COMPUTER NOTES REVIEW

REPRINTS

VOLUME ONE

April, 1975 through July, 1976

Q and A

Q. What do I need to hook my Southwest Technical Products CT1024 to my ALTAIR?

A. You need a CT-S serial interface from Southwest Tech and a SIOA interface from MITS.

Another possibility that we believe will work is a 88-PIO from MITS and a CT-L parallel interface from Southwest.

Q. What do I do about a short routine or subroutine that doesn't do what I want it to?

A. Send it to our service desk -- we won't rewrite it for you but we will tell you what you are doing wrong.

Q. What kinds of hardware problems have you been experiencing with the 8800?

A. The problems at this point have been only in kits and have been primarily solder bridges and cold solder joints in the ground or power circuits. Point of interest: Of 1000 units in the field, only 13 have been returned to date. Five of these have been at the request of the reair department to investigate ,otentially serious problems.
P.S. They weren't serious.

Q. How many options can I put in my ALTAIR 8800 before I have to beef up the power supply?

A. There are 16 slots in the basic chassis. One of these is taken up by the CPU board. The other 15 can be filled in any manner (memory, I/O, etc.) to satisfy the user's requirements.

Q. How do I use the sense switches?

A. Sense switches are inputted to AREG by an input 303 from DEVICE 377. After this the data may be manipulated in any fashion that you choose.

Q. What does MITS have in the mill as a mass storage device priced between the Audio Cassette and the Floppy Disk?

A. We do have a mag tape system in the planning stages right now, and will be announcing more on it in future newsletters.

Q. How can I contact other Users Group members?

A. If you will send a letter to Barb authorizing her to publish your name to other users, we'll have these names printed in future newsletters.

Q. When am I going to get more information on the floppy disk and line printer?

A. Look at the "New Products" page in this newsletter. (page 4)

Q. I've seen bits and pieces about software available for the 8800. What's the full story?

A. See page 3 for software details and prices.

ALTAIR CHECKOUT PROCEDURE

1. BEFORE APPLYING POWER VISUALLY (WITH THE AID OF A MAGNIFYING GLASS IF POSSIBLE) CHECK FOR SOLDER BRIDGES, COLD SOLDER JOINTS, BROKEN LANDS AND/OR WIRES AND CORRECT ORIENTATION OF COMPONENTS. 97.37452% OF ALL FAILURES CAN BE CAUGHT DURING THIS STEP.

2. With the pluggable boards out of the machine, power it up and check the terminal boards for the proper input voltages.
Check the bus for +5 volts (pins 1 & 51), +15 volts (pin 2), -15 volts (pin 52), and ground (pins 50, 100). With the boards out these should reach as follows:
(Use negative side of power supply electrolytic capacitors as ground.)

Pins 1 & 51	+10 VDC	approx.
Pin 2	+20 VDC	
Pin 52	-17 VDC	
Pins 50, 100	-0V	

Now check that none of these voltages are seen on pins adjacent to the above checked pins.

NOTE: IN NO CASE REMOVE OR INSTALL ANY BOARDS WITH POWER ON.

3. After powering the machine down, install the boards and check the output of the voltage regulators for +5VDC. Also check the output of the 12V zener on the CPU board for +12VDC.

Assuming everything above is kosher, you should now power the machine down; install the CPU plug and power it back up. The machine comes up in an undetermined state, so what you do is hold the STOP switch in the stop position and give it a RESET. Then check to see if it is protected (lit), push the PROTECT-UNPROTECT switch in the unprotect position.

4. Now you can check out the different switches and indicators. All address switches should be in the off position. Hold the RESET on--you should have all the status lights off and all data address lights on. When you release the RESET switch, all the address lights should go off. The MEMR, M1, WO, WAIT lights should be on and whatever data there is in location 0 will be displayed in the data lights.

Now to check the lights and switches for proper operation, turn each address switch on one at a time and make sure that the corresponding address light comes on when EXAMINE switch is operated. What you are doing here is checking for obvious shorts in the address bus area so only one switch should be on at a time.

The data lights should be checked in the same fashion. Only use the lower 8 switches and the DEPOSIT switch to check these.

Checking the EXAMINE NEXT and DEPOSIT NEXT is fairly simple. Just keep pressing them and observe that the address lights count up binarily.

Make sure that PROTECT switch turns on the Protect Status light and that UNPROTECT turns it off. With the PROTECT on you shouldn't be able to change the contents of memory with DEPOSIT or DEPOSIT NEXT (or instructions either).

Now you're ready to try a program. Use the one in the Operator's Manual on pages 33-38. After you load everything in be sure to RESET so that you start from LOC 0. SINGLE STEP through it first to check out the SINGLE STEP switch and then run it. Every time you stop it to examine the results be sure to RESET prior to restarting.

8800 MOD

(1) DEPOSIT PROBLEM (D/C board)
SYMPTOMS: Machine won't deposit at all or deposits all ones when using the front panel deposit switch.
FIX: Change the timing capacitors for the deposit single shot (IC G on the Display/Control board). Change C7 to 0.01MF, C8 to 0.1 MF.

(2) 12 VOLT ZENER (CPU board)
SYMPTOMS: Zener running very hot.
FIX: Change R46 on CPU board to 43 ohms (Use either a 1W or 2W resistor). This should be done only if there are four or less cards on the bus. If there are more than four cards R46 should be 33 ohms.

(3) AC SWITCH (Display/Control board).
PROBLEM: The tracks on the D/C board which connect the AC switch terminals to the pads where the switch wires are connected to the board have 115VAC on them. If these are inadvertently shorted to other tracks on the board several IC's are wiped out.
FIX:(a) remove the AC switch lines from the D/C board.
(b) Cut the tracks leading from the pads where the switch is mounted to the pads where the AC switch wires attach to the D/C board. Cut the tracks as close as possible to the switch pads so there will be no length of track with 115VAC on it.

(c) Slide about 1" of heat shrink tubing (3/32 to 7/64 diam.) over the AC switch wires. Solder these wires directly to the AC switch terminals. Slide the heat shrink tubing down so it covers the switch terminals and uninsulated ends of the switch wires. Heat the tubing once it has been properly positioned to keep it in place.

(4) CLOCK SPECS (CPU Board)
(a) Phase 1 pulse width (measured at 90% points), 60 nanoseconds minimum.
(b) Phase 2 pulse width (measured at 90% points), 220 nanoseconds minimum.
(c) Delay from leading edge (90%) of Phase one to leading edge (10%) of Phase 2. 130 nana-seconds minimum.
(d) Delay from trailing edge (10%) of Phase 2 to leading edge (10%) of phase 1. 70 nano-seconds minimum.

MAINTENANCE

by Paul Van Baalen

A good percentage of the people who have had trouble with their bits have shown this symptom. "All the data lites on at all times". Below is a list of the most common causes:

A. Check the mother board for shorts using VOM.

B. Insure that all regulated voltages are OK.

C. Make sure the memory is installed and the address strapping is correct. Be sure that the connector from the front panel for the data lines is on the CPU.

D. Are all the data lines high on the D/C board? If they aren't, check the continuity of the CPU connector.

E. Check the timing of 01 &02.

F.Check the drivers J&H on the static memory board. These tri-state drivers need a low on pins 1&15 to pass a signal. If 1&15 are high the outputs of these drivers will be high all the time.

G. See if line 68 on the bus is high all the time. If it is, backtrack thru the logic. Most probable cause is ICG on the front panel being bad or a short or solder bridge in the vicinity of ICG.

H. Check IC's U&T on D/C board to insure the signal levels are correct,i.e.; no 1.5 volt levels as opposed to ≤.5V or ≥4+VDC.

ARROWHEAD TIPS

The following tips are on construction of the Altair 8800. Page numbers refer to the Assembly Manual.

Page 11
(D/C Board): Capacitor C7 has been changed several times; you may find change notices referring to various stages of this change process, as well as extra parts. C7 was changed from .001 MFD to .0047 MFD, then to .0068 MFD. Now, the absolute last final ultimate change (as of August 10th!) makes C7 = .01 MFD and changes C8 from .01 to .1 MFD.

Page 18
(D/C Board): Don't bolt the printed circuit board to the sub-panel; the switches will hold it fine. The switches come with extra mounting nuts and extra guide washers; you can safely throw away all this extra hardware - you don't need it. Mount the switches as shown with only one nut each for best results.

Page 39
(CPU board): The wafer connector is about 5% too large to fit the board (or you might say the holes in the board are too close). To prevent the connector plug from arching, clamp it down flat while soldering it on. If you can't clamp it, then try cutting it with a hacksaw into two 4-pin connectors.

(Checkout): After turning the computer on you should reset it. To reset the computer you have to hold the stop switch raised while raising the reset switch. Release the reset switch first. (No, we don't know why, but it's traditional!)

Pages 40-46
(1K Memory): Until May, only the 1K memory board was available, and most systems were ordered with 256 words of memory on one of these boards. Now that the 2K and 4K memories are available, it isn't sensible to require some 1K boards in every system, but the instructions are still embedded in the CPU manual. CPU kits don't have any memory in them.

Page 50
(P/S Board): The bridge rectifier seems to cause more than its share of problems. Be sure the leads are clean - several of us have found that solder won't wet the leads, and it's a mess to try to clean the partially soldered assembly. Run the leads through some slcohol and/or steel wool before installing the bridge. The spacer-and-washer arrangement on Page 51 is a jig to get the bridge flat at the right position it will later be bolted directly to the chassis.

Page 58
(Chassis): MITS doesn't ask you to cut wires to close tolerances. If you follow the instructions here without trimming transformer leads, and use all the #20 wire, you'll have long loops of slack. This is good for allowing boards, etc. to be moved without breaking wires, but you may want to install the terminal lugs after consulting the wiring diagrams on Pages 59 and 62.

Page 66
(Mother board): The way the instructions spell it out, you'll have a slack loop of cable. Hold everything in place on the chassis to see the actual length required. We are including a sorted list of wires to help you check your progress. Using masking tape, group the wires in decades after protecting them with a cable clamp. Then, install them on the motherboard, by decades; 50's, 60's, 70's, 20's, 80's, 30's, 90's, and finally 40's.

Install the cable clamps by bolting them to the printed circuit board only. If you put screws through the sub-panel, then the dress-panel won't fit flush against it and you won't be able to screw the chassis into the case!

Page 68
(Expander Boards): The card guides are maybe sorta optional. They look nice, but they really aren't required to hold the boards in place - the edge connectors are plenty strong for that.

Page 74
(CPU Chip): Many people advocate postponing installation of the CPU chip until after the regulator and zener diodes on the CPU have been tested.

Page 77
(Nameplate): This beauty gets a lot of criticism: "Mine was off-color, kinda pinkish." That's a sticky plastic cover to protect it until after you've installed it. Peel the covering off afterward. The white lettering on the dress-panel can be chipped off by hard use. If you decide to protect it with clear acrylic spray, use a matte-finish product. Ours looks funny with a glossy krylon finish.

DATA/CONTROL BOARD CONNECTIONS TO SYSTEM BUS

ORGANIZED BY DECADE

0's	10's	20's	30's	40's
		20	30	41
		21	31	42
		24	32	43
		26	33	44
		27	34	45
		28	37	46 (1!)
		29	39	47
				48

50's	60's	70's	80's	90's
53	68	70	80	91
54	69	71	81	92
		72	82	93
		75	83	94
		76	84	95
		77	85	96
		78	86	97
		79	87	98
				99

MEASURING INTERRUPT ACTIVITY

by NORMAN CROWFOOT

Altair User Devises Interrupt Monitor

"During the course of completing a process-control application utilizing the Altair computer, it became necessary to obtain certain measurements of interrupt activity. I devised and implemented a hardware/software mechanism in which Computer Notes readers may be interested.

"Briefly, a four-chip circuit was constructed on a prototype card and four instructions were added to the general interrupt service routine. This mechanism is further detailed below."

Sincerely,

Norman C. Crowfoot
Lowell Observatory
Flagstaff, AZ 86001

Interrupt Monitor

General Description:

In order to accurately measure certain time-dependent interrupt parameters on a heavily-loaded Altair 8800 system, the following hardware/software additions were made.

The hardware additions consist of four integrated circuit (IC) packages on a prototype board. The function of this circuit is to latch data written by the central processing unit (CPU) to a specific memory address area. This data is further used to gate a 1 MHz pulse stream.

The software additions consist of four additional instructions inserted into a general interrupt routine. These instructions set and reset the hardware data latch.

Together these additions allow the accurate measurement of the following parameters:

- interrupt rate;
- interrupt count;
- microseconds used to process interrupts;
- percentage of total machine cycles used to process interrupts;
- interrupt response latency time

Hardware Description:

Constructed on a prototype card, the four-package circuit is diagrammed in Figure 1. The circuit responds to all memory writes with address bit 15 high; that is, all addresses greater than or equal to X'8000'. The data is latched from data line zero into IC C, a D-type flip-flop. Test point A reflects the status of this latch.

IC D continuously divides the 2 MHz ϕ_2 clock to yield a 1 MHz square wave. This 1 MHz pulse stream is then gated to test point B by the current setting of IC C.

Software Description:

The general interrupt service routine is listed in Figures 2 and 3.

Figure 2 lists the memory locations to which the Vectored Interrupt (VI) board forces the CPU to execute. A typical routine (INT1), first pushes the B/C and D/E register pairs, and then calls the general interrupt service routine (LEVEL). Following the call are three bytes of parameters for LEVEL. First is the new level mask work for the VI board, then two bytes of the address of the specific interrupt handling subroutine for the level.

Figure 3 lists LEVEL, the general interrupt service routine, and BACK, the general return routine. LEVEL completes the saving of registers, pushes the current status of the VI board (CLMASK) and sets the VI board up for the new level. Note that at this time, all previous context is pushed on the stack and that interrupts may now be enabled. The ';;;' notation in the remarks field indicates instructions for which interrupts are masked off.

LEVEL then fakes a call on the specified handler subroutine, by pushing the address of BACK and then pushing the handler address. Finally a return (RET) instruction is executed, causing a branch to the handler subroutine.

The code at location BACK is entered when the handler routine executes a RET instruction, causing the address pushed by LEVEL to be returned to. First BACK restores the VI board and location CLMASK to their previous values. Then all registers are restored and a RET instruction is executed to return control to the interrupted code.

This technique of handling interrupt context changes allows interrupts to be nested to an indefinite level. In fact, interrupts are allowed at all times, except when the VI board is being updated.

In Figure 3, the four additional instructions have been marked. They simply store a '1' at location X'8000' upon entry to LEVEL and set X'8000' to '0' upon exiting BACK.

Measurement Procedure:

Referring to Figure 1, the following procedures are used to measure various timings and counts:

- interrupt rate - attach frequency meter at test point A;
- interrupt count - attach event counter at test point A;
- microseconds used to process interrupts - attach event counter at test point B;
- interrupt response latency time - trigger 'scope on rise of CPU line PINT, measure time to rise of status line SINTA. The lines PINT and SINTA are available on the MITS bus.

Doubtless, there are many other measurements that may be made with this relatively simple setup.

-continued

Measuring Interrupt Activity

General Suggestions and Comments:

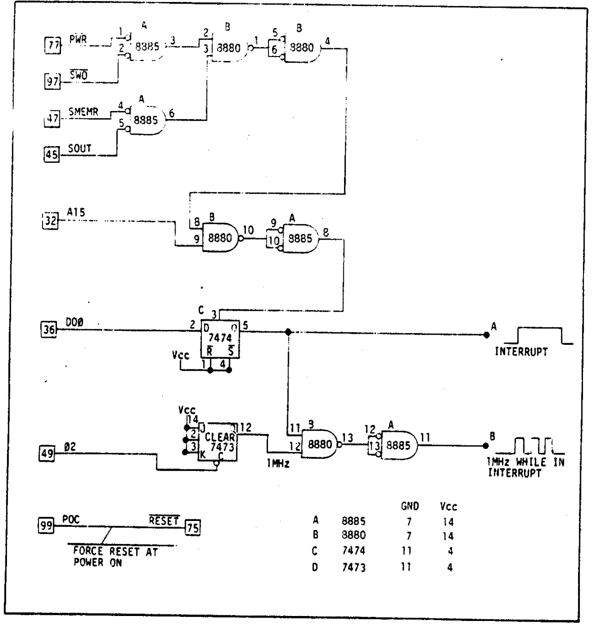
Only half of IC C, the 7474 latch, is used. Another data line may be connected to the unused half for adding further software "hooks" for more complicated measurements. Additionally, some gating logic might be added to combine several latch outputs. For instance, this could be used to determine how often the CPU is interrupted from some specific code or interrupt level.

The Power On Clear (POC), bus line 99, was tied to Reset, line 75, to cause the CPU to automatically start at location zero when power is first applied to the machine. Obviously, some type of involatile memory is required to enable this feature to work properly. And logically, one might wonder why this minor alteration was not included in the original MITS design.

We also found a subtle problem with the Intel 8214 priority interrupt control chip, used on the VI board. While it is described as a latch, it is clocked at 2 MHz and thus appears as a buffer. Too short an interrupt request signal causes the 8214 to develop a level zero interrupt, rather than the one originally requested. The interrupt request line must be held until the CPU grants the interrupt. Beware also of holding the line too long, as multiple interrupts will be generated.

A final point is that the MITS engineers, in their infinite wisdom, have reversed the level numbers on the VI board going into the 8214. This explains the apparent reversal of the VI mask bytes in Figure 2.

This work is an outgrowth of a larger project which has been supported in part by the National Science Foundation grant AST 73-05269 A01 to the Lowell Observatory.



		GND	Vcc
A	8885	7	14
B	3880	7	14
C	7474	11	4
D	7473	11	4

FIGURE 1 - HARDWARE

FIGURE 2: HARDWARE INTERRUPT LEVELS

```
*
*-------INTERRUPT LEVEL 0
        ORG     0                       ;;;ALWAYS AT LOC 0
INT0    LXI     SP,STACK                ;;;SETUP STACK POINTER
RESET   EQU     INT0                    ;;;RESTART POINT
        JMP     LEVEL0                  ;;;CONTINUE WITH STARTUP
*
*-------INTERRUPT LEVEL 1
        ORG     0'10'                   ;;;HARDWARE FIXED LOCATION
INT1    PUSH    B                       ;;;SAVE B/E REG PAIR
        PUSH    D                       ;;;SAVE D/E REG PAIR
        CALL    LEVEL                   ;;;CONTINUE WITH COMMON CODE
        OCT     316                     ;;;NEW LEVEL MASK WORD
        DEF     LEVEL1                  ;;;ADDRESS OF HANDLER SUBROUTINE
*
*-------INTERRUPT LEVEL 2
        ORG     0'20'                   ;;;HARDWARE FIXED LOCATION
INT2    PUSH    B                       ;;;SAVE B/E REG PAIR
        PUSH    D                       ;;;SAVE D/E REG PAIR
        CALL    LEVEL                   ;;;CONTINUE WITH COMMON CODE
        OCT     315                     ;;;NEW LEVEL MASK WORD
        DEF     LEVEL2                  ;;;ADDRESS OF HANDLER SUBROUTINE
*
*-------INTERRUPT LEVEL 3
        ORG     0'30'                   ;;;HARDWARE FIXED LOCATION
INT3    PUSH    B                       ;;;SAVE B/E REC PAIR
        PUSH    D                       ;;;SAVE D/E REC PAIR
        CALL    LEVEL                   ;;;CONTINUE WITH COMMON CODE
        OCT     314                     ;;;NEW LEVEL MASK WORD
        DEF     LEVEL3                  ;;;ADDRESS OF HANDLER SUBROUTINE
*
*-------INTERRUPT LEVEL 4
        ORG     0'40'                   ;;;HARDWARE FIXED LOCATION
INT4    PUSH    B                       ;;;SAVE B/E REC PAIR
        PUSH    D                       ;;;SAVE D/E REG PAIR
        CALL    LEVEL                   ;;;CONTINUE WITH COMMON CODE
        OCT     313                     ;;;NEW LEVEL MASK WORD
        DEF     LEVEL4                  ;;;ADDRESS OF HANDLER SUBROUTINE
*
*-------INTERRUPT LEVEL 5
        ORG     0'50'                   ;;;HARDWARE FIXED LOCATION
INT5    PUSH    B                       ;;;SAVE B/E REC PAIR
        PUSH    D                       ;;;SAVE D/E REC PAIR
        CALL    LEVEL                   ;;;CONTINUE WITH COMMON CODE
        OCT     312                     ;;;NEW LEVEL MASK WORD
        DEF     LEVEL5                  ;;;ADDRESS OF HANDLER SUBROUTINE
*
*-------INTERRUPT LEVEL 6
        ORG     0'60'                   ;;;HARDWARE FIXED LOCATION
INT6    PUSH    B                       ;;;SAVE B/E REG PAIR
        PUSH    D                       ;;;SAVE D/E REG PAIR
        CALL    LEVEL                   ;;;CONTINUE WITH COMMON CODE
        OCT     311                     ;;;NEW LEVEL MASK WORD
        DEF     LEVEL6                  ;;;ADDRESS OF HANDLER SUBROUTINE
*
*-------INTERRUPT LEVEL 7
        ORG     0'70'                   ;;;HARDWARE FIXED LOCATION
INT7    PUSH    B                       ;;;SAVE B/E REC PAIR
        PUSH    D                       ;;;SAVE D/E REC PAIR
        CALL    LEVEL                   ;;;CONTINUE WITH COMMON CODE
        OCT     310                     ;;;NEW LEVEL MASK WORD
        DEF     LEVEL7                  ;;;ADDRESS OF HANDLER SUBROUTINE
*
```

FIGURE 3:

```
        GENERAL INTERRUPT SERVICE ROUTINE, COMPLETE STACK FRAME
        AND UPDATE CURRENT LEVEL ON VI BOARD

        GENERAL FORM OF THE CONTEXT STACK FRAME

                ADDRESS OF RETURN HANDLER ('BACK')
                OLD CLMASK
                PSW
                H/L
                D/E
                B/C
                PC

*
*
LEVEL   XTHL                            ;;;SAVE H/L REC PAIR + GET PARM LIST
        PUSH    PSW                     ;;;SAVE PSW/A REC PAIR
 *{     MVI     A,1                     ;;;SIGNAL WE'RE IN AN INTERRUPT
        STA     IMARK                   ;;;TO EXTERNAL COUNTER BOARD
        LDA     CLMASK                  ;;;GET CURRENT LEVEL MASK
        PUSH    PSW                     ;;;SAVE AS PART OF STACK CONTEXT
        MOV     A,M                     ;;;READ NEW LEVEL MASK
        STA     CLMASK                  ;;;RETAIN FOR NEXT STACK FRAME
        OUT     VI                      ;;;INFORM VI BOARD
        EI                              ;;;OPEN UP FOR MORE INTERRUPTS
*
*-------NOW FAKE SUBROUTINE CALL ON SPECIFIED LEVEL SUBROUTINE
        LXI     B,BACK                  ;;;POINT TO RETURN POINT
        PUSH    B                       ;FORCE ONTO STACK AS A RETURN POINT
        INX     H                       ;BUMP PARM LIST POINTER
        MOV     E,M                     ;READ FIRST BYTE OF ADDRESS
        INX     H                       ;BUMP TO SECOND BYTE
        MOV     D,M                     ;READ SECOND BYTE
        PUSH    D                       ;SAVE JUMP ADDRESS
        RET                             ;THEN GO TO THE ROUTINE
*
*-------RETURN FROM INTERRUPT ROUTINE, RESTORE STACK FRAME
BACK    DI                              ;;;KEEP IT QUIET FOR A COUPLE OF CYCLES
        POP     PSW                     ;;;PULL BACK OLD LEVEL MASK
        STA     CLMASK                  ;;;RETAIN FOR NEXT INTERRUPT
        OUT     VI                      ;;;INFORM VI BOARD
        EI                              ;;;OPEN UP NOW, TRICKLY STUFF'S DONE
  {     XRA     A                       ;;;CLEAR A REG
        STA     IMARK                   ;SIGNAL THAT WE'RE THROUGH
        POP     PSW                     ;RESTORE PSW/A REG PAIR
        POP     H                       ;RESTORE H/L REC PAIR
        POP     D                       ;RESTORE D/E REG PAIR
        POP     B                       ;RESTORE B/C REG PAIR
        RET                             ;THEN EXIT TBIS INTERRUPT LEVEL
*
```

(additional instructions marked '*')

The Mainframe of the Seventies

altair 8800-b TM

The Altair 8800b, now in full production, has generated a large number of inquiries, reminiscent of the response to the original Altair 8800. Because of this, we are devoting a large portion of this month's Computer Notes to technical information on the 8800b.

When the original Altair 8800 first went out into the field, it was by far the most advanced design of its kind. By being at the vanguard of the computer movement, MITS has been in a unique position to assimilate feedback and new information from many sources: from hobby customers, from business users, from computer design industries. All of these influences have been percolating at MITS since the first Altair computer came off the line, and the current result is the Altair 8800b. We feel it will be "the mainframe of the 70's."

As anyone associated with microcomputers will tell you, the field is evolving so rapidly that keeping current is almost a day-to-day job. The Altair 8800b incorporates many new electronic and mechanical features including some of the newer integrated circuits for the 8080 family of microprocessors.

The new design features of the Altair 8800b that will be discussed here include: enhanced front panel capabilities, new Display/Control logic, the Front Panel Interface Board, the new CPU Board, added bus lines and heavy duty power supply.

NEW FRONT PANEL CAPABILITIES

Added Front Panel Switch Functions

Five new front panel switch functions have been added to the Altair 8800b computer to expand the front panel capability:

1. SLOW: Permits execution of a program at a rate of approximately 2 machine cycles per second or slower. The normal machine speed is approximately 500,000 machine cycles per second. Useful in debugging programs where it would be too time consuming to single step through the code.
2. DISPLAY ACCUMULATOR: Displays the contents of the CPU accumulator register on the front panel data LEDs.
3. LOAD ACCUMULATOR: Loads the CPU accumulator register with the data present on the lower eight front panel address switches.
4. INPUT ACCUMULATOR: Inputs the data present at an input/output device into the CPU accumulator register. The input/output device is selected on the upper eight front panel address switches.
5. OUTPUT ACCUMULATOR: Outputs the contents of the CPU accumulator register to a selected input/output device. The input/output device is selected on the upper eight front panel address switches.

Dress Panel

A new multi-color dress panel with functionally designed graphics is used in the Altair 8800b. The front surface of the dress panel has a protective sheet of mylar to insure that the graphics are not rubbed or scratched off. The LED indicators are now back-lit through the panel and the toggle switches have 50% longer handles that are flatted (instead of round) for easier use.

Front Panel I/O Capability

The 8800b has I/O channel 255, and effectively channel 254, dedicated to the front panel. As with the Altair 8800, an input from channel 255 (octal 377) will input the contents of the Sense Switches (A15--A8) to the accumulator. The 8800b has the added feature that an output to 255 will display the contents of the accumulator on the data LEDs. In addition, one can configure this I/O channel (by means of patching jumpers) so that all outputs (to any I/O channel number) are shown on the data LEDs and/or all inputs (from any I/O channel number) are shown on the data LEDs.

NEW DISPLAY/CONTROL LOGIC

Electronically the Display/Control Board has been completely redesigned. The logic design is now totally synchronous. The design approach used in the Altair 8800b is to allow the Display/Control logic to assume control of the CPU and jam the instructions necessary to implement the Front Panel functions. For example: To implement an EXAMINE, the Display/Control Board causes the CPU to execute a jump (JMP) to the address selected on the front panel address switches (AØ--A15).

In order for the Display/Control Board logic to jam instructions to the processor (that is, cause the processor to execute a specific series of instructions), two things are necessary:

1. The Display/Control logic must have control of the processor READY line (RDY). (See section entitled "New Bus Lines.")
2. The Display/Control logic must have access to the processor data bus.

If these two conditions are satisfied, the Display/Control logic can cause the processor to execute a series of instructions by successively placing the instructions on the data bus and activating the READY line to cause the processor to execute the required instructions.

The block diagram, Figure 1, summarizes the interface between the Display/Control logic, the CPU, and the Memory and I/O. On the block diagram, note that:

1. The Display/Control logic has control of the READY (RDY) line.
2. The Display/Control logic has access to the data bus through its own data input drivers (FDIØ--FDI7). By activating the Bus Control signal, it can enable its own drivers and disable the standard data input drivers (DIØ--DI7) from the memory and I/O.

-continued

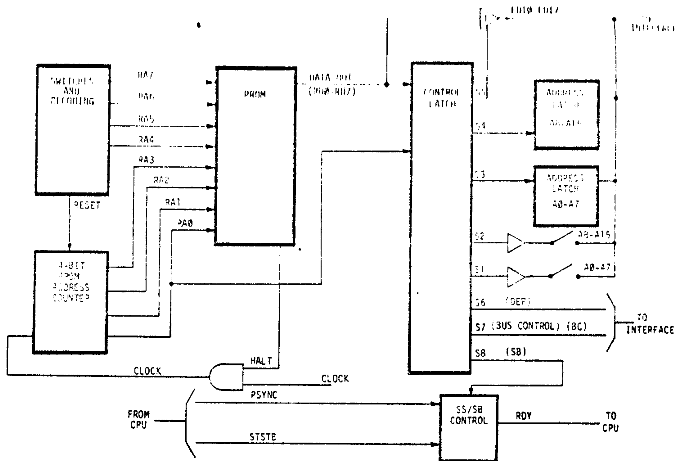


FIGURE 2. DISPLAY/CONTROL LOGIC BLOCK DIAGRAM

The block diagram, Figure 2, shows the Display/Control logic itself (from a functional block viewpoint).

1. The front panel switches are debounced and the examine, deposit, accumulator and I/O function switches (8 switches) are encoded to the upper four address lines (RA7--RA4) of the control PROM.

2. The outputs from the RUN, STOP, SINGLE STEP and SLOW switches go (in pairs) to similar circuits whose outputs (RUN and SS) control the RDY line ([RDY] = [RUN] OR [SS]). (See Figure 3.) Both of these circuits consist basically of flip-flops which, when set, force the outputs (RUN, SS) high and, when reset, force the outputs low.

   The RUN/STOP flip-flop is set asynchronously as soon as the input from the run switch (S[RUN]) goes high. This in turn causes RDY to go high and the processor will start to execute. The flip-flop will reset when the input from the stop switch (S[STOP]) goes high and the following stop condition is true:
   STOP COND =
   (PSYNC) AND (DO5 = SM1) AND (STSTB).

   This insures that the processor will stop during the M1 machine cycle of an instruction cycle.

3. The SS/SB circuit's flip-flop is set by the switch inputs from SINGLE STEP or SLOW, or by the SB input from the control PROM. This circuit, however, has two sets of stop conditions. One is associated with the SINGLE STEP/SLOW switch and one with the control PROM that generates the Front Panel functions. The latter will always stop execution after a single machine cycle has been executed. The former can be configured via patching jumpers to stop execution after either a single machine cycle or a complete instruction cycle.

4. The 8 switches which are encoded as addresses to the control PROM represent those functions that are implemented by jamming instructions to the CPU:

   EXAMINE
   EXAMINE NEXT
   DEPOSIT
   DEPOSIT NEXT
   LOAD ACCUMULATOR
   DISPLAY ACCUMULATOR
   INPUT
   OUTPUT

   Pressing one of these switches causes a unique address to be set up on the upper four address lines of the PROM (thus selecting a 16-byte sector within the PROM). At the same time, the 4-bit PROM address counter is cleared and clock pulses are applied to its input. This causes the lower four PROM address lines to begin counting from zero and continue sequentially through the 16 bytes of the selected sector. This will continue until a stop code is encountered in the PROM which will stop the address counter. The instructions stored as data in the PROM may be roughly divided into two categories:

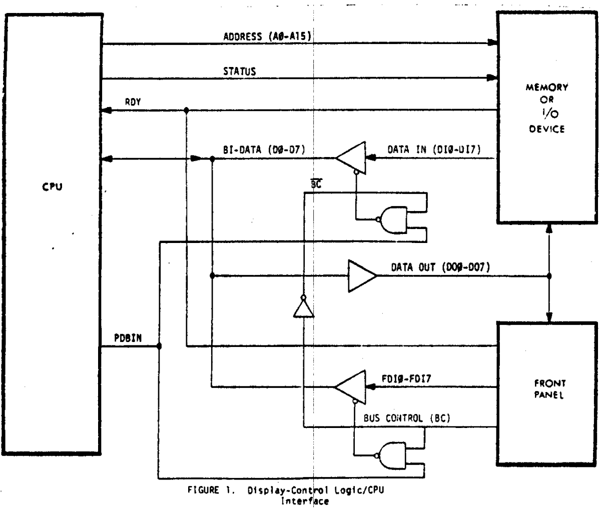


FIGURE 1. Display-Control Logic/CPU Interface

-continued

D/C Logic Control instructions at the even address locations;

"Processor" instructions at the odd address locations.

The D/C Logic Control instructions are output from the PROM and stored in the CONTROL LATCH. These instructions configure the D/C logic so the subsequent "processor" instruction can be jammed to the CPU. In general this will involve setting the SB and BUS CONTROL bits and selecting the source of the "processor" instruction. It must be noted here that by "processor" instruction we mean any of the bytes that may be required to make a complete 8080 instruction (not just the OP code). There are five sources for "processor" instructions:

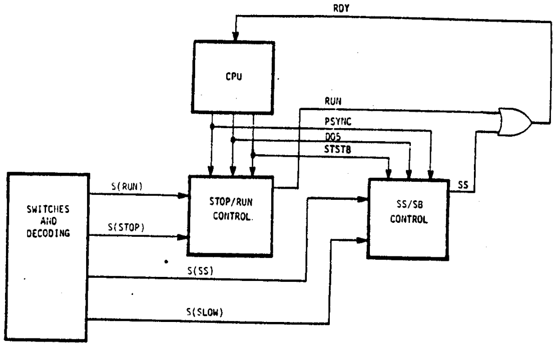


FIGURE 3. DISPLAY/CONTROL LOGIC READY LINE CONTROL CIRCUITRY

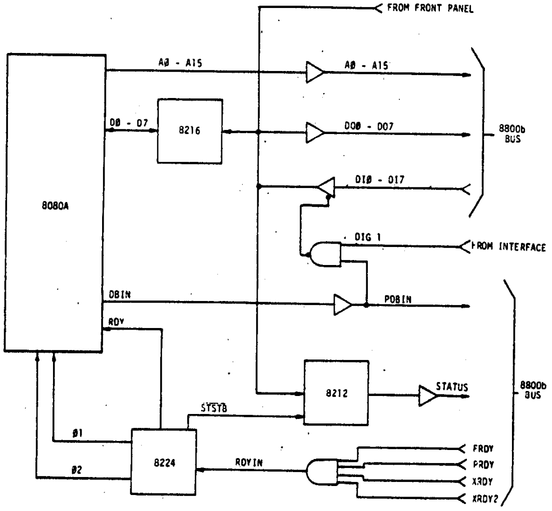


FIGURE 4. CPU BLOCK DIAGRAM

CONTROL PROM (via tri-state drivers), enable: S5

Upper address switches (A15--A8), enable: S2

Lower address switches (A7--AØ), enable: S1

Upper address latch, enable: S3

Lower address latch, enable: S4

A typical PROM sequence to complete an examine would be as follows:

1) Control Instruction: Set up SB, BUS CONTROL (BC), S5

2) "Processor Instruction": JMP = 303 (octal)

3) Control Instruction: Set up SB, BC, S1

4) "Processor Instruction": 000_8. The contents of PROM are immaterial here since data is coming from address switches AØ--A7.

5) Control Instruction: Set up SB, BC, S2

6) "Processor Instruction": A8--A15

7) Control Instruction: Clear Control Latch

8) "Processor Instruction": Stop Code for PROM address counter

FRONT PANEL INTERFACE BOARD

All the lines between the 8800b bus and the Display/Control Board are now buffered through a Front Panel Interface Board. (The bus lines no longer directly drive anything on the Display/Control Board.) The Front Panel Interface Board connects to the Display/Control Board by means of two 34-conductor ribbon cable assemblies, eliminating the wiring harness between the Display/Control Board and the bus.

-continued

NEW CPU BOARD

The CPU Board consists of four major functional blocks:

8080A CPU Chip
8224 Clock Generator Chip
8212 Status Latch
Drivers and Receivers

The diagram, Figure 4, shows the relationship between these four blocks. Several points of interest are:

1. The DIG1 signal (see section entitled "New Bus Lines") controls enabling of the input data drivers (DIØ--DI7) from the bus.

2. The ready input to the 8224 (RDYIN) is the logical product of (PRDY) AND (FRDY) AND (XRDY) AND (XRDY2).

3. The bidirectional data bus to (and from) the 8080A is completely buffered (8216s).

The 8080A, the microprocessor chip itself, exercises control over the CPU board and the rest of the system. It executes the instructions stored in memory and controls all the data transfers.

The 8224 clock generator chip provides the two-phase clock (at the specified voltage levels) required by the 8080A. In addition, it synchronizes the READY and RESET inputs to the 8080A and provides a status signal ($\overline{\text{STSTB}}$) that can be used to load the 8212 status latch. This guarantees that status data will be available as soon as possible in a machine cycle. The master timing reference for the 8224 is an external crystal (18MHz). By changing this crystal it is possible to generate the clocks used by the faster versions of the 8080A: the 8080A-1 (1.3us cycle time) and the 8080A-2 (1.5us cycle time).

The 8212 status latch outputs the status signals that define the current machine cycle for all devices attached to the bus. The status latch was used in the 8800b instead of the 8228 bus controller because it was necessary to maintain bus compatibility with the original Altair 8800.

The majority of the system bus lines either originate or terminate at the CPU board. All output lines from the board are driven by tri-state bus drivers (74367 or 74368).

ADDED BUS LINES

All of the original Altair 8800 bus lines have been maintained, and five new lines have been added:

Bus Number	Signal
12	XRDY2
58	FRDY
55	RTC
56	$\overline{\text{STSTB}}$
57	DIG1

XRDY2 and FRDY

XRDY2 and FRDY are additional ready inputs to the CPU Board. Formerly, the READY signal consisted of two inputs, PRDY and RDY. The READY signal input to the processor that determines the RUN/WAIT state of the 8080A is now defined as the logical product of these four signals:

READY = (PRDY) AND (FRDY) AND (XRDY) AND (XRDY2)

Therefore, if any of the four "ready" signals on the bus are pulled low, the READY input to the 8080A will go low, causing the CPU to enter a series of .5 microsecond wait states. The four "ready" signals on the bus are used as follows:

PRDY: Used by memory and I/O to synchronize the CPU to slower memory or I/O

FRDY: Used by the Display/Control logic

XRDY and XRDY2: External ready signals. XRDY and XRDY2 are available to devices that have to stop the processor (by pulling READY low), but must also be able to sense the state of PRDY and FRDY. (For example: DMA)

RTC

RTC is a 60Hz signal used as a timing reference by the Real Time Clock/Vectored Interrupt Board.

$\overline{\text{STSTB}}$

$\overline{\text{STSTB}}$ is a strobe signal provided by the 8224 clock generator chip. Its basic function is to strobe the 8212 status latch to allow status signals to be set up as soon as possible. This signal is also used by the 8800b Display/Control logic.

DIG1

DIG1 is a signal that controls enabling of the CPU Data Input (DI) drivers. The 8800b employs two sets of DI drivers: one is the standard set used by all memory and I/O devices; the other is used exclusively by the Display/Control logic. If G1 is defined to be the enable signal for the first set of drivers and G2 to be the enable for the second set, then:

G1 = (DIG1) AND (PDBIN)

G2 = ($\overline{\text{DIG1}}$) AND (PDBIN)

POWER SUPPLY

Specifications: The power supply furnishes the following voltages to the 8800b bus at the indicated full load currents.

8 volts at 18 amps

+18 volts at 2 amps

-18 volts at 2 amps

The +18 and -18 volt supplies are pre-regulated (series pass transistor) to provide a constant voltage to the bus over the load range of the supplies (0 - 2 amps).

The +8 volt supply is not pre-regulated. Instead, the 8 volt secondary of the transformer is tapped at 3 points. By changing the tap that drives the 8 volt supply, the bus voltage can be maintained between 7.5 volts and 9 volts over a load range of 1 amp to 18 amps.

The primary of the power transformer is tapped to allow for either 115 volt AC or 230 volt AC operation. In addition there are "HIGH LINE" and "LOW LINE" taps for 130 VAC, 100 VAC, 260 VAC and 200 VAC operation. The supply will operate at the above specifications on either 50Hz or 60Hz line frequencies.

-continued

[MISCELLANEOUS]

18-Slot Motherboard

The four-slot expander cards in the Altair 8800 have been replaced by a single piece 18-slot motherboard. The 18-slot motherboard contains 10C solder lands which comprise the 100 pin bus. The need for expander board wiring has been completely eliminated. Assembled units may be ordered with 6, 12 or 18 edge connectors.

Single Step/Slow

Single Step: The 8800b has provisions for selecting one of two modes for the single step operation by means of a patching jumper. In the first mode a single machine cycle will be executed each time the switch is activated. The second mode allows a complete instruction cycle to be executed.

Slow: The SLOW mode on the 8800b will operate in the same manner as single step as far as the mode is concerned. The speed of the slow mode is selectable by patching jumpers for three different speeds.

Data LEDs

The front panel data LEDs are driven (in the STOP mode) by the Data Out lines (DOØ--DO7). (In the Altair 8800 they are driven by the Data Input lines, DIØ--DI7.) If single step is operated in the single machine cycle mode, the correct data will be displayed on the LEDs during memory write and output machine cycles.

RESET Switch

The RESET switch on the front panel has provisions for wiring to the front panel switch enable line (instead of to ground). If this is done, the machine can be RESET only in the STOP mode.

Control PROM

The front panel control PROM (1702A) has been divided into 16 sectors, each 16 bytes long. The even addresses within any sector are used to control the front panel circuit. Since the last address must contain a stop code for the PROM Address Counter, there are 7 bytes available in each sector for machine code. This means that there is some flexibility in redefining the front panel switch functions (for special applications) by re-programming the control RPOM. The functions are constrained by the fact that there are only 7 bytes of machine code available to execute them.

altair 680-b Hardware Notes

by Steve Pollini

A. Teletype Interconnect

Some 680b users are having hassles when connecting a Teletype to the Altair 680b computer. All of the necessary connections within the computer are clearly explained in the 680b Assembly Manual. However, the terminal strip connections within the Teletype machine itself are not described. The diagram below shows the necessary interconnections between the 25-pin connector on the back panel of the Altair 680b and the 9-lug terminal strip inside the Teletype.

680b 25-pin conn.	pin 2	pin 3	pin 4	pin 5
	↕	↕	↕	↕
Teletype 9-lug term.	lug 6	lug 7	lug 3	lug 4

The Teletype must be wired for 20 ma current loop and full duplex operation.

B. RESET Function

Another aspect of 680b operation that has caused some concern is the RESET function. It is necessary to turn off the computer in order to have it RESET and RUN properly after the processor has entered an undefined state (due to the processor attempting to execute an invalid instruction). The reason that the system cannot be RESET once the processor enters this undefined state is that the BUS AVAILABLE (BA) signal line will not go high when an attempt is made to HALT the processor. This, in turn, prevents enabling of the front panel RESET function. BA was originally used for front panel enabling because it is valid (high) only when the processor is halted. This prevents the front panel RESET and DEPOSIT functions from interfering with the system while it is running a program.

To permit your computer to perform a master RESET even when the MPU will not halt while the RUN/HALT switch is in the HALT position, perform the following modification to the front panel display board:

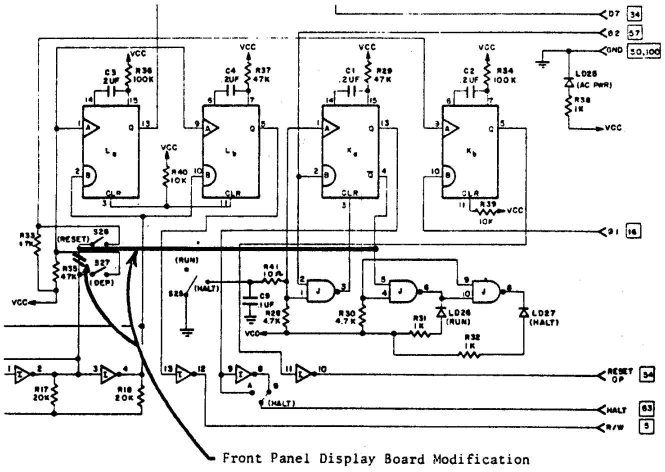
1. Delete the signal line that connects the center pins of the RESET and DEPOSIT switches.
2. Delete the signal line from IC I (pins 2 and 3) to the center pin of the RESET switch.
3. Since step 2 also deletes the signal line from the DEPOSIT switch, it is necessary to connect a jumper from the land of IC I (pin 2) to the center pin of the DEPOSIT switch.
4. Connect a jumper from IC K (pin 4) to the center pin of the RESET switch.

The diagram below shows the modification schematically.

Now the RESET function of the front panel is no longer contingent upon the state of the BA signal line, but rather upon the state of the RUN/HALT signal line.

In normal operation, IC K_a (pin 13) is high when the processor is in the HALT mode. Pin 4 ($\overline{Q}$) of IC K is then low. Since a low signal is necessary at IC L_a (pin 1) to effect a RESET, the $\overline{Q}$ output of K_a (pin 4) is fed to the RESET switch. When the processor is in the RUN mode, IC K_a (pin 13) is low and the $\overline{Q}$ output (pin 4) is high. With this high signal present on the RESET switch, no RESET can be effected.

If you wish to initiate a master RESET while the processor is in the RUN or the HALT mode, I recommend following modification steps 1, 2, and 3 above. Instead of jumpering as in step 4, jumper the center pin of the RESET switch to a ground land on the board. Remember, however, that when a RESET is effected while the processor is in the RUN mode, it will then jump back to the Monitor regardless of what program it was previously running.



Front Panel Display Board Modification
(680 Display Schematic, Sheet 4 of 4)

ALTAIR FLOPPY DISK

by Tom Durston

Available in August, the MITS Altair Disk will enable the Altair 8800 to function as a really sophisticated computer system. The disk offers the advantage of nonvolatile memory (doesn't "forget" when power turned off), plus relatively fast access to data (3/4 sec -- worst case).

The Altair disk can be separated into three parts as follows:

1) Altair Disk Controller

This part consists of two PC Boards (over 60 I.C.S.) that fit in the Altair chassis. They interconnect to each other with 10 wires and connect to the disk through a 37-pin connector mounted on the back of the Altair.

Data is transferred to and from the disk serially at 250K bits/second. The disk controller converts the serial data to and from 8-bit parallel words (one word every 32μsec). The Altair CPU transfers the data, word by word to and from memory, depending on whether the disk is reading or writing. The disk controller also controls all mechanical functions of the disk as well as presenting disk status to the computer. All timing functions are done by hardware to free the computer for other tasks. Since the floppy disk itself is divided into 32 sectors, a hardware interrupt system can be enabled to notify the CPU at the beginning of each sector.

Power consumption is approximately 1.1 amperes from the +8v (VCC) line for the two boards.

2) Disk Drive and Multiplexer

A PERTEC FD400 is mounted in an Optima case (5 1/2" high--same depth and width as computer) and includes a power supply PC board and a buff/multiplexer PC board. A cooling fan is provided to maintain low ambient temperature for continuous operation.

The disk drive has two 37-pin connectors on the back panel, one is the input from the disk controller, the other is the output to additional disk drives. Up to 16 drives may be attached to one controller, and it is possible to have more than one controller in an Altair.

The following are specifications on the disk drive:

→ Rotational speed 360 rpm
(166.7 ms/rev)

→ Access times

track to track	10 ms
head settle	20 ms
head load	40 ms
average time to read or write	400 ms

→ Head life - over 10,000 hours of head to disk contact

→ Disk life - over 1 million passes per track

→ Data transfer rate 250K bits/sec

→ Power consumption - 117VAC 80W

→ Disk specifications
hard sector - 32 sectors + index
recommend Dysan 101 floppy disk
77 tracks

3) Altair-Disk Format & Software

We use our own format, allowing storage of over 300,000 data bytes. Since the disk is hard sectored (32 sectors for each track), we write 133 bytes on each sector, 5 of which are used internally (track #, CRC) leaving 128 data bytes per sector, 4096 per track.

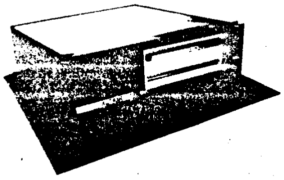
One floppy disk is supplied with each drive, extra floppies are available at $15 each. A software driver for the floppy disk is available at no charge and is supplied with the disk as a source listing.

The disk operating system--which has a complete file structure and utilities for copying, deleting and sorting files--costs extra. The Extended BASIC, which uses random and sequential file access for the floppy disk, will also be available at the same time (late July).

Altair Floppy Disk System

The ALTAIR Floppy Disk System offers the advantage of nonvolatile memory, and relatively fast access to data. The Disk Drive unit (shown below) consists of a PERTEC FD400 mounted in an Optima case. (The case is 5 1/2" high, same depth and width as the computer.) The Disk Drive unit includes a power supply PC board and a Buffer/Address/Line Driver PC Board. A cooling fan maintains the ambient temperature for continuous operation. The Disk Drive cabinet has two 37-pin connectors on the back panel. One is the input from the Disk Controller, and the other is the output to additional disk drives. Up to 16 drives may be attached to one controller.

The Disk Controller consists of two PC boards (over 60 ICs) that fit in the ALTAIR chassis. They interconnect to each other with 20 wires and connect to the disk through a 37-pin connector mounted on the back of the ALTAIR. Data is transferred to and from the disk serially at 250K bits/second. The Disk Controller converts the serial data to and from 8-bit parallel words (one word every 32 usec). The ALTAIR CPU transfers the data, word by word to and from memory, depending on whether the disk is reading or writing. The Disk Controller also controls all mechanical functions of the disk as well as presenting disk status to the computer. All timing functions are done by hardware to free the computer for other tasks. Power consumption is approximately 1.1 amperes from the +8v (VCC) line for the two boards.



ALTAIR FLOPPY DISK DRIVE UNIT

The ALTAIR Disk Format allows storage of over 300,000 bytes. Since the floppy "diskette" is hard sectored (32 sectors for each track), we write 137 bytes on each sector, 9 of which are used internally (track #, checksum) leaving 128 data bytes per sector, 4096 per track. A hardware interrupt system can be enabled to notify the CPU at the beginning of each sector. One floppy diskette is supplied with each drive; extra floppies are available for purchase. The diskette and the disk format are not IBM compatible.

A preliminary version of Disk Extended BASIC has already been shipped, and the production version will be ready to ship on March 29, 1976. Disk Extended BASIC provides the user with complete facility for reading or writing data files and saving and loading program files. All file names are eight character ASCII strings. Three file access modes are allowed: ASCII sequential input ("I" mode), ASCII sequential output ("O" mode), and random mode ("R"). In random mode, the user can read or write the nth 128 byte binary record in the file. Listed below are some of the features of ALTAIR Disk Extended BASIC:

FEATURES

NOTE: Parameters enclosed in brackets [ ] are optional.

MOUNT {disk number}, [{disk number} . . . .]
no argument means all disks
Mounts and initializes for I/O the floppy disk on drive {disk number}.

UNLOAD {disk number}, [{disk number} . . . .]
no arguments mean all disks
Closes all files on {disk number} and disables all I/O on that disk.

KILL {file name}, {disk number}
Deletes the file on the disk specified.

OPEN {mode}, [#] {file number}, {file name}, {disk number}
Opens the file in the mode given on the disk specified. The file is assigned a file number (1-15) for further I/O operations.

Mode is a string formula whose first character must be
O = sequential output
I = sequential input
R = random

CLOSE {file number}, [{file number}]
no argument means all files
Closes the file(s) given.

INPUT # {file number},
{variable list}
Reads the information on the sequential input file {file number} into the variable list specified.

PRINT # {file number}, [USING {string formula};]
{formula list}
Writes the ASCII representation of the internal format on the formulas given on the file {file number}. (Example: PRINT #1, 3) puts a space, 3, space, carriage return on the output file.

-continued

ALTAIR Floppy Disk

LINE INPUT [#] {file number},
{string variable}
Reads the complete character string up to a carriage return, into the string specified. LINE INPUT without a file number may be used to read a string from the user terminal.

GET [#] {file number}, {record number}
Performs a random read of the nth record of the file into the random file buffer associated with {file number}.

PUT [#] {file number}, {record number}
Performs a random write of the random buffer associated with {file number} to the nth record in {file number}.

FIELD [#] {file number}, {numeric formula} AS {string variable},
[,{numeric formula} . . . .]
Associates {numeric formula} bytes in the disk buffer with the {string variable} given. Subsequent CSETs and RSET may be used to place data in the random buffer, while a random read will automatically assign byte strings to string variables that have "FIELDed" into the buffer.

END and NEW both close all files.

LOAD {file name}, {disk number}[,R]
The LOAD command loads a program file into memory from {disk number}. The optional R at the end may be used to RUN the program after it is loaded. The old program and all variables are erased. LOAD can be given in a program.

SAVE {program name}, {disk number}
[,A[{line range}]]
The SAVE command saves the program on the {disk number}. The optional A can be used to save the program file in ASCII source format (using the optional line range). Otherwise, the program is saved in compressed image format, which requires less disk space and loads more quickly.

- continued

FUNCTIONS

MKI$ ({integer formula})
Returns a two byte string containing the binary representation of the {integer formula}.

MKD$ ({double precision formulas})
Returns an eight byte string containing the binary representation of the {double precision formulas}.

MKS$ ({single precision formula})
Returns a four byte string containing the binary representation of the {single precision formula}.

CVI ({string formula})
Returns an integer value which is obtained from the first two bytes of {string formula}.

CVD ({string formula})
Returns a double precision value which is obtained from the first eight bytes of {string formula}.

CVS ({string formula})
Returns a single precision value which is obtained from the first four bytes of {string formula}.

CVI, CVS, CVD file a "function call" error if the string argument is too short.

DSKF ({disk number})
Returns the number of free sectors on {disk number}. The disk must be mounted.

EOF ({file number})
Must be a sequential input file and returns a true (-1) if end of file is detected on {file number}. False (0) otherwise.

LOC ({file number})
Returns the current record number read or written on {file number}. For random files, gives the record that will be accessed if a GET or PUT without a record # is used.

LOF ({file number})
Must be a random access file and returns the last record number written on the random file {file number}. Always = 5 MOD 8.

altair disk specifications

Rotational	360 rpm (166.7ms/rev)
Access Times	Track to track, 10 ms Head settle, 20 ms Head load, 40 ms Average time to read or write, 400 ms Worst case, 1 sec
Head Life	Over 10,000 hours of head to disk contact
Disk Life	Over 1 million passes/track
Data Transfer Rate	250K bits/sec
Power Consumption	117VAC 110W
Diskette	Hard sectored, 32 sectors per track + index, Dysam 101 floppy disk, 77 tracks/diskette

COMPUTER NOTES/OCTOBER, 1975

PROM Memory Card

1. The 88-PMC PROM memory card provides up to 2K bytes of non-volatile memory for boot loaders and other programs that must be retained if power to the computer is interrupted. The card uses either 1702 or 1702A PROMs which contain 256 8-bit bytes. These PROMs are electrically programmable and erasable (using ultra-violet light) so they can be reprogrammed if necessary. The non-erasable versions (1602 and 1602A) can also be used in the card.
2. The card has a power down option consisting of four drivers that switch the PROMs on and off in pairs to reduce current drain. In the "off" state the PROMs draw about 15% to 20% of the current required in the "on" state. Except for some overlap in switching, only two of the PROMs can be in the "on" state at a time.
3. There are provisions on the card for patching the number of wait states from 0 to 3 in order to accomodate different speed devices.

COMPUTER NOTES/MAY, 1976

Product Review

MITS 4K STATIC MEMORY

By Tom Durston

Reports from our customers indicate the work we put into the design of the 88-4MCS was well worth the effort.

The features that make the 88-4MCS outstanding are:

- Solder mask on the soldered side of the PC Board. This helps prevent solder shorts during assembly.
- Dip Switch for address selection. (No address jumpers.)
- Sockets for all logic and memory I.C.s.
- A manual that includes a complete theory of operation and troubleshooting section.

The 88-4MCS has received very favorable comments on its reliability and trouble-free operation. It functions equally well with dynamic memory and our new 16K static memory (88-16MCS).

Other specifications include:

Access Time - 450 ns worst case (300 ns typical)

Power Requirement - +8V unregulated at 1.2A, max.

Memory Array - 4 each 1K x 8 bits (32 ea. 2102 A-4)

from MITS repair dept.

by Jay Miller and Dave Silva

I. TROUBLESHOOTING 4K DYNAMIC BOARDS

A. Addressing

A one-to-one relationship exists between the slope of the address straps (I2, I3, I4, I5) and the switches (A12, A13, A14, A15). With the address switch up, the strap must have positive slope from I to A to enable the board. All straps are at Vcc with the board enabled, and J8 must go low. THE BOARD IS NOW ENABLED.

Within the domain of the board, there are 4096 bytes (locations) addressed by the switches to the right of A12,13,14,15. Twelve pins about the 4060s follow these A0 through A11 switches with respect to the examine switch. The pin configuration for the 4060 is given below:

Source	Signal	4060		Signal	Source
Zener D2	Vbb	(1	22)	Gnd	Buss 50,100
Q9	A9	(2	21)	A8	Q3
Q13	A10	(3	20)	A7	Q5
Q11	A11	(4	19)	A6	Q7
J8	$\overline{CS}$	(5	18)	Vdd	7812 reg +12v
R or Z	DIN	(6	17)	CD	+12v pulse via Q1, Z
To latch	$\overline{DOUT}$	(7	16)	(not used)	
P7 or 09	A0	(8	15)	A5	P9 or 03
P5 or 011	A1	(9	14)	A4	P11 or 05
P3 or 013	A2	(10	13)	A3	P13 or 07
7805	Vcc	(11	12)	$\overline{WE}$	A4

With the appropriate switch raised, the levels at the address pins (above) rise to switched Vcc. A small square imposed on this DC level on the 2, 3, 4 and 19, 20, 21 pins is due to the driver Q being tri-stated by I13-- a function of refresh. Going behind the lines from O are the counters, E and F.

IC E:	pin 3 - 30 usec	IC F:	pin 3 - .5 msec
	pin 2 - 60 usec		pin 2 - 1 msec
	pin 6 - .12 msec		
	pin 7 - .25 msec		

Counters E and F are a function of refresh. Chip O is referred to as the artificial address driver.

B. Protect/Unprotect

Tb is the protect flop. T_b-$\overline{Q}$ is used to control the LED on the panel for protect. Pin 9, if low, allows the $\overline{WE}$ at the 4060 to fall during a write. You may verify this by chips M and A. T is set to protect with a high from M1. A .1uF cap is placed across pin 5 to ground of Tb to prevent noise transients from clocking T. Jumpers 13 and 14 should be checked. Pin 10 should no longer be common with pin 4 (Vcc) but rather with pin 11 (gnd).

C. Deposit Switches to 4060s

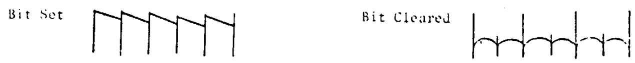
In this discussion, it is assumed that refresh is working properly.

In the Dep mode we may assign a 4060 to each address switch on the right half of the front panel. Pin 6 is the data input to the chip. If a zero is deposited into the chip in question, active high pulses are observed (pulse width approximately .6 millisec wide). Depositing a 1 presents no spikes. The data lines are inverted between the buss and the RAMs since the RAMs invert the data internally.

-continued

from MITS repair dept.

With $\overline{WE}$ active, the following forms at pin 7 ($\overline{DOUT}$) may be seen with the scope at 20 usec/div and at TTL levels:



A feature of the $\overline{WE}$ remains obscure in the stop state. The following program running will smoke it out:

```
0    333     Input sense switches
1    377
2    062     Store at location 100
3    100
4    000
5    303     Loop
6    000
7    000
```

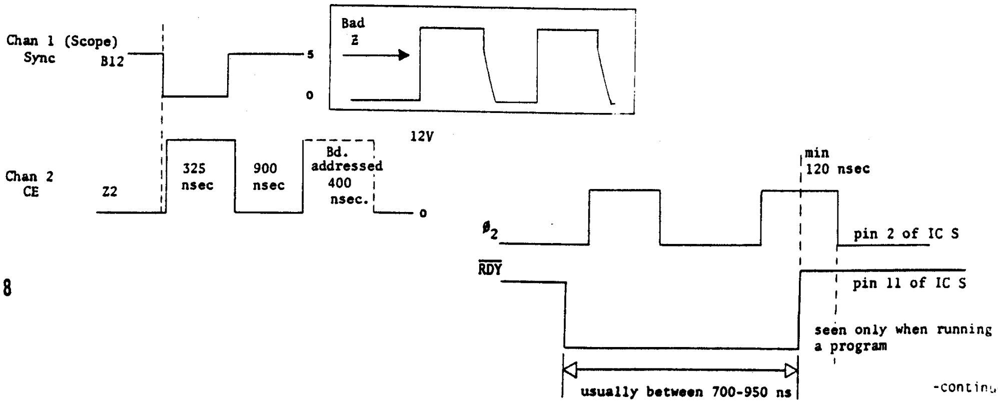
While running the unit you should see the $\overline{WE}$ line pulse low. If the line is shorted to Vcc, no pulses appear.

D. Refresh

Refresh requires reading the memory locations over short time intervals as well as providing an external write and read to the system. The $\overline{READY}$, SYNC, Ø1 and Ø2 serve to synchronize the refresh and read systems. The sync passes through A and should be pulsing only while the machine is running. K3 has negative-going spikes about 32 usec apart and should not have a square wave(due to a faulty G). In static reset state M11 is low and M12 is high due to the initial status of the 8080. On the other side of the Read/Write flop (Hb) check pin 11 of N for a strobe every 32 usec. This pulse is used to load the latch with the data available at pin 7 of the 4060s.

Chip Enable is provided by Q1 (12v level) and is strobed through L from H. The rise and fall time for this level must be minimal, hence the transistor is used. If not, usually Z is at fault (see drawing below). As the board is addressed two pulses appear at Z2 and the second disappears as other 4Ks are addressed. CE must be related to B12 as shown to insure that the addresses are stable before enabling the chips.

If the board is to be used with a Disk, any caps at C3, C4, C5 or the delay between Read/Write and the Latch must not have diamonds on them as the tolerance isn't tight enough. Also R5 and R6 are to be changed to 15K and 30K respectively. These changes should produce the forms below. If not, R5 may have to be reduced further to 13K. In essence, we must catch the Ø2 that has been getting away to leave the wait state and keep up with the Disk.



-continued

from MITS repair dept.

II. DISK PRECAUTIONS—Protecting Floppies
Protecting the Disk Drive

The following "Disk Precautions" will help prevent unnecessary abuse of the floppies or the disk system itself.

The floppies are very susceptable to magnetic fields. Keep them away from flourescent lamps, transformers and soldering irons. Store the floppies free from dust to protect the surface and save on head wear. Marking the floppies should be mandatory.

The drive should be shipped only with the safety board or strap provided. While the board or strap is installed, the door should be fixed shut. Attempting to open the door with the board in the drive will damage it.

Consider the following program:

```
0      333      Input sense switches
1      377
2      323      Out to Disk enable channel
3      010
4      303      Loop
5      000
6      000
```

By raising A8 to A11 we should be addressing another Disk on the Daisy and as a result Disk 0 should be disabled. We may also simulate an open door by raising A15, which will also disable Disk 0.

Suppose that Disk 0 is enabled and we wish to mechanically operate the head and track movement (a scratch floppy is advised until the operator is familiar with the Disk). Change byte 3 of the program from 10 to 11. We will now output to the control channel. Toggling A8 will move the head in toward the center (to track 77). Likewise toggling A9 will move it out. A10 up will load the head and A11 up will unload it.

To protect the floppy, the head should be unloaded and the Disk disabled before opening the door. This is done in BASIC by an OUT 8,255. Now the Disk may be changed quite safely. Remember that BASIC deals in decimal, unless the Q option (octal notation) is used. Using the WAIT command to read status from the Disk is straightforward. Reading and Writing data with the Disk in machine code is more complex due to the sector bookkeeping.

A reminder: the Disk is composed of 77 tracks of 4K per track and rotates at 360 rpm. There are 32 sectors per track and 128 free bytes per sector.

III. DISK ERRATUM

On Disk board #1 there is a track missing that allows unused inputs of the logic to float instead of being tied to V_{HB}. It could possibly affect the sector circuit if noise were picked up by this line. To remedy the problem, connect the top end of R16 (V_{HB}) to pin 7 of IC F2. This is easily done on the bottom (solder side) of the PC board, since a track from F2-7 passes right next to the pad for R16.

New Products

680-b-BSM 16k Static Memory Card

One of the big hits at the 1976 National Computer Conference was the MITS Altair 680b computer with 33K of memory running MITS Altair 680 BASIC. The memory board that made this possible is the 680b-BSM 16K Static Memory Card. Two 680b-BSM cards were used, providing 32K and the internal 1K of memory provided the additional 1K.

Included with the 680b-BSM and the 680b computer is a free copy of MITS Altair 680 BASIC, Assembler and Text Editor on paper tape. Altair 680 BASIC is virtually identical to the Altair 8800 8K BASIC and operates in about 7K of memory.

The 680b-BSM 16K Static Memory Card has many outstanding features, one of which is its extremely modest power consumption of 5 watts or 38 micro watts per bit. This allows operation of two 16K cards without adding a second power transformer. A DIP switch is used for address selection and test points have been installed at important signal outputs for ease of checkout and troubleshooting. Ferrite beads are used on all common supply lines for noise isolation.

For the kit builder, the use of an epoxy solder mask on areas not to be soldered increases ease of assembly, as well as sockets for all ICs, providing easy installation and removal of ICs. All of these features, plus a well-documented manual with step-by-step assembly instructions and detailed theory and troubleshooting sections, make the 680b-BSM the ideal addition to the MITS 680b computer. The care and effort we have put into the 680b-BSM will be appreciated by hobbyists and professional computer users alike.

If you are ordering the 680b-BSM card and it is your first additional card for the 680b, you will need to order the 680-MB Expander Card. The 680-MB holds up to three cards and includes a 100-pin connector for plugging in to the 680b Main Board.

Also, if you intend to add three cards to your 680b, you will need to install a second power transformer to the 680b back panel in the holes provided.

680b-BSM SPECIFICATIONS

RAM Access Time	215ns maximum
RAM Cycle Time	400ns minimum
+16v current	130ma maximum
-16v current	110ma maximum
+9v current	150ma maximum

Altair 680 BASIC, Assembler, Text Editor

Price	Free with purchase of Altair 680b and 680b-BSM

DISK HARDWARE NOTES

By Tom Durston

If you are having difficulties with your 88-DCDD hardware, follow these guidelines for servicing:

A. Controller Boards:

1. On Controller Board #1 be sure the bus strips are soldered on both the top and bottom of the P.C. Board. Do not apply pressure to bus strips after installation.

2. On Controller Board #1 jumper the top end of R16 (VHB) to the track from pin 7 of IC F2 (on back of card). This ties floating inputs of sector logic high to prevent noise pickup.

3. On Controller Board #1 check the track from Pin 9 of IC H1 where it goes through the board on the plated hole. Some P.C. Cards had shorts to the adjacent track on the back of the card.

4. On Controller Board #1 check jumper wires to be sure there are no shorts to bus strips (insulation on wires melted), and check jumper wires for correct wiring.

5. On both Board 1 and 2 check Stab Connector for shorts on fingers. File at an angle along the length of the Stab Connector and the bevel edge of the card to remove any shorts.

6. Be sure all interconnect cables are wired correctly and the pins are making good contact.

7. Check one shot timing on both boards as follows, using the Disk Test Program that appeared in April '76 Computer Notes, pages 12 and 13.

a) Controller Board #1:

FUNCTION	IC and PIN #	POSITIVE PULSE WIDTH RANGE
Read Clock Mask	IC A1 Pin 13	0.7us to 1.2us
Read Data Window	IC A1 Pin 5	2.6us to 2.9us
Sector Pulse Mask	IC E1 Pin 13	150us to 600us
Index Pulse Window	IC E1 Pin 5	3.3ms to 4.5ms
Read Clear	IC F1 Pin 13	130us to 150us
Index Pulse Verification	IC F1 Pin 5	3.3ms to 4.5ms
Sector True	IC F4 Pin 13	20us to 40us
Write Data Enable	IC F4 Pin 5	250us to 300us

b) Controller Board #2:

FUNCTION	IC and PIN #	POSITIVE PULSE WIDTH RANGE
Repeat Step OK (Status)	IC A1 Pin 13	0.4ms to 0.8ms
Step Inhibit 1 (Status)	IC A1 Pin 5	9.5ms to 11.5ms
Head Settle	IC B1 Pin 13	35ms to 70ms
Step Inhibit 2 (Status)	IC B1 Pin 5	17ms to 30ms
Trim Erase Start Delay	IC B2 Pin 13	180us to 225us
Trim Erase End Delay	IC B2 Pin 5	420us to 520us
Disk Enable Timer	IC B3 Pin 13	1.5us to 4.5us
Disk Power Disable	IC B3 Pin 5	1.5us to 4.5us

-continued

c) If the measured time constants are not within the specified tolerance, vary the resistor value for the one shot affected.

d) We have had difficulty using National 74123 ICs for B3 on Board #2. Replace with Signetics or TI ICs if you suspect problems.

8. If you are using 4K Dynamic cards, be sure they are using only one wait state. See May '76 Computer Notes, pages 9 and 10.

9. Check the Power Supply to be sure the negative peaks of the +8V unregulated do not go below +7V.

B. Disk Drive Chassis:

1. On the Buffer Card the most common difficulty is incorrect wiring or incorrectly installed ICs.

2. On the Power Supply Board be sure X1 and X3 are properly installed as indicated on the errata sheet.

3. If you suspect difficulty with the Disk Drive, DO NOT attempt to service it. Any work done on the Pertec FD-400 will void the warranty. Typical service charges for customer damaged FD-400's are $100.00.

4. Do not plug the FD-400 connector in backwards. Be sure to install the polarizing key as the instructions indicate. Plugging in the connector backwards will destroy 5-10 ICs and will cost at least $100.00 for repair.

5. If you must ship the Pertec FD-400 or complete Disk Drive unit, reinstall the Disk door block or strap. Any damage to the mechanism as a result of incorrect shipping typically costs the customer $100.00 in repair charges.

6. Our dealers now have Pertec FD-400 service manuals. If you suspect difficulty with the FD-400, contact your nearest dealer for his advice and service.

7. If you can't remedy the difficulty, don't try to save postage by just returning the FD-400 alone. Please return your complete 88-DCDD including Cables, Controller Boards, and Drive Chassis. This will allow us to check your system out completely and save you time, money, and hassle.

COMPUTER NOTES/SEPTEMBER,1975

Using the VLCT

The VLCT is a very useful octal I/O device. It consists of a 10-key keyboard (0-7, Ready, and Clear), an eight LED input register display and a three digit octal output display. The VLCT interfaces to the Altair through the 88-PIO using TTL logic levels. Data is entered into the Altair by use of a short input program, such as the one listed below.

VLCT INPUT PROGRAM

Stores data in memory starting at address specified by LXI instruction (must be different from input program.)

```
       0     041     LXI     Load H & L with starting address
       1     XXX             Least significant address byte
       2     XXX             Most significant address byte
IST:   3     333     IN
       4     000
       5     346     ANI     Test for input latch loaded
       6     002             (after 3 keystrokes)
       7     312     JZ
      10     003
                             Jump to IST
       1     000
       2     333     IN
       3     001
       4     167     MOV     (Mem H,L) ← (A)
OST:   5     333     IN
       5     000
       7     346     ANI     Test for Ready key
      20     001             pressed
       1     312     JZ
       2     015             Jump to OST
       3     000
       4     176     MOV     (A) ← (Mem H,L)
       5     323     OUT
       6     001
       7     043     INX     (H,L) + 1
      30     303     JMP
       1     003             Jump to IST
       2     000
```

NOTE: For 88-PIO address 0&1

After 3 keystrokes on the VLCT, data is automatically transferred to the input latch, where software stores it in memory. Pressing the Ready key causes the software to echo back the data to the octal display on the VLCT.

By changing the test byte after the first "ANI" you can use the Ready key to trigger storage of data to prevent automatic entry of an incorrect. code.

This program stores octal data sequentially in memory starting at the address specified by LXI. It should be used for writing machine language programs; and, for long programs, it is faster and easier than loading in through the front panel switches.

COMPUTER NOTES/JULY,1976

88-VLCT MOD

88-VLCT READY KEY MOD

PROBLEM:
Pressing "READY" key should cause one strobe pulse to PIO board "SBO" line, causing computer to output data to octal display. Noise from keyswitch bounce causes multiple pulses on "SBO" line, causing next byte entered to be echoed without pressing READY.

SOLUTION:
Change R32 from 10K to 10meg. R32 is across C6, the .01 pulse timing capacitor for the READY key. Increasing R32 to 10meg makes discharge time for C6 greater than 10ms, preventing keyswitch bounce.

NOTE:
READY key schematic is incorrect: R33, 100Ω, shown going to Vcc is actually connected to ground. R31, 47Ω, shown going to ground, actually goes to Vcc.

10meg -- MITS part number 102079

COMPUTER NOTES/OCTOBER,1975

ENGINEERING NOTE:

For proper operation of a Teletype with the 88-SIOC interface board, the Teletype must be internally wired for 20mA, full duplex operation. MITS is currently selling Teletype model #3320-3JE. Instructions for changing to 20mA, full duplex operation may be found in the wiring diagram #9336WD-B2A supplied with the Teletype (this model comes from the factory wired for 60mA, half duplex). Also, be sure that R10 on the 88-SIOC is 220 ohm, not 390 ohm. This resistor determines the 20mA loop current operating the printer part of the Teletype.

USING SERIAL I/O BOARDS

by Tom Durston

The MITS technical staff has been receiving many questions on the writing of software to handle the 88-SIO boards. Most of the confusion has been due to a lack of explanation of the fundamental concept of the Altair I/O structure. We hope the following will help answer users' questions on 88-SIO software.

Inputting data from an external I/O device: The input instruction is a two byte (1 byte = 8 bits) instruction. The first byte (333_8) is the code telling the CPU to input data from an external device (TTY, Comter, CRT terminal, etc.) and put it in the accumulator. The second byte (XXX) is the address of the I/O board connected to the desired external device. The address of the I/O board is determined by the user by seven hardwire jumpers (I1 - I7) on the I/O board. The list of addresses and jumpers is in the back of the I/O board assembly manual. After execution of this two byte input instruction, the input data present at the I/O board is transferred to the accumulator in the CPU. There are 256 I/O addresses, 000_8 through 377_8; the even numbered ones being used for status and control of the I/O board and the odd numbered ones being used for data transfer. This gives the user a maximum possible number of 127 external I/O devices; the 128th I/O device is the sense switch input (address 377).

To input valid received data from an 88-SIO board, the status channel information must be inputted and bit DØ tested. For Rev 1 88-SIO boards, DØ = Ø indicates that a new character has been received from an external device. When status bit DØ has been found to equal Ø, then the data channel may be inputted. Status bit DØ is reset when an input to the data channel is done. From this point the user may do anything desired with the data. A typical input program using address 0&1 would look like this:

```
Byte 1   000    333    IN     Input
     2     1    000           Stat address
           2    017    RRC    Test DØ, rotate Acc right
Byte 1     3    332    JC     Jump if carry (DØ = 1)
     2     4    000
     3     5    000
Byte 1     6    333    IN     Input
     2     7    001           Data address
Byte 1    10    062    STA    Store char in memory loc 000 040
     2    11    040
     3    12    000
Byte 1    13    303    JMP    Jump to beginning
     2    14    000
     3    15    000
```

Outputting data to an external I/O device: The output instruction is a two byte instruction. The first byte (323) is the code telling the CPU to output the data in the accumulator to an external I/O device. The second byte (XXX) is the address of the I/O board connected to the desired external device. When outputting to an 88-SIO board on the even numbered address line (control channel), the interrupt enable circuit may be enabled or disabled by bits DØ and D1 (interrupt is explained later in this article). When outputting to an 88-SIO on the odd numbered address line, parallel data is transferred from the accumulator to the 88-SIO and is transmitted serially to the external device.

When outputting data to an external device, a test of status bit D7 (Rev 1 88-SIO board) must be done to see if the 88-SIO board is ready to transmit a character. When status bit D7 = Ø, indicating the transmitter buffer is empty, then data to be transmitted is placed in the accumulator and outputted to the 88-SIO board on its data channel. A typical output program using address 0&1 would look like this:

```
Byte 1   000    333    IN     Input
Byte 2     1    000           Status Channel
           2    007    RLC    Test D7, rotate Acc left
Byte 1     3    332    JC     Jump if carry (D7 = 1)
     2     4    000
     3     5    000
Byte 1     6    333    IN     Input
     2     7    377           from sense switches
Byte 1    10    323    OUT    Output
     2    11    001           Data Channel
Byte 1    12    303    JMP    Jump to beginning
     2    13    000
     3    14    000
```

This program continuously transmits data from the sense switches to the external device.

-continued

Si/O CONT...

By putting together the input program and output program we can create an "echo" program so that a character transmitted to the Altair will be immediately received back.

```
Byte 1   000   333   IN     Input
     2     1   000          Status Channel
           2   017   RRC    Rotate acc right
Byte 1     3   332   JC     Jump if DØ = 1
     2     4   000          (no data)
     3     5   000
Byte 1     6   333   IN     Input
     2     7   001          Data Channel
Byte 1    10   062   STA    Store data
     2    11   040
     3    12   000
Byte 1    13   333   IN     Input
     2    14   000          Status Channel
          15   007   RLC    Rotate acc left
Byte 1    16   332   JC     Jump if D7 = 1
     2    17   013          (transmit busy)
     3    20   000
Byte 1    21   072   LDA    Load acc with
     2    22   040          stored data
     3    23   000
Byte 1    24   323   OUT    Output
     2    25   001          Data Channel
Byte 1    26   303   JMP    Jump back to
     2    27   000          beginning
     3    30   000
```

Interrupt: Using the Interrupt feature of the CPU allows the Altair to be performing a task other than monitoring the I/O ports for data transfers. To use the interrupt feature, the stack address should be specified, interrupts enabled in the CPU, interrupts enabled on the I/O board, and a subroutine written at the proper location to handle the I/O of data. Using single level interrupt (vectored interrupt hardware will be available by the end of the year), all interrupts will cause the CPU to jump to location 070 where the I/O subroutine should be located. Single level interrupt is utilized by connecting the desired interrupt mode pad ("IN", "OUT", "BH") to the "INT" pad on the 88-SIO board. A short input program that uses single level interrupt would look like this (for I/O board address 0&1):

```
Byte 1   000   061   LXI SP  Set stack pointer
     2     1   200           to 000 200
     3     2   000
           3   373   EI      Enable interrupts
Byte 1     4   076   MVI     Move 001 into
     2     5   001           accumulator
Byte 1     6   323   OUT     Output to I/O
     2     7   000           board control channel
Byte 1    10   303   JMP     Jump to self and
     2    11   010           wait for interrupt
     3    12   000

Byte 1   070   333   IN      Input from
     2    71   001           Data Channel
Byte 1    72   062   STA     Store data in
     2    73   040           loc 000 040
     3    74   000
          75   373   EI      Enable interrupts
          76   311   RET     Return to "jump to
                             self"
```

Interfacing the 88-SIO with the external device: First you must determine what interface your terminal requires. The three standard types available from MITS are EIA-RS232 (88-SIOA), TTY - 20mA loop (88-SIOC), and TTL logic (88-SIOB).

The EIA-RS232 interface on the 88-SIOA board offers only transmitted data, received data, and signal ground (circuits BA, BB and AB). If your external device requires other signals, they will have to be strapped to the proper DC voltages for proper operation. EIA interface levels are: Logic Ø: +5 to +25 volts, Logic 1: -5 to -25 volts.

The TTY - 20mA loop interface is used for Teletypes or I/O devices utilizing that interface level. The 88-SIOC levels are: Logic Ø--open circuit, no current flow; Logic 1--closed circuit, 20mA current.

The TTL logic interface uses 7400 type logic levels. The interface level for the 88-SIOB are: Logic Ø: +.4 volts, Logic 1: +2.2 volts.

Selecting the correct baud rate for your I/O device will depend on its requirements. For instance Teletypes operate at 110 baud only, the Comter operates at 110 or 300 baud, most CRT terminals operate between 110 baud and 9600 baud. (Baud = bits per second)

For Teletype, use 110 buad, 2 stop bits, and 8 data bits. For baud rates of 300 and up, use one stop bit and 8 data bits. Note that above 2400 baud the output pulse of IC "0" must be between 2.0 and 2.5 microseconds for correct operation. Varying R25 (7.5K) will obtain proper timing.

If you wish to check parity, use 7 data bits and either odd or even parity, depending on your requirements.

Selecting the address of the I/O board is up to the user with the following exceptions (used in the BASIC language software):

Address 0/1	--	I/O terminal (88-SIO)
2/3	--	Line Printer
4/5	--	Alternate terminal
6/7	--	ACR
10/11/12	--	Altair Disk
376	--	VI, RTC
377	--	Sense Switch
20,21	-	2SIO I/O Term.

NEW IO's

88-2SIO Interface

The 88-2SIO Interface board can be ordered with one port for just $115 in kit form and $144 assembled. The additional port is $24 for kit and $35 assembled. Each port provides the following features:

1) 5 signal/control lines
   a) transmit data
   b) receive data
   c) data carrier detect
   d) clear to send
   e) request to send

This allows for maximum utilization of some of the more sophisticated terminals on the market.

2) All signal/control lines are user-selectable for ±12 volt levels (RS-232), TTL levels (0-5 volts), or 20 milliamp current loop (Teletype).TM

3) Software programmable for 9 or 10 bit transmission
   a) 7 data bits + parity bit (odd, even or none) + 1 or 2 stop bits
   b) 8 data bits + 1 or 2 stop bits
   c) 8 data bits + 1 stop bit + parity bit (odd or even)

4) Full 8 bit status register
   a) received data available
   b) transmitter buffer empty
   c) carrier detect
   d) clear to send
   e) framing error
   f) received data overflow
   g) parity error
   h) interrupt request

Allows for greater control and handshaking ability.

5) Transmit and receive interrupts --disable/enable under software.

6) On-board, crystal-controlled clock for any of eight baud rates (with a single jumper): 110, 150, 300, 1200, 1800, 2400, 4800, 9600.

7) Programmable counter can provide other baud rates of 37.5, 75 and 600.

The 88-2SIO with 2 ports can interface 2 serial I/O devices, each running at a different baud rate and each using a different electrical interconnect. That is, the 2 ports can be operating entirely independently of each other (such as an RS-232 CRT terminal running at 9600 baud and a 20 milliamp Teletype running at 110 baud). Five volt power consumption is typically 520 milliamps.

88-4PIO Interface

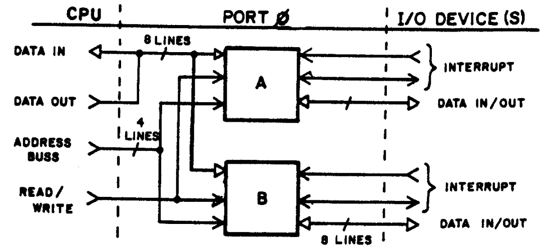
The 88-4PIO Interface board can be ordered with a single port for just $86 in kit form and $112 assembled. Three additional ports are $30 each in kit form and $39 each assembled. Each port provides the following:

1) Sixteen data lines
   a) Each line can be initialized as an input or output so that a single port can interface a terminal (8 lines in--8 lines out) or 2 input devices (such as paper tape reader and keyboard) or 2 output devices (paper tape punch and printer) or any combination for custom applications.
   b) All data lines are fully TTL compatible and in addition, 8 of the 16 lines can directly drive the base of a transistor switch (1.5v @ 1 milliamp).

2) Four controllable interrupt lines
   a) Enabled/disabled under software control.
   b) Two lines can act as outputs for ready/busy handshake.

3) Two control/status registers--Contains a status bit for each of the four interrupt lines.

4) Removable flat cable connection from board to back panel.



SIMPLIFIED BLOCK DIAGRAM (1 port)

Assuming the board is addressed at location Ø, register selection for port Ø is:

ADDRESS	REGISTER
Ø	Section A - control
1	Section A - data
2	Section B - control
3	Section B - data

Port 1 would be addresses 4,5,6,7
Port 2 would be address 8,9,10,11
Port 3 would be addresses 12,13,14,15

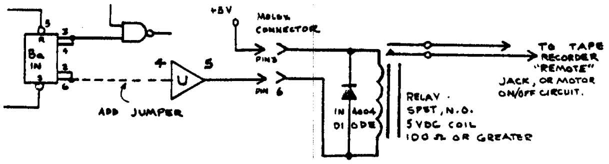
An 88-4PIO with 4 ports has 64 data lines (each group of 8 individually selectable) and consumes 500 milliamps at 5 volts - typical.

...more notes on ACR

Tape Recorder Motor Control

Another request for the 88-ACR has been for start/stop motor control for DC motor tape recorders with the subminiature phone jack marked "REMOTE." A simple way to do this is to utilize an unused flip-flop normally intended for the interrupt function on the 88-SIOB board. Since the interrupt circuit is usually not used in the 88-ACR mode, it is possible to connect the output of one of the flip-flops to an unused driver of an 8T97, using it as a relay driver.

The circuit shown below uses control channel bit DØ to turn the motor on and off. Consult page 3 of the 88-SIOB manual and the 88-SIOB schematic for detailed information on this circuit.



The other half of IC B may be used for another control function in the same manner. IC U has 3 other spare drivers that may be used.

We suggest the relay be mounted externally to prevent recorder supply voltages from interfering with the Altair 8800.

For the machine language Read/Write programs, adding the following instructions will allow use of the start/stop feature.

To turn the motor circuit on, place these instructions before the beginning of the Write and Read programs.

Location	Octal Code	Mnemonic/Description
016,374	076	MVI move immediate to accum.
375	001	Turn on motor
376	323	Output data from accum.
377	006	Control channel

NOTE: **For Write program, single step through these 4 instructions, wait appropriate time (5-15 seconds), then hit RUN.**

To turn the motor off, place these instructions before the jump to self loop at "END." Also change data in location 017,376 to 371.

Location	Octal Code	Mnemonic/Description
017,371	076	MVI
372	000	turn motor off
373	323	Output
374	006	Control channel

NOTE: **The flip-flops Ba and Bb do not have power on clear. It may be necessary to single step the motor off circuit to clear these flip-flops.**

For use with Altair 8K BASIC, use:
OUT 6,1 - to turn motor on
CLOAD or CSAVE
OUT 6,0 - to turn motor off

Keep in mind that if writing, you must turn the motor on 5-15 seconds before outputting data.

DAZZLER featured in POP 'TRONICS

A new Altair-compatible interface, the TV DAZZLER from Cromemco, is being featured in the February 1976 issue of Popular Electronics. Providing an interface between the computer and a TV set, the DAZZLER can be "used to generate action games, animated displays, educational learning drills, graphs, even light shows--all in full color."* Considering its versatility and wide variety of applications, the TV DAZZLER represents a unique and affordable concept in computer peripherals.

The basic kit costs $195 and is designed to plug directly into the Altair 8800 using direct memory access (DMA). There are two PC boards, each taking up one slot on the Altair bus. Board #1 outputs a conventional NTSC (National Television Standards Committee) color video signal, and board #2 communicates with the computer via a high speed DMA controller.

"When writing programs for the DAZZLER, it is important to remember that the TV picture is stored as a specially coded sequence in the computer memory. The DAZZLER simply interprets this code to form the image."* Communication between the computer and the DAZZLER uses two output ports (016 and 017) and an input port (016). Output port 016 turns the DAZZLER on and off and sets the starting address of the picture in the computer memory. The data output from port 017 determines the format of the picture as to normal resolution or 4X resolution, amount of memory to be used for the picture, black-and-white or color, and the color or intensity of each frame of the picture. Input port 016 uses one bit to indicate that the DAZZLER is enabled and one bit to indicate the end of a frame.

Interfacing and construction details are outlined in the PE article, along with a parts list, test program, and an octal listing for a DAZZLER Game of Life.

To obtain the schematics, etching and drilling guide and component placement diagram free of charge, send a stamped (for 3 oz.), self-addressed 9" x 12" envelope to:

Cromemco
One First Street
Los Altos, CA 94022

Prices for the TV DAZZLER:

$195-kit
$215-kit with IC sockets
$350-assembled and tested DAZZLER

NOTE: Static memory is required in the Altair 8800 when interfacing with the TV Dazzler.

*"Build the TV DAZZLER" by Terry Walker, Roger Melen, Harry Garland, and Ed Hall. Popular Electronics, Feb., 1976.

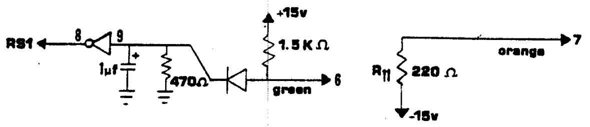
SIOC REV O MOD

In house, the SIOC Rev 0 boards underwent two significant changes. One change, the more apparent of the two, concerns the location of the interface connections between the TTY and the board as well as the nature of the interface electronically. First the position of the pins between the computer and the TTY will be redefined to be more compatible to the Rev 1 cabling. The color code is arbitrary but provided in order that the client is aware of MITS standards.

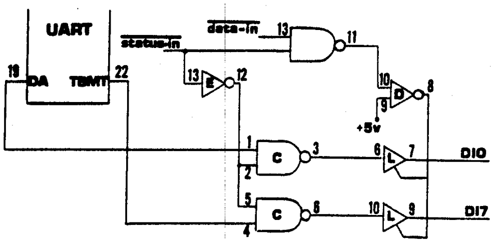
Molex Conn on Board	Color	Male & Female 25-Pin Conn.	Term Lugs in TTY(ASR-33)
4 (GnD)	Black	2	6
5 (Tran)	Red	3	7
6 (Rcv)	Green	4	3
7 (Rcv)	Orange	5	4

It might also be noted that the SIOC board is capable of only functioning with full duplex transmission. That is, a White-Blue lead and a Brown-Yellow lead should be moved from term lugs 4 and 3 respectively to lug 5. Unless some major change has been made in the interface electronics the SIOC board will have difficulty in functioning with a 60mA current loop in the TTY. The desired 20MA loop can be obtained by 1) moving a Purple lead from lug 8 to lug 9. 2) Moving the left-most terminal on R1 (a 4 connection resistor located about 8" back from the line/local switch on the base plate) to the left-most terminal connection of R1 (from the 3rd connection to the 4th connection).

The Rev 0 modified electronic interface and the Rev 1 interface appear below:



The second modification of the Rev 0 board (called hardware interrupt) is in the status provided to the data bus by the UART. The new board will pull DI0 low on the DA high condition (input status) and will pull DI7 low on favorable output status. The logic follows.



Altair 8800 Interfaces

ALTAIR 8800 INTERFACE--THANKS TO PROFESSOR KENNETH B. WIBERG OF YALE UNIVERSITY

Altair 8800 Interfaces

One of a kind interfaces are most conveniently made by wire wrapping, and wire wrapping tools are available at a reasonable price. Most wire-wrap boards are made by inserting wire-wrap IC sockets into a suitable board and making connections on the reverse side. This is inconvenient for two reasons. First, each module will then take two locations on the Altair mother board. Second, it is much easier to wire wrap on the front side of the board (where the IC's can be seen) than on the reverse.

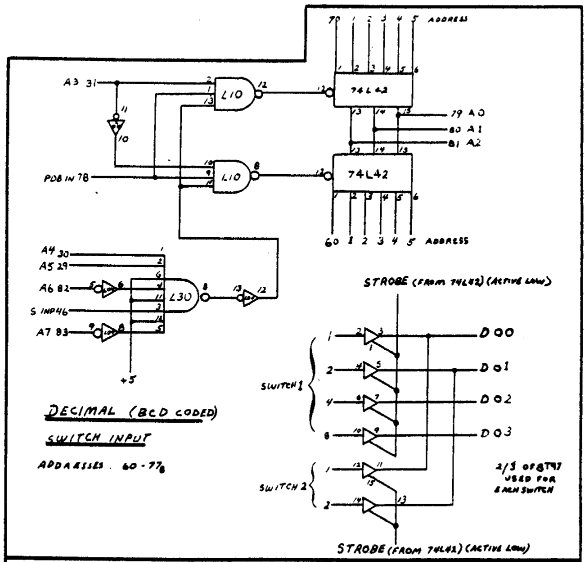
The MITS prototype board can be converted into a wire-wrap board by soldering IC sockets into the places provided, and inserting Vector T-44 mini-wrap terminals from the back side into the holes connecting with the socket pins. The terminals should be soldered from the reverse side. These terminals just fit into the holes provided. Up to 16 sixteen-pin plus 4 fourteen-pin sockets may be placed on the board.

For those who construct I/O boards, an interface which will display the contents of the accumulator is convenient. Such an interface is shown in Fig. 1. The address 377 is decoded by the 74L30, and the output is ANDed with SOUT and PWR by the 74L10. The strobe signal is inverted by a 7404 (which will drive the 7474's) and is used to latch the data on the bus in the 7475 latches. Their output are decoded by the 7447's and displayed using 7 segment LED units. The contents of the accumulator are displayed by including

OUT 377

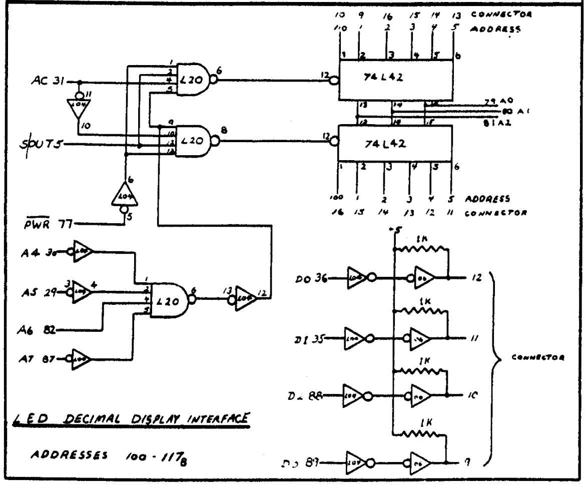
in a program.

In some of our applications, we wish to read data from BCD coded switches. Since the 8080 allows a large number of I/O addresses, it is convenient to read the contents of each switch using a unique address. As shown in Fig. 2, this can easily be done using a 74L30 to decode the four more significant address bits along with SINP, and two 74L10 gates to AND A3 (or A3) with the output of the 74L30 and PDBIN and select one of two 74L42. The 74L42's decode the three least significant address bits when enabled, giving eight possible strobe pulses from each. A given strobe pulse (negative going) is used to enable



DECIMAL (BCD CODED) SWITCH INPUT

ADDRESSES 60 - 77$_8$



LED DECIMAL DISPLAY INTERFACE

ADDRESSES 100 - 117$_8$

-continued

Vector Interrupt and Real Time Clock

Annette Milford

Two new MITS products, the 88-Vector Interrupt (88-VI) and the optional 88-Real Time Clock (88-RTC) are now being shipped to customers. Although both of these peripherals have been designed on the same printed circuit board, the Vector Interrupt may be purchased without the Real Time Clock. The 8800 can be hardwire connected for a maximum of one interrupt system. This means, of course, that it is not possible to wire an I/O board for single level interrupt and connect the 88-VI for multi-level interrupt.

VECTOR INTERRUPT

As an independent board, the 88-VI has been designed to increase the efficiency of your system. It is useful in real time applications, when it is necessary to service I/O devices on a priority basis. Specifically, the VI provides the 8800 with the capability to interrupt activity, via the Restart (RST) instruction and to allow only the highest active priority of eight levels to interrupt the 8800. A system which includes the Floppy Disk, a teletype, a line printer and an 88-VI, for example, should service the Floppy Disk before any other device. Placing the Floppy Disk at the highest priority on the 88-VI then, insures that the software necessary to process data is available to the ALTAIR 8800 as soon as possible.

The ENABLE INTERRUPT instruction of the 8800 permits the 88-VI to interrupt. After each interrupt from the 88-VI is completed, ENABLE INTERRUPT is activated again, thereby reactivating the 8800's internal interrupt. The RST instruction translates in octal code to 3A7; and "A" translates into a 3 bit code which represents one of the eight priority locations: 0, 10, 20, 30, 40, 50, 60, or 70 (octal). Restart instructions, then, are RST 0 = 307, RST 10 = 317, RST 20 = 327, etc., (octal).

The interrupt service routine for level 2 would appear as follows:

```
OCTAL LOCATION   INSTRUCTION

      20          PUSH B

      21          PUSH D

      22          PUSH H

      23          PUSH PSW

      24          JMP LEV2
```

NOTE: As soon as the interrupt RST instruction is executed, interrupts are automatically disabled.

A software device called the interrupt service handler, supervises eight interrupt service routines, thereby enabling the interruption of a lower interrupt routine by a higher one and also insuring that each lower routine is returned to and fully executed.

The RST instruction saves the current program counter in the stack, then branches to the appropriate location (0, RST 0; 10, RST 1; 20, RST 2; 30, RST 3; 40, RST 4; 50, RST 5; 60, RST 6; 70, RST 7). The correct interrupt service routine saves all CPU registers on the stack, then, if required, jumps out of the RST location to complete the rest of the program.

```
LEV2    LDA     CURLEV     ;GET LEVEL INTERRUPTED

        PUSH    PSW        ;SAVE OLD LEVEL ON STACK

        MVI     A,15Q      ;SET CURRENT LEVEL

        STA     CURLEV

        ORI     300Q       ;OR IN BITS REQUIRED BY VI BOARD
                           ;ORI 330Q SHOULD BE SUBSTITUTED
                           ;IF THE RTC IS HOOKED TO THIS LEVEL

        OUT     376Q

        EI
        .
        .
        .                  ;DEVICE SERVICE ROUTINE
        .                  ;GOES HERE
        .
        DI

        POP     PSW        ;POP OLD INTERRUPT LEVEL

        STA     CURLEV     ;RESTORE CURLEV

OFF:    ORI     300Q       ;"OR" IN BITS FOR VI

BOTH:   OUT     376Q       ;TELL VI BOARD WHAT LEVELS TO ACCEPT

        POP     PSW        ;RESTORE ALL REGISTERS

        POP     H

        POP     D

        POP     B

        EI                 ;ENABLE THE INTERRUPTS

        RET                ;RETURN FROM INTERRUPT
```

-continued

VLCT

BUSY is low active and goes low after DATA READY IN goes high only for the time constant determined by the One slot in the VLCT Receiver. As long as RESET -(BUSY) is high, the sequence generator of the VLCT send section will not count. The result is that after entering three key strokes, the READY OUT goes low signaling the 4-PIØ that DATA is ready. Your software should send the received data back to the VLCT for verification. No new data will be transmitted till the VLCT receives data back. (See "Using the VLCT", Computer Notes, Vol. 1, Issue 5.)

3. We used the same initialization program contained in the 4-PIØ manual with the following changes.

Loc. 15 005 Disables CA2, sets CA1 low active, and enables it (bit 7 becomes our DATA READY flag).

Loc. 21 055 Same as in manual except CB2 set when next "E" pulse goes high instead of when CB1 is active.

4. Our test program is as follows:

Loc.	Octal	Mneumonic	
0	333	INPUT	Read A Control Register
1	020	address	
2	346	ANI	Mask for bit 7 (data ready flag)
3	200	data	
4	312	JZ	Test and loop if (Loc. 0) not present
5	xxx	<b2>	
6	xxx	<b3>	
7	333	INPUT	Input data
10	021	address	
11	323	OUTPUT	Output data
12	023	address	
13	303	JMP	
14	xxx	<b2>	(Loc. 0)
15	xxx	<b3>	

When this program is run, the following should happen: after you enter 3 keystrokes, the octal number should appear at the DATA IN display on the VLCT and should remain until you enter another 3 keystrokes.

If that works, you are all set. Talk to your computer!

Using the VLCT with 4PIO

by Bill Kuhn

In answer to questions about use of the VLCT with the 4-PIØ, we have made the following hookup and tested it.

GENERAL PROCEDURE:

First: Decide what section of the 4-PIØ port you will use for the various signals necessary.

Second: Make an interface cable (25 pin male to 25 pin female) to connect the 4-PIØ to the proper lines on the VLCT (or if you haven't wired your VLCT you can wire its connector and eliminate the extra cable.)

Third: Initialize the port so it is ready to send and receive on the proper sections.

Fourth: Design and run a test program to check steps 2 and 3.

HERE'S WHAT WE DID

1. We chose section A of the 4-PIØ for input data lines, and CA1 as our flag for data ready at the input lines.

We chose section B for output data lines and CB1 as our signal from the VLCT requesting new data. CB2 was chosen as the signal to the VLCT that new data was ready at its inputs.

2. We made an interface cable as shown in the following chart:

NOTE: If you haven't wired your VLCT, you may wire its connector the same as the 88-4PIØ connector and eliminate the interface cable.

88-4PIØ Signal Name	Connector Pin #	Connector Pin #	VLCT Signal Name
PA 0	4	5	DØ 0
PA 1	5	6	DØ 1
PA 2	14	7	DØ 2
PA 3	15	8	DØ 3
PA 4	16	1	DØ 4
PA 5	17	2	DØ 5
PA 6	18	3	DØ 6
PA 7	19	4	DØ 7
CA 1	2	10	READY OUT
CA 2	3	not used	
PB 0	20	14	DI 0
PB 1	21	15	DI 1
PB 2	22	16	DI 2
PB 3	23	17	DI 3
PB 4	25	21	DI 4
PB 5	25	20	DI 5
PB 6	10	19	DI 6
PB 7	11	18	DI 7
CB 1	12	23	READY KEY
CB 2	13	22	DATA READY IN
Ground	6	13	Ground
		[9	RESET IN
		25	BUSY OUT

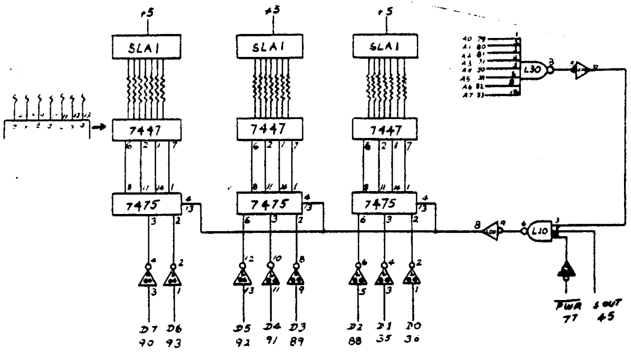
NOTE 2: We also tied Pin 9 to Pin 25 on the VLCT end of our cable to accomplish the following:

-continued

ALTAIR 8800 Interface

four 8T97 gates which are connected to the corresponding switch. Up to 16 BCD switch (16 integers) can be read in using this one interface.

Similarly, BCD data may be displayed using 7 segment units via the interface shown in Fig. 3. Here, the three 74L20 gates perform the high order address decoding function, enabling one of the 74L42 decoders. The strobe signals may be used to latch data into one of several TIL 308 display units (or the corresponding combinations of latches, decoders and 7 segment displays). The data are buffered by 74L04 and 7406 inverters in order to have sufficient drive to handle up to 16 TIL 308's.



LED OCTAL DISPLAY   ADDRESS = 377_8

VI & RTC

During this program, the following occurs: The previous interrupt level (in CURLEV) is saved on the stack. The current interrupt level is output to the VI board in order to prohibit interrupts at level 2 or levels of any lesser priority (in this case, 3, 4, 5, 6, or 7) from interrupting. The current interrupt level is saved in CURLEV. Interrupts are then re-enabled to allow execution of higher priority interrupts. At this point, the appropriate device service routine should be executed. After the service routine is completed, interrupts are disabled. The previous interrupt level, saved in CURLEV is re-stored in CURLEV and output to the VI controller. The registers are then popped off of the stack, interrupts are reenabled, and the interrupt service routine returns.

The interrupt routine is the same for all interrupt levels, except for instruction 3(MVI). The following chart indicates the correct MVI instruction for each of the eight interrupt levels. Level 0 is the highest priority interrupt level, and level 7 is the lowest. Note also that instruction 5 requires that 330 be substituted for 300 if the RTC is hooked to this level, thereby allowing the RTC to interrupt when serviced.

Interrupt Level	RST Address	Instruction
0	0	MVI A,17Q
1	10	MVI A,16Q
2	20	MVI A,15Q
3	30	MVI A,14Q
4	40	MVI A,13Q
5	50	MVI A,12Q
6	60	MVI A,11Q
7	70	MVI A,10Q

REAL TIME CLOCK

The Real Time Clock is designed for the computer system in which timing of events is critical. An interrupt is generated by the 88-RTC after a precise interval of time, thereby enabling software to time certain routines and even to generate the correct time, day, and year upon request.

The 88-RTC provides the option of one of two sources, a derivative of the 2 megahertz clock or the line frequency. Both sources offer respective advantages. The 2 megahertz clock should be used in systems that demand a fast RTC; it is selectable for time intervals down to every 100 microseconds. The line frequency (60 Hertz) on the other hand, is efficient in systems that depend upon accuracy over a long period of time. Power companies constantly adjust frequency, thus insuring a consistent source.

The table below shows the frequency and associated time interval for both sources at each of the four selectable divide rates:

SOURCE	DIVIDE RATE	DIVIDE FREQUENCY (HZ)	TIME INTERVAL
Line Frequency (60 Hertz)	1	60	16.67 milli-seconds
	10	6	166.7 milli-seconds
	100	.6	1.67 seconds
	1000	.06	16.67 seconds
10,000 Hz (a derivative of the 2 MHz system clock)	1	10,000	100 microseconds
	10	1,000	1 millisecond
	100	100	10 milliseconds
	1000	10	100 milliseconds

Note that this time interval represents the frequency at which the 88-RTC will cause an interrupt. For example, if 1000 Hz is selected, the RTC will generate an interrupt every 1000th of a second or 1000 interrupts/second.

MITS has developed a machine language program for the 88-RTC, which keeps track of hours, minutes, seconds, and 60ths of seconds in four consecutive memory locations. This program uses 8K BASIC, a USR assembly language subroutine, and an interrupt response subroutine. To execute the program, strap the RTC for line frequency in ÷ 1, and load the following program using Package I (assembler, editor, monitor). Note that Q represents octal.

After the program is loaded, BASIC must be loaded into the CPU. The "memory size" question in BASIC's initialization's dialog should be answered with 8122. All other initialization questions in BASIC should be answered as usual.

After initialization, certain modifications to BASIC must be made.

1. A JMP instruction must be put at location 70, so that the interrupt will cause a JMP to the machine language interrupt response routine. Correct branching is implemented by the following three BASIC commands:

```
POKE  56,195
POKE  57,187
POKE  58,31
```

2. The following commands allow the USR function to turn on the clock and to enable interrupts. This changes the JMP FCERR in location 72 to a JMP INIT (see symbol table).

```
POKE  73,250
POKE  74,31
```

3. In order to set the time, make these commands. (Note: Set the time a few minutes ahead to allow for the time necessary to type the commands):

```
POKE 8180, TIM (60ths of a second)
POKE 8181, TIM (seconds)
POKE 8182, TIM (minutes)
POKE 8183, TIM (hours)
```

The above commands could also be part of a BASIC program which asked for the initial tie as HHMMSSJJ (hours, minutes, seconds and jiffies -- 1 jiffy = 1/60 second).

-continued

Program for **RTC**

```
         ORG     17673Q       ;PROGRAM STARTS AT THIS MEMORY LOCATION
START:   PUSH    PSW          ;STACK ALL REGISTERS TO BE USED
         PUSH    B
         PUSH    H
         LDA     CURLEV       ;PICK UP OLD LEVEL NUMBER
         PUSH    PSW          ;SAVE IT ON THE STACK
         MVI     A,10Q        ;NEW LEVEL IS 10Q
         STA     CURLEV       ;STORE THIS AS THE NEW CURRENT LEVEL
         ORI     330Q         ;OR IN BITS NEEDED TO RESET RTC AND VI BOARD
         OUT     254          ;OUTPUT LEVEL INFO TO VI BOARD
         EI
         MVI     B,3
         LXI     H,NMB        ;GET ADDRESS OF 60TH'S OF SECONDS COUNTER
LOOP:    MOV     A,M          ;PICK UP COUNTER
         INR     M            ;INCREMENT COUNTER
         SBI     59           ;CHECK IF COUNTER IS NOW = TO 60
         JNZ     OUTLP        ;IF < 60 WE ARE DONE
         MOV     M,A          ;IF = 60 ZERO OUT COUNTER
         INX     H            ;POINT AT NEXT COUNTER
         DCR     B            ;DECREMENT NUMBERS OF COUNTERS LEFT TO CHECK
         JNZ     LOOP         ;LOOP TILL 60TH'S, SECONDS, AND MINUTES ARE DONE
         MOV     A,M          ;NOW CHECK HOURS COUNTER
         INR     M
         SBI     23           ;MAKE SURE NOT MORE THAN 24 HOURS
         JNZ     OUTLP
         MOV     M,A
OUTLP:   DI
         POP     PSW          ;POP OLD INTERRUPT LEVEL OFF STACK
         STA     CURLEV       ;STORE AGAIN AS CURRENT LEVEL
         ORI     300Q         ;OR IN CONTROL BITS FOR VI
         OUT     254          ;OUTPUT CURRENT LEVEL TO VI BOARD
         POP     H            ;RESTORE ALL REGISTERS USED
         POP     B
         POP     PSW
         EI
         RET                  ;RETURN TO INTERRUPTED PROGRAM
NMB:     DS      5
CURLEV:  DB      0
INIT:    MVI     A,360Q       ;INITIALIZE THE VI BOARD
         OUT     254
         EI
LAST:    RET
         END     TIM

UNDEFINED SYMBOLS

SYMBOL TABLE

$0200000
START 017673
CURLEV 017771
NMB 017764
LOOP 017721
OUTLP 017746
INIT 017746
INIT 017772
LAST 017777
```

EXAMPLE: If the RTC were to be set for 9:30 a.m., the commands would appear as follows:

```
POKE 8180,0
POKE 8181,0
POKE 8182,30
POKE 8183,9
```

4. In order to start the clock, type:

```
A = USR (1)
```

A printout of the correct time will be received when the following BASIC program is typed in.

```
10  DIM Z(3)
20  FOR X=1 TO 3
30  Z(X)=PEEK(8180+X)
40  NEXT X
50  PRINTZ(3);":";Z(2);":";Z(1)
RUN
Δ9Δ:Δ30 : 0
```

Δ9Δ:Δ30Δ:Δ0

New Audio Modulation Method for ACR

As evidence that we at MITS listen to our customers, we are improving the 88-ACR read and write performance. The changes described below will allow the 88-ACR to accept 2.75 times wider speed variation when demodulating tapes written with the new method. Also, demodulation (reading) of tapes written by the old method will be the same as before.

I Purpose: Make reading and writing of data on audio tapes less susceptible to errors due to speed variations, and to make adjustment of R29 (phase locked loop center frequency adjust) less critical.

II Method: Change modulator frequencies from 2225Hz/2025Hz-(200 Hz difference) to 2400Hz/1850Hz-(550 Hz difference). This change keeps the center frequency at 2125Hz, allowing the 88-ACR to demodulate (read) either type of modulation.

III Modifications to 88-ACR Modem Boards in the field:

A) Modulator - Change jumpers as follows:

1. Remove jumpers #1 & 2.
2. Connect pins 3, 4, and 5 of IC "J" together.
3. Change jumper #3 from 3B to 2A.
4. Change jumper #4 from 4B to 4A.
5. Disconnect pins 5 and 6 of IC "K" from ground (unsolder and bend out of board).
6. Connect pins 4 and 5 of IC "K" together.
7. Change jumper #5 from 5B to 2A.
8. Connect pin 6 of IC "K" to point 5A.
9. Change jumper #7 from 7B to 7A.

NOTE: The "B" row of jumper points is closest to edge of Modem Board, the "A" row of jumper points is closest to the row of numbered jumper wires (see schematic diagram in manual).

This changes the modulation frequencies to:

LOGIC 1 = 2404 Hz ± 1 Hz
LOGIC Ø = 1852 Hz ± 1 Hz

(measured at IC "H"-8)

B) Demodulator: Change R28 to 3.3K ohms, or parallel a 5.6K ohm resistor with the existing 8.2K ohm resistor.

This change increases the lock range of the phase locked loop (IC "C") for the wider frequency spread of the new modulation method. It does not affect demodulation of tapes previously recorded with the old frequencies (2225/2025 Hz).

This change allows tape speed variations between writing and reading of over 3% without readjustment of R29 (if demodulating tapes written with the new method).

IV Other Circuitry Changes Recommended for the 88-ACR.

A) Change C18 (was 5 µf electrolytic) to a 1 µf mylar or non-polarity sensitive capacitor. This prevents breakdown of C18 when reverse biased (no carrier).

B) Use the old C18 (5 µf electrolytic) to add a 5 µf capacitor: + end to IC "C" pin 9 end of R30, -- end to -12 volts. This helps stabilize adjustment of R29.

C) Change R32 to 8.2K (use old R38) and change Z1 (12 volt zener) to a 3.3K resistor. This allows the P. L. L. output (IC "C", pin 8) to pull down point "RS" to a valid logic Ø even if the system negative voltage supply is low.

D) Remove diode D4. This allows reading and writing of tapes simultaneously.

E) Optional - For indication of the carrier (2K Hz tones) a L. E. D. may be wired to points "A" and "K" on the Modem Board. Remove the jumper wire from "A" to "K", and connect the LED anode to "A", the cathode to "K". When the carrier is being received, the LED forward current is about 10MA. Use a red LED only--1.7 volts forward drop.

V Effective Date of Change

A) All COMTER II units, all assembled 88-ACR's and all repaired 88-ACR's shipped from MITS after March 1, 1976, contain the modification described above.

B) All 88-ACR kits shipped after March 15, 1976, contain the modification described above.

C) All ALTAIR BASIC and Package I cassette tapes will be made with the new modulation technique starting April 5, 1976.

VI Converting Old Tapes to the New Modulation Method: Although it is not necessary, you may wish to convert existing tapes to the new form. To do this, you need two tape recorders and:

A) Modify your 88-ACR board as indicated, including Step IV-D.

B) Identify the slower of the two tape recorders, and use it for playback of your existing tape during transfer. The play machine should be slightly slow to prevent the inputting of data faster than it can be outputted. Connect the slower machine to the "PLAY IN" circuit, and adjust R29 for the proper pattern.

C) Connect the other tape recorder to the "RECORD OUT" circuit and use it for recording the new tape.

D) Use the following program to transfer data:

Address	Octal Code	Mnemonic
000,000	333	IN
1	006	
2	017	RRC
3	332	JC
4	000	
5	000	
6	333	IN
7	007	
10	323	OUT
11	007	
12	303	JMP
13	000	
14	000	

-continued

ACR

E) Start the record machine first, then start play machine; then play program to transfer data.

F) After your tape has been transferred, check it for correct data. If your playback tape recorder was too fast, then there will be bytes dropped.

G) Once your tapes have been transferred, R29 will probably not require readjustment. This is one of the advantages of spreading the modulation frequencies.

If you have only one tape recorder, or if the above procedure does not work for you, read the old tape into memory, then write it out to tape.

Use the 88-ACR read/write programs listed in the Nov/Dec COMPUTER NOTES, pages 22 & 23. If you are rerecording an ALTAIR BASIC cassette, the test byte must be changed to 175 for version 3.1 and 256 for version 3.2.

If you need to order parts for the modification, order:

2 ea.	102085	3.3K resistor
1 ea.	100363	1.0mf mylar capacitor

88 ACR USER NOTES

by Tom Durston

Here are more helpful hints for those Altair users having difficulties loading MITS software on cassette tape.

1. Try using lower volume settings on your tape recorder during playback. Sometimes noise generated in recorders playing at maximum volume can cause errors in data. We have found that in most recorders volume settings as low as 1/3 of maximum are satisfactory.

2. If you have trouble obtaining a proper "JUMP" of address lights when beginning a bootstrap load, or you don't want to wait the 15 seconds between starting the tape playing and depressing the run switch, try this 9 step program in addition to the bootstrap loader.

This program tests for the leader bit pattern that is recorded before the checksum loader at the beginning of MITS software. The program will loop at the high addresses until the leader byte is found (10-15 seconds after start of tape) and then jump to the bootstrap loader at ØØØ,ØØØ. Approximately 10 seconds later the address lights change again, indicating proper loading of the software (for version 3.2 A3, A4, & A6 off).

1) Deposit bootstrap loader.
2) Deposit leader detector.
3) Examine 001,000.
4) Start tape and depress "RUN" on Altair.
5) 25 seconds later, Altair should jump , indicating proper loading of data.

Bootstrap Leader Detector

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
START	IN	001,000	333	input data
		1	007	from ACR
	CPI	2	376	compare data byte to
		3	256	leader byte for version 3.2 (175 for 3.1) (same as bootstrap location 1)
	JNZ	4	302	jump if data ≠ 256 (or 175)
		5	000	to "start"
		6	001	
	JMP	7	303	jump to bootstrap loader
		001,010	000	if data = 256 (or 175)
		11	000	

New Products

Teletype Call-Control Kit

The MITS Teletype Call-Control Kit provides a much lower cost and faster way to get a Teletype into your system than was previously possible. MITS has made an agreement with Teletype whereby the fully assembled mechanical portion of the Teletype will [illegible] hipped directly to you from Teletype Corp. and the PC board Call-Control Kit will be shipped from MITS. Starting in May, delivery time should be a couple of weeks as opposed to 4 to 5 months.

There are three Call-Control Kits available: 88-TYR which is supplied with the Teletype Printer only, 88-TYK which is supplied with Teletype Model KSR-33 (printer and keyboard), and 88-TYA which is supplied with Teletype Model ASR-33 (printer, keyboard, paper tape punch and reader). All three kits use the same basic printed circuit board (see parts layout, this page). All you need to do is assemble as much of the PC board as applies to the model of Teletype you have purchased, mount it to the Teletype chassis, and plug it in.

The PC board for the 88-TYR (printer) kit has a power switch, two fuses, a simple power supply and the receive circuit for 20ma current loop. Interconnect plugs and mounting hardware are also provided.

For the 88-TYK kit, a relay for line-local switching, a connector to the keyboard, and some transient damping circuitry are added.

For the 88-TYA kit, all of the above is included with the addition of another simple power supply, connections for the paper tape reader, and a circuit to control the reader by program control.

That's all there is to it. The Most complicated part of the assembly is connecting wires from the line-local relay to the PC board. The 88-TYA kit should take only 3 to 5 hours to complete. At $500.00 savings, that's $100.00 an hour for your time.

680b Paper Tape Reader Control

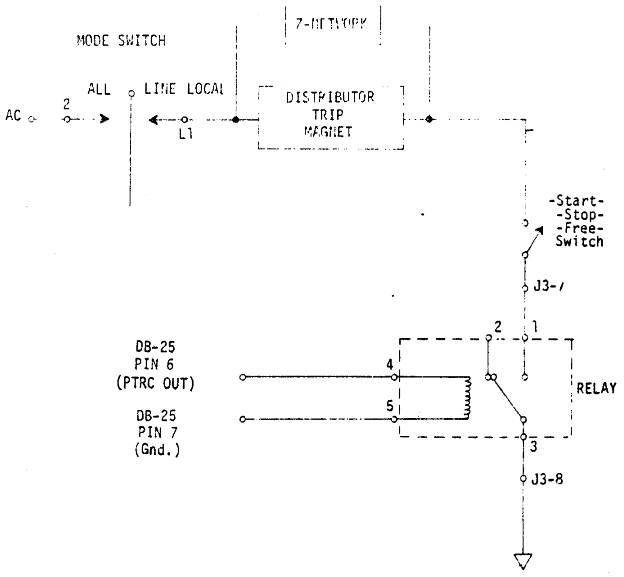
by Steve Pollini

It has been advertised that the Altair 680b has a Paper Tape Reader Control output. This output function utilizes the Request To Send ($\overline{\text{RTS}}$) output of the Motorola 6850 (ACIA) and is fully software controllable. The desirability of having such a feature is dubious until you try to generate an object tape or assemble a program with a two-pass assembler.

During the process of assembling a program, a single line of data is read from the paper tape, assembled, and then printed. If the paper tape is not halted while the computer is assembling and printing the line that was just read in, data will be lost as soon as the next line begins to be read. Therefore, it is very useful to have a paper tape reader controller on your Teletype, such as the one that is in the MITS 88-TYA.

What this circuitry essentially consists of is a relay that makes and breaks the supply current to the reader feed magnet within the Teletype. The relay is supplied with current from the 20ma current loop interface supplied on the 680b main board. The relay is driven by the $\overline{\text{RTS}}$ output of the ACIA.

Turn to page 15 for a diagram of the circuitry necessary to implement a paper tape reader controller on an ASR-3320 Teletype.



The relay used in the MITS 88-TYA is a Guardian 10 amp 1345 DC relay (form C). The coil draws approximately 5 milliamps and is therefore easily driven by the 20ma interface labelled "PTRC" (Paper Tape Reader Controller) on the 680b main board. To connect the relay to an ASR-3320, remove the jumper between pins 7 and 8 on J3 of the Teletype. Connect pin 7 of J3 to pin 1 of the relay and pin 8 of J3 to pin 3 of the Teletype.

All of the software necessary to control this output while assembling a program is contained within the Assembler. However, if you want to control this output to read in a data file from paper tape while not assembling a program, the following information will be useful.

When the $\overline{\text{RTS}}$ output goes high, the reader is turned on. When it is low, the reader is turned off.

To turn the reader on:

```
LDA  A    #$    D1
STA  A     $    F000
```

This stores D1 in the Control Register of the ACIA. Bits five and six function together such that when six is a one and five is a zero, the $\overline{\text{RTS}}$ line is high with Transmitting Interrupt Disabled. The Control Register is also set for a ÷ 16 clock, two stop bits, eight bits, and no parity.

To turn the reader off:

```
LDA  A    #$    B1
STA  A     $    F000
```

This stores B1 in the ACIA Control Register. Bits five and six function together such that when they are both zeros the $\overline{\text{RTS}}$ line is low with Transmitting Interrupt Disabled. The Control Register is also set for a ÷ 16 clock, two stop bits, eight bits, and no parity. To turn the reader off with Transmitting Interrupt Enabled, do a LDA A #$ 91 instead of a LDA A #$ B1.

For further information on the ACIA Control Register, see the 680b Programming Manual, Appendix C.

New Products

Controllable High Speed Tape Reader

88-HSR (assembled only)

Designed around a REMEX 300 character-per-second opto-reader, the 88-HSR connects directly to one port of an 88-4PIO parallel interface card or one parallel port of an Altair 680b. Software features include start/stop on character and low power standby mode. The standby mode serves two purposes: it reduces motor voltage during periods of inactivity, and also allows tapes to be read at the reduced speed of 30 cps.

The reader is enclosed in an Optima case that is the same style and colors as the Altair 8800 and 680b cases.

Specifications:

Reading Speed	300 cps or 30 cps, software selectable stops "on character"
Tapes	
a) light transmissivity	57% or less
b) thickness	.0027 - .0045 inch
c) type	standard 8-track (1-inch) and most other standard 5, 6, or 7.track
Data Output	TTL: a) less than .4 volts @ 16 ma (No Hole) 2.4 - 5 volts @ .2 ma (Hole)
	b) Plug-in compatable with 88-4PIO or Altair 680 parallel port
Drive	DC stepping motor with sprocket drive
READ Mechanism	Filament lamp to fiber optics to photo cells
	Lamp operated below voltage rating to greatly increase life
Tape Loading	easy in-line, front load
Dimensions	6 1/2 inches high, 8 1/2 inches wide, 11 inches deep
Power	50 watts, running

COMPUTER NOTES/JULY, 1976

TROUBLESHOOTING 2-SIO BOARDS

from the MITS Repair Department

by Bruce Fowler

In comparison to MITS' older I/O boards, the 2SIO Board may seem far more complex due to its programing requirement. In the earlier I/O boards, the information containing number of stop bits, type of parity, and number of bits per character was hardwired. Reset was also provided by hardware. In the 2SIO, all of this information is supplied through software. This difference only means that troubleshooting will use both software and hardware.

Special considerations:

If $\overline{\text{DATA CARRY DETECT}}$ and $\overline{\text{CLEAR TO SEND}}$ are not used but are connected through the circuitry to the Molex connectors, they must be set to a high level. This is done by jumpering S1-1 and S1-2 (or S2-1 and S2-2 for second port) of Molex pins to +5v. (Earlier errata sheet on jumping to ground should be ignored.) These jumpers are necessary for RS-232 level interfacing. If these lines are not connected, jumper the D and E pads to ground (D1, D2, E1, E2).

When using a 2SIO to load software, start the bootstrap before starting the loading device. The 2SIO's ACIA must be reset before it will accept any data. When assembling the 2SIO board, IC J is installed only if the 2SIO is to be used for 2 ports. The 2SIO requires the 3.2 version or later of BASIC and must be addressed at Location 20. Switch A-11 must be up for operation of the 2SIO at Location 20. Use the echo routine (page 101 of BASIC Manual) and the bootstrap loader (page 99 of BASIC Manual) with the 2SIO. (The echo routine is given at the end of this article).

Troubleshooting

Check the power supply levels on the voltage regulators and check for solder bridges. An easy way to check the wiring is with an ohmmeter. Use a scope, if available, to check the baud rate. The frequency is 16 times the baud rate. 110 baud should produce a square, symmetrical waveform of roughly .568 millisecond pulse width (1760 hertz).

110 baud	.568 milliseconds
300 baud	.208 milliseconds

When troubleshooting the 2SIO, use the status register information of the ACIA to indicate the problem. You can single step through the 2SIO echo routine, checking status at appropriate times. (It is necessary to check status since it is possible to echo on a 2SIO which will not respond with BASIC.) The status register indicates the condition of the ACIA at any given moment, and each bit indicates one characteristic. Status register bits appear on the data lights when instruction 333,020 of the echo program has been single stepped. ("HIGH" indicates a lit LED, and "LOW" indicates an unlit LED on the data lights.)

The status register bits are defined as follows.

Bit Ø:

HIGH - Receive data register full. A character has been received from the terminal.

LOW - Receive data register empty. No character received yet from the terminal.

Bit 1:

HIGH - Transmit data register empty. No character is being sent from the CPU to the ACIA.

LOW - Transmit data register full. ACIA has a character stored and is transmitting the character to the terminal.

Bits 2 and 3 are for use with a modem.

Bit 2:

HIGH - No carrier is present. (Pin 23, $\overline{\text{DCD}}$, of ACIA will be HIGH accordingly.) In this state, pin 23 inhibits the receiver section of the ACIA, thus no data can enter the 2SIO.

LOW - Carrier is present. (Pin 23, $\overline{\text{DCD}}$, of ACIA will be LOW accordingly.) In this state, pin 23 activates the ACIA receiver section, and ACIA is free to accept data.

Bit 3:

HIGH - Output device is not ready to receive. In this state, the ACIA's transmitter section is inhibited and the 2SIO cannot output (i.e. transmit) data. (Pin 24 of ACIA will be HIGH.)

LOW - Output device is ready to receive. In this state the ACIA transmitter section is free to output data.

The other bits, which are defined on page 8 of the Theory Manual, are not vital in troubleshooting. Thus, for proper operation, the status register should have all bits LOW except for Bit 1.

Single step the echo routine through to where the status is checked (hit single step 12 times). If you receive proper status, hit a key on the input device. Bit 0 will light up, indicating that the character has been received. If single stepping is continued, the echo routine will output the character.

When the ACIA is neither receiving nor transmitting, pins 2 and 6 of the ACIA must be HIGH. With a Teletype, pin 2 of the ACIA is LOW until the Teletype is ON LINE. If either pin is LOW, the ACIA responds as if data were being transferred.

Bad Status and Areas to Look At

If Bit 3 (or 4) and pin 23 (or 24) of the ACIA are HIGH: This indicates either bad inverters, diodes installed backwards, or S1-1 and S1-2 not tied to +5v for RS-232 levels.

If Bit 0 is HIGH before entering data, then Pin 2 of ACIA is LOW when it should be HIGH, or the ACIA hasn't been reset. The latter could be caused by the data buffers IC A and B not being enabled after hitting single step four times starting at the beginning of the echo routine.

-continued

If all bits are HIGH: ACIA is not selected due to one or more bad control signals or the output buffer to the CPU is not enabled. After single stepping 12 times (from the beginning of the echo routine) in the echo routine, the ACIA should contain:

(HIGH and LOW are in TTL levels, .8v or less for LOW, 2v or more for HIGH)

CSO	Pin 8	HIGH
RS	Pin 11	LOW
CS1	Pin 10	HIGH
R/W	Pin 13	HIGH
E	Pin 14	HIGH
$\overline{CS2}$	Pin 9	LOW
IC P	Pin 8	LOW
IC O	Pin 8	LOW
IC S	Pin 11	HIGH

NOTE: R/W (pin 13 of the 6850) is LOW for outputting. Continue single stepping 10 more tires and Pin 13 should be LOW. IC P, Pin 8, will be HIGH, while Pin 6 will be LOW. SINP will also be LOW. All other pins will be the same as before

Final Notes

If the ACIA is not reset, the Teletype may run open when it is turned on. To correct this, simply flip the Altair ON/OFF switch a couple of times. In some cases an etching error will short out SOUT. This etching error is located on the back of the 2SIO board between IC S, Pin 11, and the gold fingers. Usually SOUT is shorted to address line A6. This etching short should be cut. For those of you who bought the BASIC manual before the extended BASIC section was written, the 2SIO echo routine and bootstrap are listed below: Note that the first 4 bytes reset the ACIA and clear its internal registers. The next 4 bytes tell the ACIA what type of parity, the number of bits/character, and the interrupt information that will be used.

NOTE: There is a misprint in the Appendix, page 101, of the Extended BASIC Manual. The corrections are listed below:

025 in Location 005 is for 1 stop bit.

021, used in Location 005, is for 2 stop bits.

2SIO Echo Routine

OCTAL ADDRESS	OCTAL CODE		OCTAL ADDRESS	OCTAL CODE
000	076		013	322
001	003		014	010
002	323		015	000
003	*020		016	333
004	076		017	**021
005	021	(=2 stop bits.	020	323
006	323	025=1 stop bit)	021	**021
007	*020		022	303
010	333		023	010
011	*020		024	000
012	017			

* - Control channel

** - Data channel

USING THE STACK

The stack is a portion of memory the programmer sets aside for temporary storage of data or addresses. The stack is necessary for the proper execution of many instructions. The Stack Pointer is a 16-bit register that specifies the address in the stack that will be operated upon.

To establish the stack use the LXI SP instruction. The data byte immediately following the instruction has the least significant 8 bits of the address, and the next data byte has the most significant 8 bits of the address. For example:

LXI SP 077,000	sets the stack pointer at memory address 000 077 octal.

There are two basic operations on the stack, the PUSH and the POP. The PUSH instruction moves the contents of the specified register pair into the stack. The first register of the specified register pair goes into the stack at the address in the stack pointer minus 1 and the second register at the address in the stack pointer minus 2. The stack pointer is then decremented by 2.
For example:

PUSH D	with the stack pointer at 077 octal would move the contents of register D to memory address 076, E to 075 and then set the stack pointer to 075.

The POP instruction is the reverse of the PUSH. So the content of the stack at the address contained in the stack pointer is moved into the second register of the specified register pair. The content of the stack at the address contained in the stack pointer plus 1 is moved into the first register of the specified pair. The stack pointer is then incremented by 2.
For example:

POP D	with the stack pointer at 075 octal would move the content of 075 to register E, 076 to register D and then set the stack pointer to 077.

In addition to these instructions the stack pointer may be operated on by many of the register pair instructions.

When programming remember the following:

1) If an instruction requires a stack for proper execution be sure to provide it. (For example, the stack is necessary when using subroutines.)

2) There should be a POP instruction for every PUSH instruction. (The stack is for temporary storage.)

3) The stack pointer moves down through memory as data is added to the stack and back up as data is removed. Be sure to allow sufficient memory for the maximum possible requirements of the stack.

ALTAIR INTERRUPT STRUCTURE

By Paul Allen

In order to implement simple (one level) interrupts on the Altair, use the following procedure:

1) Enable interrupts using the EI (enable interrupt) instruction.

2) The I/O interface should pull bus line PINT low. This will cause the immediate execution of a RST 7 instruction if the CPU is halted, or as soon as the current instruction finishes if the CPU is already running. As soon as the interrupt is acknowledged, interrupts are disabled.

3) At the completion of the interrupt service routine (which should start at octal location 70), enable interrupt (EI) instruction should be executed to re-enable interrupt; and then an RET instruction should follow the EI, causing the CPU to execute the instruction after the one it was executing when the interrupt occurred.

NOTE! Since the RST instruction uses one level (2 bytes) of stack to store the return address to the "main sequence" code, the programmer should always have a stack set up if he expects interrupts to occur; and he should allocate enough stack space for the use of the interrupt service routine.

Consider the following example:

```
location 70:  PUSH  PSW     ;save A & condition codes
              PUSH  H       ;save [H,L]
              IN    10      ;read in byte from device
              LHLD  BUFPNT  ;load buffer pointer into [H,L]
              MOV   M,A     ;save byte in buffer
              INX   H       ;increment buffer pointer
              SHLD  BUFPNT  ;save updated buffer pointer
              POP   H       ;restore [H,L]
              POP   PSW     ;restore A and PSW
              EI            ;re-enable interrupts
              RET           ;return to main sequence
```

When an interrupt occurs, this simplified routine will save the A register, the condition codes and the H and L registers. Then the input byte from the interrupting device is read into A with an IN 10. Next, a buffer pointer (BUFPNT) is loaded into the register pair [H,L]. This buffer pointer addresses an area of memory where the incoming bytes are to be stored. The first byte read in will be stored in the first byte of the buffer, etc.

The buffer pointer in [H,L] is used to store the byte in the buffer. The buffer pointer is then advanced to point to the next byte in the buffer; and then the pointer is saved back in memory, so that when the next interrupt occurs the incoming byte will be stored in the correct location.

This interrupt service routine would use 3 levels or 6 bytes of stack space.

An actual interrupt service routine for an ACR or SIO board would be more complicated. It would test the status bits for the device to see whether the interrupt was caused because the device set character ready or character done. It would then either empty a character from the output buffer or store a new character in the input buffer. Also, it would take some special action when the output buffer became empty or the input buffer became full.

If the programmer wishes to ignore interrupts and is in an interruptable state because an enable interrupt instruction has been executed, he should include a disable interrupt (DI) instruction in his code.

The maximum time between the occurrence of an interrupt and the execution of the first instruction in the interrupt service is approximately twenty microseconds for dynamic memory and thirty microseconds in the static memory. This allows for the execution of the longest possible instruction plus the time required to execute the RST instruction.

Vectored Interrupts board

The Vectored Interrupt board gives the Altair eight levels of priority interrupt service. The highest priority level is zero and the lowest is seven. An interrupt on level five would cause an RST 5 (357) to be executed. If an interrupt occurred on level two while the service routine for level five was still being executed, the level two service routine would pre-empt the level five service routine and an RST 2 (327) would be executed. When the level two service routine finished, it would return to some location inside the level five service routine, where execution would continue. If a level six interrupt occurred during the servicing of the level five or level two interrupts, it would be held pending and would not be serviced until the level two and level five service routines had finished.

Software

By bill Gates

Condition Codes

There seems to be some confusion about the condition codes. These are the Boolean (true/false) flags that are set/reset depending on the results of certain instructions. They are:

Z = zero - result was 0
S = sign - the most significant bit (MSB) of the result
P = parity - the result has an even number of ones in it
C = carry - an arithmetic operation generated a carry out of the most significant bit (i.e. adding 200 to 212)
CY$_1$ = first digit carry - this is used only for BCD arithmetic and will be elaborated on next month.

It is the condition codes that determine whether conditional JMP's, CALL's and RET's will be executed (i.e. RZ, CPE, JP). JM, CM, and RM (minus) are executed if the sign flag is on. JP, CP, and RP (positive) are executed if the sign flag is off. JZ, JNZ, CNZ, RZ, RNZ (zero/no zero) depend on the zero flag just as JC, JNC, CC, CNC, RC, RNC (carry/no carry) depend on the carry flag. CPE, JPE, RPE (parity even) are executed if the parity flag is on and CPO, JPO, RPO are executed when it is off.

The condition codes do not always reflect the value in A since IN, LDA, LDAX, MOV and MVI can change A but do not affect the condition codes. Instructions like INR C, DCR L, CMP B, CPI 3, STC, CMC and DAD B affect the condition codes, but not A.

Affect carry only: STC, CMC, RAL, RAR, RLC, RRC and DAD.

Affect all but carry: INR, DCR.

Affect all: ADD, ADC, SUB, SBB, CMP, ANA, ORA, XRA, DAA and their immediate counterparts (i.e. ADI,CPI).

Use carry to affect result: CMC, RAP, RAL, ADC, SBB, ACI, SBI, DAA.

The instructions XRA, ORA, ANA, XRI ORI and ANI always reset carry.

If the condition codes do not reflect A's value (i.e. you just did a LDA or MOV into A) and you want to see if A=0, use ORA A or ANA A. CPI 0, ADI 0 and ORI 0 also work but they are 2-bytes.

The only other instructions besides the ones in the list above that use the condition codes are PUSH PSW and POP PSW. Respectively, they SAVE/RESTORE the condition codes and A on the stack.

For tricky programmers a sequence like PUSH B / POP PSW may be used to set the condition codes. This has the effect of moving B into A (MOV A, B) and moving C into the condition codes. The PSW (condition code) format is

MSB→	C	1	P	0	CY$_1$	0	Z	S	←LSB
	7	6	5	4	3	2	1	0	

Therefore if C was 201_8 before the POP PSW, zero and sign would be set and parity and zero would be unset. The bits marked '0' and '1' are constant and cannot be changed.

HINT #1

If you have a counter that can be bigger than 255 but is always less than 65535, it is convenient to use the following:

```
LXI B, count   ;set up counter
LOOP: code to be executed 'count'
      times
DCX B           ;decrement count
                ;does not affect
                ;condition codes
MOV A,B
ORA C           ;see if any bits set
JNZ LOOP        ;go back if so
```

HINT #2

For those who like to save bytes, and especially for those with 256-byte machines, (a byte is always 8 bits, which is a word on the 8800) RST's that are not used for interrupts, debug calls, monitor calls, etc. can be used to call subroutines that get called in many places (i.e. a character input subroutine). An RST is only 1 byte and a CALL is 3 bytes. Even if you have to put in a JMP so you don't overrun another RST location (0,10,20,30,40,50,60,70) you will probably save bytes.

Loading Software

Software from MITS will be provided in a checksummed format. There will be a bootstrap loader that you key in manually (less than 25 bytes). This will read a checksum loader (the 'bin' loader) which will be about 120 bytes.

For audio cassette loading the bootstrap and checksum loaders will be longer. All of this will be explained in detail in a cover package that will go out with all software.

For loading non-checksummed paper tapes here is a short program:

```
STKLOC:  DW GETNEW
         (2 bytes-#1 low byte of
                     GETNEW address
                  #2 high byte of
                     GETNEW address)

START:   LXI H,0
GETNEW:  LXI SP, STKLOC
         IN <flag-input channel>
         RAL  ;get input ready bit
         RNZ  ;ready?
         IN <data-input channel>
CHGLOC:  CPI <043 = INX B>
         RNZ
         INR A
         STA CHGLOC
         RET
                        (22 bytes)
```

Punch a paper tape with leader, a 043 start byte, the byte to be stored at loc 0, the byte to be stored at 1, - - - etc. Start at START, making sure the memory the loader is in is unprotected. Make sure you don't wipe out the loader by loading on top of it.

To run this again change CHGLOC back to CPI - 376.

.....LETTER TO THE EDITOR.....

Here are a couple of programming suggestions which may be of interest to other Altair owners.

1) An efficient way to establish a control counter in a program which is already using the accumulator (register A) and the memory address register pair (H and L) for other purposes is to use the B, C, D or E register with the decrement register (DCR) and Jump if zero or Jump if not zero (JZ, JNZ) instructions.

The desired count is first loaded into the selected register. The DCR instruction will cause the zero flag to be set when the count is exhausted and the JZ instruction will test this condition.

Note that the decrement register pair instruction (DCX) does not set the condition flags and cannot be used in this fashion. Also be aware that the DCR (E) instruction will not cause a borrow from register D.

2) A simple way to indicate "end of program" on an Altair without any I/O devices (i.e. the basic kit) is to use the "interrupt enabled" light on the front panel and an unconditional Jump instruction. The last instructions are:

```
EI
JMP  (To EI)
```

The stop switch can be actuated when the light comes on and the reset switch will turn it off. (This avoids use of the machine hanging HALT instruction.)

T. H. Schmidt
P.O. Box 9674
Stanford, CA 94305

FUN WITH ALTAIR BASIC

by Monte Davidoff

After loading BASIC, you may be ready for a little enjoyment from your Altair. Here are three short programs that illustrate some of the things that are easy to do in 8K Altair BASIC.

The first program wants two strings S$ and W$. The program finds all the occurances of W$ as a substring. (i.e. it finds all the places in S$ that are exactly the same characters that are in W$.) For example, if S$="ABCDAB" and S$="AB" then the program would say "AB" was found in S$ starting at characters 1 and 5. Now let's look at the program.

The first thing we do is input S$ and W$. Notice that we can print text in an INPUT statement also. The next thing the program does is set the variable CN to Zero. (Remember you can have two character variable names in Altair 8K BASIC). CN will be used to count how many times W$ is found in S$. Next, we use the CHR$ function to set Q$ to a string one character long. 34 is the ASCII code for a double quote. Next we use the concatentation operator "+" to put quotes around W$ and S$. This is just so the printouts will look nice. Next comes the meat of the program. I ranges from 1 to the number of characters in S$. The MID$ function will return as many characters out of S$ as there are in W$. If these are equal to W$, we have found our substring. So, we add one to the count, CN, and print a message.

Remember that if the IF statement is true, the statements following the THEN will be executed. If the IF statement is false, we skip the statements on the same line and start at ethe beginning of the next line. In 8K Altair BASIC, if there is no variable given in the NEXT statement, BASIC will assume the NEXT is for the most recent FOR loop. Line 90 just prints out how many times we found W$ in S$. This program is just intended to show what kinds of things can be done with the string functions such as MID$ and LEN.

The next program shows how you can use STR$ to make your output look nice. This program produces a table of Pascal's Triangle. This table is used in probability. It is formed by starting with a one at the top and then adding the two numbers that appear to the upper right and left to form an entry in the next row. Notice the NEXT J,I in line 60 to end both for loops. The trick comes in printing the thing out to look like a triangle. The STR$ function is used to convert the number to be printed to a string. Then the LEN function is used to find out how many characters are needed to print the number. These are added together in line 90 to see how many characters will be used to print all the numbers in the line. In line 100 we find how many spaces to print between each number. We have 72-LL positions to fill, and we want to divide the number of spaces into groups. .5 is added to round to the nearest integer. Line 120 then prints the row out using the SPC function to print the correct number of spaces. The last NEXT in line 120 ends the FOR in line 80,so we repeat this procedure for each line in the triangle. The right side does not look as nice as the left side. This is because the terminal can only print a character at discrete positions on the paper. However, the program still shows the general idea of using the STR$ function.

The third program is intended for people who already know about BASIC on an advanced level. While BASIC is a simple language, it still has the power to do complicated things. This program calculates N factorial (N!) where n!= Nx (n-1)x (N-2)... 3x2x1 and 0!=1. So 5!= 5x4x3x2x1= 120.

The program uses the fact that 0!=1 and N!=N.(n-1)!

The FOR loop in line 20 just sets up things so a table can be printed in two columns. The subroutine in line 60 prints out the result. The STR$ function is used to make the output look nice by having the exclamation mark come immediately after the number.

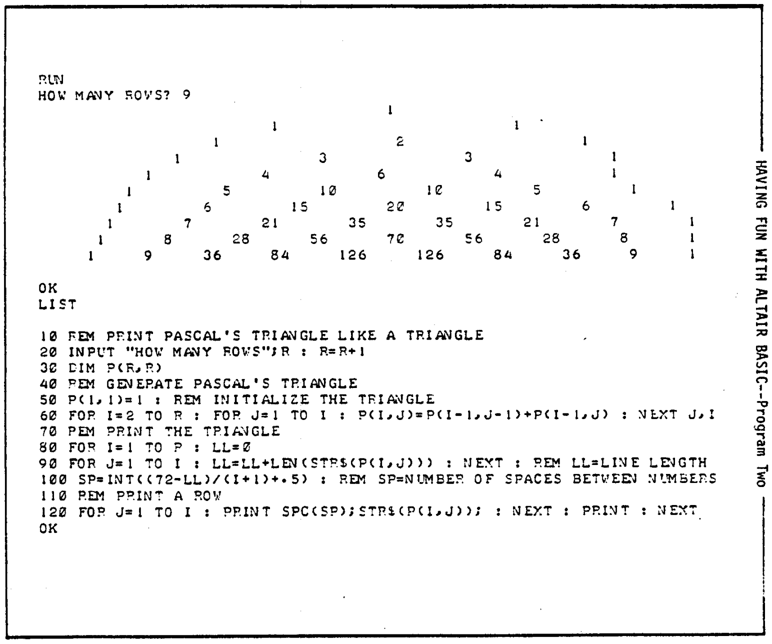
Now for the tricky part. The subroutine in line 40 calculates F=N!. In line 40, if N=o then we know F should be one and we RETURN because we are done. If N is not zero, Line 50 decreases N by one, and then GOSUB'S to line 40 again to calculate (N-1)! The amount of times GOSUB'S can be executed with out a RETURN is limited only by how m much memory you have. I ran these programs on a machine with 8K. Anyway, when the subroutine returns in line 50, the N=N+1 updates N to the value it was before the GOSUB, F=N*F updates (N-1)! to N! and then we return. If you are confused about this program, you might want to print out the values of N and F at line 40 before the IF statement. (i.e. type: 40 PRINT N,F: IF N=0 THEN F=1: RETURN). If confusion s prevails, do not worry about it. There are very few BASIC programs in which this type of trick is used. If you want to run this program in 4K Altair BASIC, change line 60 to: 60 PRINT N;"!="F,:RETURN

-continued

----------- Having Fun With Altair BASIC--Program One -----------

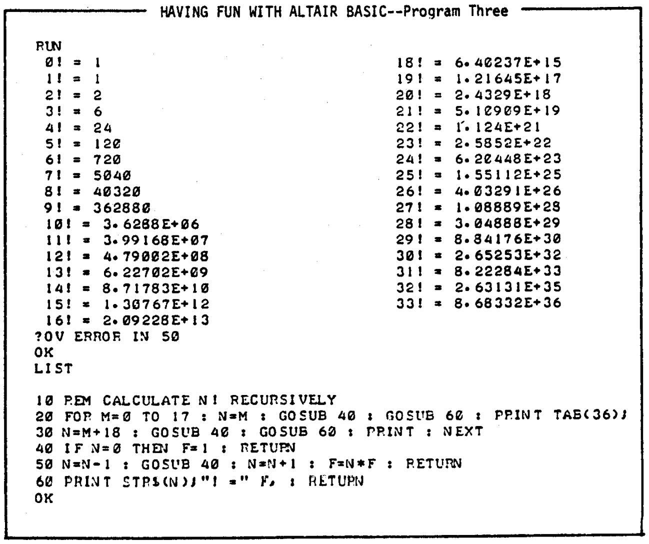
```
RUN
GIVE ME A SENTENCE? THE RAIN IN SPAIN FALLS MAINLY ON THE PLAIN
GIVE ME SOME CHARACTERS? THE
"THE" STARTS AT 1
"THE" STARTS AT 35
"THE" OCCURS 2 TIMES IN "THE RAIN IN SPAIN FALLS MAINLY ON THE PLAIN"

GIVE ME A SENTENCE? I LIKE APPLES COOKED IN TURPENTINE
GIVE ME SOME CHARACTERS? ELEPHANT
"ELEPHANT" OCCURS Ø TIMES IN "I LIKE APPLES COOKED IN TURPENTINE"

GIVE ME A SENTENCE? A MAN LOOKED AT A CAT
GIVE ME SOME CHARACTERS? A
"A" STARTS AT 1
"A" STARTS AT 4
"A" STARTS AT 14
"A" STARTS AT 17
"A" STARTS AT 2Ø
"A" OCCURS 5 TIMES IN "A MAN LOOKED AT A CAT"

GIVE ME A SENTENCE?

OK
LIST

1Ø REM SEARCH FOR A CHARACTER IN A SENTENCE
2Ø INPUT "GIVE ME A SENTENCE";S$
3Ø INPUT "GIVE ME SOME CHARACTERS";W$
4Ø CN=Ø : REM CN=NUMBER OF TIMES W$ OCCURS IN S$
5Ø Q$=CHR$(34) : SQ$=Q$+S$+Q$ : WQ$=Q$+W$+Q$
6Ø FOR I=1 TO LEN(S$)
7Ø IF MID$(S$,I,LEN(W$))=W$ THEN CN=CN+1 : PRINT WQ$ " STARTS AT" I
8Ø NEXT
9Ø PRINT WQ$ " OCCURS" CN "TIMES IN " SQ$ : PRINT
1ØØ GOTO 2Ø
OK
```

-continued



HAVING FUN WITH ALTAIR BASIC--Program Two

```
RUN
HOW MANY ROWS? 9
                                     1
                         1                         1
                   1                  2                  1
               1              3               3              1
            1           4           6            4           1
          1         5         1Ø         1Ø          5         1
        1         6        15        2Ø         15        6         1
       1        7       21       35       35        21       7        1
      1       8      28      56      7Ø       56      28      8       1
     1      9     36     84      126     126     84     36     9      1

OK
LIST

1Ø REM PRINT PASCAL'S TRIANGLE LIKE A TRIANGLE
2Ø INPUT "HOW MANY ROWS";R : R=R+1
3Ø DIM P(R,R)
4Ø REM GENERATE PASCAL'S TRIANGLE
5Ø P(1,1)=1 : REM INITIALIZE THE TRIANGLE
6Ø FOR I=2 TO R : FOR J=1 TO I : P(I,J)=P(I-1,J-1)+P(I-1,J) : NEXT J,I
7Ø REM PRINT THE TRIANGLE
8Ø FOR I=1 TO R : LL=Ø
9Ø FOR J=1 TO I : LL=LL+LEN(STR$(P(I,J))) : NEXT : REM LL=LINE LENGTH
1ØØ SP=INT((72-LL)/(I+1)+.5) : REM SP=NUMBER OF SPACES BETWEEN NUMBERS
11Ø REM PRINT A ROW
12Ø FOR J=1 TO I : PRINT SPC(SP);STR$(P(I,J)); : NEXT : PRINT : NEXT
OK
```



HAVING FUN WITH ALTAIR BASIC--Program Three

```
RUN
 Ø! = 1                               18! = 6.4Ø237E+15
 1! = 1                               19! = 1.21645E+17
 2! = 2                               2Ø! = 2.4329E+18
 3! = 6                               21! = 5.1Ø9Ø9E+19
 4! = 24                              22! = 1.124E+21
 5! = 12Ø                             23! = 2.5852E+22
 6! = 72Ø                             24! = 6.2Ø448E+23
 7! = 5Ø4Ø                            25! = 1.55112E+25
 8! = 4Ø32Ø                           26! = 4.Ø3291E+26
 9! = 36288Ø                          27! = 1.Ø8889E+28
 1Ø! = 3.6288E+Ø6                     28! = 3.Ø4888E+29
 11! = 3.99168E+Ø7                    29! = 8.84176E+3Ø
 12! = 4.79ØØ2E+Ø8                    3Ø! = 2.65253E+32
 13! = 6.227Ø2E+Ø9                    31! = 8.22284E+33
 14! = 8.71783E+1Ø                    32! = 2.63131E+35
 15! = 1.3Ø767E+12                    33! = 8.68332E+36
 16! = 2.Ø9228E+13
?OV ERROR IN 5Ø
OK
LIST

1Ø REM CALCULATE N! RECURSIVELY
2Ø FOR M=Ø TO 17 : N=M : GOSUB 4Ø : GOSUB 6Ø : PRINT TAB(36);
3Ø N=M+18 : GOSUB 4Ø : GOSUB 6Ø : PRINT : NEXT
4Ø IF N=Ø THEN F=1 : RETURN
5Ø N=N-1 : GOSUB 4Ø : N=N+1 : F=N*F : RETURN
6Ø PRINT STR$(N);"! =" F, : RETURN
OK
```

SoftWARE HINTs for 8800

by Bill Gates

When I first heard about 8-bit computers, I thought about how difficult 12-bit computers, like the PDF-8, are to program. The PDP-8 has one accumulator, 256 words directly addressable and eight instructions. Cutting down on this would make a computer unusable.

The 8080 instruction set is actually much better than that of the PDP-8. There are seven accumulators, 65,536 words directly addressable and 78 instructions. How is it possible to have all this on a computer with only 2/3 as many bit per word? One of the most important reasons is the use of multi-byte instructions. Any possible address can be specified in three byte instructions that use the second and third bytes to form an address. For ease of manipulation up to three addresses can be stored in the registers. Decrementing (DCX), incrementing (INX) or adding another number (DAD) to these addresses are all one instruction operations.

Another important thing about the 8080's instruction set is the stack, used both for storing temporary values and subroutine return addresses. Two registers can be stored in, or loaded from memory with a single PUSH/or POP instruction. When a subroutine returns, conditionally or unconditionally, no address needs to be specified since the new program counter is always taken from the top of the stack. The stack allows a programmer to be tricky and elegant using very few instructions.

4K BASIC is a good example of how compact a complicated program written for the Altair can be. Some 16-bit machines have 4K BASIC's as good as Altair 4K BASIC--but, to use the same amount of memory as Altair 4K BASIC, a 16-bit machine would have to have a 2K BASIC which is unheard of.

BCD Arithmetic

BCD stand for binary coded decimal. This is a way of storing numbers according to their decimal digits. Four bits are used to store each digit, so two digits are stored in each word. Each decimal digit is represented by its value in binary, so 0=0000, 1=0001, 2=0010, 3=0011, 4=0100, 5=0101, 6=0110, 7=0111, 8=1000, 9=1001. This leaves six possible configurations of digits which are meaningless (1010, 1011, 1100, 1101, 1110, 1111). These wasted comfinations mean that BCD is not as compact a way of storing numbers as is binary. In binary, a word can have values from 0 (all 0's) to 255 (all 1's). In BCD, a word can range from 0 (all 0's) to 255 (all 1's).

The advantage of BCD is that decimal input and output are extremely easy. With binary numbers, decimal input involves multiplying by 10 and decimal output dividing by 10. If the operations to be performed on a number are simple (addition or subtraction), BCD may be a more convenient form of storage. Simple calculators use BCD for internal storage, whereas more powerful calculators, such as those that have trigonometric functions, convert numbers to a binary format.

The Altair has a special instruction for BCD arithmetic called DAA (decimal adjust). Adding 31=0011 0001 to 56=0101 0110 will give 87=0100 0111, so the ADD instruction works fine in this case. The problem with using ADD occurs whenever you get decimal carry. Example: 46=0100 0110 + 27=0010 0111 will give 0110 1101, which is meaningless as a BCD number. If a result has a digit greater than 9, we want to have the next higher digit incremented. Also, consider 19=0001 1001 + 48=0100 1000 = 0110 0001 which is 61. The carry from the low order digit resulted in a number greater than 15, so the 10's digit was affected. Therefore, only 10 was added to the number, instead of the 16 that should have been added. The CY1 flag bit, in the PSW, is used to remember if a carry occured out of the fourth bit, i.e., whether the 10's digit was affected. The DAA instruction is always used after adding BCD numbers. DAA, the only instruction that uses CY1, looks at the number in accumulator A and reformats it as a BCD number. This is done by adding 6 if CY1 is on, checking for a digit being greater than 9 and if so, subtracting 10 from that digit and incrementing the next higher digit. If the high order digit overflows, carry is set.

```
;Routine to convert A from BCD
;to binary.  13 bytes.

;A routine which doesn't loop takes
; 14 bytes

BCDBIN:      PUSH B              ;save [B,C]
             ORA  A              ; turn carry off
             MVI  B,100 decimal

LOOPBC       INR  A              ; add to BCD #
             DAA                 ; set carry if
                                 ;equal to 100
             DCR  B              ; count down
                                 ;[B]
             JNC  LOOPBC         ;if [A] is not equal to 100
                                 ;continue to
                                 ;count down
             MOV  A,B            ;save result
                                 ;in [A]
             POP  B              ;restore [B,C]
             RET
```

If anyone has a shorter solution than the above, please send it in.
Challenge: What is the shortest Binary to BCD routine?

-continued

Hints For 8800

THE LXI TRICK

Many computers have SKIP instructions. Having multi-word instructions makes it difficult to tell how much should be skipped, so computers with multi-word instruction are seldom provided with a SKIP instruction. the LXI trick, however, allows the skipping of one or two words. Example: (from Altair BASIC)

```
ERR3:MVI     E,3                 ;set up the error #
     JMP     ERROR
ERR7:MVI     E,7                 ;set up the error #
     JMP     ERROR
ERR5:MVI     E,5                 ;
     JMP     ERROR
ERR2:MVI     E,2
ERROR:LXI    B,ERRMSG
```

change this to:

```
ERR3:        MVI  E,3
             1                   ;first byte of LXIB,
ERR7:        MVI  E,7
             1                   ;first byte of LXIB,
ERR5:        MVI  E,5
             1
ERR2         MVI  E,2
ERROR:       LXI  B,ERRMSG
```

If a jump is made to ERR3, E will be set up. Then a LXIB will be executed. B, and C, will be given garbage values depending on the instructions that follow, and the program counter will be incremented beyond the MVI E,7.

Another example: (from Basic)

```
AND:         MVI  A,1            ;set flag this is AND
             JMP  DOBOOL         ;by setting A non-zero
OR:          XRA  A              ;set flag this is OR
                                 ;by setting A to zero
DOBOOL:      .....
change to:
                                 ;set flag with A not 0
                                 ;first byte of ORI
AND:         366 OCTAL           ;XRA A does not equal 0 so will
                                   not equal zero
OR:          XRA  A
DOBOOL:      ....
```

Skip one byte:

```
MVI   A,(or other register)      ;sets register to next op code,
                                  but does not affect the condition
                                  codes.
CPI                              ; Doesn't change any registers, but
                                  condition codes are set.
ORI   A,                         ;Sets register to next op code, and
                                  will always reset the ZERO condi-
                                  tion code, and carry.
```

Skip two bytes:

```
LXI   B,(or other register pair);Sets register pair to following
                                  two bytes. Does not affect
                                  condition codes.
JC,JNZ,etc.
         (or other conditional Jump)
                                 ;if you know one of the condition
                                  codes is on or off you can skip the
                                  next two bytes by using a JMP which
                                  won't get executed. Condition codes
                                  are not affected, nor are any
                                  registers.
```

-continued

Hints For 8800

Other hints:

If you have the following sequence:

```
            CALL   SUB1
            JMP    LABL1
SUB1:       LHLD    ADR1
```

shorten it to:

```
            LXI    H,LABL1
            PUSH   H
SUB1:       LHLD   ADR1
```

This may look like a fairly special trick, but manipulating return addresses with PUSH's and POP's is a very general technique. For instance, say, a subroutine has a condition in which it wants to return to its caller's caller and not the caller. POPing off the caller's return address will handle this.

Consider this:

```
  CPI   ","          Replace this with:
  JZ    TERMN
  CPI   ":"                                           ;assuming [B,C] is
  JZ    TERMN                     LXI   B,TERMN       ;free,put the
  CPI   12                        PUSH  B             ;address to go to
  JZ    TERMN                                         ;onto the stack.
  CPI   ";"                       CPI   ","
  JZ    TERMN                     RZ                  ;go to TERMN if
                                  CPI   ":"           ;matches
                                  RZ
                                  CPI   12
                                  RZ
                                  CPI   ";"
                                  RZ
                                  POP   B             ;eliminate TERMN
                                                      ;address from the
                                                      ;stack
```

Final challenge: Write a short subroutine that finds the parity of A without using the parity flag. Give the result in the carry flag, and don't smash any registers but A. You can use the stack.

Software Notes

by Bill Gates

Though the most difficult and enjoyable part of writing a program is the design of data structures and program flow, it is also important to use the least number of instructions possible to perform each function in a program. For instance:

```
CALL SUB1      should be replaced by
RET

JMP SUB1       unless something fairly
               tricky is being done
```

with return addresses. The JMP is faster, takes one less byte, and uses no stack space. An instruction book on programming the 8008 ignores this simple fact!

JMPs should be avoided wherever possible. By rearranging code you can often avoid having an unconditional JMP by falling into the routine you were JMPing to.

The beginning programmer will use lots of SHLDs, LHLDs, STAs and LDAs when they are not necessary. The stack can be used to save temporary values in most cases. SHLDs, LHLDs, LDAs and STAs should only be used for values referenced in many different contexts within a program, i.e. an I/O parameter or the current line number.

A good technique for familiarizing yourself with the instruction set is to go out of your way to use every instruction at least once (except perhaps DAA). Go through the instruction set from time to time and look closely at the instructions you seem to use very rarely. With few exceptions (DAA, SPHL) all the instructions can be used to advantage, even in small programs. One of the most overlooked instructions is XTHL. When all the accumulators have values that must be saved and a value needs to be taken off the stack, XTHL is the only instruction that can be used.

```
Example:  ;Exchange [B,C] with [H,L]

PUSH B    ;put [B,C] on the stack

XTHL      ;[H,L] = top stack entry =
          [B,C]

          ;[H,L] goes on the stack

POP B     ;[B,C] = original [H,L]
```

Sometimes the simple way of doing things is the best. PUSH B/POP D may seem like a tricky way of setting [D,E] = [B,C], but the obvious sequence MOV D,B/MOV E,C is much faster.

Some tricks involve instruction sequences which at first sight seem meaningless. For instance: SUB A or XRA A. Subtracting A from itself or exclusive-oring A with itself are the only one-byte ways of setting A=0. MVI A,0 must still be used if the condition codes need to be preserved, but this is rare.

ADC A is equivalent to RAL, except it affects all the condition codes. SBB A sets A=0 if carry is off and A=377 if carry is on. The routine below uses this fact to convert A as a signed integer to a double byte signed integer in [H,L]:

```
MOV L,A  ;setup the low order
         ;now the sign must be
         ;"extended" by setting H=0
         ;if A=>0 and H=377 otherwise
RAL      ;Carry = 1 if A<0
         ;Carry = 0 if A=>0
SBB A    ;A=0 if old A was =>0
         ;A=377 if old A was <0
MOV H,A  ;setup the high order
```

```
The sequence:  INR E
               DCR E
```

doesn't modify any values, but it does set the condition codes (except carry) depending on what is in E. If E is being used as a flag to indicate, say, whether or not a decimal point has been seen, the zero flag is set up to do a conditional JMP.

The subject of good decimal print routines has been discussed extensively in the Altair Software Department this week. This routine is one of the four or five I wrote this week -- each with its own advantages and disadvantages. This one is fairly tricky, in that it takes a little bit of looking at to understand.

```
#1 ;
   ;Print the binary unsigned number
   ;in [H,L] in decimal, suppressing
   ;leading zeros
   ;
   ;24 bytes (25 if saves D,E)
   ;ON RETURN:
   ;A = last digit in ASCII
   ;B,D = 255 (all constants in
   ;decimal)
   ;C,E = last digit -1Ø
   ;H,L = Ø
   ;
   ;Uses up to 18 bytes of stack
   ;Total compute time up to 85
   ;milliseconds
   ;
   ;IDEA: calculate a digit, save it
   ;      on the stack, and call the
   ;      digit calculator to calcu-
   ;      late and print higher order
   ;      digits, pop the digit off
   ;      and print it.
   ;
```

-continued

Software Notes

```
DECOUT:   LXI B, -1Ø         ;CALL here.
GETDIG:   MOV D,B            ;[D,E] = -1
          MOV E,B            ;since B = 255
LOOPSB:   DAD B              ;Subtract 1Ø from [H,L] until [H,L] < 1Ø. Carry
                             ;won't be set by the last DAD when [H,L] < 1Ø.
          INX D              ;increment the count
          JC LOOPSB          ;loop subtracting
          PUSH H             ;[L] = current digit -1Ø
                             ;Save the current digit on the stack.  Change to
                             ;XTHL and add PUSH D at GETDIG to save [D,E].
          XCHG               ;[H,L] = old [H,L]/1Ø
          MOV A,H            ;Set zero flag if [H,L] = Ø
          ORA L
          CNZ GETDIG         ;If not zero, print the higher order digits and
                             ;then return here to print this digit.
          MVI A, "Ø" + 1Ø    ;A = constant to add to digit
          POP B              ;pop the digit into C
          ADD C              ;A = ASCII of digit
          JMP OUTCHR         ;Jump to the routine to print A and return.  If
                             ;OUTCHR is located next, the JMP can be eliminated.
```

Parity is used as a check to detect errors in data transmission. Each data word is given an additional bit which is set to 1 if there are an odd number of 1's in the data and Ø otherwise. When the data is received the parity bit is checked to make sure it is set properly. Thus, if you are reading a 7-bit ASCII paper tape with the 8th bit used for parity, the parity of the entire 8 bits should be even.

The reason I first thought about a parity routine for the 8080 is that the parity condition code and all the instructions related to it (JPO, JPE, RPE, RPO, CPO, CPE) are seldom used. I wondered how difficult it would be to calculate parity if the parity flag were removed. A user-settable flag would be much more useful than the parity flag. BASIC uses the parity flag in only about eight places, and all of these are special tricks. Here is the smallest parity routine I've been able to write:

```
;Enter with number in A.  1Ø bytes.
;On exit, A=Ø and all the other reg-
;isters are preserved.
;Carry is set depending on A's
;parity.
;Enter at ODDPAR for carry on to
;mean odd parity.

ODDPAR:   ADD A           ;Move a bit of A into carry.

          RZ              ;If all bits added into carry, return.

          JNC ODDPAR      ;If no bit moved into carry, rotate more.

;enter at EVNPAR for carry on to
;mean even parity

EVNPAR:   ADI 200         ;Complement the parity of the remaining bits

          JMP ODDPAR      ;Rotate more.
```

GENERAL SoftWARE UpDATE INFO

by Paul Allen

Programmed I/O

The coding technique for data input and output in which the CPU waits for completion of the I/O operation is usually termed "programmed I/O." This is by far the easiest and most common way of writing input and output subroutines for the Altair, and is used by BASIC and the Package I software.

There are usually two subroutines for each device. One that inputs a character from the device and one that outputs a character to the device. The input routine (INCHR) waits for the device's input buffer full flag to be set and then reads the character. On the Altair, the device status is in the input side of the lower I/O channel, and the data is read from that channel +1. Assuming we will return the byte read from the device in the A register, the code is as follows (for an old SIOC board--character ready bit in bit 5):

```
INCHR:   IN INCHN      ;where ICHN is the input channel

         ANI 40Q       ;TEST BIT 5=0 (Q means octal). The mask 40Q is
                       ;"anded" with the device status in the A register.
                       ;The mask (40Q) selects only bit 5.

         JZ INCHR      ;If no input data ready, loop.

         IN INCHN+1    ;Read the input byte.

         RET           ;Return from the subroutine.
```

Note that the input character routine is a "subroutine" that could be called many different places in a program by using a CALL instruction, i.e.

```
     :
     :
CALL INCHR     ;Get a character from the terminal.

CPI 15Q        ;Was it a carriage return?

JZ ENDLIN      ;If so, end of input line.
     :
     :
```

Of course, the stack pointer must be set up pointing to an area of memory set aside for use by subroutine calls and PUSH/POP and other stack manipulations. This is most easily done as follows (this code is usually placed at the start of your program):

```
START:   LXI    SP, STACK
          :
          :
          :
         DS     20       ;set aside 20 locations (10 levels) of stack space
STACK:
```

A corresponding character (byte) output subroutine for an old (REV 0) SIO board is listed below. The byte to be output is in the A register:

```
OUTCHR:  PUSH PSW        ;Save the A register on the stack.

OUTLP:   IN INCHN        ;Read the device status into the A register.

         ANI 2Q          ;See if bit 1 is = 0.

         JZ OUTLP        ;If it is, keep waiting for the terminal to finish
                         ;printing.

         POP PSW         ;Get back the saved output byte.

         OUT INCHN + 1   ;Now output the byte to the terminal.

         RET             ;Return from subroutine.
```

Often it is desirable to echo the character read from a terminal's keyboard immediately back to the terminal. The easiest way to do this is to insert

```
INECHO:  CALL INCHR
```

right before the OUTCHR routine and then call INECHO instead of INCHR. If we knew we were always going to echo the input character back to the terminal, we could have the input character subroutine (INCHR) "fall into" the output character routine (OUTCHR). This may be done by placing INCHR directly ahead of OUTCHR and also removing the RET at the end of INCHR so an "OUTCHR" will always be performed when INCHR is called.

-continued

Software Update

Slight modifications must be made to these routines if we want to use REV 1 or modified REV 0 serial I/O boards. In these boards, the character ready bit is in bit 0 of the status byte, and the character done (sent) bit is in bit 7. Also, the bits are "active low," that is, a 1 means the bit is false and a zero means the bit is true, which is just the opposite of the way the bits were set on the REV 0 board used in the previous examples. We could test bits by using an AND immediate instruction as before (i.e. replace the ANI 40Q in INCHR with an ANI 1Q and the ANI 2Q to an ANI 200Q) and changing the JZ's to JNZ's. However since the status bits are in the least and most significant bits in the status byte, we can conveniently test them by using the rotate instruction to move the bit in question into the carry flag and then using a JNC instruction to loop:

```
INCHR:  IN ICHN    ;Read status
        RAR        ;Character ready?
        JC  INCHR  ;If not, loop
        IN INCHN+1 ;Read character
        RET        ;Return

OUTCHR: PUSH PSW   ;Save character
OUTLP:  IN ICHN    ;Read status
        RAL        ;Test bit 7
        JC  OUTLP
        POP PSW    ;Get character
                   ;back in A
        OUT INCHN+1;Send it to
                   ;terminal
        RET        ;All done, return
```

Using rotates instead of ANIs saves one byte in each routine. Remember: taking care to save each byte you can will make long programs significantly shorter and faster.

PIO boards (often used for SWTPC TVTs) have the status bits "active low" like REV 1 SIO boards, but the status bits are in different positions: character ready is bit 1 and character done is bit 0, so:

```
INCHR:     IN   INCHN
           ANI  2Q
           JNZ  INCHR
           IN   INCHN+1
           RET

OUTCHR:    PUSH PSW
OUTLP:     IN   ICHN
           RAR
           JNC  OUTLP
           POP  PSW
           OUT  INCHN+1
           RET
```

If you are confused by the use of "masks," here is an explanation. If we want to make a jump on only one bit of the A register, we "and" a mask with that bit on with A. The result of the AND will be zero if that bit was zero, and non-zero if the bit was one. Here is a table of bit masks (in octal) for each bit position:

BIT	MASK
0	1 (usually use PAR to test)
1	2
2	4
3	10
4	20
5	40
6	100
7	200 (usually use RAL to test)

Note that bits 0 and 7 take fewer bytes to test than the rest because they can be rotated into the carry status bit as mentioned earlier.

It is often very useful to use bit testing and setting in a program. Suppose you are writing an assembler and you want to remember if you have seen any colons or commas on a line. You could use one bit in a register to flag the fact you had seen a colon and another bit to flag whether you had seen a comma; and you could use the other six bits of the register for six other flags. Suppose the flags were kept in the B register. Then, to set a flag (if bit=1 means set):

```
        .
        .
        .

MOV A,B    ;Get flag register in A

DRI 2      ;Mark colon seen (bit 1)

MOV B,A    ;Save flags back
```

To reset a flag:

```
MOV A,B    ;Get flag register in A

ANI 375    ;377-2
           ;Reset colon flag (bit 1)

MOV B,A    ;Save flags back
```

To test two flags:

```
MOV A,B    ;Get flag register

ANI 12Q    ;Test both bits 3 & 1
           ;(colon and comma)

JZ NETHER ;Jump to NETHER if both
           ;flags = 0

JNZ ONEFLG ;Jump to ONEFLG if one
           ;or both of two flags
           ;set.
```

To complement (invert) a flag (reset it if set, set it if reset):

```
MOV A,B    ;Get flag register

XRI 2      ;Flip (complement) bit 1

MOV B,A    ;Save flags back
```

Software Notes

by Bill Gates

Using the STACK

Every program written for the 8800, large or small, should take advantage of the stack. The stack is a stored list of data which behaves in a last in--first out (LIFO) fashion. That is, PUSH D/POP D doesn't modify [D,E] since the POP removes the same two bytes that were just pushed onto the stack. There are three things to remember in using the stack:

1) Initialize the stack pointer to the highest location in a block of free Read/Write memory. As data is pushed onto the stack the stack pointer is decremented.

2) Make sure that the amount of space set aside for the stack is sufficient to hold all the values (including return addresses) that are stored on the stack at one time. Every time a PUSH or CALL is done the stack pointer is decremented by two, so data pushed onto the stack will be stored in lower and lower locations. Every POP or RETurn increments the stack pointer by two, so the important thing is not how many values are PUSHed on during a program, but how many are on at the same time. Consider:

```
          LXI    SP, STKBOT
LOOPDO:   PUSH   D            ;save [D,E]
           :
           :
          CALL   SUB1
RETADR:    :
           :
          POP    D            ;restore [D,E]
          DCR    B
          JNZ    LOOPDO
           :
           :
          DS 4
STKBOT:
SUB1:      :                  ;no stack use
           :
          RET
```

No matter how many times the above loop is executed, [D,E] will always be saved and restored from the same memory locations, since the stack pointer is incremented by 2 by the PUSH, incremented by 2 by the CALL, decremented by 2 by the RET, and decremented by 2 by the POP. Only four bytes of stack space are set aside for this program since only 2 bytes of data (D and E) and 2 bytes of return address are ever stored on the stack at a time. If additional PUSH/POPs or CALLs are done inside the loop, more stack space would have to be set aside. Unless you are using all of memory and need to compactify as much as possible, it is a good idea to allocate a lot more stack space than you think you will ever use.

3) Unless you are being very tricky, never take more data off of the stack than is put on. This means doing more POPs and RETs than CALLs and PUSHs.

ILLUSTRATION OF STACK OPERATION

Consider the example given earlier. Assume that STKBOT = 200 octal.

```
         :
         :
     program data

174  |--------|
175  |--------| ;set aside for stack use
176  |--------| ;initial contents irrelevant
177  |________|

200  other data
               After the LXI SP, STKBOT
               SP will equal 200.
```

PUSHs and CALLs always put 2 bytes of data onto the stack as follows:

1) Decrement SP
2) Store the high 8 bits of data being PUSHed in the memory location given by SP.
3) Decrement SP
4) Store the low 8 bits of data being PUSHed into the memory location given by SP.

When the PUSH D is done: SP = 177, D is stored at 177, SP = 176, E is stored at 176.

Say RETADR = 124
When the call is done: SP = 175, high byte of Retadr = 0 and is stored at 175, SP = 174, the low byte of Retadr = 124 and is stored at 174.

So we have:

```
174  [ 124 ]
             the return address
175  [  0  ]

176  [  E  ]
             old [D,E]
177  [  D  ]
```

POPs and RETs do the "reverse" operation:

1) Pick up the low 8 bits of data in the memory location given by the SP
2) Increment the SP
3) Pick up the high 8 bits of data in the memory location given by the SP
4) Increment the SP

Exercise: Work out the details of how the RET and POP D in the example work.

Other machines that have stacks vary in the details of implementation Some machines increment the SP on PUSHs and decrement on POPs (PDP-1Ø). Other machines store before decrementing on PUSHs and increment before fetching on POPs (Altair 680). However, the basic notion of a last in-first out list to store data and return addresses remains the same.

The way subroutines are nested, that is, always returning to the most recent caller, makes the stack very natural for storing return addresses.

The fact that the stack is used to store both return addresses and data allows for some tricky programming involving manipulation of return addresses on the stack, of which a few examples were given in earlier "Software Notes." It also causes some trouble however for subroutines that wish to leave results on the stack or fetch arguments from the stack, since the return address gets in the way. The sequence:

```
MOV    C,M
INX    H
MOV    B,M
INX    H
PUSH   B
```

was used many, many times in BASIC so it was decided to use one of the RST instructions to perform this operation. It was coded as follows:

```
XTHL                       ;[H,L]=return addres
SHLD    PUSHMA+1           ;modify a JMP
POP     H                  ;get [H,L] back
MOV     C,M
INX     H
MOV     B,M
INX     H
PUSH    B
PUSHMA:
JMP     *                  ;JMP to return point
```

If only a single byte of data needs to be pushed onto the stack, put the data in A, B, D, or H and do a PUSH PSW, B, D, or H respectively, followed by a INX SP. To pop off the single byte of data do a DCX SP, POP A, B, D, or H. This puts garbage into the PSW, C, E, or L respectively. Unless the same PUSH will be used to store a large number of one byte pieces of data on the stack, it is simplest to merely do a PUSH/POP sequence and allow the extraneous byte to be stored.

-continued

Software Notes

Sometimes the amount of stack space a program requires will depend on the input to the program, for instance when BASIC evaluates complicated formulas. If this is the case, a check must be done when data is pushed on to make sure the stack isn't "overflowing." If it is, either some sort of recovery procedure must be invoked, or an error message printed. The following subroutine checks to see if the stack is pointing below STKSTP.

```
;saves all registers
CHKSTK:  PUSH   H          ;save [H,L]
         LXI    H,-STKSTP  ;won't work for STKSTP=0
         DAD    SP
         POP    H          ;[H,L]
         RC                ;return if still ok
;here on stack overflow
```

If you have a subroutine which is often passed constant arguments such as:

```
          MVI    C,3
          CALL   SUB1
           :
          MVI    C,5
          CALL   SUB1
           :

                MVI    C,7
                CALL   SUB1
                 :
      SUB1:
```

By manipulating the return address you can save one byte per call as follows:

```
          CALL   SUB1C
          DB     3    ;put constant in
           :          ;return location
          CALL   SUB1C
          DB     5
           :
          CALL   SUB1C
          DB     7
           :
 SUB1C:   XTHL        ;[H,L]=return address
          MOV    C,M  ;fetch the constant
          INX    H    ;update return address
          XTHL        ;restore the return
 SUB1:     :          ;address and [H,L]
```

This is not a useful technique in most cases, but it does give a good example of XTHL.

Notes on Disk Extended BASIC

Disk Extended BASIC is a stand-alone system which is delivered on a floppy disk. This floppy has been formatted and loaded with the utility files that print directories, format other disks and do disk diagnostics as well as Extended BASIC. The disk loader is about 100 bytes and can either reside on a PROM, be keyed in or be loaded from ACR or paper tape using the standard 20-byte bootstrap.

During initialization the number of disk buffers (maximum 8) and random access blocks (maximum 8) to be allocated are determined by the user. These numbers determine the number of files that can be open simultaneously and the number of random access files that can be open simultaneously, respectively. The disk drives that are to be brought on line are all checked for proper formatting and the locations of free sectors are stored in memory.

Each floppy can store 300,000 bytes (characters) of user information. The rest of the storage space on the floppy is used to store the file structuring and error detection information. Up to 254 files can be stored on a floppy and a single file can be up to 300,000 bytes long. A file must reside entirely on a single floppy, thus no file can be larger than 300,000 characters.

There are three modes for file access:

1) Sequential input: The file is stored as ASCII text. Numbers and strings are read as character strings in exactly the order they were typed in or written out.

2) Sequential output: Any previous contents of the file are deleted and output is done item by item in ASCII.

3) Random access: Each record is 128 characters. Numbers are written in binary, so integers take 2 bytes, single-precision numbers 4 bytes and double-precision numbers 8 bytes. Special functions return the record number of the current position in the file (LOC) and the highest numbered record currently allocated (LOF) in the file. READs and PRINTS of random access records can be intermixed. A specific record number in the file can be specified by a formula in both the READ and PRINT statements.

All the features of non-disk Extended BASIC are provided.

To use Disk Extended BASIC, 20K of memory is required since the program itself uses 16.5K and each disk buffer and random access block require another 140 bytes.

end

I/O Programs for the ACR

Input/Output programs for the 88-ACR

By Tom Durston

One request we've been getting frequently is for simple machine language programs to write and read data on tape through the 88-ACR. Listed below is a program to write and a program to read using the 88-ACR. These programs have been used in our engineering department to store lengthy test routines, and can be used for any type of data.

WRITE PROGRAM - 38 bytes

Writing data on tape through the 88-ACR is accomplished by first specifying the start address of data and the end address of data. Then a test byte (000 in this program) is written, followed by data output. The last portion of the program tests to see if the program has transmitted the last byte of data. If it has, the program jumps to the last positions in memory, and is observed by a change in the address lights on the front panel. If the program hasn't outputted the last data byte, the H & L registers are incremented by 1 and the program outputs the next byte. This program is placed in the upper portion of 4K memory with a starting address of 017,000. The location may be changed, but be sure to change all jump addresses accordingly. After recording data that includes program information, write down the start and end address on the tape cartridge along with the name and test byte of the program for identification.

When recording data at the beginning of a cassette tape, record at least 15 seconds of steady tone before running the write program (to get past the plastic leader and wrinkles in the beginning of the tape). Also, if recording more than one batch of data, leave at least 5 seconds of steady tone between batches. This program is written for 88-ACR addresses of 6 & 7.

88-ACR WRITE PROGRAM

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
	LXI	017,000	041	Load immediate H&L register pair
		1	xxx	Lo } starting address of
		2	xxx	Hi } data to be written
	LXI	3	001	Load immediate B&C register pair
		4	xxx	Lo } end address of
		5	xxx	Hi } data to be written
	MVI	6	076	Move immediate to accumulator
		7	000	Test byte to be written at beginning
	OUT	017,010	323	Output data from accumulator
		11	007	Data channel # of 88-ACR
TEST	IN	12	333	Input data to accumulator
		13	006	Status channel # of 88-ACR
	RLC	14	007	Rotate accumulator left, test for D7 true
	JC	15	332	Jump if carry (D7 not true)
		16	012	} To "TEST"
		17	017	
	MOV	017,020	176	Move contents of memory specified by H&L register to accumulator
	OUT	21	323	Output data from accumulator
		22	007	Data channel # of 88-ACR
	MOV	23	175	Move contents of L register to accumulator
	CMP	24	271	Compare accumulator vs B register
	JNZ	25	302	Jump if not zero (L ≠ B)
		26	040	} To "NEXT"
		27	017	
	MOV	017,030	174	Move contents of H register to accumulator
	CMP	31	270	Compare accumulator vs C register
	JNZ	32	302	Jump if not zero (H ≠ C)
		33	040	} To "NEXT"
		34	017	
	JMP	35	303	Jump (if L = B and H = C)
		36	375	} To "END"
		37	017	
NEXT	INX	017,040	043	Increment register pair H&L
	JMP	1	303	Jump
		2	012	} To "TEST"
		3	017	
END	JMP	017,375	303	Jump (loop to self)
		376	375	} To "END"
		377	017	

-continued

READ PROGRAM - 48 bytes

As in the write program, start and end addresses of incoming data are specified first. Next, the program looks for the test byte (000 in this program). Once the test byte is detected, the program inputs data and stores it in memory as specified by the H & L registers. The next portion of the program tests to see if the end memory address has been filled. If it has, the program jumps to the last positions in memory, and is observed by a change in the address lights on the front panel. If it is not the end, then the program increments H & L by 1 and jumps back to input another data byte. This program is placed in the upper portion of 4K of memory with a starting address of 017,000. The location may be changed, but be sure to change all jump addresses accordingly. When reading data back in, the tape and program should be started a few seconds before the start of data.

88-ACR READ PROGRAM

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
	LXI	017,000	041	Load immediate H&L register pair
		1	xxx	Lo } starting address of
		2	xxx	Hi } data to be read
	LXI	3	001	Load immediate B&C register pair
		4	xxx	Lo } end address of
		5	xxx	Hi } data to be read
TSTBT	IN	6	333	Input data to accumulator
		7	006	Status channel # of 88-ACR
	RRC	017,010	017	Rotate accumulator right (test DØ true)
	JC	11	332	Jump if carry (DØ not true)
		12	006	} To "TSTBT"
		13	017	
	IN	14	333	Input data to accumulator
		15	007	Data channel # of 88-ACR
	CPI	16	376	Compare immediate with test byte vs accumulator
		17	000	Test byte
	JNZ	017,020	302	Jump if not zero (test byte≠input byte)
		21	006	} To "TSTBT"
		22	017	
TEST	IN	23	333	Input data to accumulator
		24	006	Status channel # of 88-ACR
	RRC	25	017	Rotate accumulator right (test DØ true)
	JC	26	332	Jump if carry (DØ not true)
		27	023	} To "TEST"
		017,030	017	
DATA	IN	31	333	Input data to accumulator
		32	007	Data channel # of 88-ACR
	MOV	33	167	Move contents of accumulator to memory address specified by H&L registers
	MOV	34	175	Move contents of L register to accumulator
	CMP	35	271	Compare accumulator vs B register
	JNZ	36	302	Jump if not zero (L ≠ B)
		37	051	} To "NEXT"
		017,040	017	
	MOV	41	174	Move contents of H register to accumulator
	CMP	42	270	Compare accumulator vs C register
	JNZ	43	302	Jump if not zero (H ≠ C)
		44	051	} To "NEXT"
		45	017	
	JMP	46	303	Jump (if L = B and H = C)
		47	375	} To "END"
		017,050	017	
NEXT	INX	51	043	Increment H&L register pair
	JMP	52	303	Jump
		53	023	} To "TEST"
		54	017	
END	JMP	017,375	303	Jump (loop to self)
		376	375	} To "END"
		377	017	

Software Notes

by Bill Gates

To go with the decimal output routine given in September's issue, here is a decimal input routine. For fun, modify it so it checks for overflow. (Hint: use the carry bit generated by DAD.)

```
;routine to do decimal input (DECINP)
;return result in [H,L].  [A] contains the terminating character.
;[D,E] is smashed.  Stack use:  INCHR is called to get a character
;in [A].  Overflow is not checked.

DECINP:    LXI    H,Ø         ;initialize to zero
DECLOP:    CALL   INCHR       ;read a character into [A]
           CPI    "9" + 1     ;see if it is > "9"
          -RNC                ;return if so
           CPI    "Ø"         ;see if it is < "Ø"
           RC                 ;return if so
           SUI    "Ø"         ;[A] = numeric value of new digit
          †MOV    D,H         ;[D,E] = [H,L]
          †MOV    E,L
           DAD    H           ;[H,L] = old [H,L]*2
           DAD    H           ;[H,L] = old [H,L]*4
          †DAD    D           ;[H,L] = old [H,L]*5
           DAD    H           ;[H,L] = old [H,L]*1Ø
           MVI    D,Ø         ;[D,E] = new digit
           MOV    E,A
           DAD    D           ;add in the new digit
           JMP    DECLOP      ;get more digits

      †eliminate for octal input
```

The simplicity of loading BASIC into an Altair is important, since people without PROMs or BASIC on ROM must load it every time they power up their machine. Here are the details of how this process works:
(All numbers are octal)

The format of a binary tape of BASIC or a monitor is as follows:

```
leader = 175 currently
last byte of checksum loader (311)
next to last byte of checksum loader (172)
 .
 .   intermediate checksum loader bytes
 .

second byte of checksum loader (61)
first byte of checksum loader (363)
gap of null characters (0)
<checksum data block -- up to 256
 data bytes per block>
<additional checksum data blocks
 until all program data has been given>
<checksum go block>
```

Checksum loaders can be loaded into most pages of memory depending on location 2 of the boot and which checksum loader is on the tape. The checksum loader for 4K BASIC and the Package I monitor starts at location 7400. The checksum loader for 8K BASIC starts at 17400. Except for being relocated, these loaders are identical.

-continued

Software Notes

Checksum data block:

```
74 start character
number of data bytes in the block (Ø=256)
lower 8 bits of storage address
high 8 bits
<data bytes>
checksum byte = summation without carry of all bytes
                in the block except the 170 and count
                specification
```

Checksum: go block

```
170 start character
lower 8 bits of address to jump to
high 8 bits
```

The data block for locations Ø through 376 is the last data block on the tape so the bootstrap loader doesn't have to be keyed in again when checksum errors occur, unless the checksum error is on the final data block.

BOOT STRAP LOADER
start at location zero

```
0/      LXI   H,
         number of bytes in the checksum loader
         page number of the checksum loader

                           Set [H,L] to point to the last location
                           in the checksum loader + 1.

3/      LXI   SP,STKADR    Set [SP] so returns come back to this
                           location.  After each return [SP] is reset.

6/      IN    Ø            See if there is a character, and loop
        RAR                if not.
        RC

12/     IN    1            Read a character and see if it's leader.
        CMP   L            (Lead character = number of bytes in the
        RZ                 checksum loader)

16/     DCR   L            Store the data in the next lower location,
        MOV   M,A          and loop unless all bytes have been read.
        RNZ

21/     PCH   L            Start the checksum loader at its beginning.

22/     STKADR:  DW LOPADR
                           The stack pointer points here, so this
                           gives the address returns branch to.
```

This bootstrap loader has several advantages:

1) Leader is allowed.
2) Only 20 bytes need to be keyed in.
3) It automatically starts the checksum loader, so only one tape needs to be entered.
4) It can run from Read Only Memory.
5) It starts at a convenient location (zero).
6) It is easily relocated by changing the addresses at locations 4 and 22.
7) To load different checksum loaders, only location 2 needs to be changed.

I've written a bootloader that only takes 13 bytes of keyed-in data, but anything smaller than 20 bytes isn't easy to use.

SOFTWARE NOTES

Multi-Precision Arithmetic

By Bill Gates

On the 8080, multi-precision unsigned arithmetic is made easy by the carry bit and its affect on the instructions "ADC" and "SBB". Unsigned arithmetic treats all numbers as positive, with all zeros being the least number, and all ones being the highest. Adding and subtracting memory addresses is the most common form of unsigned arithmetic. Multi-precision arithmetic must be used whenever the range of values desired is greater than that accepted by the arithmetic unit of the computer you are using. The 8080's arithmetic units accept 2-8 bit operands, one from the [A] register and the other from B, C, D, E, H, L, contents of address in [H, L] or the byte following the arithmetic instruction (immediatemode) so anytime values greater than 255 are to be accepted, multi-precision arithmetic must be used. "DCX", "INX" and "DAD" allow 16-bit quantities to be added to, or subtracted from, so 16-bit arithmetic could be considered single precision. However, the lack of any 16-bit arithmetic instructions that use carry to affect their result make it more appropriate to consider 16-bit arithmetic multi-precision.

```
;16-bit unsigned add [H, L] = [D, E] + [H, L]
 ADD16U: DAD D
;carry is set as an overflow indicator
```

```
;32-bit unsigned add [D, E, B, C] (M + 3, M + 2, M + 1, M)

ADD32U: MOV A, C
        ADD M
        MOV C, A
        INX H
        MOV A, B
        ADC M
        MOV B, A                        ;carry is returned as an overflow
        INX H                           ;indicator
        MOV A, E
        ADC M
        MOV E, A
        INX H
        MOV A, D
        ADC M
        MOV D, A
        RET

;16-bit subtract [H, L] = [D, E] - [H, L]

SUB16U: MOV A, E
        SUB L
        MOV L, A
        MOV A, D
        SBC H
        MOV H, A
        RET

;carry indicates that [H, L] was greater than [D, E]
;32-bit subtract [D, E, B, C] = (M + 3, M + 2, M + 1, M) - [D, E, B, C]

SUB32U: MOV A, M
        SUB C
        MOV C, A
        INX H
        MOV A, M
        SBC B
        MOV B, A
        INX H
        MOV A, M
        SBC E
        MOV E, A
        INX H
        MOV A, M
        SBC D
        MOV D, A
        RET
```

-continued

```
;carry indicates [D, E, B, C] was greater than (M + 3, M + 2, M + 1, M)
;add 8-bit [A] to 16-bit [B, C] unsigned.  Result in [B, C]

AD816U:  ADD C
         MOV C, A
         ADC B
         SUB C
         MOV B, A
         ;no overflow indication is given
```

Signed Arithmetic

In signed arithmetic (2's complement) half the numbers are treated as negative and the other half as positive. A 1 followed by all zeros is the smallest number. All 1's is the largest negative number (-1), and all 0's is the smallest positive number. A zero followed by all ones is the largest number. Note that the absolute value of the smallest number (a one followed by all zeros) is larger than the largest number. This creates an overflow case for negation, and makes subtraction tricky if this special case is handled. This "special" negative number is -32768 if 16-bit signed arithmetic is used. This highest 16-bit signed number is 32767.

This signed format allows two numbers to be added through a simple DAD. The only complication is checking for overflow. The table below gives the different possibilities for adding signed numbers:

	Arg 1	Arg 2	Carry	Sign of Result	Overflow
1	pos	pos	off	neg	yes
2	pos	pos	off	pos	no
3	pos	neg	off	neg	no
4	pos	neg	on	pos	no
5	neg	neg	on	neg	no
6	neg	neg	on	pos	yes

Overflow only occurs when the result of adding two positive numbers is greater than 32767, or the result of adding 2 negative numbers is less than -32768. The formula: (⊗ means exclusive - or)

Sign of arg 1 ⊗ sign of arg 2 ⊗ carry ⊗ sign of result is 1, if and only if overflow occurred. Subtraction is merely a negation followed by an addition, unless -32768 is being subtracted (i.e. -30 - (-32768)), in which case no negation is necessary, but the sign of -32768 as an addend must be positive.

```
;16-bit signed negate [H, L] = -[H, L]

NEGIGS:XRA A                        ;get negative [L]
       SUB L
       MOV L, A
       SBB H
       SUB L
       MOV H, A                     ;[H] = -[H] - borrow if any
       SUI 128                      ;see if -32768 (decimal)
       ORA L                        ;with [H] = 128, [L] = 0
       RNZ                          ;if not, return
       JMP OVERFL                   ;overflow here

;16-bit signed add and subtract [H, L] = [D, E] = [H, L]

SUBENT:CALL NEGIGS                  ;entry from subtraction
ADD16S:MOV B, H                     ;MSB of [B] = sign of arg 1
SBZENT:DAD D                        ;do the add
       RAR                          ;MSB of [A] = carry
       XRA B                        ;XOR in sign of arg 1
       XRA D                        ;XOR in sign of arg 2
       XRA H                        ;XOR in sign of result
       RP                           ;return if MSB Ø
       JMP OVERFL                   ;otherwise there was overflow
```

-continued

Software Notes

```
       ;subtract [H, L] from [D, E]
        SUB16S:MOV A, H                     ;is it -32768?
               SUI 128
               ORA L
               JNZ SUBENT                   ;if not, just negate and add
               MOV B, L                     ;say sign is positive
               JMP SBZENT                   ;do the add
```

USR Routines

There are two ways for a "USR" routine to get the argument passed to it as a signed integer in [D, E]. The easiest is to use a CALL followed by the two byte address in locations 4 and 5. The only disadvantage to this is that it has to be changed when you get a new version of BASIC, since the address in locations 4 and 5 changes from version to version. The alternate way is long, but doesn't have to be changed when you get a new version of BASIC. It is:

```
                         LXI H, BACK LC
                         PUSH H
                         LHLD 4
                         PCHL
                 BACKLC: rest of USR routine
```

Here are 4 example USR routines written for 4K and 8K BASIC. In Extended BASIC, the argument is passed and returned in [H, L]. So appropriate modifications will have to be made to use these with Extended BASIC.

```
#1 function:  turn interrupts ON if argument is negative.
              turn interrupts OFF if argument is positive.

              CALL <address at location 4>
              MOV A, D                     ;get argument high order
              ORA A                        ;set MINUS if negative
              DI                           ;assume positive
              RP                           ;return if so
              EI                           ;otherwise, turn interrupts on
              RET

#2 function:  Delay for 11.5u seconds * argument + overhead

              CALL <address at location 4>
   LOOPDL:    DCX D                        ;decrement the argument
              MOV A, E                     ;is [D, E] = 0?
              ORA D
              JNZ LOOPDL                   ;if not, continue looping
              RET

#3 function:  Execute instruction or instructions in [D, E].  Return
              value of [A].

              CALL  address at location 4  ; get argument in [D, E].
              XCHG
              SHLD INSTRS
              LXI, H 0                     ;save the stack pointer
              DAD SP
              SHLD STORSP + 1
              LXI SP, PSWLOC + 1           ;set up to read the USR ac-
              POP PSW                      ;cumulators, fetch the USR
              POP H                        ;accumulators
              POP D
              POP B
INSTRS:       DS 2
              PUSH B                       ;store the USR accumulators
              PUSH D
              PUSH H
              PUSH PSW
              MOV B, A                     ;return the contents of [A].
              XRA A                        ;as the result in [A, B]
STORSP:       LXI SP, 0                    ;restore the stack pointer
              LHLD 6                       ;convert [A,B] and re-enter
              PCHL                         ;the BASIC program
PSWLOC:       DS 8                         ;the accumulators can be set
                                           ;up and examined by using
                                           ;PEEK's and POKE's on these
                                           ;locations
```

-continued

#4 function: dispatch to one of several subroutines depending on the high 8-bits of the argument

```
            CALL <address at location 4>
            LXI H, TBLLOC                   ;point at dispatch table
            MVI B, 0                        ;get dispatch offset
            MOV C, D                        ;in [B, C]
            DAD B                           ;add in 2 * offset since
            DAD B                           ;table entries are 2 bytes
            MOV A, M                        ;fetch the dispatch address
            INX H                           ;into [H, L]
            MOV H, M
            MOV L, A
            PCHL                            ;dispatch

TBLLOC:     DW USRZER                       ;address to go to on zero
            DW USRONE                       ;on one
            DW USRTWO
```

<u>Note on allowing interrupts</u>

To allow interrupts a program must always leave 16-bytes of free stack space. If multiple interrupts can come in 16 *(maximum number at once) bytes must be left free. Also no tricks involving INX SP or DCX SP can be used. The third example USR routine is not interruptible, since an interrupt following the LXI SP would not work.

Slot Machine Game For MITS BASIC

by Jon Walden

This program is written using the combinations and percentages suggested by Donald D. Spencer on pages 219-223 of his book, "Game Playing with Computers".

SLOT MACHINE IS SET UP WITH 3 REELS, 20 SYMBOLS EACH REEL:

	REEL 1	REEL 2	REEL 3	SYMBOL	EQUIVALENT
CHERRIES	4	6	0	=	1
ORANGES	5	4	7	0	2
BELLS	4	6	5	1	3
LEMONS	3	2	4	#	4
WATERMELONS	3	1	3	+	5
BARS	1	1	1	$	6

PAYOFFS ARE AS FOLLOWS: (A = ANY SYMBOL)

COMBINATION			PAYOFF	NUMBER OF POSSIBLE WAYS
=	A	A	$ 3	400
=	=	A	5	240
0	0	$	6	20
1	1	0	8	168
#	#	#	10	24
+	+	$	15	3
0	0	0	18	140
+	+	+	20	9
$	$	$	200	1

PAYOFF AVERAGES $70.49 FOR EVERY $80 PUT IN; NET LOSS IS $9.51; THE HOUSE MAKES 11.89%. (CASINOS ARE THOUGHT TO MAKE BETWEEN 3% AND 50% WITH THE AVERAGE BETWEEN 11% and 12%).

EACH TIME THE REELS SPIN, YOU ARE BETTING A DOLLAR. THE PROGRAM ESTABLISHES THE REEL EQUIVALENTS WITH RANDOM NUMBERS, PRINTS OUT THE SYMBOLS, THE PAYOFF (IF ANY), AND SUMMARIZES YOUR FINANCIAL POSITION AT THAT POINT. (REMEMBER IF YOUR WINNINGS ARE $20 AND YOU WIN A $5 PAYOFF, YOUR NEW WINNINGS ARE $24--$25 MINUS THE DOLLAR YOU BET.)

THE PROGRAM IS WRITTEN IN "MITS" BASIC AND USES THE FOLLOWING VARIABLES:

K	PAYOFF COUNT
L	LOSSES (MONEY PUT IN)
N	NUMBER OF RANDOMS IGNORED
P	PAYOFF
Q	EQUIVALENT OF ALL THREE REELS
R(3)	EQUIVALENT OF INDIVIDUAL REELS
S(6)	SYMBOL EQUIVALENT TABLE
W	WINNINGS (TOTAL PAYOFFS)
X/Y	"FOR" LOOP CONTROLLED VARIABLES
Z	IGNORED RANDOMS
D$	DECISION
R$(3)	SYMBOL FOR INDIVIDUAL REELS
S$(6)	SYMBOL TABLE

I'd like to say something brief about MITS Basic: I think it's great! Comments have been made about it being slow and about certain clumsy features. But the agility to play with bits, to sense ports and to use single ASCII codes is long overdue! I hope the use of "INP" in this program will encourage other programmers to work on new data input methods (especially for games). Having to "hit return" after each entry is a drag!

-continued

Reprinted from the February, 1976, issue of Interface Magazine.

Slot Machine Game

```
100 DATA = ,0,1,#, + ,$
110 FOR X = 1 TO 6 READ S$(x).S(X) = X NEXT
120 IF NOT (INP(0)<128) G010 120
130 OUT 1,12
140 PRINT 1AB(6),">>>>>> SLOT MACHINE<<<<<<" PRINT PRINT
150 N = INT (500*RND (8)) + 1
160 FOR X = 1 10 N Z = RND(8) NEXT
170 PRINT "PRESS THE SPACE BAR TO GET REPEATED"
180 PRINT "REEL SPINS. (EACH SPIN COSTS YOU $1.)"
190 PRINT PRINT "PRESS 'Q' WHEN YOU'RE READY TO QUIT."
200 PRINT PRINT "PRESS ANY OTHER KEY TO GIVE YOURSELF"
210 PRINT "A BREAK. THE SPACE BAR WILL GET YOU"
220 PRING "GOING AGAIN ... GOOD LUCK!!!"
230 IF NOT (INP(1) = 32 OR INP (1) = 81) GOTO 230
240 IF INP (1) = 81 GOTO 750
250 L = L + 1
260 FOR X = 1 TO 3:R(X) = INT(20*RND(8)) + 1:NEXT
270 IF R(1)< 6 THEN R(1) = 2:GOTO 330
280 IF R(1)<10 THEN R(1) = 1:GOTO 330
290 IF R(1)<14 THEN R(1) = 3:GOTO 330
300 IF R(1)<17 THEN R(1) = 4:GOTO 330
310 IF R(1)<20 THEN R(1) = 5:GOTO 330
320 R(1) = 6
330 IF R(2)   7 THEN R(2) = 3:GOTO 390
340 IF R(2)  13 THEN R(2) = 1:GOTO 390
350 IF R (2)  17 THEN R(2) = 2 GOTO 390
360 IF R (2)  19 THEN R(2) = 4.GOTO 390
370 IF R(2) = 19 THEN R(2) = 5:GOTO 390
380 R(2) = 6
390 IF R(3) < 8 THEN R(3) = 2:GOTO 440
400 IF R(3)<13 THEN R(3) = 3 GOTO 440
410 IF R(3)<17 THEN R(3) = 4 GOTO 440
420 IF R(3)<19 THEN R(3) = 5 GOTO 440
430 R(3) = 6
440 Q = 100*R(1) + 10*R(2) + R(3)
450 IF Q = 666 THEN P = 200:GOTO 550
460 IF Q = 555 THEN P =  20:GOTO 550
470 IF Q = 222 THEN P =  18 GOTO 550
480 IF Q = 556 THEN P =  15:GOTO 550
490 IF Q = 444 THEN P =  10:GOTO 550
500 IF Q = 332 THEN P =   8.GOTO 550
510 IF Q = 226 THEN P =   6.GOTO 550
520 IF INT (Q/10) = 11 THEN P = 5:GOTO 550
530 IF INT (Q/100) = 1 THEN P = 3:GOTO 550
540 P = 0
550 FOR X = 1 TO 3.FOR Y = 1 TO 6
560 IF R (X) = S(Y) GOTO 580
570 NEXT Y
580 R$(X) = S$(Y):NEXT X
590 W = W + P:IF P = 0 GOTO 610
600 K = K + 1
610 PRINT:PRINT "REELS." TAB(9);R$(1);TAB(12);R$(2);TAB(15);R$(3)
620 IF P = 0 GOTO 700
630 IF P = 200 GOTO 660
640 IF NOT (INP(0)<128) GOTO 640
650 OUT 1,7:PRINT "PAYOFF: $";P:GOTO 700
660 FOR X = 1 TO 75
670 IF NOT (INP(0)<128) GOTO 670
680 OUT 1,7:NEXT
690 PRINT "JACKPOT!!!!!! $";P
700 ON SGN(W-L) + 2 GOTO 710,720,730
710 PRINT "SO FAR YOU'VE LOST $";L-W:GOTO 740
720 PRINT "SO FAR YOU'RE EVEN":GOTO 740
730 PRINT "SO FAR YOU'VE WON $",W-L
740 PRINT:GOTO 230
750 PRINT:PRINT "TIMES PLAYED: ";L:PRINT "NUMBER OF PAYOFFS: ";K
760 PRINT "AMOUNT PAID: $";W
770 ON SGN (W-L) + 2 GOTO 780,790,810
780 PRINT "TOTAL LOST $";L-W:PRINT "WANNA TRY AGAIN, SUCKER?":GOTO 9999
790 PTINY "YOU BROKE EVEN. TOO BAD,":PRINT "THE NEXT ONE MIGHT HAVE"
800 PRINT "BEEN THE BIG ONE!":GOTO 9999
810 PRINT "TOTAL WON: $";W-L:PRINT "YOU BUY THE DRINKS!"
9999 END
OK
```

MAINTENANCE SOFTWARE

By Harvey Lee

Maintenance software serves two primary purposes. First, it aids in identifying problem areas that exist within a computer system. Second. by not finding any problems, it should give a high degree of confidence in the system. With these thoughts in mind, we use several different programs in the process of checking out repair and production computers.

Let us consider how one short program can be of use in this regard.

Sense Switch Read

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
STRT	IN S.SW	000,000	333	Input sense switch data (I/O channel 377)
		001	377	to the accumulator
	STA 100	002	062	Stores the contents of the accumulator
		003	100	in memory location 100 (octal)
		004	000	
	JMP STRT	005	303	jump to the start of the program
		006	000	
		007	000	

This program reads the sense switches (A8 through A15), then stores the information in memory location 100. Thus, when memory location 100 is displayed, the data lights should reflect the positions of the sense switches. For example, if switch A8 were up, then D0 would be on. Or if A15 was down, then D7 would be off.

What this program tells the technician depends on his knowledge of the computer. If the technician has trouble entering or checking the program in the computer's memory, he knows the proper portions of the D/C board, CPU board, and memory boards to check for the problem.

After loading the program, he should then single step through the program. By observing as it executes the program step by step, he can often identify a problem area on the CPU board or on the D/C board.

The last thing he does is to run the program, stopping to examine the memory for each of the sense switches. This assures proper sense switch reading, which is necessary when loading BASIC.

If the program fails to execute properly, the technician then begins his troubleshooting in the appropriate area.

If the program will run and single step, but yields the wrong data at memory location 100, he has several possible problem areas. First a visual inspection is made on the CPU board to verify R9 through R16 are 4.3K ohm resistors and 1C"Q" is a TI 74123. Next the timing relationship of Ø1 and Ø2 are checked. If this checks properly, he will then check the logic on the D/C board, starting at 1C"U" pin 8 (a logic low level that has a high going pulse when the sense switches are being read).

If the computer will operate correctly when either running or single stepping, but not both, the technician should begin his troubleshooting on the CPU board at 1C"R" pin 8 (normally high; goes low to enable DI data bus) and checks logic levels of 1C"O". If the CPU checks out, he will then proceed to the appropriate area on the D/C board to continue his troubleshooting. If improper operation results in writing alternating bit patterns through memory (usually 071 with 000) he begins his troubleshooting with 1C"O" on the CPU. These problems are generally caused by improper enabling of the DI data bus.

As can be seen, this program while short can be of great benefit to the technician. It aids him in identifying the area in which to start his troubleshooting. If he fails to find a problem, he feels confident in the proper operation of the majority of the computers circuits. He is now ready to use more complex programs in checking out the computer.

ALTAIR DISK TEST PROGRAMS

by Tom Durston

Listed below are some Altair Disk Test programs that will check out all the normal functions of the Disk Drive. These check-out procedures will also be included in the Altair Disk Theory of Operation manual.

A. Disk Read/Write Test Program

This program writes data on disk on sector Ø of the track it is positioned on, then reads the data back, stores it in memory, then outputs it to an I/O device. It is used for testing all read/write functions.

WRITE: The number of write data bytes is set by the position of the sense switches (maximum of 22Ø$_8$). Write data consists of:

1st byte = 377_8 (D7 = 1 - sync bit)
2nd byte = data on sense switch
3rd byte = 2nd - 1
4th byte = 2nd - 2
.
.
.
"n"th byte = 001
last byte = 000

If sense switch is set to ØØØ, program will stop.

READ: The read data is stored in memory, starting at address 001,236_8 and consists of the data written by the write program

OUTPUT: After the read program, the data is outputted to a terminal (Teletype, CRT, etc.). The output program is set to output on channel 1. To obtain a useful output pattern, change the sense switches until a desirable pattern is printed. The characters printed will consist of all printable ASCII characters in reversed order (as in 987654321 and zyxwvu . . .). This pattern repeats itself and is easily observed for errors.

B. Stepping Program

This program steps the disk head out 77 times to track Ø and then in 77 times to track 76, continuously repeating with the computer in the run mode.

This program is useful for testing the disk enable, MH status, track Ø status, and stepping functions of the disk.

While stepping with this program, the head is unloaded, so it may be run continuously without wear on the read/write head surface. A squeaking sound caused by the head load mechanism is normal in this test.

To loop with the read/write program, see next section.

For stepping program, disk drive address of ØØØ is used. To change disk drive tested, the address is contained in location (001,001).

Looping With Stepping Program

To check the read/write and step functions simultaneously, the two programs may be run together by changing:

1) Data in locations (000,154) and (000,155) to 037. 001 as indicated.

2) Data in location (001,034) to 303 as indicated.

-continued

Disk Test Programs

Start the program at (001,000), the start of the stepping program.

The disk head will step out to track Ø.

The head will then load and a write/read will occur. The head will then unload and output will take place. After output, the head will step in once, starting the write/read sequence again. After this repeats 76 times, the head is stepped out to track Ø, and it begins again. **

NOTE: ** 1) For read/write program, disk drive address of ØØØ is used. To change disk drive tested, the address is contained in location (000,001) and (000,150).

2) Output device addresses are in locations (000,133) (status) and (000,141) (data).

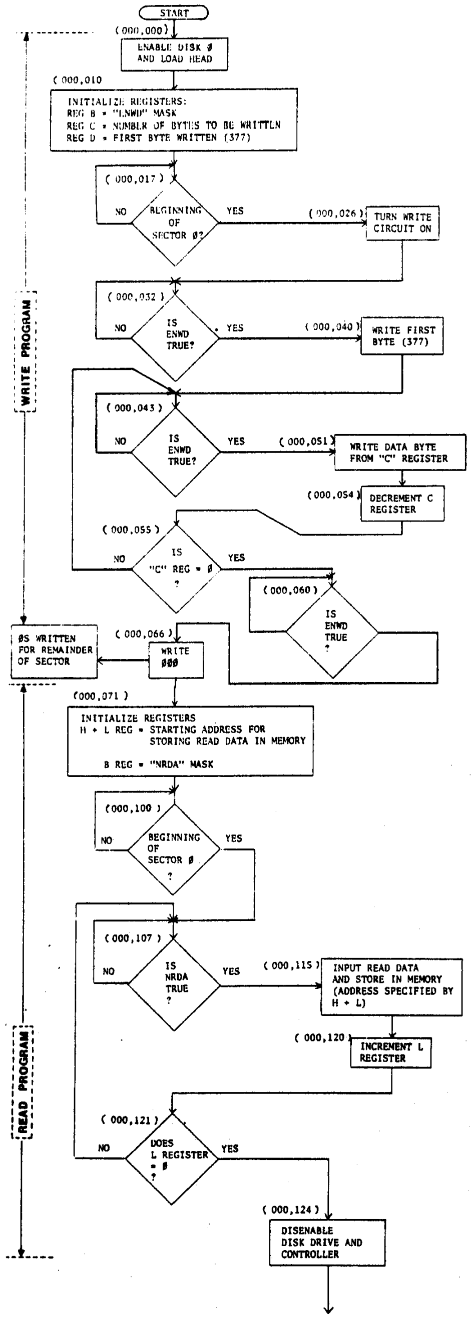
READ/WRITE PROGRAM

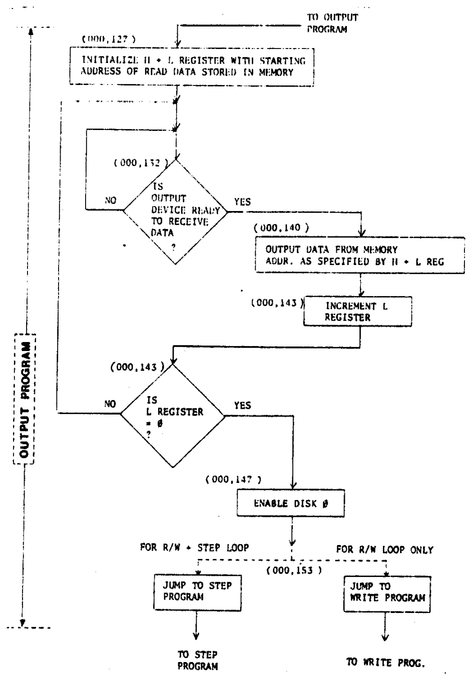
TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
	MVI(A)	000,000	076	
		1	000	Disk drive address
	OUT	2	323	
		3	010	Disk controller enable channel
LDHD	MVI(A)	4	076	
		5	004	Load head bit
	OUT	6	323	
		7	011	Disk function control channel
WRTLP	IN	10	333	Input # of bytes to be written
		11	377	Sense switch
	MOV(C)←(A)	12	117	Store in "C" reg.
	MVI(D)	13	026	Store in "D" reg.
		14	377	First write byte
	MVI(B)	15	006	Store in "B" reg.
		16	001	"ENWD" status mask
WSECT	IN	17	333	Write sector test
		20	011	Sector position channel
	CPI	21	376	
		22	300	Ø sector
	JNZ	23	302	Jump if not start of Ø sect.
		24	017	to "WSECT"
		25	000	
	MVI(A)	26	076	
		27	200	Write enable bit
		30	323	
		31	011	Disk function control channel
FBYT		32	333	First byte test
		33	010	Disk status channel
	ANA(A)/(B)	34	240	Test for "ENWD" status
	JNZ	35	302	Jump if "ENWD" false (=1)
		36	032	to "FBYT"
		37	000	
	MOV(A)(D)	40	172	Move 377 into accum.
	OUT	41	323	Output first byte
		42	012	Disk data channel
INDAT	IN	43	333	Start of write data sequence
		44	010	Disk status channel
	ANA	45	240	Test for "ENWD" status
	JNZ	46	302	Jump if "ENWD" false (=1)
		47	043	to "WDAT"
		50	000	
	MOV(A)←(C)	51	171	Move "DATA" byte to accum.
	OUT	52	323	
		53	012	Disk data channel
	DCR(C)	54	015	Decrement "DATA" byte
	JNZ	55	302	Jump if data byte = Ø.
		56	043	to "WDAT", write another byte
		57	000	
WZT	IN	60	333	Start of zero byte
		61	010	Output sequence
	ANA(A)←(B)	62	240	Test "ENWD" (last byte written)
	JNZ	63	302	Jump if "ENWD" false
		64	060	To WZT
		65	000	
	XRA(A)(A)	66	257	Zeros accumulator
	OUT	67	323	Output zero byte
		70	012	Disk data channel (end of write, start of read)
	LXI	71	041	Load H+L reg. with:
		72	236	Starting addr. to store read data
		73	001	
	MVI(B)	74	006	Store in "B" reg.
		75	200	"NRDA" mask
	NOP	76	000	
	NOP	77	000	
RSECT	IN	100	333	Read sector test
		101	011	Sector position channel
	CPI	102	376	
		103	300	Ø sector
	JNZ	104	302	Jump if not start of Ø sect.
		105	100	to "RSECT"
		106	000	
RDTST	IN	107	333	Start of "NRDA" test
		110	010	Disk status channel
	ANA(A)/(B)	111	240	Test for "NRDA" status
	JNZ	112	302	Jump if "NRDA" false (=1)
		113	107	to "RDTST"
		114	000	
	IN	115	333	Input read data
		116	012	Disk data channel
	MOV(M)←(A)	117	167	Store data in memory (H+L)
	INR(L)	120	054	Increment L reg. (mem addr)
	JNZ	121	302	Jump if L reg. ≠ 0
		122	107	to RDTST
		123	000	
	MOV(A)←(D)	124	172	Move 377 byte to accum.
	OUT	125	323	Disenable disk by output logic 1 on
		126	010	D7 to disk enable chan. (end of read start of output)
	LXI(H+L)	127	041	Load H+L with:
		130	236	Starting addr of data stored by read program
		131	001	
OTST	IN	132	333	Test output device for busy
		133	000	Status chan. of terminal
	RLC	134	007	Test bit Ø, rotate into carry
	JC	135	332	Jump if carry (bit Ø = 1)
		136	132	to "OTST"
		137	000	
	MOV(A)←(M)	140	176	Move data from mem(H+L)
	OUT	141	323	Output data
		142	001	Data channel for term
	INR(L)	143	054	Increment L register
	JNZ	144	302	Jump if L reg ≠ 0, output another byte
		145	132	to "OTST"
		146	000	
	MVI(A)	147	076	Enable disk
		150	000	
	OUT	151	323	
		152	010	
	JMP	153	303	
NOTE		*154	004	To "LDHD"
		*155	000	
		156		
		157		

*--For R/W-step loop change
Data at (000,154) to 037
Data at (000,155) to 001

-continued

Disk Test Programs





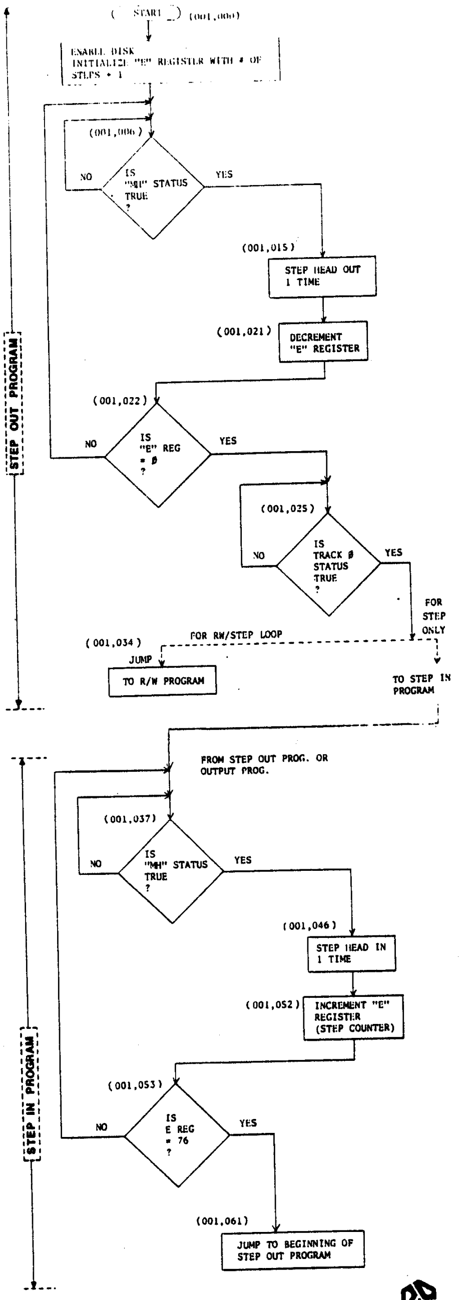
-continued

Disk Test Programs

STEPPING PROGRAM

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
STEP	MVI(A)	001,000	076	
		1	000	Disk drive addr ∅
	OUT	2	323	Output (data-∅∅∅) to
		3	010	Disk CTRL enable channel
NOS	MVI(E)	4	036	Initialize E register
		5	115	=77 (number of steps + 1)
SOUT	IN	6	333	Test "MI" status bit (move head)
		7	010	Disk status channel
	ANI	10	346	Test
		11	002	D1 mask
	JNZ	12	302	Jump if "MI" false (D1=1)
		13	006	To "SOUT"
		14	001	
	MVI(A)	15	076	
		16	002	Bit D1=1 (step out)
	OUT	17	323	Output (data 002) to
		20	011	Disk function control channel
	DCR(E)	21	035	Decrement step counter (E reg.)
	JNZ	22	302	Jump if E reg ≠ 0
		23	006	to "SOUT"
		24	001	
TZ	IN	25	333	Test for track ∅ status
		26	010	Disk status channel
	ANI	27	346	Test
		30	100	D6 mask
	JNZ	31	302	Jump if track ∅ false (D6=1)
		32	025	to "TZ"
		33	001	
LOOP	NOP	*34	000	
	NOP	35	000	
	NOP	36	000	
SIN	IN	37	333	Test "MH" status bit (move head)
		40	010	Disk status channel
	ANI	41	346	Test
		42	002	D1 mask
	JNZ	43	302	Jump if "MH" false (D1=1)
		44	037	to "SIN"
		45	001	
	MVI(A)	46	076	
		47	001	Bit D∅ = 1
	OUT	50	323	Output (Data 001) to
		51	011	Disk function control channel
	INR(E)	52	034	Add 1 to "E" register
	MVI(A)	53	076	
		54	114	76 steps
	CMP(A)/(E)	55	273	Compare "E" reg. to 76
	JNZ	56	302	Jump if "E" reg. ≠ 76
		57	034	To "Loop"
		60	001	to "Loop"
	JMP	61	303	Jump if "E" reg. = 76
		62	004	to "NOS"
		63	001	
		64		
		65		
		66		
		67		

*--Change to 303 for Step + R/W loop



Software Initialization of Parallel and Serial I/O Boards

By Patrick N. Godding

In an attempt to encompass as many different applications as possible, MITS has created two new peripheral interface boards, the 88-4PIO and the 88-2SIO. The boards are extremely versatile, which has led to some confusion in their software intialization requirements. This article will help to explain the software operation of the boards.

88-4PIO

NOTE: All address references are in octal.

There are two sections in each port, "A" and "B", and three registers in each section of each port: the Control Register, the Data Direction Register and the Data Register.

The Control Register is always accessed by an even address (address line AØ = Ø). Assume an 88-4PIO with only one port, addressed at starting location 020. In this case, the "A" Section Control Register is 020 and the "B" Section Control Register is 022. Execution of an input instruction (INP = 333) followed by an I/O address of 020 would transfer the contents of the "A" Section Control Register into the accumulator. Execution of an output instruction (OUT = 323) followed by an I/O address of 022 would transfer the contents of the accumulator into the "B" Section Control Register.

The Data Direction Register and the Data Register have the same address: using the above example, the "A" Section Data Direction Register and the Data Register are at address 021 and the "B" Section Data Direction Register and Data Register are at address 023. These addresses are always odd (AØ = 1). The Control Register determines which one of the other two Registers will be selected:

Control Register Bit 2	I/O Address	Register Selected
Zero (0)	0 2 1	"A" Section Data Direction
Zero (0)	0 2 3	"B" Section Data Direction
One (1)	0 2 1	"A" Section Data Register
One (1)	0 2 3	"B" Section Data Register

This brings us to the first step of the Port Initialization: write a zero into bit 2 of both Control Registers. In fact, since at this point the other bits of the Control Registers have no effect, simply write the Registers with all zeros:

```
076    Load Accumulator with zeros
000

323    Output zeros to "A" and "B"
020    Section Control Registers
323
022
```

When the above routine has been executed, an I/O instruction followed by the even address will select one of the Data Direction Registers (DDR). The DDR has only one purpose: to define each of the data lines as an input or an output. Normally this Register is accessed only during initialization. It is an 8-bit, write-only register with each bit defining a particular data line: Bit Ø defines data line Ø, etc. When a DDR bit is set to a one, the associated data line acts as an output. When the DDR bit is reset to a zero, the data line acts as an input.

Suppose we wish to interface a parallel keyboard and CRT display unit with the port. The routine given below sets up the "A" Section to act as an 8-bit input for the keyboard and the "B" Section to act as an 8-bit output for the CRT:

Note that this is a continuation of the above program and the accumulator is zeroed.

```
323    Output zeros to the "A" Section DDR
021    to make the data lines inputs

076    Load accumulator with ones
377

323    Output ones to "B" Section DDR to
023    make data lines outputs
```

The above sequence is only an example. Any combination of inputs and outputs is possible: e.g., the "A" Section could be set up for 4 inputs and 4 outputs and the "B" Section for 8 inputs, giving a total of 12 inputs and 4 outputs.

When one section is going to be used for input and the other for output, it might be desirable to use the "B" Section as output because not only is it TTL compatible (as the "A" Section is) but it also has higher sourcing current. This means that it can directly drive a transistor switch by sourcing 1 milliamp at a minimum of 1.5 volts. One application would be interfacing to a device using relays for its inputs. The "B" Section could directly drive transistors used to turn the relay coils on and off.

The last step in the initialization is to again write into the Control Registers. All the data bits of the Control Register are important for this final write. Remember that we're using address 020, so that the "A" Section Control Register is address 020 and the "B" Section Control Register is 022.

Before explaining the effect that the Control bits have on the operation of the port, let's briefly discuss terminal interface communications. Assume that we have a keyboard and a printer. We are going to interface the keyboard to the "A" Section of the port and the printer to the "B" Section. The keyboard produces 7 data bits with each key stroke and a strobe pulse signaling when a key is active. The strobe pulse becomes a "Handshake" signal to inform the CPU when there is valid data at the input section. The keyboard also has an input for a busy signal. Once a key is depressed and the strobe signal is sent, it is up to the port to maintain a busy signal back to the keyboard until the data has been received by the CPU. After the receipt of the valid data signal the CPU inputs the data and resets the busy signal indicating that it is ready to receive new data from the keyboard.

The printer operates in a similar manner. When the printer is ready to receive new data it indicates this by a ready-to-receive signal to the interface port. When the CPU receives this signal it sends out valid data along with a signal indicating to the printer that there is valid data. Once the data has been received by the printer it again sends out its ready-to-receive signal which then resets the CPU valid data signal. (See block diagram.)

The table below shows the bit function for the control registers.

Bit #	7	6	5	4	3	2	1	0
Function	Interrupt Request		C2 Control			DDR Control	C1 Control	

—continued

Software Initialization

The bits of both the "A" and "B" Section Control Register are defined in the tables below:

CONTROL BITS 1	CONTROL BITS 0	C1 INPUT	STATUS BIT 7	IRQ OUTPUT
0	1	Active low	Set high when C1 is active	Disabled -- remains high
0	1	Active low	Set high when C1 is active	Goes low when bit 7 goes high
1	0	Active high	Set high when C1 is active	Disabled -- remains high
1	1	Active high	Set high when C1 is active	Goes low when bit 7 goes high

CONTROL BITS 5	4	3	C2	STATUS BIT 6	$\overline{\text{IRQ}}$
0	0	0	Active low	Set high when C2 is active	Disabled -- remains high
0	0	1	Active low	Set high when C2 is active	Goes low when bit 7 is high
0	1	0	Active high	Set high when C2 is active	Disabled -- remains high
0	1	1	Active high	Set high when C2 is active	Goes low when bit 7 is high

B SECTION CONTROL BITS			CB2	
5	4	3	CLEARED	SET
1	0	0	Low when E pulse goes high, following a write of B data channel	High when CB1 is active
1	0	1	Low When E pulse goes high, following a write of B data channel	High when next E pulse goes high
1	1	0	Always low when bit 3 is low	
1	1	1		Always high when bit 3 is high

A SECTION CONTROL BITS			CA2	
5	4	3	CLEARED	SET
1	0	0	Low after E pulse, following read of A data channel	High when CA1 is active
1	0	1	Low after a read of A data channel	High following next E pulse
1	1	0	Always low when bit 3 is low	
1	1	1		Always high when bit 3 is high

As an example of the use of these tables, consider the keyboard/printer discussed above. The complete initialization of the port for the "A" and "B" Sections is given below, with particular emphasis on the control word of each section.

First we must designate the "A" Section as the Input Mode and the "B" Section as the Output Mode.

```
076   Load accumulator with zeros
000

323   Output
020   "A" Section Control

323   Output
022   "B" Section Control

323   Output
021   "A" Section DDR

076   Load accumulator with ones
377

323   Output
023   "B" Section DDR
```

Access Data Direction Register and set up all "A" Section data lines as inputs and "B" Section data lines as outputs,

"A" SECTION CONTROL WORD

```
076   Bits 0 and 1 equal 0 and 1 respectively -
046   making CA1 low (active) and disabling inter-
323   rupts. Bit 2 equals a 1 - enable the Data
020   Register. Bits 3, 4, and 5 equal 001 re-
      spectively - defining CA2 as an output to
      act as a busy signal to the keyboard.
```

CA2 will go low when the keyboard strobe signal forces the C1 input active. CA2 will go back high after the CPU reads the "A" Section Data Channel. In this application the CA2 signal indicates the CPU is busy when it is low. When it is high the keyboard is free to send new data.

"B" SECTION CONTROL WORD

```
076   Again bits 0 and 1 equal 0 and 1 and operate
045   as in the above "A" Section.
323   Bit 2 equals 1 - enable the "B" Data Register.
022   Bits 3, 4, and 5 define CB2 as an output busy
      signal to the printer. The printer pulls CB1
      low when it is ready for new data. This
      action causes CB2 to go high, indicating that
      the CPU is busy. When the CPU writes a data
      word out to the printer, CB2 goes low to tell
      the printer that valid data is available.
```

CPU OPERATION

Bits 6 and 7 of the Control Register operate as status bits in the above example. Since CA2 and CB2 are used as outputs, status bit 6 is not used. The following routine is an echo program which inputs a character from the keyboard and then outputs the character to the printer. The CPU monitors status bit 7 for indication that data is available from an input section or the output section is ready to receive data.

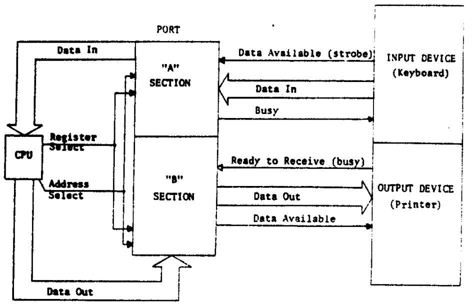
Assume that the port has been initialized per the above routine.

```
000   333   Input
001   020   "A" Section Control Register
002   346   AND Immediate
003   200   Bit 7
004   312   Jump if result zero
005   000   to location 0
006   000
007   333   Input
010   021   "A" Section Data Register
011   062   Store Accumulator
012   100   at location 100
013   000
014   333   Input
015   022   "B" Section Control Register
016   346   AND Immediate
017   200   Bit 7
020   312   Jump if zero
021   014   to location 014
022   000
023   072   Load Accumulator
024   100   from location 100
025   000
026   323   Output
027   023   "B" Section Data Register
030   333   Input
031   023   "B" Section Data Register
032   303   Jump unconditional
033   000   to location 0
034   000
```

-continued

Software Initialization

Execution of this program causes the following: "A" Section Control Register is input to the accumulator and tested to see if bit 7 has gone high. If high, "A" Section Data Channel is input to the accumulator and then stored at location 100. Then the "B" Section Control Register is tested to see if bit 7 is high. If it is, it indicates that the printer is not busy and the stored character is then loaded from location 100 into the accumulator and output to the "B" Section Data Channel. Then the "B" Section Data Register must be input to clear the status flag, bit 7. The program then jumps to the beginning to wait for a new character.



Port/Device Communications

This has been a description of one port. The 88-4PIO has provisions for a total of four ports, each operating identically to the above port. The only addition for the extra ports is the I/O address used for Register selection. The complete addressing for 4 ports on the above board (starting address 020) is as follows:

PORT #	SECTION	ADDRESS	REGISTER
0	A	0 2 0	Control
		0 2 1	Data
	B	0 2 2	Control
		0 2 3	Data
1	A	0 2 4	Control
		0 2 5	Data
	B	0 2 6	Control
		0 2 7	Data
2	A	0 3 0	Control
		0 3 1	Data
	B	0 3 2	Control
		0 3 3	Data
3	A	0 3 4	Control
		0 3 5	Data
	B	0 3 6	Control
		0 3 7	Data

88-2SIO

Each port on the 88-2SIO contains two Registers: a Control/Status Register and a Data In/Out Register. As with the 88-4PIO, the ports must be initialized. The access scheme is the same as the 4PIO. The Control/Status Register is the even address (A∅ = ∅) and the Data I/O Register is the odd address (A∅ = 1).

Assume the board is addressed at starting location 010. Then the Register addresses become:

010	Control/Status Register	Port ∅
011	Data I/O Register	Port ∅
012	Control/Status Register	Port 1
013	Data I/O Register	Port 1

The first step in initialization is to reset the port. This is accomplished by writing a 1 into the first two bits of the Control Register:

```
076
003   Write 1's into bit ∅ and bit 1 of
323   the Control Register
010
```

BIT	7	6	5	4	3	2	1	0
FUNCTION	REC INT	X-MIT	INT	WORD SELECT			RESET/DIVIDE	

The table above shows the bit functions of the control register.

The next step is to define the clock divide ratio. The on-board clock generator produces a frequency that is 16 times the actual baud rate (the rate marked 110 on the board is actually 16 times 110, or 1760 Hertz). If the baud rate needed for your particular terminal is one of the baud rates marked on the board, use the ÷16 mode. If the baud rate needed is one of the other five available (27.5, 37.5, 75, 450 or 600), use the ÷64 mode. Page 5 of the 88-2SIO Theory Manual shows which rate to select on the board for the above five rates. Note that both the baud rate and the word select are a function of the I/O device being used. The port is programmed to be compatible with the device, not vice-versa.

The chart below shows the function of bits ∅ and 1.

BIT 1	BIT 0	FUNCTION
0	0	÷ Clock by 1
0	1	÷ Clock by 16
1	0	÷ Clock by 64
1	1	Master Reset

Bits 2, 3, and 4 define the form of the serial word transmitted and received. Again, this is a function of the device. An example is a Teletype that has a reader and punch (ASR). The port should be programmed for 8 data bits, 2 stop bits and no parity. The chart below shows the serial word combinations available:

DATA BIT			FUNCTION		
4	3	2	Number of Data Bits	Number of Stop Bits	Parity
0	0	0	7	2	Even
0	0	1	7	2	Odd
0	1	0	7	1	Even
0	1	1	7	1	Odd
1	0	0	8	2	None
1	0	1	8	1	None
1	1	0	8	1	Even
1	1	1	8	1	Odd

continued

Software Initialization

Bits 5 and 6 control the transmit interrupt and the Request-To-Send ($\overline{\text{RTS}}$) output signal. The $\overline{\text{RTS}}$ signal is used to turn the reader on and off, under software control, on the new TYA Kit. With the 2SIO, a "Handshake" type communication previously described in the 4PIO operation is usually not needed. When the port receives a serial data word and converts it to parallel for the CPU, a status flag is automatically set to indicate valid data and if interrupts are enabled, it also generates an interrupt. The transmit portion works in the same way. Normally, the "Handshake" signals on the 2SIO would only be used in a modem connection. This is also true with terminals. There is generally no need for handshake signals in a single-user, stand-alone system. This is why the 88-2SIO Manual indicates that if $\overline{\text{CTS}}$ and $\overline{\text{DCD}}$ are not needed, which is usually the case, connect them directly to ground (at the D and E pads) and to nothing else.

Bit 7 controls the receive interrupt.

DATA BIT			FUNCTION
7	6	5	
X	0	0	$\overline{\text{RTS}}$ = low, transmitting interrupt disabled.
X	0	1	$\overline{\text{RTS}}$ = low, transmitting interrupt enabled.
X	1	0	$\overline{\text{RTS}}$ = high, transmitting interrupt disabled.
X	1	1	$\overline{\text{RTS}}$ = high, transmits a break level on the transmit data output. Transmit interrupt disabled.
0	X	X	Receive interrupt disabled.
1	X	X	Receive interrupt enabled.

X = does not matter.

The example below shows the complete control word for a CRT terminal that operates at 9600 baud, uses 7 data bits, 1 stop bit and even parity. This example disables both receive and transmit interrupts:

Control Bit	7	6	5	4	3	2	1	0
State	0	1	0	0	1	0	0	1

Thus, to initialize the port:

```
076   Load
111   Control Word

323   Out
010   Control Register
```

Remember that the previous output for resetting the port has been executed prior to this initialization.

The CPU looks at status information exactly the same as with the 4PIO, but the pertinent bits are in a different location within the status register.

STATUS REGISTER BIT #	FUNCTION
0	Receive Data Register full - when high, indicates that valid data is ready to be input into the accumulator.
1	Transmit Data Register empty - when high, indicates that the CPU may output data.

We can now write an echo program for port Ø of a 2SIO at starting address 010:

```
000    333    Input
001    010    Status Register

002    346    AND Accumulator
003    001    with bit Ø

004    312    Jump if no data
005    000    to Ø
006    000

007    333    Input
010    011    Data Register

011    062    Store at
012    100    100
013    000

014    333    Input
015    010    Status Register

016    346    AND with
017    002    bit 1

020    312    Jump if buffer
021    014    full to 14
022    000

023    072    Load from
024    100    100
025    000

026    323    Out to
027    011    Data Register

030    303    Jump to get
031    000    new data
032    000
```

Hopefully, this information has cleared up some misconceptions about the I/O boards. The software is a little more complex, but this allows hardware compatibility with many different devices, and also allows interfacing more than one device per card.

SPECIAL NOTE: With the new I/O boards, a program cannot interrupt from a Halt state. A mod to change this exists, but is fairly complex. If your application does not lend itself to changing the Halt instruction to a Jump-self instruction, please contact me and I'll be happy to send you the mod.

Patrick N. Godding
Program Manager

correction

"Software Initialization of Parallel and Serial I/O Boards", Computer Notes, June 1976

Note at the end of the I/O article it was stated that an interrupt could not occur in the HALT state if the new I/O boards are used. (This is true only on the 88-4PIO. The 88-2SIO works normally in the HALT condition.)

STAR TREK LIVES!

by Lynn Cochran Reprinted from *Interface* June, 1976

Lynn Cochran, a graduate of Cal Poly, Pomona, has been active in amateur radio including the design of digital controls for repeater systems, and has recently become involved in the hobby computer field. He is presently employed in San Francisco as a communications engineer for a large oil corporation..

The computer game of STARTREK (based on NBC's popular TV show) has appeared in many versions since the late sixties. Among the most popular versions are those written by Mike Mayfield in the HP contributed program library, and by David Ahl in the book 101 BASIC COMPUTER GAMES.

These instructions are for a version of STARTREK optimized for use with MITS 8-K BASIC (a listing of the program is included). I rewrote it to fit in a minimum of memory with the result that it just fits in my 12-K Altair 8800 system with the 8-K Basic interpreter (version 3.1). In rewriting it to get it to fit within 12-K, I've combined the "Warp Engines" and "Photon Torpedoes" routines, left out printed-out instructions, and eliminated the "Status Report" command (each device now tells you how many years are required for its repair when you try to use it). It also runs a little slower than on the big machines; it takes about 4 seconds to set up a new Quadrant, and a whole 10 seconds to initialize the Galaxy, but the feeling you get from saving the known universe makes it worth it. Running this program is also a good way to test memory; it finds problems no memory-checker program will (it uses just about everything). It would be a good bet to put your best memory in the bottom 8-K to minimize the fatality of any 'memory lapses.' The program does not use ATN, so Basic can be initialized without it if memory space is needed.

I have included a 'Block Diagram' of the STARTREK program which provides, on one page, an idea of how the program works. The diagram shouldn't be taken too literally; not shown are the damaged device routines, or the fact that the "Warp Engines" and "Torpedo" commands are processed by a single routine.(Block Diagram--page 18)

Also included is a list of the variables used in the program; they may come in handy if you try to decipher the strange techniques I've used to crunch STARTREK into 12-K of memory.

Because I feel that other people may want access to copies of this program, either to modify for their own use, or to use as is, I want to see this program freely distributed.

Lynn Cochran

INSTRUCTIONS FOR STARTREK

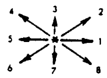
It is Stardate 3421 and the Federation is being invaded by a band of Klingon "pirates" whose objective is to test our defenses. If even one survives the trial period, Klingon Headquarters will launch an all out attack. As Captain of the Federation Starship 'ENTERPRISE', your mission is to find and destroy the invaders before the time runs out.

The known galaxy is divided into 64 quadrants arranged like a square checkerboard, 8 on a side. Each quadrant is likewise divided into 64 sectors arranged as an 8 by 8 square. Each sector can contain a Klingon (K), star (*), Starbase (B), the Enterprise herself (E), or empty space (.). Each sector is also numbered; a Starbase in sector 3-5 is 3 rows down from the top of the Short Range Scan printout, and 5 sectors to the right. Docking at a Starbase is done by occupying an adjacent sector, and reprovisions your Starship with energy and photon torpedoes, as well as repairing all damages.

Your Starship will act on the following commands:

Command 1

Warp Engine Control is used to move the Enterprise. You will be asked to set the distance (measured in Warps), and the course for the move. Each move that you make with the Enterprise from one sector to another, or from one quadrant to another, costs you one stardate (one year). Therefore, a 30 year game means you have 30 moves to win it in.



Course--A number from 1 to 8.999 indicating a direction (starting with a 1 to the right and increasing counterclockwise). To move to the left, use a course of 5. (A course of 3.4 is halfway between 3 and 4; a course of 8.75 is three-quarters of the way from 8 to 1.

Warp--One Warp moves you the width of a quadrant. A warp of .5 will move you halfway through a quadrant; moving diagonally across a quadrant to the next will require 1.414 Warps. Warp 3 will move you 3 quadrants providing nothing in your present quadrant blocks your exit. Once you leave the quadrant that you were in, you will enter Hyperspace; coming out of Hyperspace will place you randomly in the new quandrant. Klingons in a given quadrant will fire at you whenever you leave, enter, or move within that quadrant. Entering a course or warp of zero can be used to return to the command mode.

Command 2

A Short Range Sensor will print out the quadrant you presently occupy showing the content of each of the 64 sectors, as well as other pertinent information.

Command 3

The Long Range Sensor Scan summarizes the quadrant you are in, and the adjoining ones. Each quadrant is represented as a 3-digit number; the first (hundreds) digit is the number of Klingons in that quadrant while the middle digit is the number of Starbases, and the units digit is the number of stars. An entry of 305 means 3 Klingons, no Starbases, and 5 stars.

Command 4

Fire Phasers; the portion of the Enterprise's energy that you specify will be divided evenly among the Klingons in the quadrant and fired at them. Surviving Klingons will retaliate. Phaser fire bypasses stars and Starbases, but is attenuated by the distance it travels. The arriving energy depletes the shield power of its target. Energy is automatically diverted to the shields as needed, but if you run out of energy, you'll get fried.

Command 5

Photon Torpedo Control will launch a torpedo on a course you specify which will destroy any object in its path. Range is limited to the local quadrant. Expect return fire from surviving Klingons.

Command 6

The Galactic Records Section of the ship's computer responds to this command by printing out a galactic map showing the results of all previous sensor scans.

VARIABLES USED WITHIN THE STARTREK PROGRAM

A = Command:	A=0	Just arrived in quadrant
	A=1	Warp Engine Control
	A=2	Short Range Sensor Scan
	A=3	Long Range Sensor Scan
	A=4	Phaser Control
	A=5	Photon Torpedo Control
	A=6	Print Galactic Map

-continued

E = Present energy

E0 = Initial energy

P = Number of torpedoes left

P0 = Initial number of torpedoes

T = Present year

T0 = Starting year

T9 = Ending year

K = Number of Klingons in present quadrant

K9 = Total number of Klingons in the galaxy

B = Number of Starbases in present quadrant (one or zero)

B9 = Total number of Starbases in the galaxy

S = Number of stars in present quadrant

S9 = Initial energy of each Klingon

C = Course entered by player

W = Warp entered by player

H = Hit on Enterprise or Klingon in units of energy

I,J = Matrix variables in FOR-NEXT statements

N,X,Y,X1,Y1,X2,Y2 = Multiple-use variables

S1,S2 = Enterprise sector coordinates (down, right)

Q1,Q2 = Enterprise quadrant coordinates (down, right)

Q$ = Characters used in printing a Short Range Scan (EKB*)

C$ = Condition; Green, Yellow, Red, or Docked

E$ = Multiple-use string variable

D(I) = Number of years for repair of each device where the value of I determines the device.

D$(I) = Descriptive character string for each device where the value of I determines the device. See statements 30 through 70.

The value of I for each device is:

I=0 Warp Engines
I=1 Short Range Sensors
I=2 Long Range Sensors
I=3 Phasers
I=4 Photon Torpedoes
I=5 Galactic Records

K1(I) = Sector coordinate (down) for Ith Klingon in quadrant

K2(I) = Sector coordinate (right) for Ith Klingon in quadrant

K3(I) = Units of energy left in Ith Klingon in quadrant

S(I,J) = Sector Matrix. The values of both I and J are permitted to range from 0 to 7. Together, a given set of values for I and J selects one of the 64 stored numbers that the dimensioned variable S (I,J) has. S(I,J) is used to store the contents of the 64 sectors of the quadrant that the Enterprise is presently in. When the sector is printed out using the Short Range Scan, the value of I associated with each sector determines how far down from the top that sector is printed, while the value of J determines how far to the right it is printed. The contents of each element in the matrix S(I,J) determines what is printed in the associated sector:

1=Empty space (.)
2=Enterprise (E)
3=Klingon (K)
4=Starbase (B)
5-Star (*)

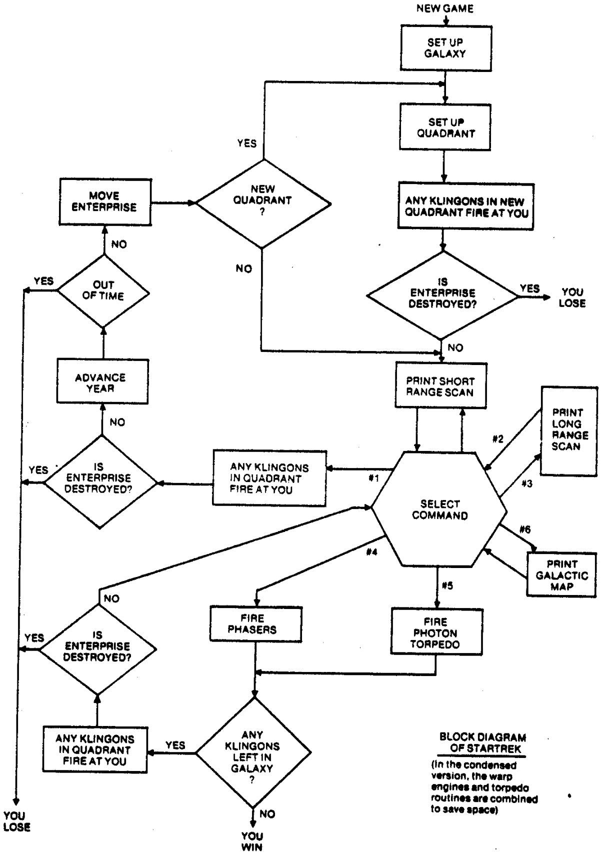
Q(I,J) = Quadrant Matrix. Like the sector matrix, this matrix is also defined to have 64 elements; each one stores a number which indicates how many Klingons, Starbases, and stars are in the associated quadrant. No attempt is made to store actual positions of Klingons, Starbases, or stars in any quadrant but the one that the Enterprise is currently in. For the remaining quadrants, only the number of Klingons, Starbases, and stars are stored (in the quadrant matrix).

When the enterprise moves to a new quadrant, the number of Klingons, Starbases, and stars for that quadrant are looked up in the quadrant matrix, and then that many Klingons, Starbases, and stars are randomly positioned along with the Enterprise in the sector matrix when the new quadrant is set up (statements 230 to 300). The numbers stored in the quadrant matrix are what are printed out when Long Range Scans and Galactic maps are printed. When the galaxy is first set up, each element in the quadrant matrix has a negative value. As the Enterprise roams the galaxy scanning quadrants, the values for those quadrants are made positive. The Galactic Map routine (statements 1300 to 1350) only prints positive valued quadrants; i.e. only those previously scanned by the Enterprise. Typical values stored for a quadrant are:

-305 means unexplored, 3 Klingons, no Starbases, and 5 stars

16 means mapped by sensors, no Klingons, 1 Starbase, and 6 stars

-continued



BLOCK DIAGRAM OF STARTREK

(In the condensed version, the warp engines and torpedo routines are combined to save space)

-continued

Star Trek Lives

```
NULL 0
CLEAR 50
NEW


10 REM  ** STARTREK **   (3/14/76)
10 REM  A GAME OF INTRAGALACTIC WARFARE BASED ON NBC'S POPULAR TV SHOW
10 REM   ADAPTED FOR ALTAIR 8K BASIC (VERSION 3.1) BY L E COCHRAN
10 REM   AND REWRITTEN TO FIT (WITH 8K BASIC) WITHIN 12K OF MEMORY
10 REM    (EXPECT A 4 SEC PAUSE TO SET UP EACH QUADRANT, AND
10 REM      10 SEC AFTER "WORKING")


10 DIM D(5),K1(7),K2(7),K3(7),S(7,7),Q(7,7),D$(5)
20 Q$=". EKB*"
30 D$(0)="WARP ENGINES"
40 D$(1)="SHORT RANGE SENSORS"
50 D$(2)="LONG RANGE SENSORS"
60 D$(3)="PHASERS"
70 D$(4)="PHOTON TORPEDOES":D$(5)="GALACTIC RECORDS"
80 INPUT"PLEASE ENTER A RANDOM NUMBER";E$:I=ASC(E$)
90 I=I-11*INT(I/11):FOR J=0 TO I:K=RND(1):NEXT:PRINT"WORKING-"
100 DEF FND(N)=SQR((K1(I)-S1)^2+(K2(I)-S2)^2)
110 GOSUB 610:GOSUB 450:Q1=X:Q2=Y:X=8:Y=1:X1=.2075:Y1=6.28:X2=3.28
120 Y2=1.8:A=.96:C=100:W=10:K9=0:B9=0:S9=400:T9=3451:GOTO 140
130 K=K+(N<X2)+(N<Y2)+(N<.28)+(N<.08)+(N<.03)+(N<.01):K9=K9-K:GOTO 160
140 T0=3421:T=T0:E0=4000:E=E0:P0=10:P=P0:FOR I=0 TO 7
150 FOR J=0 TO 7:K=0:N=RND(Y):IF N<X1 THEN N=N*64:K=(N<Y1)-Y:GOTO 130
160 B=(RND(Y)>A):B9=B9-B:Q(I,J)=K*C+B*W-INT(RND(Y)*X+Y):NEXT J,I
170 IF K9>(T9-T0) THEN T9=T0+K9
180 IF B9>0 THEN 200
190 GOSUB 450:Q(X,Y)=Q(X,Y)-10:B9=1
200 PRINT LEFT$("STARTREK ADAPTED BY L.E.COCHRAN 2/29/76",8):K0=K9
210 PRINT"OBJECTIVE:  DESTROY";K9;"KLINGON BATTLE CRUISERS IN";T9-T0;
220 PRINT"YEARS.":PRINT" THE NUMBER OF STARBASES IS";B9
230 A=0:IF Q1<0 OR Q1>7 OR Q2<0 OR Q2>7 THEN N=0:S=0:K=0:GOTO 250
240 N=ABS(Q(Q1,Q2)):Q(Q1,Q2)=N:S=N-INT(N/10)*10:K=INT(N/100)
250 B=INT(N/10-K*10):GOSUB 450:S1=X:S2=Y
260 FOR I=0 TO 7:FOR J=0 TO 7:S(I,J)=1:NEXT J,I:S(S1,S2)=2
270 FOR I=0 TO 7:K3(I)=0:X=8:IF I<K THEN GOSUB 460:S(X,Y)=3:K3(I)=S9
280 K1(I)=X:K2(I)=Y:NEXT:I=S
290 IF B>0 THEN GOSUB 460:S(X,Y)=4
300 IF I>0 THEN GOSUB 460:S(X,Y)=5:I=I-1:GOTO 300
310 GOSUB 550:IF A=0 THEN GOSUB 480
320 IF E<=0 THEN 1370
330 I=1:IF D(I)>0 THEN 620
340 FOR I=0 TO 7:FOR J=0 TO 7:PRINT MID$(Q$,S(I,J),1);" ";:NEXT J
350 PRINT"  ";:ON I GOTO 380,390,400,410,420,430,440
360 PRINT"YEARS =";T9-T
370 NEXT:GOTO 650
380 PRINT"STARDATE=";T:GOTO 370
390 PRINT"CONDITION:  ";C$:GOTO 370
400 PRINT"QUADRANT=";Q1+1;"-";Q2+1:GOTO 370
410 PRINT"SECTOR =";S1+1;"-";S2+1:GOTO 370
420 PRINT"ENERGY=";E:GOTO 370
430 PRINT D$(4);"=";P:GOTO 370
440 PRINT"KLINGONS LEFT=";K9:GOTO 370
450 X=INT(RND(1)*8):Y=INT(RND(1)*8):RETURN
460 GOSUB 450:IF S(X,Y)>1 THEN 460
470 RETURN
480 IF K<1 THEN RETURN
490 IF C$="DOCKED" THEN PRINT"STARBASE PROTECTS ENTERPRISE":RETURN
500 FOR I=0 TO 7:IF K3(I)<=0 THEN NEXT:RETURN
510 H=K3(I)*.4*RND(1):K3(I)=K3(I)-H:H=H/(FND(0)^.4):E=E-H
520 E$="ENTERPRISE FROM":N=E:GOSUB 530:NEXT:RETURN
530 PRINT H;"UNIT HIT ON ";E$;" SECTOR";K1(I)+1;"-";K2(I)+1;
540 PRINT" (";N;"LEFT)":RETURN
550 FOR I=S1-1 TO S1+1:FOR J=S2-1 TO S2+1
560 IF I<0 OR I>7 OR J<0 OR J>7 THEN 580
570 IF S(I,J)=4 THEN C$="DOCKED":E=E0:P=P0:GOSUB 610:RETURN
580 NEXT J,I:IF K>0 THEN C$="RED":RETURN
590 IF E<E0*.1 THEN C$="YELLOW":RETURN
600 C$="GREEN":RETURN
610 FOR N=0 TO 5:D(N)=0:NEXT:RETURN
620 PRINT D$(I);" DAMAGED.";
630 PRINT" ";D(I);"YEARS ESTIMATED FOR REPAIR.":PRINT
640 IF A=1 THEN RETURN
650 INPUT"COMMAND";A
660 IF A<1 OR A>6 THEN 680
670 ON A GOTO 710,310,1250,1140,690,1300
680 FOR I=0 TO 5:PRINT I+1;"= ";D$(I):NEXT:GOTO 650
690 IF D(4)>0 THEN PRINT"SPACE CRUD BLOCKING TUBES.";:I=4:GOTO 630
700 N=15:IF P<1 THEN PRINT"NO TORPEDOES LEFT":GOTO 650
```

-continued

```
710 IF A=5 THEN PRINT"TORPEDO ";
720 INPUT"COURSE (1-8. 9)"; C: IF C<1 THEN 650
730 IF C>=9 THEN 710
740 IF A=5 THEN P=P-1: PRINT"TRACK. "; : GOTO 900
750 INPUT"WARP (0-12)"; W: IF W<=0 OR W>12 THEN 710
760 IF W<=. 2 OR D(0)<=0 THEN 780
770 I=0: PRINT D$(I); " DAMAGED, MAX IS  2 "; : GOSUB 630: GOTO 750
780 GOSUB 480: IF E<=0 THEN 1370
790 IF RND(1)>. 25 THEN 870
800 X=INT(RND(1)*6): IF RND(1)>. 5 THEN 830
810 D(X)=D(X)+INT(6-RND(1)*5): PRINT"**SPACE STORM, ";
820 PRINT D$(X); " DAMAGED**": I=X: GOSUB 630: D(X)=D(X)+1: GOTO 870
830 FOR I=X TO 5: IF D(I)>0 THEN 860
840 NEXT
850 FOR I=0 TO X: IF D(I)<=0 THEN NEXT GOTO 870
860 D(I)=. 5: PRINT"**SPOCK USED A NEW REPAIR TECHNIQUE**"
870 FOR I=0 TO 5: IF D(I)=0 THEN 890
880 D(I)=D(I)-1: IF D(I)<=0 THEN D(I)=0: PRINT D$(I); " ARE FIXED!"
890 NEXT: N=INT(W*8): E=E-N-N+. 5: T=T+1: S(S1, S2)=1
900 Y1=S1+. 5: X1=S2+. 5: IF T>T9 THEN 1370
910 Y=(C-1)*. 785398: X=COS(Y): Y=-SIN(Y)
920 FOR I=1 TO N: Y1=Y1+Y: X1=X1+X: Y2=INT(Y1): X2=INT(X1)
930 IF X2<0 OR X2>7 OR Y2<0 OR Y2>7 THEN 1110
940 IF A=5 THEN PRINT Y2+1; "-"; X2+1,
950 IF S(Y2, X2)=1 THEN NEXT: GOTO 1060
960 PRINT: IF A=1 THEN PRINT"BLOCKED BY ";
970 ON S(Y2, X2)-3 GOTO 1040, 1020
980 PRINT"KLINGON"; : IF A=1 THEN 1050
990 FOR I=0 TO 7: IF Y2<>K1(I) THEN 1010
1000 IF X2=K2(I) THEN K3(I)=0
1010 NEXT: K=K-1: K9=K9-1: GOTO 1070
1020 PRINT"STAR"; : IF A=5 THEN S=S-1: GOTO 1070
1030 GOTO 1050: 2L29E76C
1040 PRINT"STARBASE"; : IF A=5 THEN B=2: GOTO 1070
1050 PRINT" AT SECTOR"; Y2+1; "-"; X2+1: Y2=INT(Y1-Y): X2=INT(X1-X)
1060 S1=Y2: S2=X2: S(S1, S2)=2: A=2: GOTO 310
1070 PRINT" DESTROYED!"; : IF B=2 THEN B=0: PRINT".   .  .GOOD WORK!";
1080 PRINT: S(Y2, X2)=1: Q(Q1, Q2)=K*100+B*10+S: IF K9<1 THEN 1400
1090 GOSUB 480: IF E<=0 THEN 1370
1100 GOSUB 550: GOTO 650
1110 IF A=5 THEN PRINT"MISSED!": GOTO 1090
1120 Q1=INT(Q1+W*Y+(S1+. 5)/8): Q2=INT(Q2+W*X+(S2+. 5)/8)
1130 Q1=Q1-(Q1<0)+(Q1>7): Q2=Q2-(Q2<0)+(Q2>7): GOTO 230
1140 I=3: IF D(I)>0 THEN 620
1150 INPUT"PHASERS READY:  ENERGY UNITS TO FIRE"; X: IF X<=0 THEN 650
1160 IF X>E THEN PRINT"ONLY GOT"; E: GOTO 1150
1170 E=E-X: Y=K: FOR I=0 TO 7: IF K3(I)<=0 THEN 1230
1180 H=X/(Y*(FND(0)^. 4)): K3(I)=K3(I)-H
1190 E$="KLINGON AT": N=K3(I): GOSUB 530
1200 IF K3(I)>0 THEN 1230
1210 PRINT"**KLINGON DESTROYED**"
1220 K=K-1: K9=K9-1: S(K1(I), K2(I))=1: Q(Q1, Q2)=Q(Q1, Q2)-100
1230 NEXT: IF K9<1 THEN 1400
1240 GOTO 1090
1250 I=2: IF D(I)>0 THEN 620
1260 PRINT D$(I); " FOR QUADRANT"; Q1+1; "-"; Q2+1
1270 FOR I=Q1-1 TO Q1+1: FOR J=Q2-1 TO Q2+1: PRINT"    ";
1280 IF I<0 OR I>7 OR J<0 OR J>7 THEN PRINT"***"; : GOTO 1350
1290 Q(I, J)=ABS(Q(I, J)): GOTO 1340
1300 I=5: IF D(I)>0 THEN 620
1310 PRINT"CUMULATIVE GALACTIC MAP FOR STARDATE"; T
1320 FOR I=0 TO 7: FOR J=0 TO 7: PRINT"  ";
1330 IF Q(I, J)<0 THEN PRINT"***"; : GOTO 1350
1340 E$=STR$(Q(I, J)): E$="00"+MID$(E$, 2): PRINT RIGHT$(E$, 3);
1350 NEXT J: PRINT: NEXT I: GOTO 650
1360 PRINT: PRINT"IT IS STARDATE"; T: RETURN
1370 GOSUB 1360: PRINT"THANKS TO YOUR BUNGLING, THE FEDERATION WILL BE"
1380 PRINT"CONQUERED BY THE REMAINING"; K9; "KLINGON CRUISERS!"
1390 PRINT"YOU ARE DEMOTED TO CABIN BOY!": GOTO 1430
1400 GOSUB 1360: PRINT"THE FEDERATION HAS BEEN SAVED!"
1410 PRINT"YOU ARE PROMOTED TO ADMIRAL": PRINT K0; "KLINGONS IN";
1420 PRINT T-T0; "YEARS.   RATING="; INT(K0/(T-T0)*1000)
1430 INPUT"TRY AGAIN"; E$: IF LEFT$(E$, 1)="Y" THEN 110
```

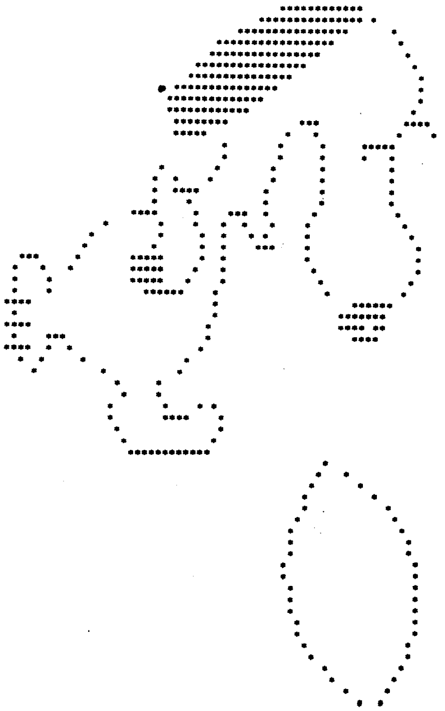
GOOD GRIEF!

This program was adapted from a program appearing in the H-P BASIC Program Library Handbook, June 1972. It will run without modification on both 680 and 8800 8K ALTAIR BASIC.

```
LIST

9000 REM ***** SNOOPY ***** DEMONSTRATION PROGRAM *****
9001 REM *****VERSION 1 *****7/31/69 *****
9002 REM  PRINTS APICTURE OF SNOOPY ON THE TTY
9003 DIM L(80)
9004 FOR T =1 TO 63
9005 LET L(T)=0
9006 NEXT T
9007 LET M=0
9008 READ K,K1
9009 IF K<0 THEN 9018
9010 IF K=1000 THEN 9110
9011 IF K=999 THEN 9112
9012 IFK<M THEN 9014
9013 LET M=K1
9014 FOR Y=K TO K1  -
9015 LET L(Y)=1
9016 NEXT Y
9017 GOTO 9008
9018 FOR I = 1 TO 63 STEP 3
9019 IF I>M THEN 9024
9020 IF L(I)=1 THEN 9027
9021 IF L(I+1)=1 THEN 9038
9022 IF L(I+2)=1THEN9043
9023 PRINT "   ";
9024 NEXT I
9025 PRINT
9026 GOTO 9004
9027 IF L(I+1)=1 THEN 9031
9028 IF L(I+2)=1 THEN 9036
9029 PRINT "*  ";
9030 GOTO 9024
9031 IF L(I+2)=1 THEN 9034
9032 PRINT "** ";
9033 GOTO 9024
9034 PRINT "***";
9035 GOTO 9024
9036 PRINT "* *";
9037 GOTO9024
  'R     L(I+2)=1 THEN 9034
  'C PRINT " * ";
 .40 GOTO 9024
9041 PRINT " **";
9042 GOTO 9024
9043 PRINT "  *";
9044 GOTO 9024
9045 DATA 999,0
9046 DATA 40,51,-1,0
9047 DATA 37,51,53,53,-1,-1
9048 DATA 34,49,56,56,-1,-1
9049 DATA 32,47,57,57,-1,-1
9050 DATA 30,45,58,58,-1,-1
9051 DATA 28,43,59,59,-1,-1
9052 DATA 27,41,60,60,-1,-1
9053 DATA 25,39,60,60,-1,-1
9054 DATA 24,37,60,60,-1,-1
9055 DATA 24,35,59,59,-1,-1
9056 DATA 25,32,43,45,58,61,-1,-1
9057 DATA 26,29,41,41,45,45,57,57,62,62,-1,-1
9058 DATA 32,32,40,40,46,46,53,56,-1,-1
9059 DATA 32,32,40,40,46,46,52,52,56,56,-1,-1
9060 DATA 23,23,31,31,39,39,46,46,56,56,-1,-1
9061 DATA 22,22,25,25,30,30,39,39,46,46,56,56,-1,-1
9062 DATA 22,22,26,28,38,38,46,46,56,56,-1,-1
9063 DATA 22,22,27,27,38,38,46,46,56,56,-1,-1
9064 DATA 19,22,28,28,33,35,38,38,45,45,57,57,-1,-1
9065 DATA 15,15,23,23,28,28,32,32,35,35,39,39,44,44,58,58,-1,-1
9066 DATA 13,13,23,23,29,29,32,32,36,36,38,38,44,44,59,59,-1,-1
9067 DATA 12,12,22,22,29,29,32,32,37,39,44,44,60,60,-1,-1
9068 DATA 3,5,11,11,20,23,29,29,32,32,44,44,60,60,-1,-1
9069 DATA 2,2,6,6,10,10,20,23,28,28,32,32,45,45,60,60,-1,-1
9070 DATA 2,2,7,7,20,23,27,27,32,32,46,46,59,59,-1,-1
9071 DATA 2,2,7,7,21,26,31,31,47,47,58,58,-1,-1
9072 DATA 2,4,31,31,51,56,-1,-1
9073 DATA 2,2,31,31,50,55,-1,-1
9074 DATA 2,4,30,30,50,52,54,55,-1,-1
9075 DATA 2,2,8,9,30,30,51,54,-1,-1
9076 DATA 2,4,7,7,10,10,29,29,-1,-1
9077 DATA 3,3,6,6,12,12,27,27,-1,-1
9078 DATA 5,5,15,15,26,26,-1,-1
9079 DATA 17,17,23,23,-1,-1
9080 DATA 18,18,23,23,-1,-1
9081 DATA 16,16,24,24,29,29,31,31,-1,-1
9082 DATA 16,16,24,27,30,30,-1,-1
9083 DATA 17,17,32,32,-1,-1
```

```
9084 DATA 18,18,31,31,-1,-1
9085 DATA 20,30,-1,-1
9086 DATA 47,47,-1,-1
9087 DATA 46,46,50,50,-1,-1
9088 DATA 45,45,52,52,-1,-1
9089 DATA 44,44,54,54,-1,-1
9090 DATA 44,44,56,56,-1,-1
9091 DATA 43,43,57,57,-1,-1
9092 DATA 42,42,58,58,-1,-1
9093 DATA 42,42,59,59,-1,-1
9094 DATA 42,42,59,59,-1,-1
9095 DATA 41,41,60,60,-1,-1
9096 DATA41,41,60,60,-1,-1
9097 DATA 42,42,60,60,-1,-1
9098 DATA 42,42,60,60,-1,-1
9099 DATA 42,42,60,60,-1,-1
9100 DATA 43,43,60,60,-1,-1
9101 DATA 43,43,60,60,-1,-1
9102 DATA 44,44,59,59,-1,-1
9103 DATA 45,45,59,59,-1,-1
9104 DATA 47,47,58,58,-1,-1
9105 DATA 48,48,57,57,-1,-1
9106 DATA 50,50,56,56,-1,-1
9107 DATA 52,52,55,55,-1,-1
9108 DATA 1000,0
9109 DATA 1000,0
9110 PRINT"*********************WE HOPE YOU LIKE IT******************"
9111 STOP
9112 PRINT "A PICTURE OF SNOOPY FROM THE ALTAIR 680B SYSTEM"
9113 PRINT
9114 GOTO 9008
9999 END
OK
```

```
A PICTURE OF SNOOPY FROM THE ALTAIR 680B SYSTEM
```



```
*********************WE HOPE YOU LIKE IT*****************
```

CHANGING CSAVE AND CLOAD I/O PORTS FOR 8K BASIC (VERSION 3.2)

By Tom Durston

By altering locations in 8K BASIC, it is possible to change the ACR "CSAVE" and "CLOAD" I/O ports. You may want to do this to use a different type of storage device for "CSAVE" or "CLOAD" or you may want to use the 2SIO board with the COMTER II.

The following is a listing of the portion of 8K BASIC (version 3.2) that handles "CSAVE" and "CLOAD", and the locations that may be changed. You may change the octal code either by "EXAMINE" and "DEPOSIT", or convert the octal address and code to decimal and use "PEEK" and "POKE". Note that if you POKE incorrectly, you may kill BASIC and will have to reload it.

LOCATION	OCTAL CODE	I/O PORT	2SIO CHANGES
010007	333		
010010	006	CLOAD STATUS PORT	- - - - - -
010011	346		
010012	001		
010013	302	- - - - - - - - - - - -	312
010014	007		
010015	020		
010016	333		
010017	007	CLOAD DATA PORT	- - - - - -
010020	311		
010021	315		
010022	024		
010023	020		
010024	365		
010025	333		
010026	006	CSAVE STATUS PORT	- - - - - -
010027	346		
010030	200	- - - - - - - - - - - -	002
010031	302	- - - - - - - - - - - -	312
010032	025		
010033	020		
010034	361		
010035	323		
010036	007	CSAVE DATA PORT	- - - - - -

To find the "CSAVE" and "CLOAD" locations for other versions of BASIC, enter "CLOAD" and carriage return. Then stop the computer and examine the addresses in that vicinity for a pattern similar to the one listed above.

Assembling from the Edit Buffer with Package II

By Mark Chamberlin

The typical program development process usually involves the following steps:

1. Load the Editor.
2. Use the Editor to enter the program into the Edit Buffer.
3. Output the program from the Edit Buffer to an I/O device (e.g. paper tape or cassette tape).
4. Load the Assembler.
5. Assemble the program from the tape.

However, to expedite the program development procedure, Package II allows the user to assemble a program directly from the Edit Buffer. This saves the time spent outputting the program from the Edit Buffer to an I/O device and reading it back into the Assembler.

The steps outlined below constitute a general procedure for creating a program file using the Editor and then using the Assembler to assemble the program directly from the Edit Buffer.

Step 1:

›ad the Monitor.

Step 2:

Load the Editor.

Step 3:

Type E to return to the Monitor.

Step 4:

Use the Monitor's DEP command to modify the contents of locations 5124-5125Q and 5530-5531Q. These locations contain the starting and ending addresses of the Edit Buffer, respectively. This step is necessary because the default location of the Edit Buffer is directly above the Editor, and Version 2 of the Assembler (AM2) loads directly above the Editor. In the sample program given here ("ASC"), the Edit Buffer has been moved to start at 12K and end at 16K-1. Note that 12K is 30000Q which is 0011000000000000 in binary.

Split into 8 bit bytes, this address becomes:

```
00110000  00000000

 high      low
 order     order
 byte      byte
```

Converting the bytes to octal yields:

```
00 / 110 / 000    00 / 000 / 000

0     6     0      0     0     0
```

Thus, the high order byte is 060 (octal) and the low order byte is 000 (octal).

The 8080 must always have addresses stored with the low order byte first and the high order byte stored second. Therefore, the command:

```
DEP  5124

Ø

6Ø

↑Z (control Z - not echoed)
```

is used to change the starting address of the Edit Buffer. Similarly, the ending address of the buffer is changed (see sample).

Step 5:

Restart the Editor by typing EDT and enter the program into the Edit Buffer. (See the Package II Manual for details on the use of the Editor.)

Step 6:

When the program has been entered, type E to return to the Monitor.

Step 7:

Type the command OPN FIL,EB,A. This opens the symbolic device "FIL" to the Edit Buffer in ASCII mode.

Step 8:

Load and run Version 2 of the Assembler (AM2).

Step 9:

Type FILE to tell the Assembler to read and assemble the program from the symbolic device FIL. (In this case, the contents of the Edit Buffer.)

Note that the last line of the program is a RUN directive which tells the assembler to execute the code that it assembled.

The sample program "ASC" accepts characters from the Teletype and prints the ASCII value of the character in octal. Control is returned to the Monitor when a $ is typed.

Memory Map

When using the Editor and Version 2 of the Assembler in the fashion outlined above, it is necessary to plan memory use carefully. Below is the memory map for the above example.

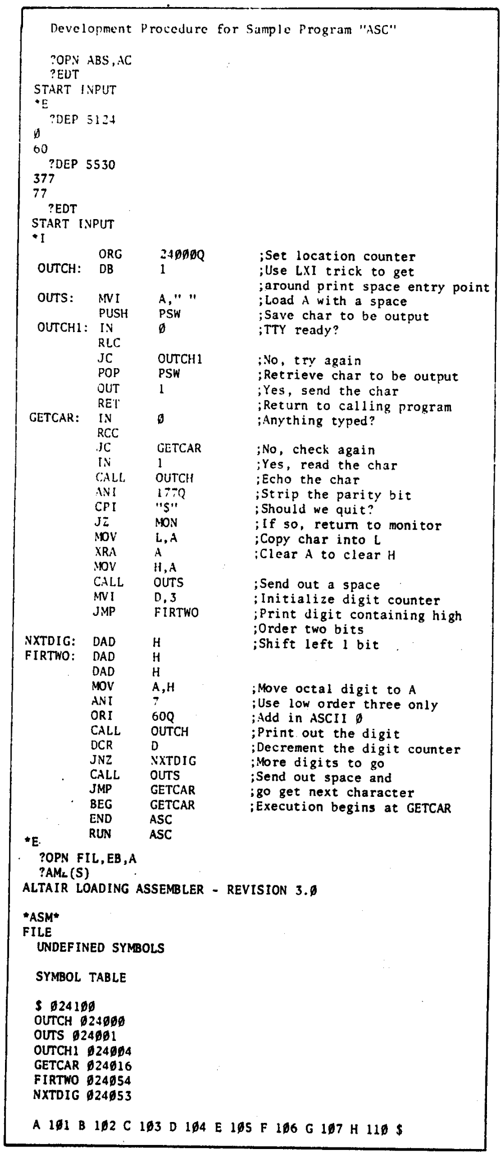
377777Q*	Edit Buffer
30000Q*	User Program Area
24000Q*	Assembler Symbol Table
17041Q	Assembler (AM2)
11553Q	Editor (EDT)
5100Q	Monitor

*User defined addresses

Memory Map
for Program Development

-continued

Development Procedure for Sample Program "ASC"



```
   Development Procedure for Sample Program "ASC"

   ?OPN ABS,AC
   ?EDT
 START INPUT
 *E
   ?DEP 5124
 Ø
 60
   ?DEP 5530
 377
 77
   ?EDT
 START INPUT
 *I
           ORG     24ØØØQ        ;Set location counter
  OUTCH:   DB      1             ;Use LXI trick to get
                                 ;around print space entry point
  OUTS:    MVI     A," "         ;Load A with a space
           PUSH    PSW           ;Save char to be output
  OUTCH1:  IN      Ø             ;TTY ready?
           RLC
           JC      OUTCH1        ;No, try again
           POP     PSW           ;Retrieve char to be output
           OUT     1             ;Yes, send the char
           RET                   ;Return to calling program
 GETCAR:   IN      Ø             ;Anything typed?
           RCC
           JC      GETCAR        ;No, check again
           IN      1             ;Yes, read the char
           CALL    OUTCH         ;Echo the char
           ANI     177Q          ;Strip the parity bit
           CPI     "$"           ;Should we quit?
           JZ      MON           ;If so, return to monitor
           MOV     L,A           ;Copy char into L
           XRA     A             ;Clear A to clear H
           MOV     H,A
           CALL    OUTS          ;Send out a space
           MVI     D,3           ;Initialize digit counter
           JMP     FIRTWO        ;Print digit containing high
                                 ;Order two bits
NXTDIG:    DAD     H             ;Shift left 1 bit
FIRTWO:    DAD     H
           DAD     H
           MOV     A,H           ;Move octal digit to A
           ANI     7             ;Use low order three only
           ORI     60Q           ;Add in ASCII Ø
           CALL    OUTCH         ;Print out the digit
           DCR     D             ;Decrement the digit counter
           JNZ     NXTDIG        ;More digits to go
           CALL    OUTS          ;Send out space and
           JMP     GETCAR        ;go get next character
           BEG     GETCAR        ;Execution begins at GETCAR
           END     ASC
           RUN     ASC
*E
  ?OPN FIL,EB,A
  ?AML(S)
ALTAIR LOADING ASSEMBLER - REVISION 3.Ø

*ASM*
FILE
  UNDEFINED SYMBOLS

  SYMBOL TABLE

  $ Ø241ØØ
  OUTCH Ø24ØØØ
  OUTS Ø24ØØ1
  OUTCH1 Ø24ØØ4
  GETCAR Ø24Ø16
  FIRTWO Ø24Ø54
  NXTDIG Ø24Ø53

  A 1Ø1 B 1Ø2 C 1Ø3 D 1Ø4 E 1Ø5 F 1Ø6 G 1Ø7 H 11Ø $
```

Reentering the Editor

Should it ever be necessary to reenter the Editor to modify the text left in the Edit Buffer from the last edit session, the R parameter should be used.

For example, to modify the sample program "ASC" after returning to the Monitor, the command EDT (R) would be used to restart the Editor.

Do not use the command EDT to reenter the Editor or the contents of the Edit Buffer will be lost.

In other words, use the command EDT to create a new program file, and use the command EDT (R) to modify a program that is already in the Edit Buffer.

Altair BASIC Biorhythm Program

Reprinted from Personal Systems
(The San Diego Computer Society Newsletter)
June, 1976

I wrote this program in MITS 8K BASIC, and it runs on my 12K ALTAIR 8800.

I don't think that anyone will get rich selling a computerized biorhythm chart (at least at the small users' level), but I think it can be used as a means of showing what our computers can do and might serve as a programming example for the novice programmer (also it's possibly good for a laugh from the more experienced "bit-diddlers").

Well, enough rambling! For what it's worth, here's my version of a biorhythm program.

Thank you,
Henry O. Arnold, Jr.

(Editor's note: Great!)

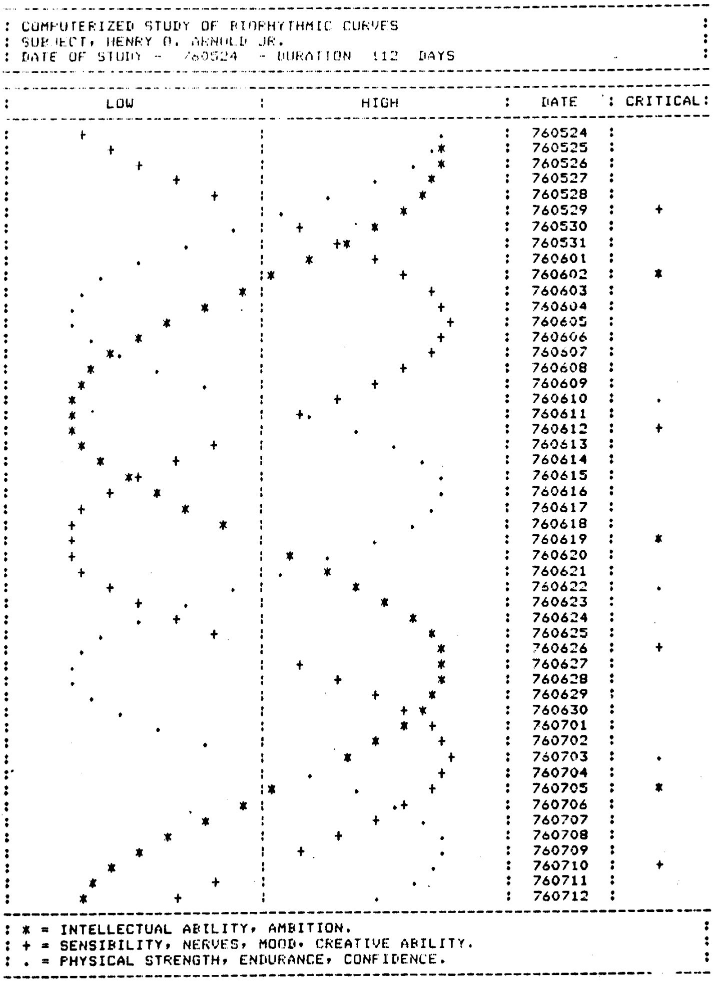
```
LIST

1 LET R1 = (360/33)/57.2958
2 LET R2 = (360/28)/57.2958
3 LET R3 = (360/23)/57.2958
50 DATA 0,31,59,90,120,151,181,212,243,273,304,334
51 DATA 365
60 DIM L$(50)
75 RESTORE
100 PRINT "ENTER BIRTHDATE,CURRENT DATE (YYMMDD)"
125 LET F1 = 0
150 LET J6 = 1
200 INPUT D1,D2
205 LET D9 = D2
206 PRINT "ENTER DURATION"
207 INPUT J5
210 PRINT "ENTER NAME OF SUBJECT"
220 INPUT A$
230 GOSUB 12000
300 IF D1 > D2 THEN PRINT "INVALID DATES":GOTO 200
400 LET X1 = D1
500 GOSUB 1000
550 LET Y1 = X2:LET M1 = X3:LET D1 = X4
600 LET X1 = D2
625 GOSUB 1000
650 LET Y2 = X2:LET M2 = X3:LET D2 = X4
800 GOTO 4000
1000 LET X2 = INT(X1/10000)
1100 LET X3 = INT(X1/100)-(X2*100)
1200 LET X4 = X1-((X3*100)+(X2*10000))
1300 RETURN
4000 LET D4 = (INT((Y2-1)*365.25)-INT((Y1-1)*365.25))
4100 FOR I = 1 TO M1
4200 READ J1
4300 NEXT I
4400 RESTORE
4500 FOR I = 1 TO M2
4600 READ J2
4700 NEXT I
4800 LET J1 = J1+D1
4900 LET J2 = J2+D2
5000 LET L1 = (Y1/4)-(INT(Y1/4))
5100 IF L1 = 0 THEN LET L1 = 1:GOTO 5300
5200 LET L1 = 0
5300 LET L2 = (Y2/4)-(INT(Y2/4))
5400 IF L2 = 0 THEN LET L2 = 1:GOTO 5600
5500 LET L2 = 0
5600 IF M1 > 2 THEN LET J1 = J1+L1
5700 IF M2 > 2 THEN LET J2 = J2+L2
5800 LET D4 = D4+J2-J1
6000 LET D1 = (D4-(INT(D4/33)*33))
6100 LET D2 = (D4-(INT(D4/28)*28))
6200 LET D3 = (D4-(INT(D4/23)*23))
6300 FOR L3 = 1 TO 50
6350 FOR I = 1 TO 50
6360 LET L$(I) = " "
6370 NEXT I
6400 LET X = SIN(R1*D1)
6500 LET Y = SIN(R2*D2)
6600 LET Z = SIN(R3*D3)
6700 LET L$(X*20+25) = "*"
6800 LET L$(Y*20+25) = "+"
6900 LET L$(Z*20+25) = "."
6950 PRINT ":  ";
7000 FOR I = 1 TO 50
7050 LET L$(25) = ":"
7100 PRINT L$(I);
7200 NEXT I
7205 PRINT ": ";
7207 GOSUB 10000:PRINT D5;" :    ";
7210 IF D1 = 0 THEN LET C = 1:PRINT "* ";
7215 IF D1 = 16 THEN LET C = 1:PRINT "* ";
7220 IF D2 = 0 THEN LET C = 1:PRINT "+ ";
7225 IF D2 = 14 THEN LET C = 1:PRINT "+ ";
7230 IF D3 = 0 THEN LET C = 1:PRINT ". ";
7235 IF D3 = 12 THEN LET C = 1:PRINT ". ";
7240 IF C = 1 THEN LET C = 0
7250 PRINT
7300 LET D1 = D1+1
7400 LET D2 = D2+1
7500 LET D3 = D3+1
7600 IF D1 = 33 THEN LET D1 = 0
7700 IF D2 = 28 THEN LET D2 = 0
7800 IF D3 = 23 THEN LET D3 = 0
7900 LET J2 = J2+1
7920 LET J6 = J6+1
7950 IF J5<J6 GOTO 8300
8000 NEXT L3
8050 LET F1 = F1+1
8100 GOSUB 14500
8125 PRINT:PRINT
8150 GOSUB 12000
8200 GOTO 6300
8300 LET F1 = F1+1
8350 GOSUB 14500
8400 FOR I = 1 TO 60:PRINT:NEXT 1
8500 GOTO 75
10000 RESTORE
10100 FOR I = 1 TO 13
10150 LET J4 = J3
10200 READ J3
10250 IF J2 > 59 THEN LET J3 = J3+L2
```

-continued

Biorhythm Program

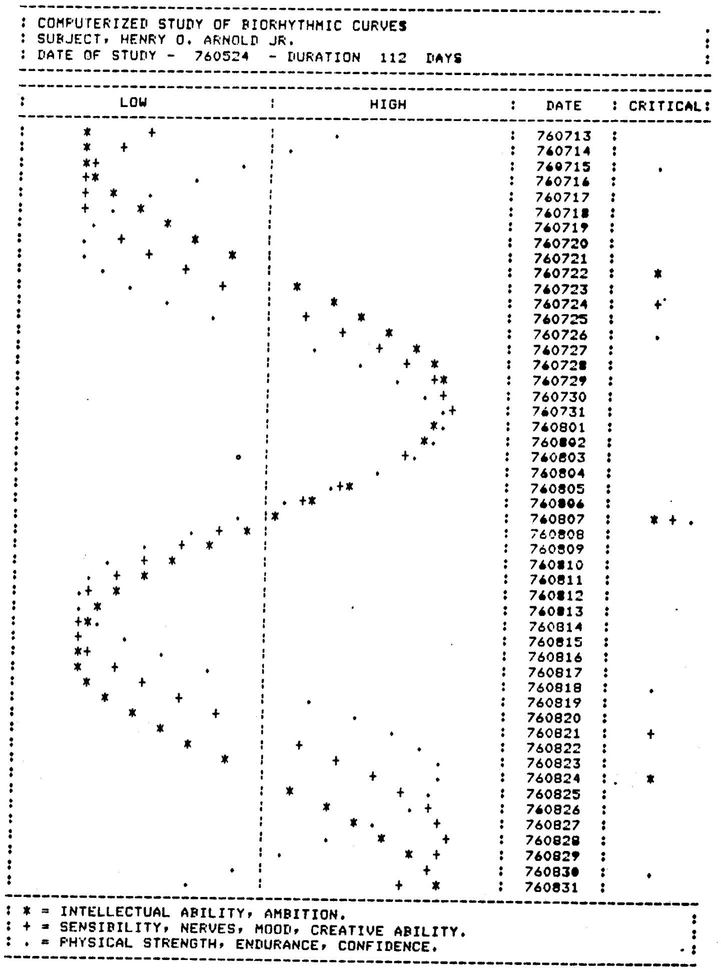
```
10300 IF J2 <=J3 GOTO 11000
10400 NEXT I
10500 LET Y2 = Y2+1
10510 LET L2 = (Y2/4)-(INT(Y2/4))
10520 IF L2 = 0 THEN LET L2 = 1:GOTO 10600
10530 LET L2 = 0
10600 LET J2 = J2-365
10700 GOTO 10000
11000 LET M2 = I-1
11100 LET D6 = J2-J4
11150 IF J2 = 60 THEN LET D6 = D6+L2
11200 LET D5 = Y2*10000+(M2*100)+D6
11300 RETURN
12000 FOR I = 1 TO 75
12100 PRINT "-":
12200 NEXT I
12250 PRINT
12300 PRINT ": COMPUTERIZED STUDY OF BIORHYTHMIC
12400 GOSUB 13600                       CURVES";
12500 PRINT ": SUBJECT, ";A$;
12600 GOSUB 13600
12700 PRINT ": DATE OF STUDY - ";D9;" - DURATION":J5
            " DAYS";
12800 GOSUB 13600
12810 FOR I = 1 TO 75:PRINT "-";:NEXT I:PRINT
13200 FOR I = 1 TO 75:PRINT "-";:NEXT I
13210 PRINT
13250 PRINT ":           LOW
            HIGH          :";
13260 PRINT "   DATE   : CRITICAL":
13400 PRINT ":"
13500 FOR I = 1 TO 75
13510 PRINT "-";
13520 NEXT I
13530 PRINT
13540 RETURN
13600 LET J = 75-POS(X)
13700 FOR I = 1 TO J-1
13800 PRINT " ";
13900 NEXT I
14000 PRINT ":"
14100 RETURN
14500 FOR I = 1 TO 75:PRINT "-";:NEXT I:PRINT
14600 PRINT ": * = INTELLECTUAL ABILITY, AMBITION.";
14700 GOSUB 13600
14800 PRINT ": + = SENSIBILITY, NERVES, MOOD, CREATIVE ABILITY.";
14900 GOSUB 13600
15000 PRINT ": . = PHYSICAL STRENGTH, ENDURANCE, CONFIDENCE.";
15100 GOSUB 13600
15150 FOR I =  1 TO 75:PRINT "-";:NEXT I:PRINT
15200 PRINT TAB(31);"PAGE ";P1
15400 RETURN
OK
RUN
ENTER BIRTHDATE,CURRENT DATE (YYMMDD)
? 410906,760524
ENTER DURATION
? 112
ENTER NAME OF SUBJECT
? HENRY O. ARNOLD JR.
```

-continued



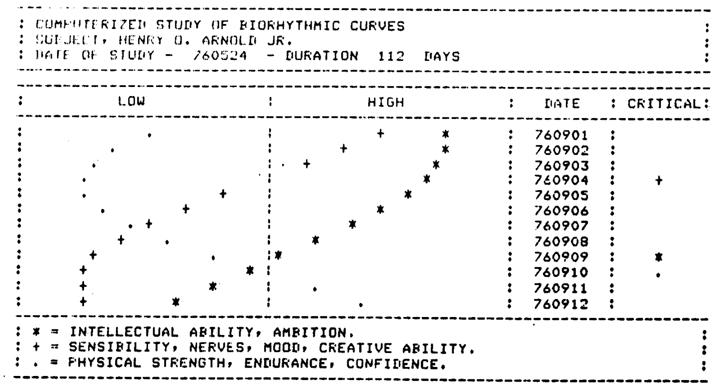
```
---------------------------------------------------------------------------
: COMPUTERIZED STUDY OF BIORHYTHMIC CURVES                                :
: SUBJECT, HENRY O. ARNOLD JR.                                            :
: DATE OF STUDY -  760524  - DURATION  112  DAYS                          :
---------------------------------------------------------------------------
---------------------------------------------------------------------------
:          LOW             :          HIGH           :   DATE   : CRITICAL:
---------------------------------------------------------------------------
:       +                  :                  .      :  760524  :
:          +               :                 .*      :  760525  :
:             +            :               .  *      :  760526  :
:                 +        :           .     *       :  760527  :
:                     +    :      .         *        :  760528  :
:                          : .            *          :  760529  :    +
:                       .  :   +     . *             :  760530  :
:                  .       :       +*                :  760531  :
:             .            :    *      +             :  760601  :
:         .                :*             +          :  760602  :    *
:       .                * :                 +       :  760603  :
:      .             *   . :                  +      :  760604  :
:      .         *         :                   +     :  760605  :
:        .    *            :                  +      :  760606  :
:          *.              :                 +       :  760607  :
:        *      .          :              +          :  760608  :
:       *            .     :           +             :  760609  :
:      *                   :       +                 :  760610  :    .
:      * .                 :   +.                    :  760611  :
:      *                   :         .               :  760612  :    +
:       *             +    :             .           :  760613  :
:         *       +        :                .        :  760614  :
:            *+            :                  .      :  760615  :
:          +    *          :                  .      :  760616  :
:       +          *       :                 .       :  760617  :
:      +               *   :               .         :  760618  :
:      +                   :           .             :  760619  :    *
:      +                   :  *   .                  :  760620  :
:       +                  : .    *                  :  760621  :
:          +            .  :         *               :  760622  :    .
:             +    .       :            *            :  760623  :
:             .   +        :               *         :  760624  :
:         .           +    :                 *       :  760625  :
:       .                  :                  *      :  760626  :    +
:      .                   :   +              *      :  760627  :
:      .                   :       +          *      :  760628  :
:        .                 :           +     *       :  760629  :
:           .              :              + *        :  760630  :
:               .          :              *  +       :  760701  :
:                    .     :           *      +      :  760702  :
:                          :        *          +     :  760703  :    .
:                          :    .             +      :  760704  :
:                          :*        .       +       :  760705  :    *
:                   .    * :             .+          :  760706  :
:                    *     :           +    .        :  760707  :
:                *         :       +          .      :  760708  :
:             *            :   +              .      :  760709  :
:          *               :                 .       :  760710  :    +
:        *            +    :               .         :  760711  :
:       *         +        :           .             :  760712  :
---------------------------------------------------------------------------
: * = INTELLECTUAL ABILITY, AMBITION.                                      :
: + = SENSIBILITY, NERVES, MOOD. CREATIVE ABILITY.                        :
: . = PHYSICAL STRENGTH, ENDURANCE, CONFIDENCE.                           :
---------------------------------------------------------------------------
```

-continued



```
---------------------------------------------------------------------
: COMPUTERIZED STUDY OF BIORHYTHMIC CURVES
: SUBJECT, HENRY O. ARNOLD JR.                                           :
: DATE OF STUDY -  760524  - DURATION  112  DAYS                         :
--------------------------------------------------------------------------
--------------------------------------------------------------------------
:         LOW              :         HIGH           :   DATE   : CRITICAL:
--------------------------------------------------------------------------
:      *      +            :      .                 :  760713  :
:      *   +               : .                      :  760714  :
:      *+               .  :                        :  760715  :    .
:      +*          .       :                        :  760716  :
:      +  *   .            :                        :  760717  :
:      +  .  *             :                        :  760718  :
:       .       *          :                        :  760719  :
:      .   +       *       :                        :  760720  :
:      .      +        *   :                        :  760721  :
:        .        +        :                        :  760722  :    *
:           .         +    :  *                     :  760723  :
:               .          :      *                 :  760724  :    +
:                    .     :   +     *              :  760725  :
:                          :       +    *           :  760726  :    .
:                          :    .      +   *        :  760727  :
:                          :         .    +  *      :  760728  :
:                          :             .   +*     :  760729  :
:                          :                . +     :  760730  :
:                          :                  .+    :  760731  :
:                          :                 *.     :  760801  :
:                          :                *.      :  760802  :
:                          :              +.        :  760803  :
:                          :           .            :  760804  :
:                          :      .+*               :  760805  :
:                          : . +*                   :  760806  :
:                       .  :*                       :  760807  :    * + .
:                  .  +  * :                        :  760808  :
:             .   +  *     :                        :  760809  :
:         .   +  *         :                        :  760810  :
:       .  +  *            :                        :  760811  :
:      .+  *               :                        :  760812  :
:      . *                 :                        :  760813  :
:      +*.                 :                        :  760814  :
:      +    .              :                        :  760815  :
:      *+       .          :                        :  760816  :
:      *   +         .     :                        :  760817  :
:       *     +            :                        :  760818  :    .
:         *       +        :    .                   :  760819  :
:            *        +    :         .              :  760820  :
:               *          :             .          :  760821  :    +
:                  *       :   +            .       :  760822  :
:                      *   :       +          .     :  760823  :
:                          :           +      .     :  760824  :    *
:                          :  *           +  .      :  760825  :
:                          :      *        . +      :  760826  :
:                          :         * .      +     :  760827  :
:                          :      .     *      +    :  760828  :
:                          : .             *  +     :  760829  :
:                       .  :                 +      :  760830  :    .
:                  .       :              +   *     :  760831  :
--------------------------------------------------------------------------
: * = INTELLECTUAL ABILITY, AMBITION.                                     :
: + = SENSIBILITY, NERVES, MOOD, CREATIVE ABILITY.                       :
: . = PHYSICAL STRENGTH, ENDURANCE, CONFIDENCE.                          :
--------------------------------------------------------------------------
```

-continued



```
------------------------------------------------------------------------------
: COMPUTERIZED STUDY OF BIORHYTHMIC CURVES                                   :
: SUBJECT, HENRY O. ARNOLD JR.                                               :
: DATE OF STUDY -  760524   - DURATION  112  DAYS                            :
------------------------------------------------------------------------------
------------------------------------------------------------------------------
:          LOW              :          HIGH            :   DATE    : CRITICAL:
------------------------------------------------------------------------------
:             .             :           +       *      :  760901   :
:         .                 :       +           *      :  760902   :
:       .                   :   +              *       :  760903   :
:      .                    :                *         :  760904   :    +
:      .               +    :              *           :  760905   :
:        .        +         :           *              :  760906   :
:           . +             :        *                 :  760907   :
:          +    .           :    *                     :  760908   :
:       +            .      :*                         :  760909   :    *
:      +                  * :                          :  760910   :    .
:      +             *      :    .                     :  760911   :
:      +         *          :         .                :  760912   :
------------------------------------------------------------------------------
: * = INTELLECTUAL ABILITY, AMBITION.                                         :
: + = SENSIBILITY, NERVES, MOOD, CREATIVE ABILITY.                           :
: . = PHYSICAL STRENGTH, ENDURANCE, CONFIDENCE.                              :
------------------------------------------------------------------------------
```

index

COMPUTER
NOTES
REVIEW

VOLUME I

ALTAIR 8800

ALTAIR 8800b

I/O DEVICES

SOFTWARE

COMPUTER NOTES REVIEW
VOLUME I
TABLE OF CONTENTS

ALTAIR 8800

ALTAIR 8800b

ALTAIR 680b

MEMORY DEVICES

I/O DEVICES

2450 Alamo S.E. / Albuquerque, New Mexico 87106

COMPUTER NOTES REVIEW
VOLUME I

SOFTWARE