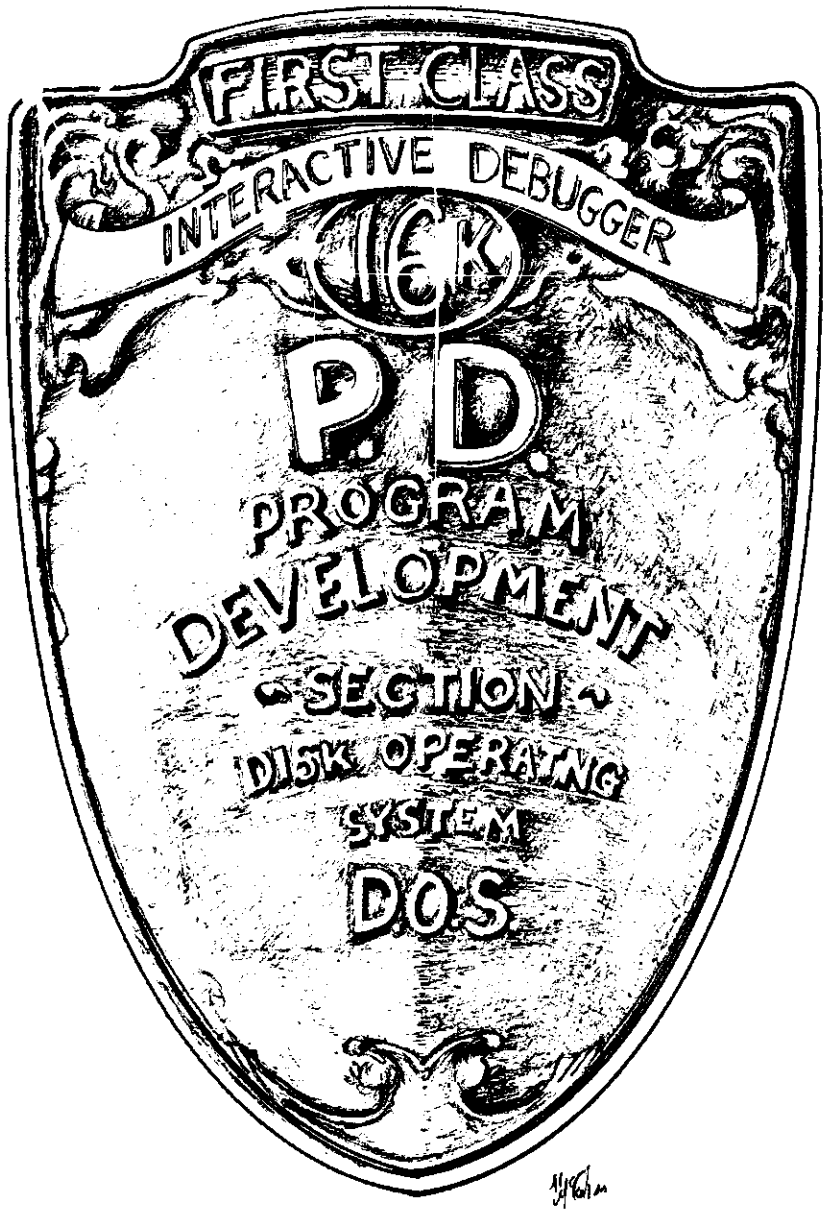
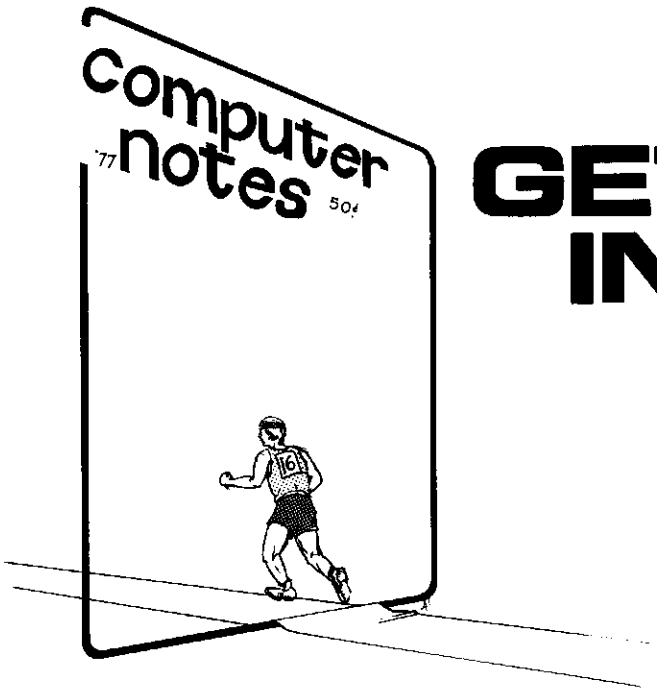
computer notes

50¢

JULY '77

Volume 3 Issue 2





GET INTO CN

The race is on to submit quality manuscripts on applications, troubleshooting, interfacing, software and a variety of other computer-related topics.

The price for accepted articles is now **$25 to $50** per typeset magazine page. Honorariums are based on technical quality and suitability for **CN's** readership. All submissions are subject to editing to fit space requirements and content needs.

Articles submitted to **C.N.** should be typed, double-space, with the author's name, address and the date in the upper left-hand corner of each numbered page. Authors should also include a one-sentence autobiographical statement about their job, professional title, previous electronic and/or computer expereince under the article's title. Authors should retain a copy of each article submitted.

All illustrations, diagrams, schematics and other graphic material should be submitted in black ink on smooth white paper. Prints and PMT's are acceptable. No pencil drawings unless properly "fixed." No halftone or wash drawings.

Whenever possible, art should be done to finished size. Complicated drawings should be submitted oversize for reduction to format by **C.N.**

All artwork should be mailed flat, never folded. Unless requested, graphics are not returned. Sketches, roughs and "idea" drawings are generally not used.

Photos, charts, programs and figures should be clearly labelled and referred to by number within the text of the manuscript.

Only clear, glossy black and white photos (no Polaroid pictures) will be accepted. Photos should be taken with uniform lighting and sharp focus.

Program listings should be recorded with the darkest ribbon possible on blank white paper.

COMPUTER NOTES is published monthly by MITS, Inc., 2450 Alamo SE, Albuquerque, NM, 87106, (505) 243-7821. A free year's subscription is included with every purchase of an Altair™ computer. Regular subscriptions can be ordered from the MITS Customer Service Dept. for $5 per year in the U.S. and $20 per year for overseas. Single copies are available for 50¢ each at all Altair Computer Centers. Send articles, questions, comments and suggestions to Editor, *COMPUTER NOTES*, MITS, Inc.

© **MITS**, Inc. 1977 (Volume 3, Issue 2, July)
2450 Alamo S.E., Albuquerque, New Mexico 87106

CASE STUDY

School Focuses Instruction on Altair 8800

By Susan Dixon
MITS

Microprocessor fundamentals, assembly of an Altair 8800 microcomputer and troubleshooting techniques form the major part of a 1640-hour computer maintenance course currently offered by the Electronics Department at the Indian Polytechnic Institute (SIPI) in Albuquerque.

"The class is exclusively hardware-oriented," explained instructor Marvin Seal. Each student is responsible for assembling a section of the Altair computer system — CPU board, display/control, motherboard, etc. Programming instruction is limited to initializing the system and general utility routines, he said. Since computer programming is not offered at SIPI, we help interested students obtain scholarships at local schools that do offer it, he added.

Seals said classes are small because some students don't want to devote over 1600 hours to the class. "Many others are anxious to return to their families on the reservation after graduation, and there are few computer-related jobs on most of the reservations," he explained. However, he said most of the students who have completed his course are now employed by AT&T on Navajo reservations throughout New Mexico.

Seals emphasized the importance of individualized instruction in his class. "It works much better than lectures, because students can progress at their own rate," he said. He also said that he coordinates his instruction with a job placement counselor to meet each student's vocational needs and interests.

Seals uses a large BITRAN-6 tube component computer to introduce students to the theory of computer operation. He follows up with the operation and assembly of the smaller and more powerful Altair 8800 microcomputer.

When students learned how to operate the Altair 8800, he said human errors made toggling from the front panel an unreliable process of initialization. "So I had them interface the unit with an ASR-33, not only to provide a more accurate method of initialization but to expose them to the problems which may be encountered when interfacing."

Continued

EDITOR
Andrea Lewis

ASSISTANT EDITOR
Linda Blocki

PRODUCTION
Al McCahon
Steve Wedeen
Lucy Ginley
Beverly Gallegos
Alice Regan

CONTRIBUTORS
Susan Dixon
Harley Dyk
Bruce Fowler
Doug Jones
Ken Knecht
Ron Scales

mits

2450 Alamo S.E.
Albuquerque, New Mexico 87106

IN THIS ISSUE

Writing Machine Helps Prepare Manuscripts

By Ken Knecht

Knecht is chief engineer at the Chicago City College TV Production Center. He's the author of DESIGNING AND MAINTAINING THE SMALL TV AND CATV STUDIO.

I wrote the following word processing program for my Altair 8800 to help in typing manuscripts for magazine articles. I hate to type, but at least this program permits me to correct mistakes and rearrange sentences without having to retype an entire page.

I used Disk Extended BASIC for the program so that the lines could be stored on disk. (Available memory would not hold a large document.) Since the items are searched for sequentially, two software-controlled tape recorders could be used to replace the two major disc files.

Before beginning an explanation of the program, it's important to understand several definitions. "Document" is the material the user wants to process. "Line" is a sentence or part of a sentence in the document. "Number" means the reference line number (**not** the BASIC program line number) added to each document. "Block # XX" refers to flow chart element # XX.

I used four major symbols in the program. The "&" at the end of a line is the flag for the end of a paragraph. The ")" at the end of a line means end of the document. The ") " at the beginning of a line means this line is to be printed as entered and not justified. The "^" at the end of a line indicates the user wants to re-enter that line because it was entered wrong.

Originally, I wanted to list the lines in random access records, using the record numbers as line numbers. However, to take full advantage of the random access, I would have to leave several records open between each line to permit adding or moving lines. In order to save disc space, each line would have to be nearly 128 bytes long to fill each record. This would mean breaking up sentences and losing the flexibility of sentence deletion, addition and rearrangement. Since discs are limited to about 2000 records, I'd end up with a disc capacity of about 700 lines, which wasn't

School Focuses Instruction on Altair 8800

Continued

During the various stages of assembly, students observe waveforms around the chips, look at the buses and learn troubleshooting techniques and the use of diagnostic equipment, Seals explained. "They apply these techniques to problems encountered during assembly and to problems that I plant in the system."

When the Altair 8800 is complete, Seals said it will be mounted on a cart with I/O provided via a TELETYPE.™ "Then everyone in the school will want to use it," he chuckled. The Optical Technology and Numerical Processing Departments have already expressed an interest in assimilating the Altair computer into their curriculum for process control, he said.

Seals said one of the future goals of the computer maintenance program is to assemble and initialize Altair microcomputers (some will include floppy disk drives) to be used for record keeping, bookkeeping, attendance, etc. in many other departments. Ultimately, Seals said that he hopes to develop an entire course of study revolving around Altair microcomputers. It would be particularly valuable because microprocessors are now used in many different types of equipment and are even found in automobiles, televisions and sewing machines, he said.

Although Seals does not currently have his own Altair computer, he said that he would like to buy one and use it at home for personal banking, record keeping, a computerized sprinkler system and as a control monitor for a fire alarm. "I'd also like to teach my kids programming, games and how to hook it up to a model train," he said with a smile.

enough for some of the material I wanted to produce. Random files would have increased speeds for printing and finding error lines. But I'm usually off doing something else during the printing routine anyway. So I decided to use the sequential disk file access method instead. I've never run out of disc space, even though several versions of the program and a few large files are stored on the word processing disc.

FLOWCHART

First the program is initialized by setting the variables, counters, arrays, etc.

Then the program prints the line number and goes to the LINE INPUT statement (block #2). This statement accepts any alphanumeric characters, including commas, double quotes, etc., up to a carriage return. If the INPUT statement was used, each line would have to be enclosed in double quotes, and double quotes could not be imbedded in the text.

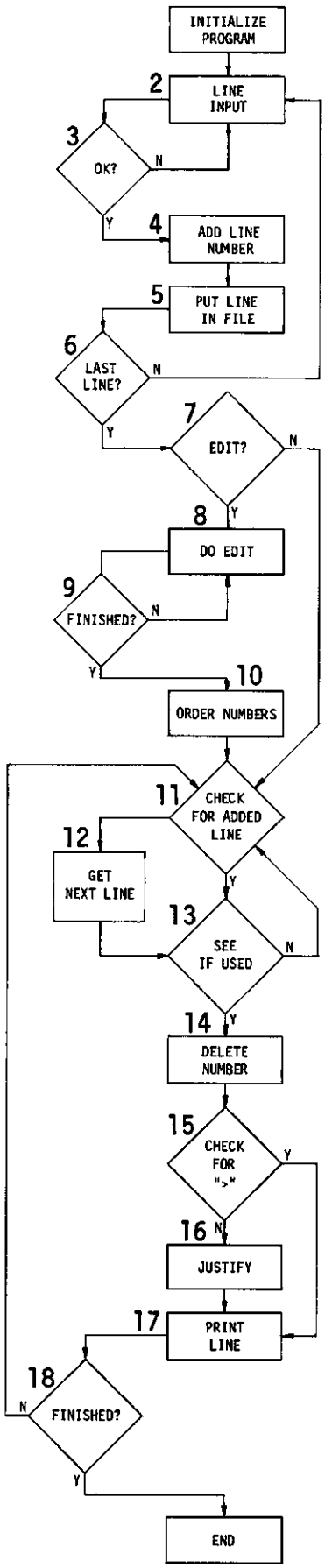
The program checks the line to be sure it is 120 bytes or less (I leave 8 bytes for the line number and "/: flag) and doesn't end with a '\'. Either requests the line to be re-entered. (See block #3.)

The next block adds a number to the line (the same one printed before the LINE INPUT), a space and a "/" and prints it into the disc file. I numbered the lines in increments of 10. Check to see if it was the last line (completed with a "}"). If not, loop back to block #2. Otherwise, proceed to the editing.

Although it's not shown in the flow chart, at this point the user can print out the file with line numbers. This is useful if the user wants to return to the file later and re-edit it.

Block #7 asks if editing is desired. If so, it jumps to block #8. Otherwise, it goes to block #11.

Block #8 sets the edit flag (ED=1) and then asks for the type of edit — add line, add null line, replace line, delete line, move line or finished. The request jumps to the proper routine. If deletion or replacement is desired, it asks for the line number, retrieves the line from the file if requested to do so and prints it. If the line is to be deleted, it sets an array element to the line number and the second dimension of that array to "2" (originally this was set to "0"). Array elements are picked by adding one to a counter at each edit and using the counter number as the array number. To replace the line, the program asks for the new line, adds the old line number and prints it into an error



FLOWCHART

file. It then puts the line number in the array, leaving "0" in the second dimension. Finally, the block loops back to the edit request statement.

When adding a line, block #1 asks for the new line number and stores it in the array. Block 11 also checks to see if there are any characters in B$. If any of them end with a "&" or "}" or are over 60 characters long, put B$ in A$ and jump to the statement which checks to see if the line is over or under 60 characters long. Then proceed from there.

If B$ is a null line or less than 60 characters long without a "&" or "}", go on to block #11 as previously described. If A$ begins with a "}", B$ must be printed, then A$. If not, B$ is added to A$ (A$=B$+" "+A$). Then go on to the program line where A$ is checked to see if it is over or under 60. Proceed from that line.

Then it asks for the new line, adds the line number to it and prints it into the error file. Moving a line is handled as a deletion and an added line. A null line edit requests the line number, puts it into the array and puts a "1" in the second dimension. Then it loops back to the edit request statement.

If the editing session is finished, both files are closed and the program goes to block #10 to put the array line numbers in order. If the line number appears in the array more than once, only the last appearance is retained with its flag in the second dimension.

At this point the edit flag, highest line number, highest array counter number and the array are stored in a third file. This permits editing or printing the file again later. These are the only variables required to restart the program with an old file.

Now the program starts printing the file. The first statement is PRINT CHR$(12), which operates the formfeed of my printer and sets it to the beginning of the next page. My paper is perforated every 11 inches to permit tearing it into standard size sheets for storage or use.

The line counter is reset to its initial setting (in M4 case 10). If the edit flag is not set, the program goes back to block #14. Otherwise, the array is checked for any line numbers less than the current line number (an added, moved or null line). If one or more are found, the lowest is used. Then the array second dimension is checked for a "1" or "2". If there's a "1", the program prints a

Continued

Writing Machine Helps Prepare Manuscripts

Continued

carriage return and goes back to block #11. If a "2", it goes directly back to block #11. If a "0", it retrieves the line from the error file by comparing the line number with the numbers added to each error line until a match is made. This procedure is followed through the rest of the error file, using the EOF to exit. It allows picking up any subsequent changes to that line.

If the program finds no lesser numbers, the next line is input from the document file, using the LINE INPUT #1, A$ statement. Then its line number is compared to array numbers to see if there is a match. If so, the second array dimension is checked for a "2" or "0". If "2", then the line has been moved, so the program goes back to block #11. If "0", then this line has been replaced. So the matching line number and line from the error file are retrieved and put into variable A$. (A$ stores the line to be used in the rest of the program.) This procedure is followed through the rest of the error file in case there was a subsequent change to that line. Then the program goes to block #15.

In block #15 the now unneeded line number is stripped off and checked for a ")". If one is found, the program jumps to block #17 and prints the line. If there is no ")", then it goes to block #16, where the justification is performed.

The justification routine first counts the characters in the line. If there are less than 60, it checks for a closing "&" or ")". If either is found, the flag is deleted and the line is printed. If the line is over 60 characters long, it starts counting backwards from the 60th character to the first space. Then everything to the right of the space is put in variable B$, to the left in A$. Next, it counts the spaces in A$, dividing up the previous count from the first space. The justification is performed by adding the extra spaces to existing spaces until there are 60 characters in the line. Then the line is printed and the program goes back to the beginning, block #11.

MODIFICATIONS

Before making any additions to this program, I suggest running it as is with a variety of input lines to be sure it handles all edited lines properly. One thing — be sure to clear variables when finished with them. Don't rely on the next input to rest them, because that variable might be checked again in some obscure part of the program, causing a number of problems. If problems do occur, adding test lines to read the contents of A$ and B$ will help locate bugs. Be sure to put the program line number in the test printout so you can see precisely where you are in the program. If all else fails, use the TRON statement to follow the program. However, it will print an unholy amount of numbers during the character scanning routines, sorting routines, etc.

To save time, use integer variables in the loops. (For example, FOR X%=1 TO M) My program is about 200 lines long and prints a justified line every four seconds. If not justified, the lines print much faster. Try to get as much accomplished as possible each time you scan a line, since these scans take the most program running time. I use only two scans — one where the spaces are counted and one where the spaces are added — to justify the line.

Once this part of the program is running properly, the following options can be added: a line counter to start a new page after 58 lines (or whatever) have been printed, a statement or two to bring page numbers at the top of each page, a routine for single or double spacing lines (don't forget to add extra blank lines after each paragraph) and routine to permit printing a title (and subtitle) at the top of each page, lower case, etc.

The lines can also be typed on an upper case only CRT or TTY. Be sure to flag each letter which is to remain upper case (I use a "\" and then print hard copy on an upper and lower case printer. Just scan the line and add 32 (decimal) to each unflagged alphabetic character to make it lower case. For example:

Z2=ASC(F$)+32:F$=CHR$(Z2) where F$ is the character scanned. If F$ is non-alphabetic or preceded by a "\", then this program line can be skipped. The line width and page length can be varied by setting them with an INPUT statement.

If you want to add formatting routines, then all your formatting questions can be put in between blocks 10 and 11 or at the beginning of the program. I put all the instructions in a disk file since they take up a lot of memory if included in the program.

My line deletion routine just scans the first eight characters in the line until a "/" is reached. The "/" was added with the line number. It assumes the next character is the first character in the line. Keep a count of the number of characters scanned, then use a RIGHT$(A$,LEN(A$)-X) statement to delete them.

SOME DAYDREAMING

One feature I intend to implement soon will permit editing within the lines. Since I haven't been able to figure out how to do this in BASIC, it probably requires a machine language program.

Another useful feature would search for a given word in the text and then change it or print the line in which it occurs. Altair BASIC's INSTR statement is useful for this.

These are just two of many possible features than can be added to your version.

For a commented listing of my version of the program with periodic fixes, send $5.00 to:

Ken Knecht
539 Addison
Chicago, Ill. 60613

Missing Track on Disk Controller Board

On the new Altair Disk Controller PC Board 1, there is a track missing between IC B1 (pins 6, 7 and 10) and VHJ supplied by R22. This PC error may cause read errors in high noise situations, since IC B1 inputs are left floating.

To correct this problem, connect a jumper wire from IC B1 (pin 10) to the far end of R22 (the right end when viewed from the component side).

This modification applies to both the Altair 88-DCDD Floppy Disk Controller and the Altair 88-MDS Minidisk Controller.

GLITCHES Correction

Please note the following corrections to GLITCHES "(Window) Program Isolates System Faults" (May CN, p. 21.)

Table 1

Location	Octal Code	
000	333	INPUT
001	( )	data address

In Table 2 location 013, the Octal Code should be 010.

In Table 3 location 027, the Octal Code should read 012.

Inverse Assembler Makes Machine Language Programs Understandable

By DOUG JONES
2271 North Mill
North East, PA 16428

For every computer and machine-language assembler that exists, someone has probably come up with an inverse assembler. The assembler will put a program together; the inverse assembler will pick it apart.

For example, you may want to modify an object tape or study some of the subroutines. Unfortunately, a source listing is often unavailable or it is proprietary material. The inverse assembler can be used to translate the binary object code into symbolic Assembly Language.

Of course, an inverse assembler is no substitute for a source listing. For example, the inverse assembler renders the following instructions

LDA A ANSW PUT THE ANSWER IN A REG BEQ NOBRANCH IF ZERO TO NO

as

LDA A $1234
BEQ $3F

This is better looking than B 6 12 34 37 3F, but it doesn't include all the information in the source code.

The inverse assembler described in this article is a BASIC language program that produces the Assembly Language equivalent of a machine language program.

The inverse assembler is capable of:

1. Inverse assembling the BASIC interpreter.
2. Inverse assembling itself.
3. Operating on any user program out of the memory bounds of either the interpreter or BASIC program.

Although the first two are interesting to analyze, the third capability is the most useful.

The following program has two modes of operation — the IA mode (inverse assembler) and the T(text) mode. The IA mode assumes that the data being operated on is machine op-codes. It gives the appropriate English language equivalent, computes bias addresses and shows addressing modes. The text mode assumes the same data being operated on is text and gives the ASCII equivalent of each.

The following summary of the 6800 addressing modes should clarify the program and its listings.

INHERENT/ACCUMULATOR

These are one-byte instructions.

Prints as:

0303 4F CLR A /O/

Since all bytes are not machine op-codes, an ASCII equivalent, if applicable, is printed between slashes.

RELATIVE

These are two-byte instructions and used during conditional branches. The second byte indicates a positive or negative direction from (PC + 2). Decimal range is limited to -128, +127.

Prints as: 0300 33 OF BHI ($OF-) $0311 /"./

The address in parentheses is the relative branch value. The hexadecimal value following it is the actual calculated value to where it will branch.

DIRECT

The second byte contains the operand address. The addressed value is in the range 0 A 255.

Print as: 030C D7 71 STAB $71

INDEXED

The numeric value of the second byte is added to the 16-bit index register. The combined value points to the operand.

Prints as: 0402 E5 03 BIT B $03,X

IMMEDIATE

The second (and sometimes third) byte is the operand.

Prints as: 0311 C6)# OC LDA B #$OC

The exceptions to the two-byte rule are the three-byte instructions CPX, LDS and LDX. The decimal value of the exceptions are 140, 142 and 206 respectively.

EXTENDED

These are three-byte instructions where the second and third byte form a 16-bit extended operand address.

Prints as: 0316 BD 0834 JSR $0834 /..4/

UNDEFINED

Hexadecimal values that are unassigned op-codes.

Prints as: 040A 3D * /=/

The following is a brief explanation of the program operation. Users may wish to modify parts or use some of these ideas for a BASIC inverse assembler on a CPU other than the 6800.

LINE **EXPLANATION**

1-32 Contain the 255 defined and undefined machine op-codes. An example of a DATA statement entry is the following:

3ADDA

where 3 tells the BASIC inverse assembler that this is an extended address op-code and that the next two bytes are to be gathered and printed. ADDA is the ASCII coded string that is to be printed as ADD A.

50-90 Establish the IA or T mode. Start and finish addresses in hexidecimal are loaded into string variables and then converted to (decimal) numeric values.

110-120 The 255 op-codes are put into string array OP (255).

340-350 Print op-code 1, 3 or 4 letters.

360 Main Main branch, depending on addressing modes. These routines fold back into each other.

O Inherent 370 Final print statement.

1 Relative 900.370 Calculate and print branch actual address from relative value.

2 Indexed 500.370

3 Extended 600.400.370

4 Immediate 700.400.370

5 Direct 800.400.370

390 A check to see if ST, the present address pointer, has surpassed the finish address.

2000-2020 ASCII equivalent string loading of op-code.

2060 Suppresses any print if there is no ASCII equivalent.

3000-3010 Load string for ASCII equivalent.

5000-6060 Complete sub-program for (T)ext mode disassembly.

9000-9120 Numerical Base Conversion.*

10000-10010 String length adjustment, prints leading zero.

11000-11010 Check on 3-byte immediate op-codes.

*__BASES,__ by Matthew Smith, **Computer Notes**, August 1976.

Continued

Inverse Assembler Makes Machine Language Programs Understandable

Continued

When answering a T to the initial question, followed by the hexadecimal start and finish address, the program will be a text disassembler.

Running text will be noted fluently. Non-ASCII bytes are printed and noted as hexadecimal value.

Example data:

54 48 45 20 43 41 54 20 49 53 20 46 41 54 2E FF 00

Prints as: THE CAT IS FAT $2E, $FF, $00

Sample run #2 is rather interesting in that it is a partial text disassembly of this simple program. A portion of it is packed by the BASIC interpreter. These packed words appear as hexadecimal equivalents. It is easy to spot the packed values and the text. Example from the run:

360 ON(V+1)GOT)00370,900,500,600,700,800

Prints as:

$01, $68, $90, (V $A1, 1) $88, 370,900,500,600,700,800$00,

Inverse Assembling a User Program

Let's assume you need an inverse assembly of a machine language program that normally occupies a 1000 length address block of continguous memory from 0000 to 03FF. Load the program in the normal manner. Then just above 03FF, write a small relocation program to produce a copy of the user program at address 4000 to 43FF.

Now load the BASIC interpreter — be sure to initialize usable memory to somewhere below 4000. Load and run the IA BASIC program. The start and finish hex addresses will be 4000 and 43FF respectively.

It's necessary to make mental adjustments to some of the addresses. For example: JSR $02AB. This subroutine actually appears at address 42AB. Actual addresses need this correction factor. Relative addresses are calculated correctly for the off-set.

```
MSG     FCC     /THE END/
        FCB $0D, $0A
        FDB $04         EOT
```

TABLE 1

6800 ADDRESSING MODE	#BYTES OF MACHINE CODE	BASIC FLAG INDICATOR
Inherent	1	0
Accumulator	1	0
Relative	2,	1
Direct	2	5
Indexed	2	2
Immediate	2*	4
Extended	3	3

*Exceptions CPX, LDS and LDX are 3-byte instructions.

```
RUN
(T)EXT OR (IA)? IA
START & FINISH ADDRESS (IN HEX)? 300,31A
                    BASIC INVERSE ASSEMBLER
0300  22 OF       BHI ($OF) $0311                    /".!
0302  39          RTS                                /9/
0303  4F          CLR A
0304  DF 6E       SIX $6E
0306  DB 6F       ADD B $6F
0308  99 6E       ADC A $6E
030A  97 70       STA A $70
030C  D7 71       STA B $71
030E  DE 70       LDX $70
0310  39          RTS                                /9/
0311  C6 0C       LDA B #$0c
0313  7F 0111     CLR $0111
0316  BD 0834     JSR $0834                          /..4/
0319  BD 0893     JSR $0893
```

Sample Run #2

```
RUN
TEXT OR (IA)? T
START & FINISH ADDRESS (IN HEX)? 2000,3000
    BASIC TEXT DISSASSEMBLER
,2,( $BA, ($BB, ( $B9, (ST $A1, 2))) $A2, 1)) $00, '$01, R$8C, 9000: $8C, 10000 $00,      $01,
T $95, "      ";: $95, $BF, (T$,3); $00, Y $01, $5E, $8A, $BA, (T$) $A9, 4 $9E, $95, " "; $C0,
(T$,1); $00, $7D, $01, $68, $90, (V $A1, 1) $88, 370,900,500,600,700,800 $00, $9C, $01, $72,
ST $A9, ST $A1, 1: $8A, A$ $AA, $A8, "" $9E, $95, $9A, 35) ;A$; $00, $AB, $01, $7C, $95, :
$8A, FX $AA, ST $9E $80, $00, $B4, $01, $86, $88, 300 $00,      $D2, $01, $90, ST $A9, ST
$A1, 1: $8C, 1000: $8C, 10000: $88, 370 $00, $FE, $01, $F4, ST $A9, ST $A1, 102, X $95, "$";
$00, !' $02, $62, ST $A9, ST
BREAK IN 9050
OK
```

THE BASIC PROGRAM

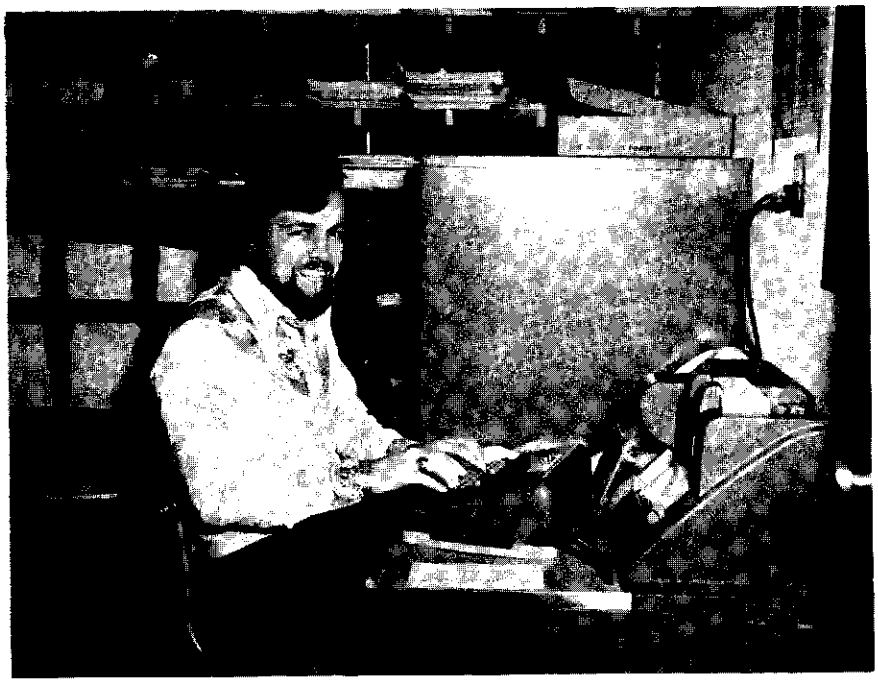
```
FORN=1TO50:?CHRS$(0);:NEXT:LIST

 1 DATA0*,0NOP,0*,0*,0*,0*,0TAP,0TPA
 2 DATA0INX,0DEX,0CLV,0SEV,0CLC,0SEC,0CLI,0SEI
 3 DATA0SBA,0CBA,0*,0*,0*,0*,0TAB,0TBA
 4 DAT)*,0DAA,0*,0ABA,0*,0*,0*,0*
 5 DATA1BRA,0*,1BHI,1BLS,1BCC,1BCS,1BNE,1BEQ
 6 DATA1BVC,1BVS,1BPL,1BMI,1BGE,1BLT,1BGT,1BLE
```

```
 7 DATA0TSX,0INS,0PULA,0PULB,0DES,0TXS,0PSHA,0PSHB
 8 DATA0*,0RTS,0*,0RTI,0*,0*,0WAI,0SWI
 9 DATA0NEGA,0*,0*,0COMA,0LSRA,0*,0RORA,0ASRA
10 DATA0ASLA,0ROLA,0DECA,0*,0INCA,0TSTA,0*,0CLRA
11 DATA0NEGB,0*,0*,0COMB,0LSRB,0*,0RORB,0ASRB
12 DATA0ASLB,0ROLB,0DECB,0*,0INCB,0TSTB,0*,0CLRB
13 DATA2NEG,0*,0*,2COM,2LSR,0*,2ROR,2ASR
14 DATA2ASL,2ROL,2DEC,0*,2INC,2TST,2JMP,2CLR
15 DATA3NEG,0*,0*,3COM,3LSR,0*,3ROR,3ASR
16 DATA3ASL,3ROL,3DEC,0*,3INC,3TST,3JMP,3CLR
17 DATA4SUBA,4CMPA,4SBCA,0*,4ANDA,4BITA,4LDAA,0*
18 DATA4EORA,4ADCA,4ORAA,4ADDA,4CPX,1BSR,4LDS,0*
19 DATA5SUBA,5CMPA,5SBCA,0*,5ANDA,5BITA,5LDAA,5STAA
20 DATA5EORA,5ADCA,5ORAA,5ADDA,5CPX,0*,5LDS,5STS
21 DATA2SUBA,2CMPA,2SBCA,0*,2ANDA,2BITA,2LDAA,2STAA
22 DATA2EORA,2ADCA,2ORAA,2ADDA,2CPX,2JSR,2LDS,2STS
23 DATA3SUBA,3CMPA,3SBCA,0*,3ANDA,3BITA,3LDAA,3STAA
24 DATA3EORA,3ADCA,3ORAA,3ADDA,3CPX,3JSR,3LDS,3STS
25 DATA4SUBB,4CMPB,4SBCB,0*,4ANDB,4BITB,4LDAB,0*
26 DATA4EORB,4ADCB,4ORAB,4ADDB,0*,0*,4LDX,0*
27 DATA5SUBB,5CMPB,5SBCB,0*,5ANDB,5BITB,5LDAB,5STAB
28 DATA5EORB,5ADCB,6ORAB,5ADDB,0*,0*,5LDX,5STX
29 DATA2SUBB,2CMPB,2SBCB,0*,2ANDB,2BITB,2LDAB,2STAB
30 DATA2EORB,2ADCB,2ORAB,2ADDB,0*,0*,2LDX,2STX
31 DATA3SUBB,3CMPB,3SBCB,0*,3ANDB,3BITB,3LDAB,3STAB
32 DATA3EORB,3ADCB,3ORAB,3ADDB,0*,0*,3LDX,3STX
40 CLEAR1000
50 INPUT"(T)EXT OR (IA)";A$
60 INPUT"START & FINISH ADDRESS (IN HEX)";C$,T$:PRINT
80 LETBT=F:C$=T$:GOSUB9000:FX=F:IFFX  =STTHEN60
100 IFA$="T"THENBT=16:BF=10:GOT05000
110 DIMOP$(255)
120 FORN=0TO255:READOP$(N):NEXT
140 PRINT:PRINTTAB(10);"BASIC INVERSE ASSEMBLER":PRINT
300 YP=PEEK(ST):V=VAL(LEFT$(OP$(YP),1))
302 T$=MID$(OP$(YP),2,LEN(OP$(YP))-1)
310 GOSUB2000:BT=16:BF=10
312 C$=MID$((STR$(ST)),2,(LEN(STR$(ST))-1))
320 GO$UB9000:GOSUB8000:PRINT"      ";
331 C$=MID$((STR$(YP)),2,(LEN(STR$(YP))-1))
332 GOSUB9000:GOSUB10000:PRINT"      ";
333 IFV=0THENPRINT"      ";:GOT0340
334 C$=MID$((STR$(PEEK(ST+1))),2,(LEN(STR$(PEEK(ST+1)))-1))
335 GOSUB9000:GOSUB10000:GOSUB11000:IFFL:=1THEN337
336 IFV  3THENPRINT"      ";:GOT0340
337 C$=MID$((STR$(PEEK(ST+2))),2,(LEN(STR$(PEEK(ST+2)))-1))
338 GOSUB9000:GOSUB10000
340 PRINT "      ";:PRINTLEFT$(T$,3);
350 IFLEN(T$)=4THENPRINT"  ";RIGHT$(T$,1);
360 ON(V+1)GOTO370,900,500,600,700,800
370 ST=ST+1:IFA$  "THENPRINTTAB(35);A$;
380 PRINT:IFFX SITHENEND
390 GOTO300
400 ST=ST+1:GOSUB1000:GOSUB10000:GOT0370
500 ST=ST+1:GOSUB1000:PRINT"4";:GOSUB10000:PRINT",X";:GOT0370
600 PRINT"$";
610 ST=ST+1:GOSUB1000:GOSUB10000:GOT0400
700 PRINT"#$";:IFFL=1THENFL=0:GOT0610
710 GOTO400
800 PRINT"$";:GOT0400
900 TP=ST+2:ST=ST+1
910 GOSUB1000:PRINT" ($";:GOSUB10000:PRINT") ";
920 IFF 127THENF=F-256
930 TP=TP+F
940 BT=16:BF=10:C$=MID$((STR$(TP)),2,(LEN(STR$(TP))-1))
950 GOSUB9000:PRINT"$";:GOSUB8000:GOT0370
1000 BT=16:BF=10:C$=STR$(PEEK(ST)):GOSUB 9000:RETURN
2000 A$="/":GOSUB3000:IFV=0ORV=5THENGOT02050
2020 GOSUB11000:D=ST:ST=ST+1:GOSUB3000:ST=D
2050 A$=A$+"/"
2060 IFA$="/.../"ORA$="/../"ORA$="/./"THENA$=""
2070 RETURN
3000 IFPEEK(ST) 31ANDPEEK(ST) 91THENA$=$+CHR$(PEEK(ST)):RETURN
3010 IFPEEK(ST) 32ORPEEK(ST) 90THENA$=A$+".":RETURN
5000 PRINT:PRINTTAB(10);"BASIC TEXT DISSASSEMBLER":PRINT
6000 FORK=STTOFX:A=PEEK(K)
6030 IFA 31ANDA 91THENPRINTCHR$(A);
6040 IFA 32THENC$=MID$(STR$(A),2,LEN(STR$(A))-1):GOSUB6100
6050 IFA 90THENC$=RIGHT$(STR$(A),3):GOSUB6100
6060 NEXTK:END
6100 GOSUB9000:PRINT" $";:GOSUB10000:PRINT", ";:RETURN
8000 IFLEN(C$)=1THENPRINT"000";
8010 IFLEN(C$)=2THENPRINT"00";
8020 IFLEN(C$)=3THENPRINT"0";
8030 PRINTC$;:RETURN
9000 E=):B$="":D$="0123456789ABCDEF":BT=INT (BT):BF=INT(BF):01=1
9002 D=0:02=2:00=0
9030 FORY=O1TOLEN(C$):FORX=O0TOBF-01
9040 IFMID$(C$,Y,O1)=" "THENY=Y+01:GOT09040
9050 IFMID$(C$,Y,01)=MID$(D$,X+01,01)THEN9070
9060 NEXT:END
9070 D=D*BF+X:NEXTY:C$="":F=D
9080 B$=B$+STR$(INT((D/BT-INT(D/BT))*BT)):D=INT (D/BT)+.01
9084 IFD =01/BTTHEN9080
9090 FORX=LEN(B$)TOO2STEP-O2
9100 IFMID$(B$,X,01)=" "THENX=X-01:GOT09100
9110 C$=C$+MID$(D$,VAL(MID$(B$,X-01,02))+01,01)
9120 NEXT:RETURN
10000 IFLEN(C$)=1THENPRINT"0";
10010 GOSUB8030:RETURN
11000 IFYP=140ORYP=142ORYP=206THENFL=1
11010 RETURN
OK
```

PUT A MICRO IN YOUR SCHOOL

Getting Started with Zero Bucks

By Harley Dyk
Math-Electronics Instructor
Mona Shores High School
Norton Shores, MI 49441



The way we were — Harley Dyk sitting at the keyboard of the retired Model 26 keypunch. (Photo by LCDR F. Wayne Brown.)

I was introduced to computer programming in the summer of 1970 while taking graduate courses in mathematics at Central Michigan University. An introductory course in FORTRAN on the good old IBM 1130 very quickly made an addict of me. Keypunching all your own cards (the system was strictly batch) and losing 10 points on every problem for each time your program failed to run correctly was quite far removed from having your own microprocessor in your basement, but compared to nothing it was great and I was hooked.

Starting a High School Computer Class with Zero Bucks

Upon returning to teaching in the fall, I immediately began looking for computer time for three reasons. First, if I could so easily become hooked on programming and enjoy it so much, why couldn't (and shouldn't) high school students have the same opportunities and experience? Also, I felt that any student who wanted to learn the basics of programming or was curious about computers should have the opportunity to be exposed. Last, I needed some computer time to satisfy my addiction.

My search turned up the fact that our intermediate school district was the cheapest source of computer time. Our school district was paying a fixed amount per student for scheduling, report cards, etc. and could use the computer for instructional purposes at no additional cost. To make the arrangement irresistible, they agreed to do the keypunching. Since my school board favors most proposals that have little or no cost attached, the fall of 1971 found me teaching a new course called Computer Math. The FORTRAN language was used for two reasons: This was the only language I knew, and the computer center had a little used FORTRAN compiler on the shelf for the Honeywell 200.

Things went as well as could be expected. In spite of shuffled card decks, dropped program decks, lost programs, coding forms returned unpunched, computer breakdown, Murphy's Law, etc., some programming was taught. But who could complain? The price was right. But after a year the computer center moved to new facilities 10 miles away. Consequently, it was impossible to arrange transportation of coding forms, have them punched, program decks run and returned by the next day. If you think it is bad to wait 24 hours to find out you used a comma where you needed a period, try 48 hours! Solution? Lease a keypunch, find a student to keypunch, offer her a credit in keypunching and presto, back to 24 hour turnover.

Unfortunately, the computer center was not on my route home, and as time went on, finding someone to pick up and deliver cards twice a day became more difficult and expensive. We were now spending about $150 a month on the keypunch, card delivery and cards. The only good thing about our setup was that we had no limit on how many programs we could run each day.

It seemed like a logical time to begin thinking about and examining alternatives. Was something better available for $150 a month? A check of high schools in the area fielded the fact that one high school had decided that buying a computer time via phone line and Teletype was getting too expensive and had leased a $14,000 DEC PDP-8 with two Teletypes on time-sharing but at considerably more than $150 a month.

Dream

January, 1975 — enter the Altair 8800 computer via **Popular Electronics.** I wondered if there really was a garage in Albuquerque where this invention was being put into producton. September of 1975 found me researching what was available for use in teching my class. At this point I was convinced that we needed and could afford our own system. My goal — to get a computer system in my room by September, 1976 no matter how much work and time it took.

Computer Math student Sheila Beaver at Teletype and Altair 8800A. (Photo by Gary Reed.)

Conception

After seeing a DEC Classic and PDP-8 in action at other schools and attending the MITS Computer Caravan, I wrote a proposal for my school board to act on. My proposal outlined my rationale and presented three systems: the DEC Multiuser/11V03 at $20,000 (four simultaneous users — three in BASIC and one in FORTRAN if desired), the DEC Classic with mark sense card reader at $17,000 (FORTRAN and BASIC) and two Altair 8800s at about $7,000 (BASIC only).

Difficult Pregnancy

I gave the superintendent the proposal in January of 1976 and was scheduled for the March 8th board meeting to present my ideas. Unfortunately, three board members were out of town for the meeting and the other four did not feel adequately familiar with my proposal, so it was tabled until March 22nd when hopefully all members would be present for a more fruitful discussion. The March 22nd meeting was more productive, but still it was obvious that no one really had an idea how microprocessors were being used in high schools. My proposal was eventually tabled indefinitely — the money situation was not clear at that time and a millage (tax adjustment) election was coming.

The next few months found me educating my principal and assistant principal and later the school board president (an engineer familiar with computers). I simply took them to a high school that was using a DEC Classic configuration and exposed them to the enthusiastic instructor and students.

June brought a millage defeat and a last ditch effort. I rewrote my proposal, eliminating the more expensive DEC systems and asking for the Altair system. Having originally approached the school board with options was fast becoming a wise decision, because the least expensive option was becoming more acceptable. Since I had written my original proposal, MITS had introduced the 16K static memory board, improved the 8800 and Michigan now had its first computer store in Ann Arbor. I also had the chance to talk to a couple of Altair owners and was convinced that it could do what we needed. I mailed copies of my new proposals to the board members and superintendent and waited.

About the middle of July (a new superintendent had assumed his duties July 1), I received a phone call indicating that some money had been located and that I should come in for further discussion. July

Continued

PUT A MICRO IN YOUR SCHOOL
Getting Started with Zero Bucks
Continued

Computer Math student Mark Tietsort at CRT and Altair 8800A. (Photo by Gary Reed.)

27 found me at a third board meeting. Now I found out that had the administration and I come with a recommendation to buy a $20,000 system, we probably would have been given approval. But we were proposing the Altair system and were given an OK to proceed, pending a checkout of MITS and the local store. They checked out fine. Finally, on August 5th, I grabbed the phone and ordered our computer system (before anyone could change their mind). The order was for two Altair 8800As, each with 16K static memory, serial I/O board, ACR board and RQ-413 Panasonic recorder. One computer was to have an ASR-33 Teletype (for needed hard copy) and the other an ADM/3 Lear-Siegler CRT. We also ordered 8K BASIC, Extended BASIC and a supply of Maxwell UDXL C-60 cassette tapes. Total cost was about $7000.

While waiting for the system, the Computer Store rushed me the BASIC manual, and I boned up, since I had not written any programs in BASIC. There was no real need for panic — school was not to start for at least four weeks. I found the transition from FORTRAN to BASIC very natural, easy and enjoyable. The fine manual on MITS BASIC helped a great deal. I doubt if switching from BASIC to FORTRAN would be as painless.

Birth

Our system was tentatively scheduled to be delivered September 1, but the Computer Store was experiencing labor (growth) pains and was not receiving shipments from MITS when expected, so birth was not quick and painless. The fact that I am three hours from the Computer Store did not help either.

The Teletype™ was the first item to arrive. I had ordered the ASR-33 directly from the Teletype Corp., from whom I received excellent cooperation. The Teletype was delivered to my room at school exactly 28 days after I had phoned in the order!

Labor Day found me picking up the first computer (sweet labor!) from a brother-in-law of the store owner. He had just come back from Ann Arbor and was only about 45 minutes from me. The store had still not received any assembled units so had sent me their demo and were assembling a kit for me. The next week found me furiously programming in my basement by night while we began a week long unit on flowcharting in my class. The keypunch machine was still in my classroom, but we ignored it.

In late September the Computer Store put on a demonstration in my room for area high school instructors, showing the 8800A, 8800B, 680, disk, etc. I was supposed to get the second computer then and keep the demo so I could be totally up and running. However, a bad ACR board prevented this. A few weeks later the second system was delivered (students had been saving programs on cassette from the CRT computer). However, this ACR board would not allow us to save programs, only load them. At least we could load BASIC and load programs from the other computer and use the paper tape feature of the Teletype to save programs. Some time later we got another ACR board and were totally up and running. Birth was complete!

Early Childhood

In spite of a difficult pregnancy and birth, early childhood has made it all worthwhile. Once our system was complete, we had no hardware problems. I have loaded and saved hundred of programs on cassette and have had only one failure which may have been due to my error. The computers are left on all the time (the terminals are turned off at night) so that loading Extended BASIC (which takes from 8 to 10 minutes) does not have to be done each morning. Occasionally, BASIC may start acting strange or die, so each computer is loaded probably on an average of every two or three weeks. We have been using Extended BASIC Ver. 3.2 which gives us 5984 bytes for programming and 100 bytes for string space. This is adequate memory for most programs, but some games written by my students could have used more memory. I am sure that if we had 64K someone would still complain! We have found a few flaws in the Extended BASIC Ver. 3.2, but they are nothing that we couldn't program around. The flaws should be fixed in the 4.0 version, which we have on order.

The computers are housed in a 10' by 10' office adjacent to my classroom, so programmers can come during any hour without bothering the class in session. They have been used an average of five hours a day. It was decided to try time and material for maintenance on everything except the Teletype, on which we have a service contract for $216 for the nine month school year.

I was always under the impression that FORTRAN was a more powerful language than BASIC, but I have been able to do all my old FORTRAN programs in MITS BASIC and am very impressed with it. Some programs took more steps in BASIC, but the majority were easier to do in BASIC. I am no longer a member of STAB (Society to Abolish BASIC); as a matter of fact, I think STAB is dead.

The students love the computers; two or three of them are seriously considering purchasing their own system. One student has one on order for graduation. He thinks it is great that computers cost less than new cars, but he laments the fact that he can not take his girl out in a computer. The computers have been in such demand that the students created a number system to determine whose turn it was to work on the computer.

Conclusion

The main reason I wrote this article was to urge hobbyists (who may also be parents) to begin insisting that their junior high and senior high schools have computer classes and facilities. Many teachers (particularly math teachers) have had some education in programming but are not familiar with the cost and capabilities of microprocessors. If you fall in this category, get off your duff and get your hands on a microprocessor. Find out if you can appropriate a couple of thousand for a system for your class. It's worth the effort! You will wonder why you didn't do it before. True, a system will cost less next year, but it is affordable now. Can you put a price on the education this year's students will miss—

Hobbyists, help that teacher get started. Maybe he is hardware shy, maybe he would be ready to move if someone in the community would be willing to help him get a system up and running. You can now buy a system (decent quality by my estimation) with 12K memory, 8K BASIC, connected to a keyboard, 16 line with 64 character video monitor and cassette for under $1300. True, the computer itself is in kit form. But again, find someone to help; help is available if you look, or better yet, put it together yourself. With some assembly (maybe 50 hours), you should be able to set up two systems, one as above, another like it but with a Teletype for about $3000. You could provide 4000 student hours on a computer in two years — not bad at less than $1 per hour!

Our baby is healthy, but my students have indicated in a recent survey I distributed that he must grow. They think we will need at least another terminal or system next year (due to increasing interest and more students in the class). We could use faster hard copy, and a floppy disk would be very useful. OK, where are the catalogues? Hm . . . time-sharing, line printer, floppy disk, Dec-Writer II, more memory . . . Say, when is the next board meeting?

GLITCHES

Spot Altair 88-PMC Problems

By Bruce Fowler

MITS

To use MITS Altair PROMs, they must be installed in a working Altair PROM memory board. Occasionally, an 88-PMC kit board may create problems for the user. This article gives a few hints for tracking down and eliminating 88-PMC problems.

Troubleshooting:

First, measure the output of the +5v and -9v voltage regulator on the 88-PMC board. If the voltages are incorrect, lift the output pin from the board and measure them again. If correct, look for shorts. Replace the voltage regulator if the voltages are incorrect.

The remaining problem areas for the 88-PMC board can be divided into four areas: (1) the chip select circuit, (2) the VGG switching circuit, (-) the PROM circuit and (4) the wait state circuit. Each area is described in detail below.

CHIP SELECT CIRCUIT:

Since 8 PROMs can be mounted on one board and the data outputs of all PROMs are in parallel, some sort of chip select circuit is necessary to enable only one PROM at a time and disable the other seven. This is done by applying a TTL low signal on pin 14 of the PROM to be enabled. A PROM is disabled by applying a high signal to pin 14. Depending on address bits A8-A10, the chip select circuit applies a low signal to one of the 8 PROMs, ICs A-H. To test and fix the chip select circuit, do the following

1. Examine the first memory location of the board from the front panel. BS (board select), pin 9 of IC K, should go low and stay low. Otherwise, check to see that IC K is working and that the address strapping jumpers are correctly installed.
2. Keeping address bits A11-A15 the same, change bits A8-A10 and examine the resulting locations. Monitor pin 14 (chip select) of the 8 PROM sockets (ICs A-H). The TTL levels should correspond to those in Table 1. If a discrepancy is found, look for shorts and opens. Pull up the suspected output in pin of IC J (pins 9, 10, 11, 13, 14, 15, 16, or 17) and recheck it with Table 1. If still wrong, check IC I. If IC I operates correctly, replace IC J.

3VGG SWITCHING CIRCUIT:

To save power, the power to 1702 PROMs is reduced when they are not selected. This is done by putting +5v on pin 16 (VGG) of the 1702A PROM for the reduced power state dn-9v on pin 16 for the active state. The VGG switching circuitry is tested in a similar manner to the chip select circuit. However, not that two PROMs will have -9v supplied to them simultaneously for an active VGG. To test the VGG circuit, do the following:

1. Redo step 2 above using Table 2 while monitoring pin 16 (VGG) of the PROM sockets (IC A-H).
2. For any invalid VGG level, examine the corresponding circuit for it. The VGG switching circuit for PROMs A and B is shown in Figure 1. All VGG switching circuits are similar.
   a. A low at either diode cathode should produce about 4.2v at the base of A1. Otherwise, replace diodes D1 and D2 if no shorts or opens are found. If no lows are present at either diode cathode, then transistor Q1 base should be about +5.2v. Otherwise, replace D1 and D2 if no shorts can be found.
   b. The 4.2v voltage at the base of Q1 should turn on Q1 producing about 5v at the collector of Q1. Otherwise, replace Q1 if no shorts or opens are found. If no lows are present at either diode D1 or D2 cathode, transistor Q1 collector should be about -9v.
   c. The 5v at the Q1 collector should turn on transistor Q2, causing Q2's collector to go to -9v. Otherwise, replace Q2 if no shorts or opens are found. The

Continued

GLITCHES
Spot Altair 88-PMC Problems

Continued

voltage at transistor Q2's collector should be about 5v if no lows are present at either D1's or D2's cathodes. Otherwise, check resistor R3's resistance and look for shorts. If these are satisfactory replace Q2.

PROM CIRCUIT

To check the PROM circuit, a listing of the data in each PROM is needed. Examine the PROM locations and compare the data lights which represent the PROM output data with the listing. A11-A15 must correspond to the Prom Board's address.

If any locations do not match the listing, then the most probable causes are shorts or opens on the data or address lines on the back of the board. Trace out these lines very carefully. Check power and ground conductors to the PROMs. Data buffers IC V and W should be enabled by a low at pin 1. Compare the input with the output for these buffers to make sure they are the same.

WAIT STATE CIRCUIT

The Wait State circuit is a little harder to troubleshoot. Since the PROM access time is too slow for the memory read time supplied by the 8080A, wait states are forced by the PROM board to add on extra clock periods (500nsec for each period) to the memory read time. The PROM board generates a wait state by pulling Ready low during period 2 of the read cycle. If Ready is low when the 8080A samples it with 02, the 8080A waits one clock period and then samples Ready again. When Ready goes high, the 8080A completes the machine cycle. If no wait state occurred, the 8080A would get invalid data from the slow PROM. Jumpers J6 and J7 are usually strapped for 1 wait state fro the 1702 PROMS. Regardless of how they are strapped, jumpers must be present for at least one wait state. To check the wait state circuit and repair it, do the following:

1. Strap the board for the highest possible location (174000 octal) and run the program in Table 3. Monitor pins 13 and 14 of IC W. If low pulses (about 500nsec wide) do not occur, then no wait states are being generated by the PROM board.
2. Stop the computer and examine the first location of the PROM board. Compare the TTL levels on the board with Table 4 for the stopped state. Check logic operation of any gates whose outputs disagree with Table 4.
3. To check the logic operation of IC P and S, remove jumpers J6 and J7. Activate the Run switch on the front panel and then the Stop Switch. 01 (a 2 Megahertz singal) should be present at pins 8 and 12 of IC P and pin 12 of IC S. If not, check these gates. Don't forget to replace J6 and J7.
4. To check the other ICs, run the program in Table 3 while checking for proper logic signals at the gates. Make sure the signals match the description in the Altair 88-PMC Theory of Operation Manual.

This information should at least give the kit builder a start. For further help, please contact MITS or the Repair Department of your nearest Altair Computer Center.

Table 1

Sockets, pin 14

A10	A9	A8	A	B	C	D	E	F	G	H
LOW	LOW	LOW	LOW	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH
LOW	LOW	HIGH	HIGH	LOW	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH
LOW	HIGH	LOW	HIGH	LOW	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH
LOW	HIGH	HIGH	HIGH	HIGH	HIGH	LOW	HIGH	HIGH	HIGH	HIGH
HIGH	LOW	LOW	HIGH	HIGH	HIGH	HIGH	LOW	HIGH	HIGH	HIGH
HIGH	LOW	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	LOW	HIGH	HIGH
HIGH	HIGH	LOW	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	LOW	HIGH
HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	LOW

Table 2

Sockets, pin 16

A10	A9	A8	A	B	C	D	E	F	G	H
LOW	LOW	LOW	-9V	-9V	5V	5V	5V	5V	5V	5V
LOW	LOW	HIGH	-9V	-9V	5V	5V	5V	5V	5V	5V
LOW	HIGH	LOW	5V	5V	-9V	-9V	5V	5V	5V	5V
LOW	HIGH	HIGH	5V	5V	-9V	-9V	5V	5V	5V	5V
HIGH	LOW	LOW	5V	5V	/5V/	5V	-9V	-9V	5V	5V
HIGH	LOW	HIGH	/5V/	5V	/5V/	5V	-9V	-9V	5V	5V
HIGH	HIGH	LOW	5V	5V	5V	5V	5V	5V	/-9V/	/-9V/
HIGH	HIGH	HIGH	5V	5V	5V	5V	5V	5V	09V	-9V

*Valid only if examingIng PROM board's address.

Extra Voltage Needed

Using the TTY (20mA loop) interface with the Lear Siegler ADM-3A CRT terminal and the Altair 680b computer requires additional voltage to the interface circuit. The following diagram illustrating this connection was contributed by Bob Burnett of the Houston Altair Computer Center.

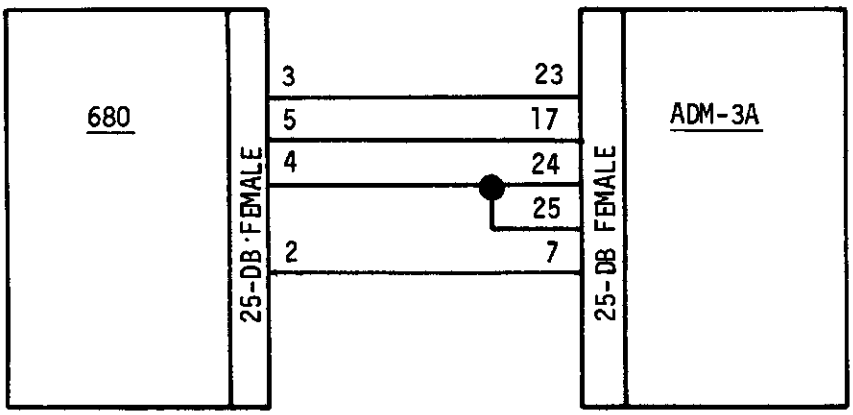


Table 3

Location	Op code	
000	072	Move to ACC
001	000	data from
002	377	PROM
003	062	Store in
004	040	Location 040
005	000	
006	303	Repeat
007	000	
010	000	

Table 4

IC	Pin	Stopped State*	RUN State
K	9	LO	pulses
S	4	HI	pulses
V	1	LO	pulses
W	1	LO	pulses
W	15	LO	pulses
S	2	01	01
S	8	LO	pulses
S	10	HI	HI
L	3	02	pulses
M	6	HI	pulses
P	6	HI	pulses
P	12	LO	pulses
P	8	HI	pulses
M	3	LO	pulses

*Valid only if Board Location is examined.

Software Winners Announced

The following are software contest winners for submissions in March, April and May, 1977.

Best Programs

First Place #7014
Title: Complex Number Interpreter for BASIC
Author: Dr. John J. Herro
Dayton, Ohio

Second Place: #7009
Title: Checkbook Balancing Program
Author: Loring C. White
Reading, Mass.

Computer Club Formed

The Canadians recently formed a new microcomputer club. For more information contact:

Mr. Gerard Bouchard
Dept. Electro
College de Chicoutimi
534, rue Jacques-Cartier est
Chicoutimi, Que
G7H 1Z5

ALTAIR COMPUTER CAPABILITIES SHOWCASED AT NCC

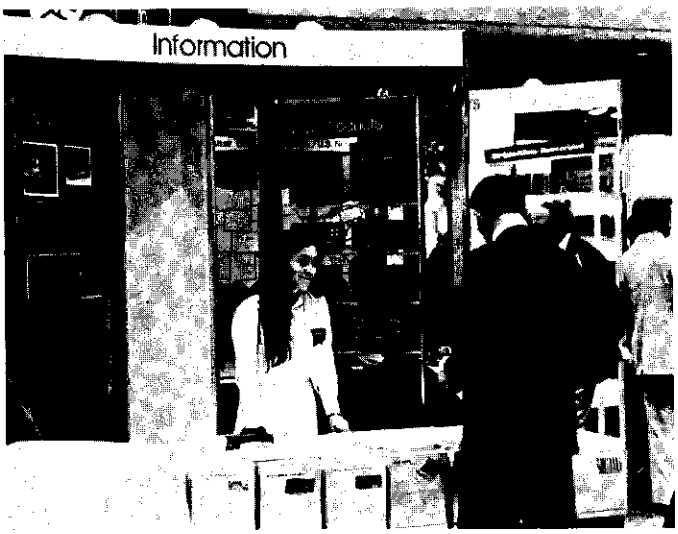
(All photos by **Steve Wadeen.**)

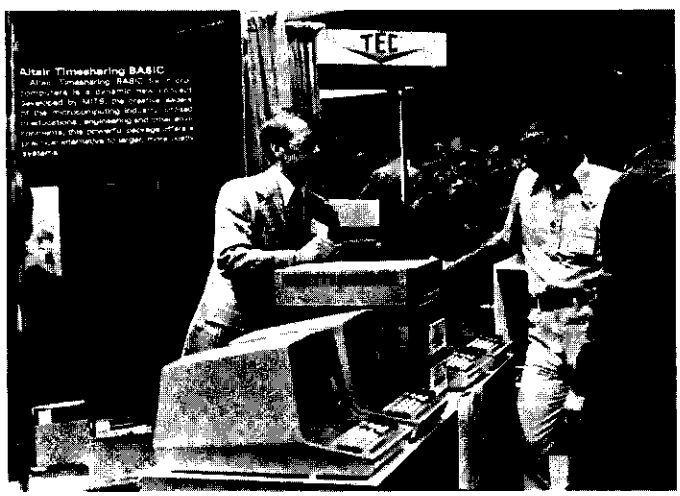
MITS Dealer Sales Manager **Dave Carey** keeps things running smoothly at the Altair computer booth.

Thousands of enthusiastic conventioneers saw the versatility of Altair computer systems demonstrated at the NCC in Dallas, June 13-16. The MITS booth featured Altair Timesharing BASIC, Altair Industrial and Scientific control systems and the Altair Business System. The interest generated by these complete, yet affordable systems clearly established the Altair computer as much more than just a personal computer.

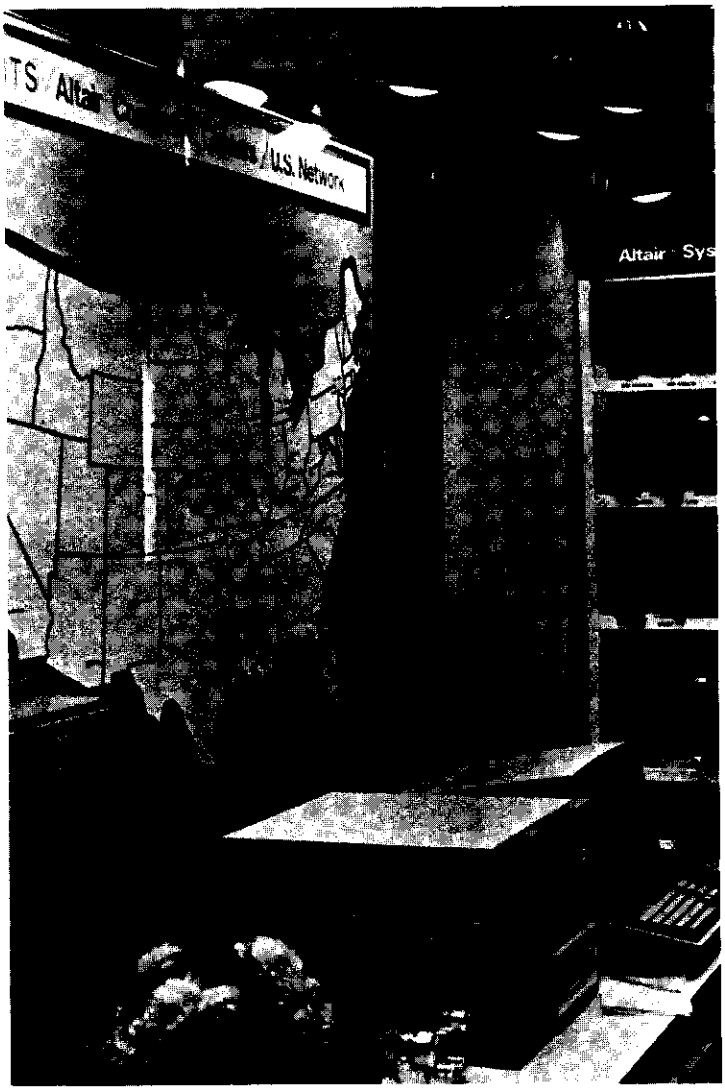
Several new Altair peripherals were also introduced at the NCC, including the Altair Hard Disk, Altair Minidisk, Altair AD/DA board, a diagnostic card and a 16K memory board.

Pam Duran, head of the MITS Technical Writing Department, was a familiar face all week at the Altair computer booth. Pam gave out information packets and answered questions about Altair computer products for thousands of interested conventioneers.

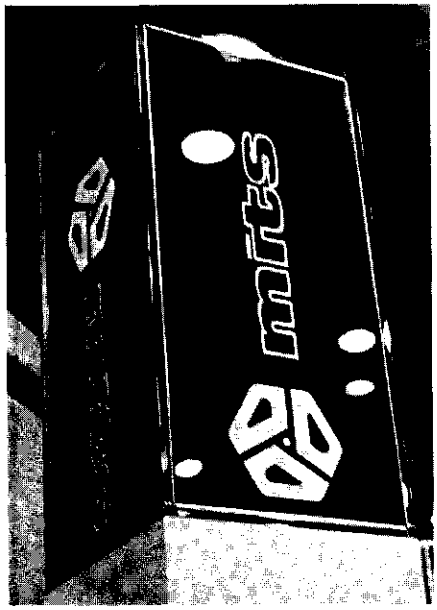




Grey Hodges, Altair Computer Center owner from Charlotte, North Carolina, explains how to load Altair Timesharing Disk BASIC.



Wall map displays location of 23 Altair Computer Centers.



MITS founder **Ed Roberts** (center) meets with MITS General Manager **Eddie Currie** (right) and MITS Vice-President of Engineering **Pat Godding.**

MITS software specialist **Chuck Vertrees** demonstrates the versatility of Altair Timesharing BASIC.

MITS Vice-President of Advertising **Andrea Lewis** watches a demonstration at a nearby booth.

MITS engineer **Ron Scales** demonstrates the use of Altair AD/DA boards on the Altair "bionic wall."

Computer Courses Offered

Beginning this fall, St. John's University, Jamaica, N.Y., is offering a series of intensive short courses on low-cost personal computing. Each course provides a well-rounded body of information on successful implementation and use of small computer systems. Information on both hardware and software design as well as numerous applications of personal computing in education, recreation, business, etc. is also included. No computer expertise is required.

The first course will meet Tuesdays, 6-8 p.m., September 27-October 18. The cost is $20. The following list is a partial course outline.

1. OVERVIEW OF PERSONAL COMPUTING: What is a personal computer (PC)? PC systems analysis.
2. DESIGN OF MICROCOMPUTER-BASED SYSTEMS: General and functional systems design, hardware design (interfacing CPU, memory, peripherals, etc.) software development (assembly and high-level languages).
3. IMPLEMENTATION OF PC SYSTEMS: How to acquire PC hardware and software, how to develop PC applications, analysis of the "buy or make" decision.
4. PC APPLICATIONS: Microcomputer-based information and business systems, personal man-computer problem-solving and decision-making systems, a low-cost PC-based investment decision system, and the PC as the user's intelligent problem-solving assistant.
5. THE PC INDUSTRY: Effect of the PC revolution on the computer and information industry, new business opportunities in personal computing.
6. LOOK INTO THE FUTURE

For further information contact:
Dean Patrick Basilice
Evening Division
St. John's University
Jamaica, NY 11439
(212/969-8000, Ext. 101)

Altair 680b Requires Phase 2 Clock Mod

By Ron Scales

The Altair 680b main board supplies 02 clock to the system bus through CMOS 4050 buffer. This buffer allows multiple board operation without overloading the MPU 02 clock. It has a propagation delay of approximately 100ns on the high to low transition, which is the clocking edge of 02. This causes only one problem in the Altair 680b.

When using a Turnkey 680b with a parallel port of the Universal I/O card, control of the 6820 PIA is impossible because of the falling edge of the 02 clock signals on the Data lines. The regular front panel provides enough load to delay the address and data signals for proper timing relationships. But the Turnkey front panel has no effect upon the address and data lines, so the falling edge of the 02 clock occurs too late relative to the Data bus signals.

To fix this problem, cut pin 4 of I.C. PP where it goes into the PC board. Then lift it up out of the way and jumper the pad 4 to pad 5 on the PC board. This connects the MPU 02 to the bus and removes any delay caused by the CMOS 4050 IC. This modification works with all versions of the Altair 680b computer and all combinations of plug-in PC boards for the 680b.

Correction

FOR SALE:

Altair 8800

Unused, fully assembled and factory tested. $450 or best offer. Call Dave (evenings): (515)-279-3683.

P-7440 Programmer

Unused, fully assembled. Contact: Steve Jacobus, 5 Ruppell's Road, Clinton, NJ 08809

Altair 680b

Partially built, $395.

TV Terminal III

Includes computer and manual cursor, screen read, UART serial interface, 2K memory, power supply, ASCII keyboard, audio cassette inte-face and enclosure. $300.

CONTINUED

Cartrivision Color VTR

Includes cabinet and removable test rack. $325.

Video Monitor

$75

Console

Heath Kit Station Console Model SB-I. $70.

Altair 8800

Barely used, 8K memory, serial I/O, cassette interface, includes documentation. $1395.

Contact: Norm Nason, 8608 Lvueline Ave., Canoga Park, CA 91306, (213)-341-1275

Classified Ads

The following lines should be added to the program "Circuit Analysis Applications Expanded to Run with Altair BASIC" (p. 14 + June, CN). These corrections were provided by Doug Jones, 2271 North Mill North East, Pa. 16428.

```
5Ø1 ON TT GOTO 52Ø, 51Ø,
54Ø, 56Ø, 146Ø IF PR AND
1Ø24 THEN GOSUB 255Ø,
214Ø PRINT "GM. BRANCH";
CL(I)+1; "TO BRANCH;
RW(I)+1: GOTO 217Ø
```

As a result of the above modifications, the output of the sample program will be slightly different. The "Equivalent Current Vector" (p. 21) will be the second item of output.

There will also be an extra line of output under "Branch Currents" (p. 20).

```
Branches 5-6  4.53E-Ø3
              4.446E-Ø3
```

The last item in the "Branch Currents" line 1-4 will be 7.34E-Ø5.

With these corrections the program will run on an Altair 8800 a or b. However, the following additional changes must be made for the program to operate properly on an Altair 680.

Using the Text Editor, remove all constants. These must be replaced with their numeric equivalents.

Example: Line 8Ø K3=3:
K4=4: K5=5 etc.

Replace K3 with 3, K4 with 4 etc. These changes must be made throughout the entire program, with the exception of line 6Ø; it is part of a string and should not be altered.

Settle for Less

—Less waiting in line for that one dogeared copy of **Computer Notes** that has been passed around until it is ready to fall apart.
—Less anxiety about missing an issue and having to depend upon the charity of others.
—Less money. It costs you less to subscribe.

Settle for fewer hassles all the way around by having **CN** delivered flat, protected by an envelope, directly to your mailbox every month.

computer notes

mits
2450 Alamo S.E.
Albuquerque, New Mexico 87106

Please send me a 1 year subscription to **Computer Notes.**
$5.00 per year in U.S. $20.00 per year overseas.

NAME:_____

ADDRESS:_____

CITY:_____ STATE:_____ ZIP_____

COMPANY/ORGANIZATION_____

☐ Check Enclosed
☐ Master Charge
☐ BankAmericard/Visa

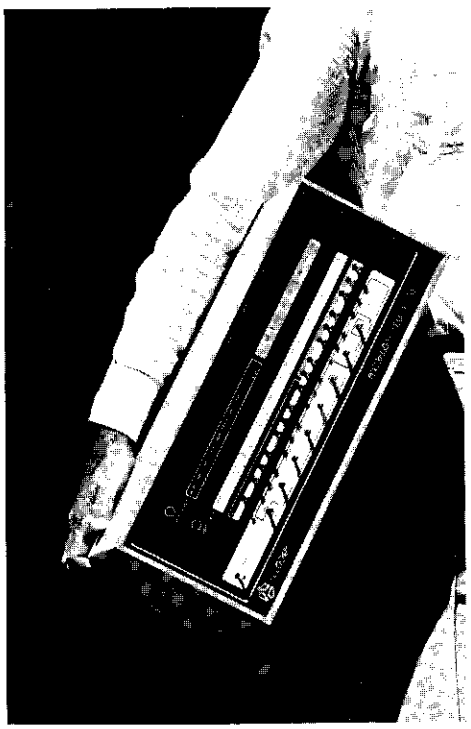
MC or BAC/Visa #_____

Signature_____

Computer Notes is published in Albuquerque by **MITS, Inc.**, the altair™ people.

Computing has come a long way in the past three years. Computer systems, which previously were only attainable on a time-sharing or rental basis, have now become accessible to the general public. Innovative technology and proficient design are just some of the factors that spawned the growth of personal computers. The Altair™ 8800 microcomputer from MITS, Inc., was the initial result of these developments and the pacesetter of this new industry.

Altair Systems may be adapted for many diverse uses. We can assist and support your hardware/software needs in any aspect of computer operation. Altair microcomputers have been utilized in research, educational programs, process and dedicated control as well as many original, user-generated applications.



Altair computer mainframes start as low as $395. Memory and interface capabilities are expandable through the use of Altair plug-in modules. Some of our recent additions permit process control, inexpensive mass storage and analog to digital conversion.

Even if you have never had any programming experience, Altair BASIC language can easily be learned and implemented. Within a short period of time, you will be solving complex problems without difficulty.

Altair microcomputer systems are readily available from any one of our nationwide dealers or through factory mail-order. Further information may be obtained by consulting your local Altair Computer Center or contacting us directly.

mits

The Personal Approach to Low cost Computing

MITS, Inc. 2450 Alamo, S.E. Albuquerque, N.M. 87106