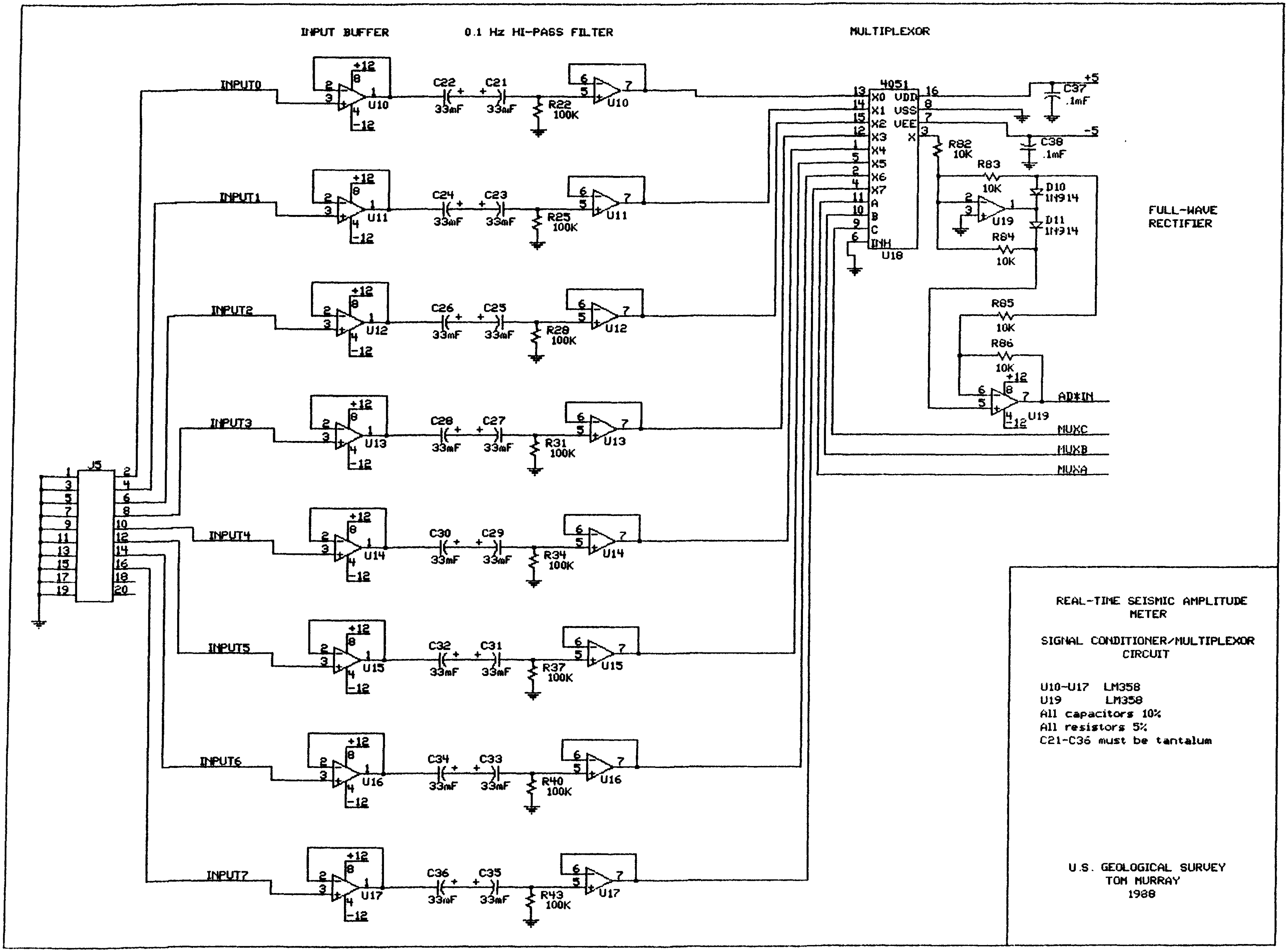

NOTE 1) * D5, D6 and C13 are located on the solder side of the board.
2) Pins 5 and 6 of U2 were switched during board production.
3) D1 is an LM385-1.2 for ± 2.5 volt output discriminators.
D1 is an LM336-2.5 for ± 5.0 volt output discriminators.
4) The pin configuration for the Model 102 system bus as listed in the 102 manual is for the 40 pin DIP socket that was used on the Model 100, not the mass termination connector used on the 102. J1 shows the proper pin-outs for the Model 102.
INTERFACE TO THE RADIO SHACK MODEL 102 SYSTEM BUS
WITH
8 BIT ANALOG-TO-DIGITAL CONVERTER
U.S. GEOLOGICAL SURVEY
TOM MURRAY
1988
A/D ADDR
BIT6
BIT0
74HC688
74HC138
ADC0803
NSC810
ICL7662
LM78L05
LM7806
LM385-1.2
REF VOLTAGE
PIA ADDR
32-63
64-95
96-127
+12 IN
MODEL 102 POWER

FIG. 6


INPUT BUFFER
0.1 Hz HI-PASS FILTER
MULTIPLEXOR
FULL-WAVE RECTIFIER
INPUT0
INPUT1
INPUT2
INPUT3
INPUT4
INPUT5
INPUT6
INPUT7
J5
4051
U18
AD*IN
MUXC
MUXB
MUXA
REAL-TIME SEISMIC AMPLITUDE METER
SIGNAL CONDITIONER/MULTIPLEXOR CIRCUIT
U10-U17 LM358
U19 LM358
All capacitors 10%
All resistors 5%
C21-C36 must be tantalum
U.S. GEOLOGICAL SURVEY
TOM MURRAY
1988

APPENDIX A

The following program was written for an RSAM connected to another computer via the RS-232 port. Data is transfered to the host computer at ten-minute intervals. The previous day's data is also stored in RSAM's memory for later transfer to the host.

This program:

1) Displays the one-minute RSAM averages for each of the 8 channels on the Model 100/102's screen. Also displays the number of RSAM "events" in the current ten-minute period for channel 1

2) Calulates RSAM "events" for channels 1-3.

3) Transfers the 10-minute averages and the number of events in the 10-minute period to a host computer via the RS-232 port

4) Saves the 10-minute data in its memory for later transfer to the host computer. Because of limited memory, only data from the current day and previous day can be stored. Data prior to that is erased to make room for the newer data.

RSAM PROGRAM

The data held im memory can be dumped over the RS-232 port by pushing function key F7 and entering in the days for the data you wish dumped.

lines 5-64 load in the machine code that actually gets the data from the RSAM board. after collecting the required number of samples, it returns to BASIC.

```
5 CLEAR 512,62600: PRINT "LOADING MACHINE CODE"
12 OUT 39,0:OUT 36,15
20 DATA 00,00,00,00,32,91,F4,E6,00,D3
21 DATA 20,06,01,CD,F6,F4,D3,05,22,92
22 DATA F4,EB,0E,10,12,13,0D,C2,A8,F4
23 DATA 2A,92,F4,06,32,CD,F6,F4,DB,05
24 DATA 32,90,F4,79,3C,E6,07,D3,20,06
25 DATA 01,CD,F6,F4,D3,05,3A,90,F4,86
26 DATA EB,12,13,EB,D2,D4,F4,34,23,0C
27 DATA 79,E6,07,C2,B1,F4,3A,91,F4,3D
```

in the following line the second value from the left (0D) determines the sample rate. 06 is about 100 samples/second/station 0D about 50 samples/second/station

```
28 DATA C8,32,91,F4,2A,92,F4,06,0D,0E
29 DATA A6,0D,C2,EB,F4,05,C2,E9,F4,C3
30 DATA B1,F4,05,C2,F6,F4,C9
52 FOR I%=1 TO 107
54 READ A$: HI%=ASC(LEFT$(A$,1))
56 LO%=ASC(MID$(A$,2,1))
58 IF HI% > 60 THEN HI%=16*(HI%-55) ELSE HI%=16*(HI%-48)
60 IF LO% > 60 THEN LO%=LO%-55 ELSE LO%=LO%-48
62 HI%=HI%+LO%: POKE 62607+I%,HI%
64 NEXT I%
```

real program begins

```
90 MAXFILES=3
```

get it so this program will run on a reset

```
190 IPL "RSAM.BA"
```

print the time and date on the screen. if it is incorrect, the user will have to stop the program, set the correct time and date (see the Model 102 manual) and restart the program.
OD$, OH$, and OM$ hold values for the time and date that are compared with the current time and date to see if its time to do something.

```
200 PRINT "TIME IS "+TIME$+"    DATE IS "+DATE$: TI$=TIME$: OH$=LEFT$(TIME$,2):
OM$=MID$(TIME$,5,1):OD$=DATE$
210 PRINT "IF TIME OR DATE IS INCORRECT HALT       EXECUTION OF THE PROGRAM (
SHIFT BREAK)  AND ENTER CORRECT TIME AND DATE (PAGE   17 OF THE MANUAL)"
211 PRINT " "
```

calculate the julian day from the date. note that the model 102 does not know about leap years. this means the date may be off in leap years. since the julian day is only incremented at the end of a day and not re-calculated from the date, this shouldn't be a problem unless the program is restarted with the wrong initial date.

```
220 V1%=VAL(MID$(DATE$,1,2)):V2%=VAL(MID$(DATE$,4,2)):V3%=VAL(MID$(DATE$,7,2))
230 JL%=(V1%-1)*31+V2%
235 IF V1%<3 THEN GOTO 260
240 JL%=JL%-(V1%-3)\2-3
245 IF V1%>8 THEN JL%=JL%+V1% MOD 2
```

```
250 IF V3% MOD 4=0 THEN JL%=JL%+1
260 PRINT USING "JULIAN DAY IS ###";JL%
```

set up for the next time to send data to the host

```
265 GOSUB 4000
```

see if we are to create a new set of data files or if one already exists for the current julian day, in which case we just append to it.

```
270 GOSUB 5000
```

F7 is the only function key that does anything. it initiates the routine that dumps data from the RAM files to the host.

```
280 ON KEY GOSUB 9000,9000,9000,9000,9000,9000,7000
```

open the RS-232 port for dumping data to the host.
2400 baud, 7 bit, no parity, 2 stop bits, no XON/XOFF

```
300 OPEN "COM:67N2D" FOR OUTPUT AS 2
```

start declaring and initializing the variables

```
A%(8)  the running totals of amplitude returned by the machine code.
DT%(11) a buffer used to transfer the data to the data stream (DS$).
EC%(3,8) the ring buffer for the last 3 2-second values collected for
         of the 8 channels
EO%     pointer into EC% for the spot of the value taken two times
EP%     pointer into EC% for the spot for the next value to go
EQ%(8) the total events in the 10 minute period for each of the 8
       inputs
MT#(7) the running total for the amplitudes for each minute.
M1% to
   M8% the multiplier for each channel in determining events
PT%    sets how many samples/station to take each time the machine code
       subroutine is entered.  125 is the maximum value.
RT#(7) the running total for the amplitudes for the 10 minute period.
SF$    an output format for the screen.
SM%    the running total of how many time the machine code was entered
       each minute. multiply it by PT% to get the running total of samples
       taken in the minute.
ST#    the running total of times the machine code was was entered in the
       10 minute period.  It times PT% is the running total of samples
       taken.
T1% to
   T8% the thresholds for each channel for determining events
```

```
405 SM%=0
410 DIM RT#(7)
412 DIM EC%(3,8)
413 DIM MT#(7)
414 DIM DT%(11)
415 ST#=0
420 DIM A%(8)
432 FOR I= 0 TO 7
433 A%(I)=0:EQ%(I)=0:RT#(I)=0
436 EC%(0,I)=20000 : EC%(1,I)=EC%(0,I)
437 NEXT I
```

events occur if the current running total recieved from the machine code exceeds the one collected twice prior by a factor of Mx% andits value exceeds a threshold Tx%.
the defaults for Mx% are two. That for Tx% is number of samples*5
(the number returned is a running total and the average of 5 is usually a decent event)

```
438 M1%=2: M2%=2: M3%=2: M4%=2: M5%=2: M6%=2: M7%=2: M8%=2
439 T1%=625: T2%=625: T3%=625: T4%=625: T5%=625: T6%=625: T7%=625: T8%=625
```

DS$ is the data stream sent to the PS2 every 10 minutes or stored in RAM every 20 minutes.
in DS$:
- 30 indicates the specific RSAM unit (insert different numbers for Quito, Vancouver etc.)
- YR is the year
- DAY is the julian day
- HR is the hour
- MN is the minute
- DATA is the data for the various stations
- > indicates valid data (never becomes invalid with the RSAM but is used in processing the low-data-rate telemetry data)

```
440 DS$="@30YRDAYHRMNDATA>DATA>DATA>DATA>DATA>DATA>DATA>DATA>@
445 DS$=DS$+"DATA>DATA>DATA>@"
450 FOR I=1 TO 11
460 DT%(I)=9999
470 NEXT I
475 SF$="  ###.#    ###.#    ###.#    ###.#"
477 EP%=2 : EO%=0
```

set the number of samples taken each time the machine code is called to its maximum (125).

```
478 PT%=125
```

---

call the machine code and sample for 125 times. the resulting running total of the samples is stored in A%

```
480 CALL 62612,PT%,VARPTR(A%(0))
```

increment the times we've gotten 125 samples (SM%)

```
482 KEY ON : SM%=SM%+1
485 KEY STOP
```

and increment the running total of the amplitude for the minute.
check for events. an event occurs if the value returned by the machine code exceeds the value returned twicw prior by a factor of Mx% and exceeds the threshold Tx%. If there was an event, the value returned 2 seconds ago is set to its upper limit to prevent counting the event twice.

```
490 J%=0 : MT#(J%)=MT#(J%)+A%(J%):EC%(EP%,J%)=A%(J%):IF (A%(J%) >M1%*
    EC%(EO%,J%)) AND (A%(J%) > T1%) THEN EQ%(J%)=EQ%(J%)+1:
    EC%((EO%+1)MOD3,J%)=20000
491 J%=1 : MT#(J%)=MT#(J%)+A%(J%):EC%(EP%,J%)=A%(J%):IF (A%(J%) >M2%*
    EC%(EO%,J%)) AND (A%(J%) > T2%) THEN EQ%(J%)=EQ%(J%)+1:
    EC%((EO%+1)MOD3,J%)=20000
492 J%=2 : MT#(J%)=MT#(J%)+A%(J%):EC%(EP%,J%)=A%(J%):IF (A%(J%) >M3%*
    EC%(EO%,J%)) AND (A%(J%) > T3%) THEN EQ%(J%)=EQ%(J%)+1:
    EC%((EO%+1)MOD3,J%)=20000
493 J%=3 : MT#(J%)=MT#(J%)+A%(J%):EC%(EP%,J%)=A%(J%):IF (A%(J%) >M4%*
    EC%(EO%,J%)) AND (A%(J%) > T4%) THEN EQ%(J%)=EQ%(J%)+1:
    EC%((EO%+1)MOD3,J%)=20000
494 J%=4 : MT#(J%)=MT#(J%)+A%(J%):EC%(EP%,J%)=A%(J%):IF (A%(J%) >M5%*
    EC%(EO%,J%)) AND (A%(J%) > T5%) THEN EQ%(J%)=EQ%(J%)+1:
    EC%((EO%+1)MOD3,J%)=20000
495 J%=5 : MT#(J%)=MT#(J%)+A%(J%):EC%(EP%,J%)=A%(J%):IF (A%(J%) >M6%*
    EC%(EO%,J%)) AND (A%(J%) > T6%) THEN EQ%(J%)=EQ%(J%)+1:
    EC%((EO%+1)MOD3,J%)=20000
496 J%=6 : MT#(J%)=MT#(J%)+A%(J%):EC%(EP%,J%)=A%(J%):IF (A%(J%) >M7%*
    EC%(EO%,J%)) AND (A%(J%) > T7%) THEN EQ%(J%)=EQ%(J%)+1:
    EC%((EO%+1)MOD3,J%)=20000
```

```
497 J%=7 : MT#(J%)=MT#(J%)+A%(J%):EC%(EP%,J%)=A%(J%):IF (A%(J%) >M8%*
    EC%(EO%,J%)) AND (A%(J%) > T8%) THEN EQ%(J%)=EQ%(J%)+1:
    EC%((EO%+1)MOD3,J%)=20000
```

reset the pointers for the two previosly collected 2 second values.

```
500 EP%=(EP%+1)MOD 3: EO%=(EO%+1) MOD 3
```

see if we are at the end of our minute yet, otherwise get some more data

```
520 IF MID$(TIME$,5,1) <> OM$ THEN OM$=MID$(TIME$,5,1):GOSUB 700
530 GOTO 480
```

---

the subroutine for once/minute processing

increment the number of times the machine code was entered in the current 10 minute period

```
700 ST#=ST#+SM%
```

put the actual number of samples taken into SM%

```
703 SM%=SM%*PT%
```

compute the ten minute running total (RT#), the one minute average amplitude (MT#)

```
705 FOR I%= 0 TO 7
710 RT#(I%)=RT#(I%)+MT#(I%):MT#(I%)=MT#(I%)/SM%
720 NEXT I%
```

print the one minute info on the screen

```
730 TR$=DATE$+" "+TIME$+"  "+"SAMPLES= "
740 PRINT USING "\                           \####";TR$,SM%
755 PRINT USING SF$;MT#(0),MT#(1),MT#(2),MT#(3)
760 PRINT USING SF$;MT#(4),MT#(5),MT#(6),MT#(7)
765 PRINT USING "###  EVENTS ON CHANNEL 1";EQ%(0)
```

reset things back to zero and re-initialize the PIA on the RSAM board.

```
767 FOR I%=0 TO 7 :MT#(I%)=0:NEXT I%:SM%=0
768 OUT 39,0:OUT 36,15
```

and see if we are in a new 10 minute period in which case we do the 10 minute processing

```
769 IF OT$<>MID$(TIME$,4,1) THEN GOSUB 800
770 RETURN
```

---

the once every 10 minutes processing

set ST# to the total number of samples taken

```
800 ST#=ST#*PT%
```

compute the 10 minute averages and put them in DT% (the output buffer)

```
805 FOR I%=0 TO 7
810 RT#(I%)=RT#(I%)/ST#:DT%(I%+1)=RT#(I%)*10:NEXT I%
```

gosub 2000 to output the data.

```
812 GOSUB 2000
```

set things back to zero

```
820 FOR I%=0 TO 7
825 RT#(I%)=0:EQ%(I%)=0:NEXT I%
830 ST#=0
```

reset OT$ by going to 4000 so we will know when the next 10 minute period is entered.

```
840 GOSUB 4000:RETURN
```

---

subroutine to output the data over the RS-232 line and to the RAM memory files

see if we are in a new day. if so go take care of closing and opening the data files (gosub 6000)
JL% hold the current julian day

```
2000 IF OD$ <> DATE$ THEN JL%=JL%+1 : GOSUB 6000
2002 OD$=DATE$
```

take care of the end of the year.

```
2010 IF JL%=366 THEN JL%=1
```

put the year, julian day, and time in DS$

```
2015 MID$(DS$,6,3)="000"
2016 JL$=STR$(JL%) : V1%=LEN(JL$): V1$=RIGHT$(JL$,V1%-1)
2020 MID$(DS$,10-V1%,V1%-1)=V1$
2021 MID$(DS$,9,2)=MID$(TIME$,1,2)
2022 MID$(DS$,11,2)=MID$(TIME$,4,2)
2033 MID$(DS$,4,2)=RIGHT$(DATE$,2)
```

the events for channels 1-3 go into DT% slots 9-11

```
2035 FOR I%=9 TO 11
2037 DT%(I%)=EQ%(I%-9) : NEXT I%
```

put in the data (from DT%) into DS$

```
2040 FOR I%=1 TO 11
2050 V%=8+I%*5
2051 OS$=STR$(DT%(I%)):LS%=LEN(OS$)-1
2053 MID$(DS$,V%,4)="0000"
2065 V$=RIGHT$(OS$,LS%):MID$(DS$,V%+4-LS%,LS%)=V$
2070 MID$(DS$,V%+4,1)=VL$(I%)
2071 NEXT I%
```

send it over the RS-232 line and indicate that the one line is all the data for now. the data is also written out to the current RAM data file (#3).

```
2075 OPEN F$+".DO" FOR APPEND AS 3:PRINT #3,DS$:CLOSE 3
2090 PRINT #2,DS$ :PRINT #2,"END OF DATA"
2100 PRINT DS$
2180 GOSUB 4000 : RETURN
```

---

reset OT$ so we will know when we have reached the next 10 minute mark

```
4000 OT$=MID$(TIME$,4,1):RETURN
```

---

the routine to search and see if data files are currently in RAM, does one exist for the current julian day already, and which one should be erased if necessary.

the file names are A.DO and B.DO

```
5000 F$="@"
```

first open them for append. this creates them if they don't exist, but won't erase them if they do.

```
5005 FOR I%=0 TO 1
5010 F$=CHR$(ASC(F$)+1)
5015 OPEN F$+".DO" FOR APPEND AS 3:CLOSE(3):NEXT I%
```

now see if one was already there for the current day. the first line in the file should have the julian day for that file.

```
5020 F$="@" : FP$="A" : OJ$="9999" :GOSUB 5070
5025 FOR I%=0 TO 1
5030 F$=CHR$(ASC(F$)+1)
```

an EOF indicates it was just created.

```
5035 OPEN F$+".DO"FOR INPUT AS 3: IF EOF(3) THEN CLOSE(3) : GOTO 5050
5037 INPUT #3,LS$:CLOSE(3)
```

we keep track of the oldest file (smallest julian day). if there are no newly created files and one doesn't exist for the current julian day, then we must erase the oldest file. FP$ holds the name for the oldest file.

```
5040 IF LS$<OJ$ THEN OJ$=LS$:FP$=F$
```

routine 5070 converts JL% to YD$. if it matches with the files LS$ then it is the RAM file for today.

```
5042 GOSUB 5070: IF LS$=YD$ THEN RETURN
```

if not try the next one

```
5045 NEXT I%
```

executing here means we open a new file for today. if it contained data from a previous day, that data is lost.

```
5047 F$=FP$
5050 OPEN F$+".DO" FOR OUTPUT AS 3:PRINT #3,YD$:CLOSE (3)
5060 RETURN
```

---

a routine to get the ascii representation of JL% into YD$

```
5070 YD$="000":JL$=STR$(JL%):V1%=LEN(JL$):MID$(YD$,5-V1%,V1%-1)=RIGHT$(JL$,V1%-1
)
5075 RETURN
```

---

the routine to close a data file and open a new one. it is called at the end of each day.

```
6000 CLOSE 3
```

increment to the next file if we are at B.DO, the next one is A.DO.

```
6005 F$=CHR$(ASC(F$)+1) : IF F$>"B" THEN F$="A"
```

opening the file for output erases the old data and allows us to start with an empty file.

```
6010 OPEN F$+".DO" FOR OUTPUT AS 3
```

put in the julian day for the file

```
6015 GOSUB 5070 : PRINT #3,YD$: CLOSE 3
6020 RETURN
```

---

routine to dump data from the files thru the RS-232 line. entered by pressing function key F7.

find out what days are to be dumped. 7200 has the routine to get the julian day number, but will time out if nothing is entered within about 10 seconds.

```
7000 PRINT " ": PRINT USING "TODAY IS JULIAN DAY ####";JL%:PRINT" "
7003 PRINT "JULIAN DAY IS "+STR$(JL%)
7005 PRINT "INITIAL JULIAN DAY OF DATA TO          BE DUMPED ?"
7010 GOSUB 7200
```

if GD%=-1 something was amiss, so just go back to normal operation.

```
7012 IF GD%=-1 THEN GOTO 7100 ELSE ID%=ST%
7015 PRINT "ENDING JULIAN DAY OF DATA TO BE         DUMPED ? "
7020 GOSUB 7200
7022 IF GD%=-1 THEN GOTO 7100 ELSE ED%=ST%
```

close the current data file and search through the files looking for a match with day we want to dump.

```
7025 CLOSE 3
7030 FOR X%=ID% TO ED%
7035 X$="000":V$=STR$(X%):V1%=LEN(V$):MID$(X$,5-V1%,V1%-1)=RIGHT$(V$,V1%-1)
```

cycle through the files looking for a julian day match

```
7040 FP$="@"
7045 FP$=CHR$(ASC(FP$)+1)
```

B.DO is the last file so we would go to 7065 if we haven't matched by then

```
7047 IF FP$>"B" GOTO 7065
```

get the julian day for the file (the first line)

```
7050 OPEN FP$+".DO" FOR INPUT AS 3: IF EOF(3) THEN CLOSE(3):GOTO 7045
```

if it does match, dump the data by going to 7500

```
7055 INPUT #3,YD$:IF YD$=X$ THEN PRINT "DUMPING DATA FOR DAY "+X$:
     GOSUB 7500:GOTO 7070
```

try the next file

```
7060 CLOSE 3:GOTO 7045
7065 PRINT "NO DATA FOR DAY "+X$
7070 NEXT X%
```

send the host the end of data dump message to let the host know it can start processing the data.

```
7080 PRINT "END OF DATA DUMP"
7085 PRINT #2,"END OF DATA DUMP"
```

reopen the current data file and return

```
7100 RETURN
```

---

the routine to get the julian days for the data dump, but still time out if nothing is entered.

```
7200 ST%=0:GD%=-1

    you have until I=300 to enter the data

7205 FOR I=1 TO 300

   get the number you have pressed on the keyboard

7210 V1$=INKEY$

    if its a carriage return (CHR$(13)) then we are done

7215 IF CHR$(13)=V1$ THEN GOTO 7250

   if its a number, add it to ST% and put it on the screen

7220 IF V1$=>"0" AND V1$<="9" THEN ST%=ST%*10+VAL(V1$):CALL 19268,ASC(V1$):GD%=1

   if its out of range, abort (set GD% to -1)

7225 IF ST%>366 THEN PRINT" ILLEGAL VALUE, OPERATION ABORTED":GD%=-1:GOTO 7270
7230 NEXT I
7250 IF GD%=-1 THEN PRINT " TIME OUT ON INPUT":GOTO 7270
7255 IF ST%<1 OR ST%>366 THEN PRINT " ILLEGAL VALUE, OPERATION ABORTED":GD%=-
1:GOTO 7270
7270 PRINT " ":RETURN
----------------------------------------------------------------------------------
    send the data from a file over the RS-232 line.

    are we at the end of the file yet??

7500 IF EOF(3) THEN CLOSE (3):RETURN

   read it in from the file and send it out

7505 INPUT #3,HS$:PRINT #2,HS$:GOTO 7500
----------------------------------------------------------------------------------
    a phony subroutine for keyboard inputs

9000 RETURN
********************************************************************************
```

APPENDIX B

RSAM data acquisition module

the following is the assembly/machine code for the subroutine that does the data acquistion for the Real-time Seismic Amplitude Monitor (RSAM) running on a Radio Shack Model 100/102. it will sample 8 stations at the rate of up to 100 samples/second/station. the running total of each stations data is returned in an integer array whose address is passed to this subroutine. dividing this number by the number of samples taken (also passed to the subroutine) gives the average amplitude.

It is called from BASIC with the statement:

```
CALL 62612,S%,VARPTR(N%(0))
```

where S% is the number of samples/station desired
N% is the array the the data will be returned in

NOTE:

1) the range of an integer in Model 100/102 BASIC is +32,000 to -32,000. if the number of samples times the 255 (the largest value possible returned by an 8 bit A/D) is greater than +32,000 the returned integer value will be negative and software in BASIC will have to convert it to the proper floating point magnitude and sign. if the returned value is greater than 64,000 it will roll-over and start counting up from 0 again.

   if you keep the number of samples to 125 or under you will not have any problems.

2) N% has to be an integer array. N or N! or N# will not work.

3) to keep the Model 100/102 from overwriting this code, one of the first statements in the BASIC program should be

   ```
   CLEAR 512,62600
   ```

   8085 machine code is not relocatable. if you wish to do so, the address jumps and calls have to changed.

4) the switch settings on the RSAM board are set for an address of 5 for the A/D and 32-64 for the NSC810.

****

the code!

****

in the following documentation, values in the first column are the hex value for the instruction/data located at the address in RAM indicated by the next two columns (column 2 has the hex address, column 3 the decimal equivalent). column 4 is an address label and column 5 the assembly code instruction.

VAR1 is just a spot to hold values periodically. it is used somewhat as another register
SAMPLES holds the number of samples/station left to collect
ADDRESS holds the address of the initial byte in the integer array passed to this subroutine

```
00 F490 62608 VAR1:
00 F491 62609 SAMPLES:
00 F492 62610 ADDRESS:
00
------------------------------------------------------------------------------
                start of program

        A has the number of samples we are to take

32 F494 62612 START:    STA     put A in SAMPLES
91
F4

        start the conversion of first channel now.  we can then do some
        initialization while that is happening instead of just waiting.

        I/O port 32 is the location of the multiplexor.  the value outputted
        there will be the channel to be sampled.

E6 F497 62615           ANI     zero A
00

        select channel 0

D3 F499 62617           OUT,32d
20

        a short delay (register B determines the length) is called to allow
        the signal to settle.

06 F49B 62619           MVI,B   load 1 into B
01
CD F49D 62621           CALL    call SHRT_DLY
F6
F4

        the A/D is at location 5.  an output to it starts the conversion
        process.

D3 F4A0 62624           OUT,5
05
------------------------------------------------------------------------
        while its converting we can set all the values of N% (the passed
        array) to zero.

        store the location in ADDRESS for further use.

22 F4A2 62626           SHLD    store hl in ADDRESS
92
F4
EB F4A5 62629           XCHG    exchange DE and HL

        C is the counter for the length of the array in bytes.  its set
        for 16 (8 bytes x 2 bytes/integer)

0E F4A6 62630           MVI,C   move 16 into c
10

        A still has zero in it
```

```
12 F4A8 62632 INIT:     STAX,A  store a at address in DE

       just keep moving up in memory from the initial ADDRESS until C is 0

13 F4A9 62633           INX,D   increment DE
0D F4AA 62634           DECR,C  decrement C
C2 F4AB 62635           JNZ     jump to INIT if not done
A8
F4

       get the initial ADDRESS back in HL

2A F4AE 62638           LHLD    load HL from ADRRESS
92
F4
-----------------------------------------------------------------------------

       start collecting the data

       we have a conversion in process so we wait 50 SHRT_DLYs for the
       conversion to be complete

06 F4B1 62641 GET_DATA: MVD,B   put 50 into B
32
CD F4B3 62643           CALL    call SHRT_DLY
F6
F4

       and read the data

DB F4B6 62646           IN A    read a/d into A
05

       store it temporarily while we get the next conversion going

32 F4B8 62648           STA     store a in VAR1
90
F4

       C always has the last channel sampled

79 F4BB 62651           MOV A,C move C into A
3C F4BC 62652           INC A

       we are set up for only 8 channels

E6 F4BD 62653           ANI,A   and A with 7
07
D3 F4BF 62655           OUT,A   set multiplexor to next channel
20

       another short delay for things to settle

06 F4C1 62657           MVI,B   move 01 into B
01
CD F4C3 62659           CALL    call SHRT_DLY
F6
F4
       start the conversion
```

```
D3 F4C6 62662            OUT A    start a/d conversion
05

       get our last reading back from VAR1

3A F4C8 62664            LDA      load A from VAR1
90
F4

       HL contains the address of the low byte of the integer
       add the value in HL to A

86 F4CB 62667            ADD M    add memory (address in HL) to A
EB F4CC 62668            XCHG

       store it back into the same spot

12 F4CD 62669            STAX,D   store a into address DE

       increment up to the high byte of the integer and put it in HL

13 F4CE 62670            INX D    increment DE
EB F4CF 62671            XCHG

       if we have a carry we have to increment the high byte

D2 F4D0 62672            JNC      jump if no carry set to NO_CARRY
D4
F4
34 F4D3 62675            INR,M    increment memory in location HL
-----------------------------------------------------------------
       increment HL to the low byte of the next integer

23 F4D4 62676 NO_CARRY: INX,H    increment HL

       increment C to the last channel sampled (it still had the
       previous one)

0C F4D5 62677            INR,C    increment C
79 F4D6 62678            MOV A,C  move C into A
E6 F4D7 62679            ANI      and A with 7
07

       if we're up to channel 8 we are done with this pass.
       otherwise get the next channels data (GET_DATA)

C2 F4D9 62681            JNZ      jump if not zero to GET_DATA
B1
F4

       getting here means we've completed a pass of sampling the
       eight channels.

       see if we are to make another pass (i.e. samples isn't 0)

3A F4DC 62684            LDA      load A from SAMPLES
91
F4
3D F4DF 62687            DCR A    decrement A
```

```
        if samples is now zero, we are done and return

C8 F4E0 62688             RZ      return if zero

        otherwise set up for another set of samples

32 F4E1 62689             STA     store a in SAMPLES
91
F4
2A F4E4 62692             LHLD    load HL from ADDRESS
92
F4
        delay an appropriate amount for 100 samples/second rate

06 F4E7 62695             MVI B   move 06 into B
06
0E F4E9 62697 MSEC_DLY: MVI C   move 166 into C
A6
0D F4EB 62699 MSEC_LOOP:DCR C   decrement C
C2 F4EC 62700             JNZ     jump if not zero to MSEC_LOOP
EB
F4
05 F4EF 62703             DCR B   decrement B
C2 F4F0 62704             JNZ     jump if B is not zero to MSEC_DLY
E9
F4
        now get the next set of samples

C3 F4F3 62707             JNP     goto GET_DATA
B1
F4
------------------------------------------------------------------------

        a subroutine for a short delay to allow channels to settle

05 F4F6 62710 SHRT_DLY: DCR B   decrement B
C2 F4F7 62711             JNZ     jump if B not zero to SHRT_DLY
F6
F4
C9 F4FA 62714             RET
```

APPENDIX C - PARTS LIST

Part	Value	Part Number
C1-C2	10mF	Electrolytic, 16 volt
C3-C4	100mF	Electrolytic, 16 volt
C5-C6	10mF	Electrolytic, 16 volt
C7	150pF	Ceramic disk
C8-C12	0.1mF	Mallory CK05BX104K
C13	33mF	Kemet T352-F336K-010AS
C22-C35	33mF	Kemet T352-F336K-010AS
C37-C38	0.1mF	Mallory CK05BX104K
RN1	100K	Bourns 4610x-101 100K
R2	10k	Pot
R3	2.7k	5%, 1/4 watt
R4	10K	5%, 1/4 watt
R22-R43	100K	5%, 1/4 watt (8 resistors total)
R82-R86	10K	5%, 1/4 watt
D1		LM 385-1.2 for ±2.5 volt inputs
		LM 336-2.5 for ±5.0 volt inputs
D2-D3		1N914
D4		1N5818
D5-D6		1N914
D10-D11		1N914
U1		National Semiconductor NSC810
U2		74HC138
U3		CD4011B
U4		74HC688
U5		ICL7662
U6		78L05CP
U7		LM7806CK
U8		ICL7662
U9		ADC0803LCN
U10-U17		TL022
U18		CD4051B
U19		LM358