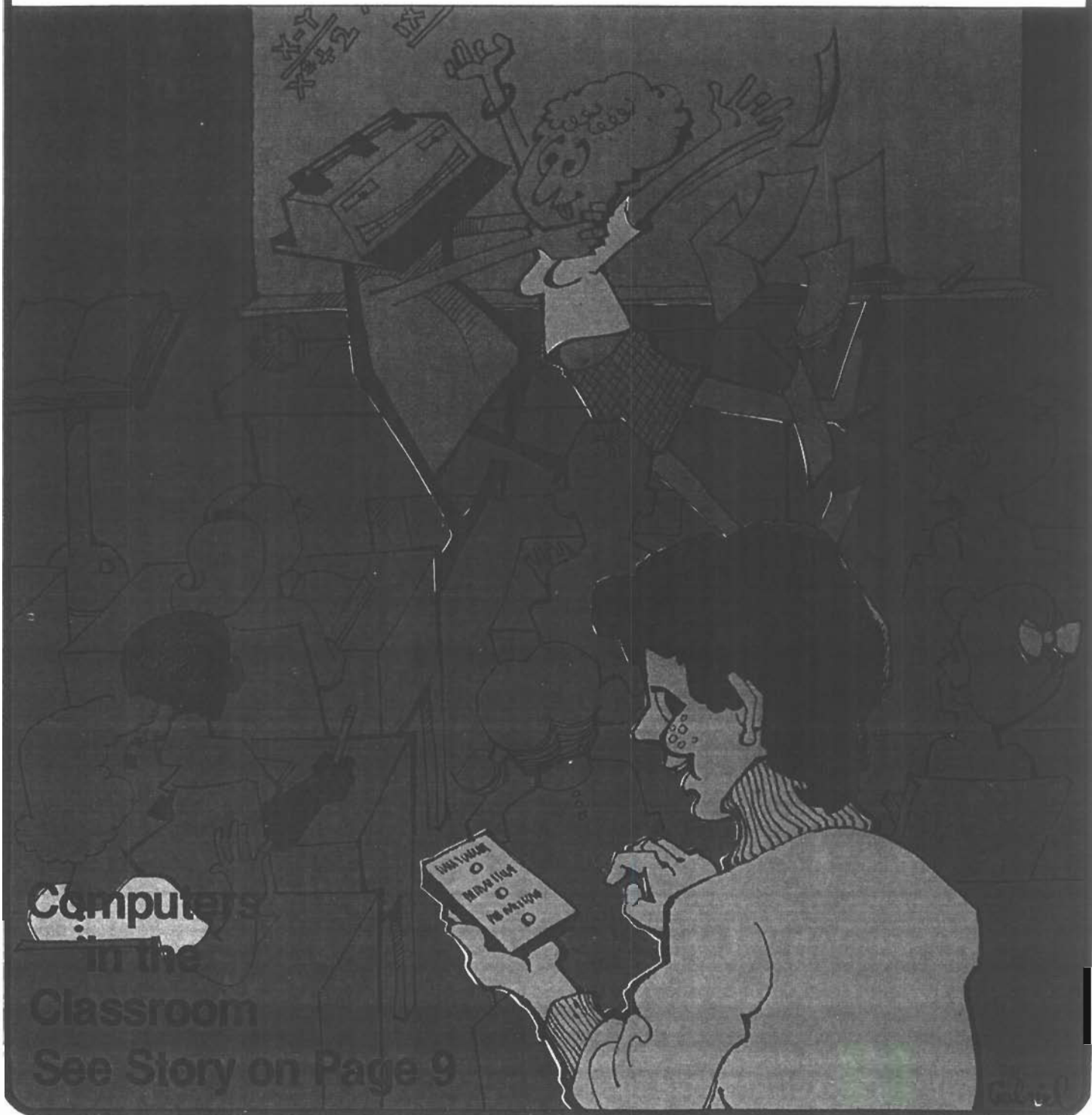


# computer notes

50¢

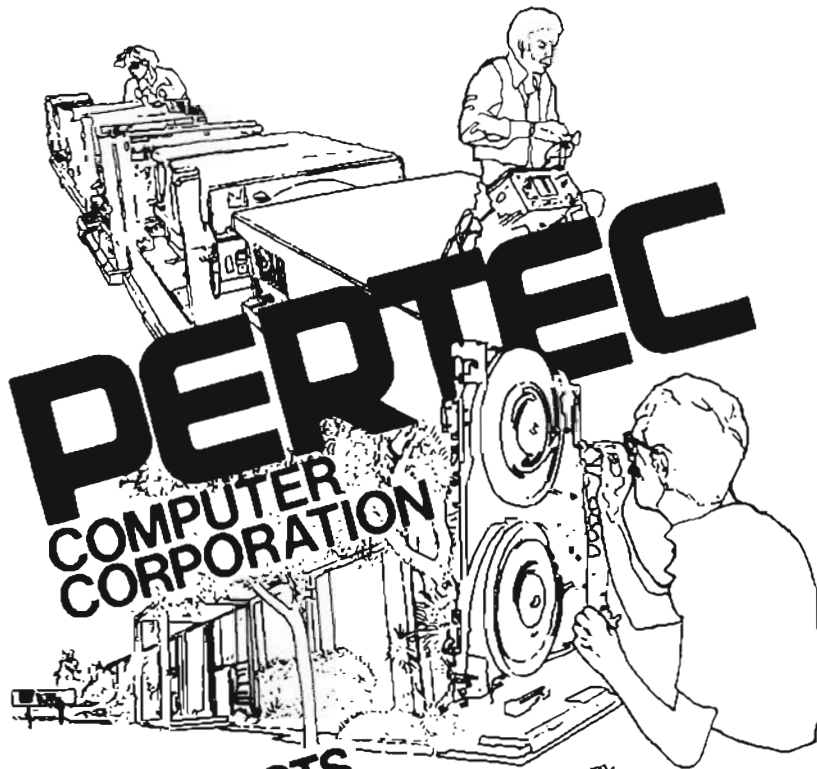
Jan/  
Feb  
'78

Volume 3 Issue 7



Computers  
in the  
Classroom

See Story on Page 9



## PEOPLE & PRODUCTS TO MEET EVERY CHALLENGE

As the world turns to computers, Pertec Computer Corporation turns to the expertise and imagination of the career professionals in the vital field of peripheral servicing - Pertec does it all, and in 10 short years, gaining the success of over \$100 million in annual sales.

Conveniently located in the West San Fernando Valley, our dynamic division continually seeks new talent to help us meet the computer demands of tomorrow. And the career fields we offer are as limitless as our challenges, including opportunities in Engineering, Manufacturing, Management, Administration, Data Processing, Business Development and Finance.

If you're looking for Pertec's kind of challenge, tell us why - write or mail your resume today to:

9610 DeSoto Avenue  
Chatsworth, California 91311

*We are an Affirmative Action Employer M/F/H*



Computer Notes is published bi-monthly by PERTEC COMPUTER CORPORATION. A free year's subscription is included with every purchase of a MITS®/Altair™ computer. Regular subscriptions can be ordered from PERTEC COMPUTER CORPORATION FOR \$2.50 per year (\$5.00 for two years) in North America and \$10 per year for overseas. Single copies are available for 50¢ each at all MITS COMPUTER CENTERS. Entire contents copyright, ©1978, PERTEC COMPUTER CORPORATION. Send articles, questions, comments, and suggestions to Editor, Computer Notes, PERTEC COMPUTER CORPORATION.

©1978 PERTEC COMPUTER CORPORATION  
20630 Nordhoff Street  
Chatsworth, CA 91311  
(213) 998-1800  
Editor: Marsha Sutton  
Volume 3, Issue 7, January/February 1978

Products or services, other than those of PERTEC COMPUTER CORPORATION, advertised in Computer Notes magazine are not necessarily endorsed by PERTEC COMPUTER CORPORATION, thereby relieving itself of all legal responsibility.

# In This Issue

System Timing Modification for the MITS/Altair 88-DCDD Floppy Disk	4
by Tom Durston	
A. Copy/Rewrite Procedure	4
by Gale Schonfeld	
B. Single Drive BASIC Diskette Rewrite Procedure	5
by Charles Verrees	
C. Single Drive DOS Diskette Rewrite Procedure	6
by Drew Einhorn	
D. Easy Floppy Disk Alignment Check	6
by Tom Durston	
More on the KCACR	7
by Doug Jones	
Union County Career Center, Update 1977	9
by James Gupton, Jr.	
MITS/Altair CPU Modification	11
by Darrel Van Buer	
Demonstration Program	14
by Ken Knecht	
A BASIC Memory Test	15
by Dave Culbertson	
FDOS-III: The Latest from Perlec Computer Corporation	16
Tic Tac Toe Modification	17
by John Trautschold	
Practical Programming, Part II	18
by Gary Runyan	
Modifying MITS BASIC for ASCII I/O	20
by John Palmer	
MITS Newest Business System	21
Book Review	22
10,000 Visit MINI/MICRO '77	24
by Marsha Sutton	
Introducing the Compact Attached Computer	26
Machine Language to BASIC Converter	27
by Richard Ranger	

# Submittal Specifications

Articles submitted to **Computer Notes** must be typed, double-spaced, with the author's name, address, and the date in the upper left corner of each numbered page. Authors should also include a brief autobiographical statement about their job, professional title, and previous electronic and/or computer experience on a separate sheet of paper. 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 PMTs are acceptable. No pencil drawings, unless properly "fixed", and no halftone or wash drawings can be accepted.

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 labeled and referred to by number within the text of the manuscript.

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

Program listings should be recorded with the darkest ribbon possible on blank white paper. A paper tape for each program submitted must also be included.

# The Latest with CN

**Computer Notes** is changing.

The top news for this issue is to announce the completion of the eventful relocation of the **Computer Notes** editorial office. The office has been moved from MITS® in Albuquerque to PERTEC COMPUTER CORPORATION in California. And to occupy this newly located office is a completely new editorial staff — myself. My name is Marsha Sutton, and I am the new editor of **Computer Notes**. The address of the new office can be found on the inside front cover.

Several further changes will be forthcoming, of which you should take note. **Computer Notes** will continue to sell for 50¢ a copy, but it will be published every other month, rather than monthly. Subscriptions will now be \$2.50 a year and \$5.00 for two years. Those of you who are current subscribers will have your expiration date advanced to provide you with the number of issues for which you have paid (\$5.00 buys 12 issues).

To establish better channels of communication among the readers is one of **CN's** primary functions, and this can be accomplished by utilizing the magazine to relay interesting and pertinent information. Rest assured I will do my utmost in organizing and compiling **Computer Notes**; but to publish a magazine composed of quality material properly, I need your input...regularly! I encourage and appreciate any and all stories, photographs, suggestions, and letters that you can offer. Send anything you have to my attention at the new address; all articles will be warmly received. Please don't hesitate to contact me on any issue, including problems or questions — I love mail, and it's always nice to be reminded that people are really out there!

Please adhere to the submittal specifications given on the inside front cover, as it will make my job much easier. Also, for all articles submitted in the future, please be sure to enclose a brief history of yourself each time, even if you have contributed articles to the magazine in the past.

A new policy has been established regarding payment for articles for outside contributors. Authors will be paid approximately \$35 per page for articles accepted, but this pay rate is subject to change, depending upon the degree of technical content, accuracy, neatness, journalistic style, amount of editing required, and the regularity with which the author submits articles to the magazine. Also, equipment or other unusual means of payment are no longer negotiable items — payments will be made by check only.

That's about all for "The Latest". Again, I would like to re-emphasize the extent to which I depend upon the contributions of all of you **CN** readers to produce the magazine. It is essential that I hear from you with whatever you may have to say, particularly with quality articles that there is a demand to publish. And besides, where else could you get the thrill of seeing your name in lights (well, in print at least) before the proverbial public?!

Thanks for your past support, and I hope to be hearing from you soon.

Marsha Sutton  
Editor

P.S. I would like to express my special thanks to Tom Antreasian and Susan Blumenthal for helping me get my "bit" together in this — my first issue as Editor of **Computer Notes**.

# System Timing Modification for the MITS<sup>®</sup>/Altair<sup>™</sup> 88-DCDD Floppy Disk

By Tom Durston

To increase diskette interchangeability from drive to drive and to minimize disk I/O errors, two time constants on the 88-DCDD Controller Board #1 have been re-evaluated. The effect of this timing change is to center the data within the sector. This allows a greater tolerance of disk drive misalignment.

A diskette written with the new write delay should be marked "NWD" for identification purposes. All BASIC and DOS diskettes shipped from MITS<sup>®</sup> after August 31, 1977 are written with this new write delay and are marked "NWD". These diskettes are compatible with unmodified systems.

To utilize the new write delay, the Read Clear Timing must be changed as indicated later in this article. If a system does not require diskette interchange capabilities and if there has been no difficulty with disk I/O errors, the complete modification is not necessary. However, it is advised that the write delay be changed as described in step IIA. The modification is strongly recommended for multiple drive systems or single drive systems where diskette interchange is required.

A modification kit (MITS Part #103678) is available at no charge to owners of MITS/Altair<sup>™</sup> 88-DCDD Floppy Disk Systems. If an owner does not have the facilities for performing the modification, Controller Board #1 can be returned for complete modification at no charge. However, R5, the Read Clear one shot timing resistor, will not be replaced, but the correct resistor for R5 will be returned with the board and should be installed upon completion of re-writing or copying the diskettes, as indicated in step IIC of the modification procedure.

An important feature of the modification includes changing the timing IC to 74LS221. This was done because the 74LS221 is more stable and predictable than the 74123. It also eliminates the need for trimming or adjusting the timing resistors.

## I. PARTS REQUIRED (Included in the FDSK Modification Kit)

2 each	74LS221 IC	MITS Part #101466	(F1, F4)
1 each	6.65K 1% resistor	MITS Part #102225	(R5)
1 each	12.1K 1% resistor	MITS Part #102226	(R12)
1 each	4.32K 1% resistor	MITS Part #102227	(R11)
1 each	8.45K 1% resistor	MITS Part #102228	(R6)

## II. MODIFICATION PROCEDURE (Controller Board #1 Only)

### A. Change the Write Clear one shot timing from 280 $\mu$ s to 389 $\mu$ s.

1. Remove R11 and R12.
2. Install a 4.32K, 1% resistor in the R11 position, and a 12.1K, 1% resistor in the R12 position.
3. Remove IC F4, and install a 74LS221 in its place.

4. If available, use an oscilloscope to measure the positive pulse width at TP8 (IC F4, pin 5). This step is not mandatory, due to the timing predictability of 74LS221. The pulse width should be in the range of 355 $\mu$ s to 425 $\mu$ s (389 $\mu$ s NOM  $\pm$  10%) when the drive is enabled and a diskette is installed.

B. Copy all diskettes using the procedure listed in Article C that follows. If the Read Timing is not being changed, it is not necessary to copy the diskettes.

### C. Change the Read Clear one shot timing from 140 $\mu$ s to 214 $\mu$ s.

1. Remove R5 and R6.
2. Install a 6.65K, 1% resistor in the R5 position and an 8.45K, 1% resistor in the R6 position.
3. Remove IC F1, and install a 74LS221 in its place.

4. If available, use an oscilloscope to measure the positive pulse width at TP5 (IC F1, pin 13). This step

is not mandatory, due to the timing predictability of the 74LS221. The pulse width should be in the range of 195 $\mu$ s to 230 $\mu$ s (214 $\mu$ s NOM  $\pm$  10%) when the drive is enabled and a diskette is installed.

D. Change schematic notation to coincide with the modification.

For step 3 in parts A and C, if ICs F1 and F4 are not socketed, remove the soldered ICs by cutting all the pins. Carefully remove each pin one by one. Clean the holes by using solder wick or a solder removing tool. Do not remove the plated portion of the hole. When soldering the new ICs in place, solder each pin on both sides of the PC board to ensure proper feed-through connection.

## A. Copy/Rewrite Procedure

By Gale Schonfeld

The following procedures are recommended for copying disk software with the new disk Read/Write modification using a multiple drive system.

## B. Single Drive BASIC Diskette Rewrite Procedure

By Charles W. Vertrees

**CAUTION:** All disk software copying should be done **AFTER** the Write modification has been made but **BEFORE** the Read modification is made.

### METHOD I — Using Disk BASIC "PIP" Utility Program.

If the user has Disk BASIC, versions 3.3, 3.4, 4.0, or 4.1, use the PIP utility program provided on the system diskette to copy onto a new diskette. A PIP program listing, and instructions on its use, are included at the end of this article.

**STEP 1:** Load Disk BASIC. Initialize the system for at least two disk drives (i.e., HIGHEST DISK NUMBER should be answered with 1 or higher).

**STEP 2:** MOUNT the diskette with BASIC and PIP on it. Do not attempt to MOUNT a diskette that is new and has never had BASIC or files on it.

**STEP 3:** LOAD PIP and type RUN.

**STEP 4:** Use the PIP Copy command to copy the old diskette (with BASIC and the files) onto the new diskette. COP will take approximately 30 minutes.

**STEP 5:** Check the new diskette by re-loading BASIC (from the new diskette), by MOUNTing, and by printing a directory of files. This will confirm that everything was copied satisfactorily.

**STEP 6:** Make the disk Read modification.

### METHOD II — Using Disk BASIC "PIP" and DOS.

If the user has Disk BASIC and DOS (Disk Operating System), Disk BASIC and PIP can be used to copy the DOS diskette. Follow the procedure described in Method I, noting the following exceptions:

**STEP 3:** LOAD PIP, but UNLOAD the diskette with BASIC on it before RUNning PIP. Place the DOS diskette in the drive where BASIC was previously located. It is not necessary to MOUNT to copy with PIP. RUN PIP, and proceed with STEP 4 of Method I.

**STEP 5:** Check the new diskette by loading DOS, by MOUNTing, and by issuing a directory command. If possible, run several of the programs, and proceed with STEP 6 of Method I.

The following program illustrates how to copy a diskette onto itself by changing the write delay timing with which each sector of the diskette is written. The program is necessary in order to take advantage of the changes to the read and write time delays that are being made on the MITS/Altair 88-DCDD Disk Controller cards. Together, the program and hardware changes will alter the physical position within a sector of a diskette from which the data is written and read.

This program works by buffering an entire track of data at a time. This is done by allocating the string array A\$ with one element for each sector on a track. The data on a specific track is then read into this array and verified by re-reading each sector to ensure that it was read correctly the first time. If for some reason the data for a given sector will not verify, the sector will read into the array again and then re-read a second time to verify. This process is repeated until verification occurs. Once an entire track has been read and verified, the data is then written back onto the same physical track of the diskette. To ensure that the entire operation is done correctly, the new written data is then re-read and compared against the original data. Again, if a specific sector will not verify, it is re-written from the original data and re-read to verify the write. This process will continue until all re-written data on the track is verified.

```
100 CLEAR 137*34
110 PRINT:PRINT"DISK SELF COPY"
120 ' GET TO TRACK ZERO
130 OUTB,0
140 IF (INP(8)AND 64) <> 0 THEN WAITB,2,2 OUT9,2:GOTO140
150 ' DO IT FOR ALL 77 TRACKS
160 FORT=0T076
170 PRINT:PRINT"READ T":T
180 DIM A$(31)
190 FOR S=0 TO 31 ' READ & COMPARE ALL SECTORS
200 A$(S)=DSKI$(S)
210 B$=DSKI$(S)
220 IF B$ <> A$(S) THEN PRINT"REREAD T":T,"S":S:GOTO 200
230 NEXT S
240 PRINT:PRINT"WRITE T",T
250 FOR S=0 TO 31 ' WRITE NEW TRACK
260 DSK0A$(S),S
270 NEXT S
280 FOR S=0 TO 31 ' CHECK NEW DATA
290 B$=DSKI$(S)
300 IF A$(S)<>B$ THEN PRINT"REWRITE T":T,"S",S:DSK0A$(S),S:GOTO 290
310 NEXT S
320 ' GOTO NEW TRACK
330 ERASE A$
340 IF T=76 THEN 360
350 WAIT B,2,2 OUT 9,1
360 NEXT T
370 CLEAR 200
380 PRINT:PRINT"THAT SHOULD DO IT"
390 END
```

The program should work without encountering many REREAD or REWRITE errors if the disk drive is in correct operating condition and if there is nothing wrong with the diskette. If a large number of these errors are encountered, this usually indicates that there is something physically wrong with the drive (alignment, transport, etc.) or with the diskette.

To use this program, first make the modifications to the write time delay circuit on the controller boards. Then bring up BASIC and enter this program, which can be saved on the diskette. The program must now be run on all diskettes on which programs or data that may be needed for future reference currently exist. Next, make the modifications to the read time delay circuitry on the controller boards. This entire procedure should greatly reduce the frequency of disk I/O errors due to drive alignment problems.

**NOTE:** This program takes about 30 minutes to run. It can run faster by increasing the amount of string space cleared in line 100. Currently, 4658 (137\*34) bytes, the minimum amount required, are cleared. This should be changed to clear as much string space as memory will allow after loading the program. Make sure the diskette is up to speed before RUN is typed.

## C. Single Drive DOS Diskette Rewrite Procedure

By Drew Elhorn

A program which runs under DOS using only a single floppy disk drive allows an update of the Write Timing of the diskettes. This is now available free of charge to those who have purchased a copy of DOS prior to December 1, 1977. Send a copy of the invoice or a proof of purchase of DOS to MITS, and request the DOS Rewrite Diskette.

In order to update the Write Timing on the diskettes, perform the following procedure. This procedure assumes only one disk drive is available.

STEP 1: Perform the modifications to the Write circuits of the Disk Controller (reference to stop number IIA or hardware modification).

STEP 2: Put the old DOS diskette in Drive 0. Bootstrap, and perform initialization as usual. Do not MNT it.

STEP 3: Remove the old DOS diskette from Drive 0.

STEP 4: Place the diskette containing Write Time Delay update program in Drive 0.

STEP 5: Issue the command MNT 0.

STEP 6: Run the Write Time Delay program by typing TIMING in response to the "." PROMPT. If there is more than one drive and if the diskette is in a drive other than 0, the command is RUN TIMING n, where n is the drive number.

STEP 7: The program will type out CHANGE WRITE TIME DELAY ENTER DEVICE NBR. Type 0, and do not hit RETURN.

STEP 8: Remove the diskette from drive 0, and place the diskette to be re-written in drive 0.

STEP 9: Hit RETURN. The program will begin executing. It will first DSM the diskette and then go around a loop 77 times, once for each track into memory. The entire track will then be compared with the contents of memory with the diskette. Any sector which does not compare will be re-read and re-compared, until they match. The entire track will be re-written with the new Write Time Delays and will then be compared with memory. Any sector that does not compare will be re-written and re-compared. When this process is completed,

the program will go to the next track. When the last track is finished, the diskette is MNTed. It takes approximately 3 minutes.

STEP 10: If there is more than one diskette to update, perform a DSM 0 command, and go to step 4.

STEP 11: Perform the modifications to the Read Circuits of the Disk Controller.

## D. Easy Floppy Disk Alignment Check

By Tom Durston

The following procedure simplifies the Index sensor alignment check on the floppy disk drives by using signals obtained on Controller Board #1. This eliminates the need for disassembling the drive chassis. The procedure is based on using Read Clear (TP-5) as a reference signal and on observing Serial Read Data going into IC G1, pin 1 or 2.

This method allows an easy check of the relative sector alignment between data written on the diskette and the drive alignment. If necessary, this method may be used to misalign the drive to match the misalignment on the diskette, allowing reading of data.

Note that this procedure only shows Index sensor and Stepper skew alignment and does not show Track Offset alignment (Cats' Eye Pattern). For a full drive alignment check and adjustment, the procedure listed in the 88-DCDD manual should be used. Only the Index sensor should be adjusted using the procedures listed here.

Shown here are two procedures for checking drive or diskette alignment. For easy control of the head position, the Disk Exercisor Program listed on page 118 of the 88-DCDD manual is recommended. A dual trace oscilloscope is required for these tests.

### 1. INDEX SENSOR ALIGNMENT CHECK

- Connect scope channel 1 probe to TP-5 (F1-13) Read Clear. Sensitivity = 2v/Div.
- Connect scope channel 2 Probe to IC G1, pin 1 or 2; Serial Read Data. Sensitivity = 2v/Div.
- Set sync to channel 1, positive edge trigger.
- Display channel 2 only.
- Set time base to 50 $\mu$ s or 20 $\mu$ s per Div.
- Run Exercisor program, insert alignment diskette, and seek tracks with Index BURST.

Observe the 40 $\mu$ s low pulse representing the Index BURST. This low pulse is typically 4 $\mu$ s slower than the actual Index BURST seen at the Read amplifier in the drive. If the low pulse is not seen, the drive is probably severely misaligned. Consult the 88-DCDD manual for drive alignment instructions, beginning on page 116.

### 2. RELATIVE ALIGNMENT CHECK

This procedure may be used to check alignment between a drive and a diskette with data on it. If a diskette is giving I/O errors due to drive misalignment when it was written, the problem can be eliminated by temporarily misaligning the drive to position the data correctly.

- Connect scope channel 1 Probe to TP-5 (F1-13), Read Clear. Sensitivity = 2v/div.
- Connect scope channel 2 Probe to IC G1, pin 1 or 2, Serial Read Data. Sensitivity = 2v/div.
- Set sync to channel 1, positive edge trigger.
- Display both channels.
- Set time base to 50  $\mu$ s/Div.
- Run Exercisor program, insert diskette to be checked, and seek 0 and 76.

Channel 1 should show the Read Clear pulse (140 $\mu$ s old, 214 $\mu$ s new), which indicates the length of time the Read circuit is turned off. When Read Clear is low, it allows the Read circuit to start searching for the Sync Bit, the first logic 1 in the data field.

Channel 2 should show the Serial Read Data. Normally, it consists of several logic 1 pulses 50 to 100 $\mu$ s after the beginning of the Sector. The data

# More on the KCACR

By Doug Jones

field starts with the Sync Bit 250 to 350 $\mu$ s (old timing) or 350 to 500 $\mu$ s (new timing) after the beginning of the sector. The logic 1 pulses after the beginning of the sector are caused by the noise written by the Write circuit being turned on when that sector was written. There should be a long period (250-400 $\mu$ s) of all logic 0 from the noise pulses to the Sync Bit.

For optimum timing, Read Clear should go low halfway between the noise pulses and the Sync Bit. The Read Circuit will generate errors if the noise pulses occur after Read Clear goes low or if the Sync Bit and Data occur before Read Clear goes low.

If necessary, the Index Sensor may be temporarily adjusted to allow proper reading of a diskette by centering the low time of Serial Read Data as described earlier. Note the original position of the Data, so the Index Sensor may be returned to normal. Check both inner and outer tracks of the diskette in order to compensate for skew in the data.

*Program on page 27*

## About the Authors

*Tom Durston is the MITS Engineering Program Director and is involved primarily with peripheral interface design. A MITS employee for five years, Durston studied Electrical Engineering at the University of Virginia and the University of New Mexico.*

*Gale Schonfeld has been employed by MITS for two years and is the Software User Specialist. She is currently pursuing a Bachelor of Science degree at the University of New Mexico in Electrical Engineering/Computer Science.*

*Chuck Vertrees is the Director of Software for MITS. He has a B.S. in Electrical Engineering from the University of New Mexico and is currently studying for a Masters in Computer Science.*

*Drew Einhorn holds a degree in Mathematics from the University of Oklahoma. He has been employed by MITS for two years as a scientific programmer.*

The announcement of the new MITS<sup>®</sup> KCACR board (Kansas City Audio Cassette Recording) for their MITS/Altair<sup>™</sup> microprocessor was indeed a welcome relief for me and for a still ailing papertape reader. With the installation of this single board, a world full of holes and spilled chad has turned into neat little plastic boxes each with a cassette tape. The chaos of rattle-rattle-checksum error has turned into absolute quiet, broken only occasionally by an eject-click.

Regarding the hardware, the board occupies one slot of the 680 expander board. Its features include CMOS logic for low power consumption, and it uses total digital logic without a single potentiometer or adjustment. The input/output is at 300 baud, allowing a speed tolerance of 20%.

The software that is supplied with the KCACR is, likewise, quite good. MITS' CSAVE BASIC is supplied on an audio cassette tape and its features still amaze me. A bootstrap loader PROM chip that fits into one of the PROM sockets on the main board is also supplied. Since this chip has no name, I will refer to it as the KCACR MONITOR. A large portion of this article will concentrate on this chip.

Since there are many things to discuss about the KCACR and related software, I have organized this article into four sections, all intending to help you gain the most from the hardware and software of the KCACR.

This writing will appear at times to be a collage of software tidbits that have appeared over the last year in *Computer Notes*, I would like to give credit where it is due. My thanks to Mark Chamberlin (I literally stole his PUNBAS routine) and to Ron Scales for his help on a rather sticky interrupt problem.

### I. Inverse Assembly of the KCACR MONITOR

After putting this new PROM chip on my 680 processor board, it was nice to see its two primary functions work well. A (J)ump to FD00 will allow a load of a Motorola-formatted audio cassette tape through the new port, and a (J)ump to FD74 will allow a properly-formatted dump of any selected portion of memory. And it really works quite well.

But curiosity started to get the better of me. Exactly how does it work, I asked myself. Are there any useful subroutines in it that can be called by other programs? Are there any provisions for turning off the motor on a checksum error? I wanted to know the answers to these and other questions.

I ran a 680 Inverse Assembly ("Inverse Assembler Makes Machine Language Programs Understandable", by Doug Jones; *Computer Notes*, July 1977) on it and produced the listing that is shown. The comment, labels, and a bit of doctoring-up was done using the EDITOR.

I received answers to my initial three questions and they were "well", "yes", and "no". It may not turn off the tapedeck motor on a checksum error, but there are some useful routines in it that are easily called from an assembly language program. If you spend a few minutes and study the KCACR MONITOR program, perhaps you will spot some useful subroutines or learn a new programming technique, such as the following question illustrates about the KCACR MONITOR. The problem is, "If BADDR (address \$FD59) is a subroutine that required a JSR to enter, how do you exit?"

### II. Comparing the KCACR MONITOR to the 680 MONITOR

Table 1 compares the addresses of the major subroutines of both MONITOR programs, and, interestingly enough, both sets of subroutines function identically except that they address different ports. For example, you wish to send a letter to the teletype port

```
C6 XX LDA B #'(letter)
```

```
BD FF81 JSR OUTCH
```

;680 PROM MONITOR address.

On the other hand, you wish to send a letter to the KCACR

```
C6 XX LDA B #'(letter)
```

```
BD FDF5 JSR OUTCH
```

;KCACR MONITOR address.

The 680 PROM MONITOR manual will give you register usage on all of the other subroutines mentioned in Table 1. Beware, for there are some hidden "GOTCHAs", at least they always seem to get me. A call to INCH does not return an 8-bit character; parity has been stripped off of it...will I ever learn?

### III. Preparing Your Other Software for Use with the KCACR

You may wonder why there is a need for converting your original BASIC or EDITOR to KCACR. CSAVE BASIC is good; as a matter of fact, I use it 99 percent of the time. But, naturally, my favorite demonstration program needs the extra few hundred bytes that the original BASIC has in usable memory. Also, since I am generally trying to emerge from the papertape world, the EDITOR and the EDITOR/ASSEMBLY are naturals to convert to KCACR format.

The PUNBAS ("680 Software News", by Mark Chamberlin; Computer Notes, November 1976) program was easily converted to a PUNKCR program for these purposes. Let me warn you of several sticky areas.

#### BASIC

V1.0 R3.2 9/25/76

Load in the PUNKCR program.

Load BASIC, but do not initialize.

Make patches to BASIC's CONT statement (see "680 Software News" article).

Dump BASIC to cassette by doing a (J) 4000.

#### Later Revisions

Check last load line of BASIC tape for address of last byte.

Load PUNKCR program.

Load BASIC, but do not initialize.

Adjust LDX statement at \$4000 for last byte address.

Dump BASIC to cassette by doing a (J) 4000.

#### EDITOR

R1.0 9/30/76

Load PUNKCR program.

Load EDITOR program, but do not initialize.

Dump EDITOR to cassette by doing a (J) 4005.

#### EDITOR/ASSEMBLER

R1.0 9/30/76

Load PUNKCR program.

Load EDITOR/ASSEMBLER, but do not initialize.

Make patch correction ("Software Tidbits", by Mark Chamberlin; Computer Notes, April 1977) to EOR statement by depositing an \$88 at address \$03A7.

Dump EDITOR/ASSEMBLER to cassette by doing a (J) 400A.

I suggest using fifteen-minute per side tapes. I also suggest following MITS' advice to dump the same thing to both sides of the tape to save rewind wear and tear. These are your big programs, so you will need to buy three tapes.

ROUTINE	680 MONITOR	KCACR MONITOR
BADDR	FF62	FD59
BYTE	FF53	FD4B
INCH	FF00	FD62
INHEX	FF0F	FD36
OUT2H	FF6D	FDE3
OUTCH	FF81	FDF5
POLCAT	FF24	-no equivalent-

Table 1. Subroutine Address Comparison

### IV. Techniques of Using Other Software with the KCACR

Table 2 shows a cross-reference of INCH and OUTCH subroutine calls and their respective addresses in both MONITOR programs. For example, you have finished a long session with the EDITOR, and you wish to store your buffer on cassette for future use.

Exit EDITOR with X\$\$

Set nulls .M 00CE 00 10

Adjust OUTCH call .M 01EE FF FD

.N 01EF 81 F5

Return EDITOR .J 010A

After this point, you will not be able to see echo, since it is being sent to the KCACR port.

To dump, type FFEF\$\$

At some later date, you may wish to reload this tape into, for example, the EDITOR/ASSEMBLER.

Initialize E/A by

doing .J 0107

Exit the EDITOR with X\$\$

Adjust INCH call .M 0184 FF FD

.N 0185 00 62

Jump directly to APPEND function .J 19FF.

Your load is completed when the terminal begins to rattle in response to some impulses on the KCACR. Next, hit the computer RESET, and readjust the INCH call .M 0184 FF FF

.N 0185 62 00

Return to EDITOR .J 010A.

Using the full editing features, check the first and the last few lines of your buffer. It is likely that the first line will be FFEF\$\$, which can easily be killed.

Table 2 also shows the PEEK and POKE cross-references for the same subroutine calls. For example, if you are in BASIC and if you wish to send some of your PRINT statements to the KCACR, do a POKE 2222,253: POKE 2223,245. To return the print to the teletype port, do a POKE 2222,255: POKE 2223,129.

Be alerted that the POKE/PEEK addresses shown here are for the subroutine addresses; the JSR command (\$BD) is found one address prior to those. For example, if you wanted to NOP the OUTCH call out of CSAVE BASIC, you would have to NOP three consecutive addresses beginning at \$08BB.

The technique in BASIC of alternately writing to the teletype, the KCACR, or NOPping the OUTCH call (writing to the bit bucket) might prove a useful technique for debugging a program.

In summary, many ideas have been presented in this article, some of which are good and some you may consider not so good. I hope you will be able to improve on both. But, no doubt about it, MITS has a good product with the KCACR board.

Program on page 29

	BASIC	CSAVE BASIC
INCH	\$0420 P1056	\$042E P1070
OUTCH	\$08AE P2222	\$08BC P2236
POLCAT	\$061C P1564	\$0627 P1575
	EDITOR	EDITOR/ASSEMBLER
INCH	\$0169	\$0184
OUTCH	\$01EE	\$022E
APPEND	.J 0689	.J 19FF
	680 MONITOR	KCACR MONITOR
INCH	\$FF00 P255,000	\$FD62 P253,098
OUTCH	\$FF81 P255,129	\$FDF5 P253,245
POLCAT	\$FF24 P255,036	--no equivalent--

Table 2. INCH and OUTCH Cross Reference



# Union County Career Center, Update 1977

By James Gupton, Jr.

During 1977, *Computer Notes* carried two articles dealing with the high school students in Union County, North Carolina and their experiences in the construction of a MITS®/Altair™ 680B microprocessor computer. To the MITS/Altair computer owner who receives *Computer Notes*, there must be some questions as to why high school students should receive such prominence. The answer is simple. The Union County Career Center is the first vocational education center in North Carolina to offer an adult level curriculum with computer construction and programming. Most educational institutions that include computer operation in their programs have relied on "breadboarded" computer circuits of limited program capabilities, whereas the MITS/Altair system was far better suited for the tone of the Union County Career Center's Electronics program. This article will center on the student assembly of the 680B-BSM 16K RAM circuit board for adapting the 680B to BASIC language programming.

There are no words to explain the feeling of apprehension when entrusting almost \$700 in circuit components to the semi-skilled hands of high school students. It is similar to the feeling a father has the first time his son asks for the family car! As a teacher of electronics, I try to develop manual skills in my students by having them solder printed circuit boards, and these young adults usually prove themselves to be quite capable, if only given the chance. So I entrusted my students with the assembly of the 680-BSM memory expansion circuit board.

The students needed to have the proper tools for such delicate soldering tasks. Panavice, Inc., by way of Bert McCabe, their Executive Vice-President, generously donated the original "Panavice" with the Model 315 circuit board holder. This provided an ideal method of holding the large circuit board of the 680-BSM memory circuit. The Ungar Princess soldering iron with a gold-plated micro-precision solder tip proved to be well worthwhile in soldering the infinite number of integrated circuit socket pins of the RAM circuits. Even so, a magnifying glass was diligently used to examine each IC socket for solder bridges and cold solder joints.

Upon the completion of the assembly of the 680-BSM circuit board, the students and I discovered a fault that seems universal in the MPU computer field. The engineers that write the assembly manuals assume that the assembler of the product possesses a certain level of knowledge in computer technology. Let me illustrate with the 680-B I/O port. The 680-B assembly manual provides the assembler with a variety of I/O port choices but offers nothing to aid in the selection of the proper I/O port. The 680-B assembly manual states connector finger-test points but does nothing to identify which side or what end is considered a starting point for pin counting.

As we at Union County Career Center soon discovered, the 680-B MPU computer alone is worthless, unless one memorizes the entire ASCII binary code. Programming instructions require alphanumeric designations that cannot be made without an appropriate keyboard, so we procured an inexpensive Radio Shack keyboard and associated components, only to learn that we must have a parallel interface to use it. We stipulated a cassette BASIC program in hopes that

there would be no need for an additional interface. But, to our dismay, in order to use the cassette program, we discovered that our 680-B must also have the 680-B KCACR interface. Should we wish to use the Assembly language and editor or the Editor programs included with the 680-BSM, we must then purchase a tape reader and a parallel interface circuit board. If not, these programs will be useless to our 680-B system. Presently, our 680-B computer just sits there, doing absolutely nothing!

The important point is that we have learned, the hard way, what is involved in getting into microprocessor-based computer programming. This has been a valuable lesson for both student and teacher. We have learned that more is required than the basic computer unit to run a program. We now know that an ASCII keyboard and its interface, as well as a CRT terminal or printer, are essential. We also learned that the KCACR interface is needed to record any programs on tape, and, primarily, we have learned that there is no substitute for quality in circuit boards and components, by having Mike Jones at Computer Stores



*Kevin Stewart solders RAM IC sockets to 680-BSM circuit board.*

of the Carolinas test out the workability of our computer with his peripherals in the store. We now know that the student assembly of the 680-B and the 680-B-BSM was 100 percent—well, 99 percent (due to one bent lead of an IC) perfect!

As the Electronics Teacher at Union County Career Center, I must confess to earlier doubts of my students' capabilities. They deserve a great deal of credit for operating the computer as well as they did. We also thank Mike Jones for his presentation to the NCAEDS (North Carolina Association Educational Data Systems), in which the MITS project at Union County Career Center was mentioned and where the Reprints of *Computer Notes* on Union County activities were so well received. This level of microprocessor-based computer activity has previously been directed exclusively to

junior colleges and technical education centers and is most unique at the high school level.

There are but two more stops before we at Union County Career Center can program our 680-B MPU computer, and they are the acquisition of a KCACR Interface and a terminal. These will be the subjects of future articles in *Computer Notes*. Perhaps the obstacles we encounter and overcome will serve as guides to the upcoming generation of home computer novices.

### About The Author

*James Gupton is a free-lance writer and an electronics teacher at the Union County Career Center in North Carolina.*

## For Sale

Monroe #326 Beta "Scientist" Programmable Calculator with Model #392 Digital Tape Unit.

Tape drive is fully controllable by program, permitting automatic read-in of program overlays plus programmable storage/retrieval of data. It is in like-new condition, with full documentation and fitted Attaché carrying case.

Original cost: \$1300.

Asking price: \$650.

#### Contact:

Gene Szymanski  
693 Rosedale Road  
Princeton, N.J. 08540  
(609)-924-8856



*Jeff Benton positions DIP switch on 680-BSM circuit board*



*John Martin inserts integrated circuits into their respective sockets.*

# MITS®/Altair™ CPU Modification

By Darrel Van Buer

Since its introduction, MITS® has had various aspects of its computer design criticized. One of the more severe problems with the original design concerns the circuitry used to generate the  $\phi 1$  and  $\phi 2$  clock signals for the CPU chip. The MITS/Altair™ design predated the availability of the 8224, so a 74123 dual one-shot was used. Problems with this circuitry have been met with a variety of fixes, the most unusual being to glue an aluminum foil heat sink to the IC. Parasitic Engineering Company offers a fix kit based on a better quality one-shot. Because I was interested in trying the 2.5 MHz and 3 MHz versions of the 8080 microprocessor, I studied the board and the Intel data sheets to learn how to substitute an 8224 clock generator IC on the MITS board. While it proved to be a rather complex modification, it can be made in an hour or two, as described here.

The modification involves the removal of all the existing clock circuitry and replacing it with the standard Intel circuit and several components, to be certain that the right signals are available on the bus. The ready latching circuit, however, was not used, because this function was already performed on the board and would have increased the complexity of the modification.

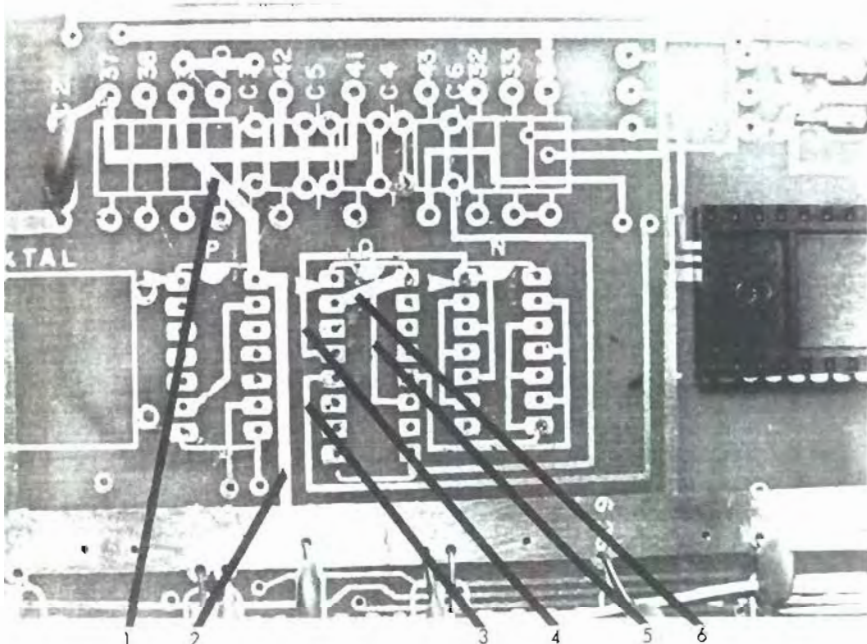


Figure 1. PC Land Cutting—Top Left Component Side

Before starting, you should carefully remove the CPU chip to a piece of conductive foam to void any static damage. The first procedure in the modification is the removal of all unwanted circuitry from the board. Since none of these components will be re-used, the main concern is that

they be removed without damaging the PC board or the plated holes. If you have good de-soldering equipment, such as solder suckers and IC de-soldering tools, use them. If not, the safest method for removing the ICs, is to cut all the leads and then unsolder each lead separately. The parts to be removed are given later in this article. If sockets were used for the ICs, they must also be removed. All of these parts are located in the top left corner of the board, between the regulator and the CPU socket, and above the large power bus running across the middle of the board. All but one of the parts in this area, capacitor C2—nearest the regulator, should be removed. When this step has been completed, the board should appear as shown in Figure 1.

The second major step in the modification is to cut through the PC foil paths on the board in eleven places. In making these cuts, remove a small segment with a sharp knife or scraper, taking care to avoid damaging other parts of the board. Table 1 summarizes these cuts. The first six cuts are also shown in Figure 1, which illustrates the top left area of the board. The seventh cut is shown in Figure 3. The four remaining cuts are shown in Figure 2, which are on the back of the board near the removed parts.

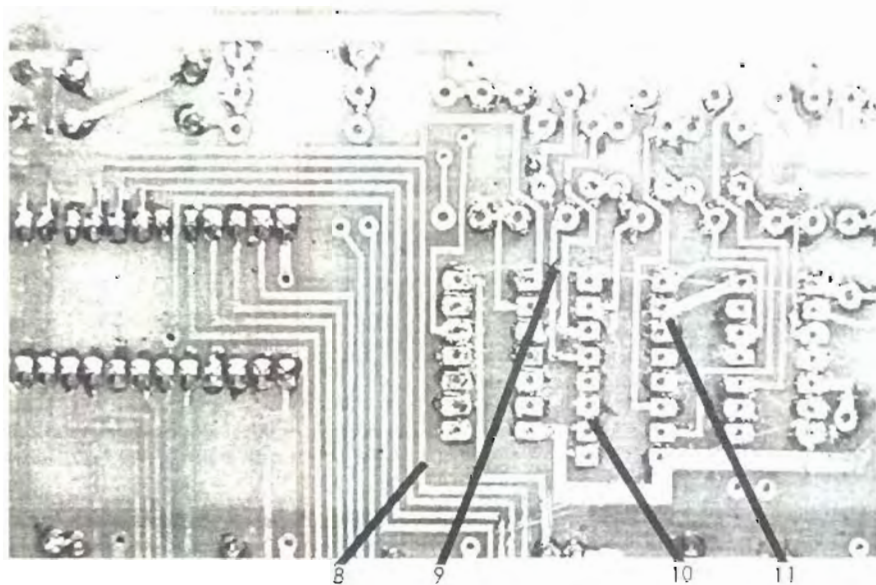


Figure 2. PC Land Cutting—Top Left Back Side

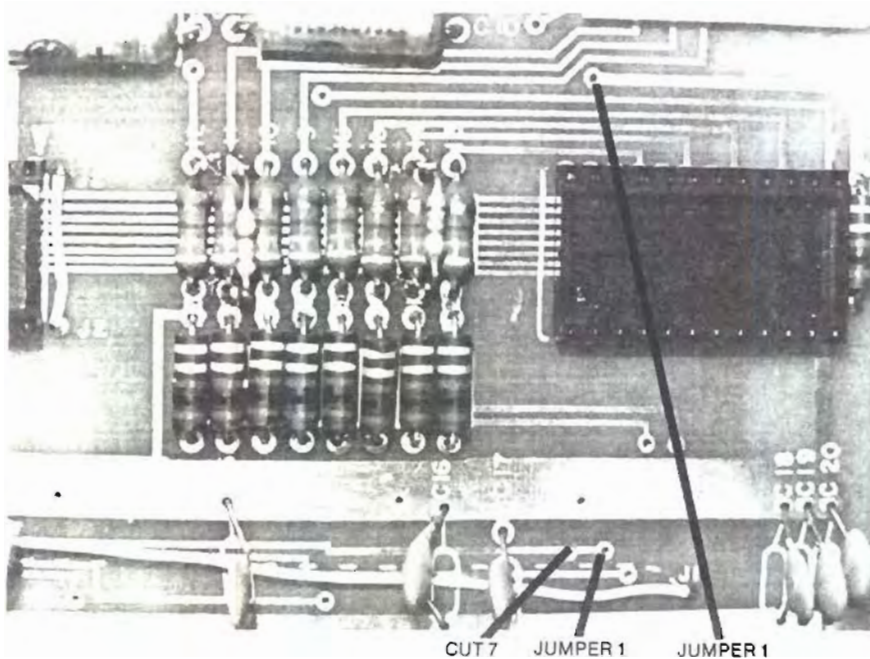


Figure 3. Modifications to Top Right of Board

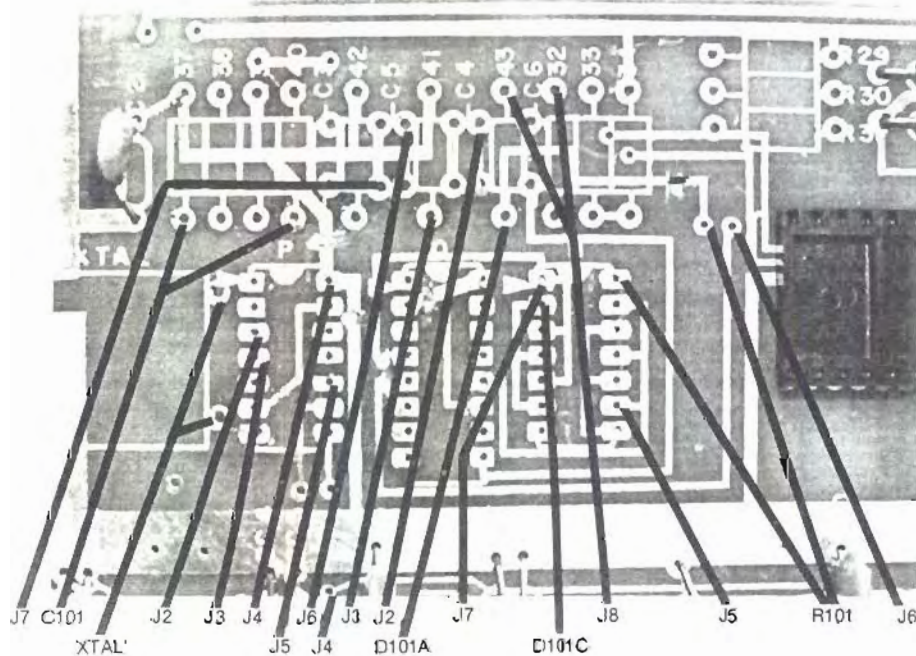


Figure 4. Components and Jumpers—Top Left

The third major step is the addition of the five parts listed in the Parts List. Table 2 summarizes the locations for these additions, and Figure 4 shows the locations of these changes (IC Q' replaces IC Q in the same orientation). C101 can be used to trim the crystal frequency to high accuracy. If tuning

is not needed, a fixed capacitor in the same range of values may be substituted. When mounting the trimmer, note that several alternate holes can be used to allow for size. Many trimmer designs will require the soldering of short pieces of wire to the lugs before mounting to the board.

While not required, the use of a socket for the 8224 is recommended.

The final step is the installation of thirteen jumpers on the board. All of the eight jumpers installed on the top of the board run between plated-through holes. Some of the holes are those left by the removal of components, and the others are extra holes on the board for other connections. The jumpers installed on the back of the board have one or both connections made by wrapping the end of the wire around the pin of an IC or socket and then soldering it in place. The locations of all jumpers are given in Table 3, and Figure 3 shows the location of jumper 1. Figure 4 shows the positions for the remaining jumpers on the top side of the board. All jumpers on the bottom of the board are shown in Figure 5.

Figure 6 shows the finished conversion. At this point, the CPU chip can be returned to the board, and the board can be re-installed. Trimming the crystal frequency with capacitor C101 is the only adjustment that may need to be made. This adjustment is not necessary, as the crystal will generally oscillate within 0.1% for any setting. Critical adjustment can be made with a high grade frequency counter or by zero-beating with WWV at 5 MHz or 10 MHz over a shortwave radio. The 2 MHz frequency will vary by as much as a few hundred Hertz, as mainframe bus loading and instruction sequences change.

The amount of noise present on many lines of the MITS/Altair bus is proportional to the length of the bus that the signal has traveled from its source. I have graphically witnessed, with an oscilloscope in my system, noise becoming unacceptable after 10 inches of signal travel. You can cut all noise levels in half by locating the CPU card in the center of the occupied part of the bus, since neither signal travels as far from source to destination. My system has been running reliably for more than six months with these modifications.

Figure 7 gives the schematic diagram for the modified clock circuit. Note that the 18 MHz oscillator output has been brought out as the CLOCK signal. In the standard MITS/Altair system design, this pin is only 2 MHz, so a slight modification in the conversion is needed if any cards in your system require the latter frequency. To supply  $\phi 2$  to this pin, omit jumper J5, listed in Table 3. Alternate jumper J5 belongs on top of the board from the hole for pin 10 of IC P (one end of old J5) to the other hole for the upper end of C5 (one end of jumper J6).

## PARTS LIST

IC Q' - 8224 clock generator  
 XTAL' - 18 MHz  
 C101 - 3 to 10 pF trimmer  
 R101 - 2.2k  
 D101 - 1N914

## UNPARTS LIST (parts to be removed)

IC N - 7406  
 IC P - 7404  
 IC Q - 74123  
 XTAL - 2 MHz  
 C3 - .01  
 C4 - 10 pF  
 C5 - 100 pF  
 C6 - 20 pF  
 R29 - 470  
 R30 - 470  
 R31 - (none)  
 R32 - 470  
 R33 - 470  
 R34 - (none)  
 R37 - 1k  
 R38 - 330  
 R39 - 1k  
 R40 - 330  
 R41 - 13k  
 R42 - 6.2k  
 R43 - 680

### A. Traces on top of board in part removal area

1. Trace from P-14 to upper resistor array
2. Trace from P14 to +5V bus
3. Trace from Q-5
4. Trace from Q-4 to N-1
5. Trace from Q-11 to Q-16
6. Trace from Q-2 to Q-16

### B. Trace elsewhere on top of board

7. The upper of two traces which pass under SC17, just to the right of SC17

### C. Traces on bottom of board in parts removal area

8. Trace from A-12 (CPU) where it passes N-8
9. Trace from Q-13 below R43
10. Trace from Q-10 to Q-11
11. Trace from Q-2 to Q-3

NOTES: P-14 denotes pad for pin 14 of IC P. R43 denotes pads left by removal of R43. Other references are similar.

Table 1 PC Land Cutting

## About The Author

Darrel Van Buer was awarded an M.S. in Computer Science by Iowa State University in 1975. He is currently studying for his Ph.D. at UCLA and has been involved in personal computing since the introduction of the MITS microcomputers.

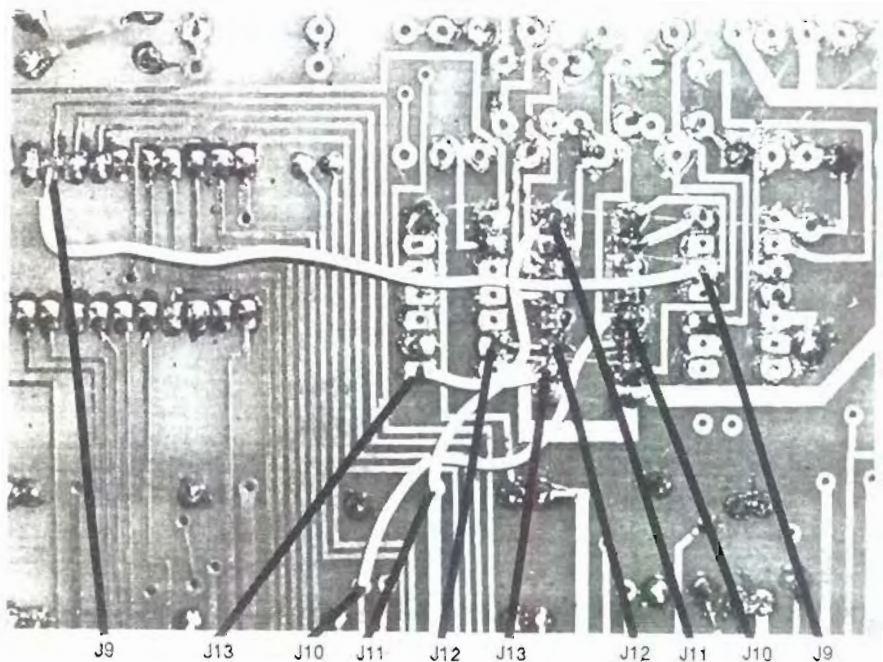


Figure 5. Jumpers—Back of Board

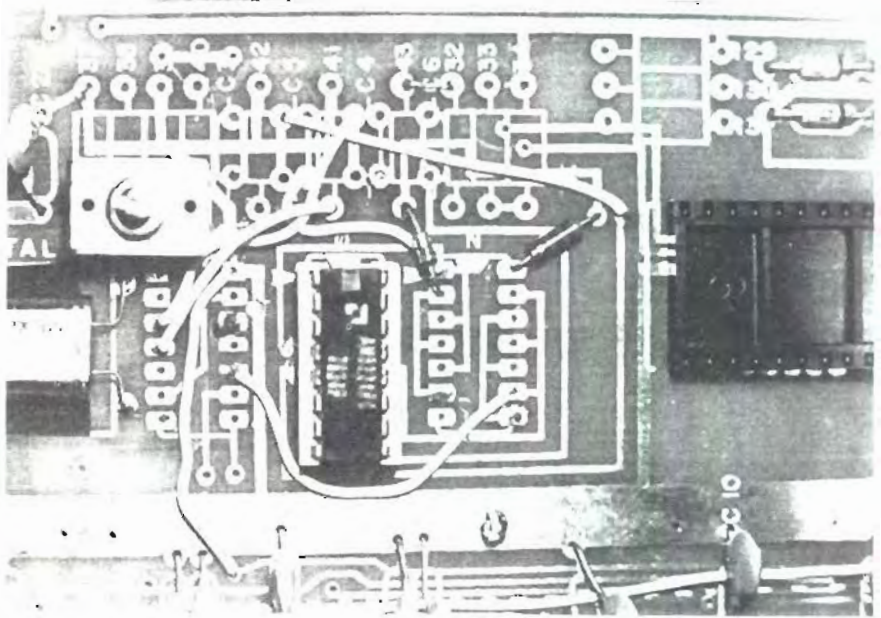


Figure 6. Finished Conversion

Part	First Location	Second Location
IC Q'	Same position and orientation as IC Q	
XTAL'	Same position as XTAL (insulate case from board)	
C101	Lower pad of R37 or R38	Lower pad of R39, R40, or C3
D101	Cathode (banded end) to N-2	Anode to lower pad of R43
R101	N-14	Unmarked pad above and to the right of N-14, the left two to the left of CPU

NOTE: Positions are the same as for Table 1.

Table 2. Component Additions

# Demonstration Program

By Ken Knecht

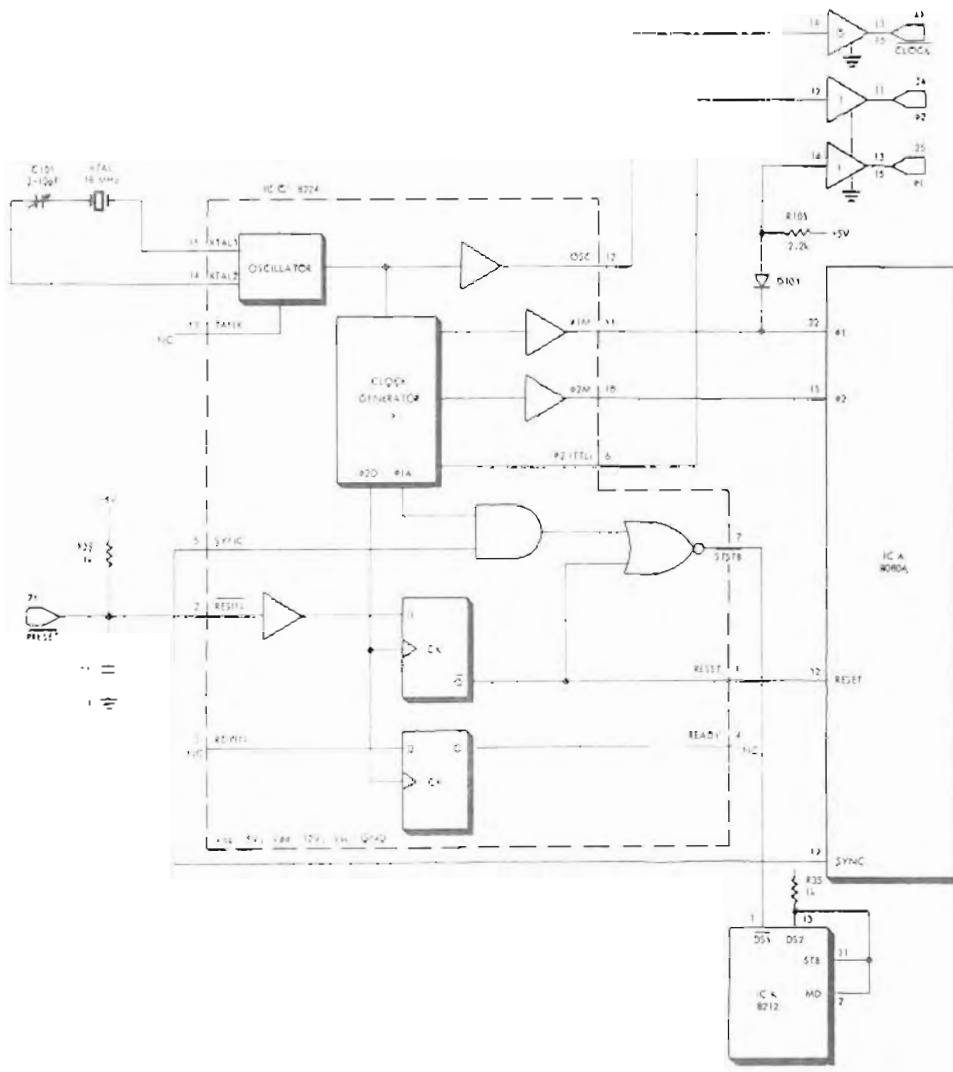


Figure 7. Schematic Diagram

A nice computer system is always fun to show to computer-less friends. Unfortunately, most programs are a bit complicated for simple demonstrations. By the time the StarTrek rules have been explained, most people have usually lost interest or are totally confused. Therefore, I wrote the following program, which makes a rather good demonstration and permits others to run the program themselves.

Very little detail about the program is required, other than to mention that the "#" following some of the variables indicates that it is double precision. If your BASIC does not support this function, omit the "#"s from the variables and change lines 165, 230, 260, and 290 to read "NOT OVER 8 DIGITS" rather than "NOT OVER 16 DIGITS". Other than this change, the program should run in any BASIC.

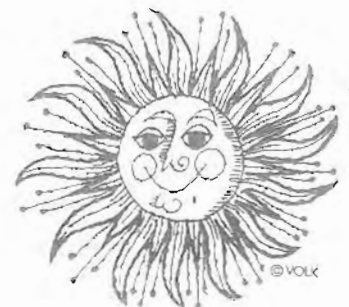
The program is very simple. It consists of a string variable to remember the user's name, some simple math problems, counting letters in a string, and the tried and true "Guess the Number" game. The latter seems to be the most popular.

In any event, when the program is finished, you'll have fun answering the many questions.

Program on page 32

## About The Author

Ken Knecht currently heads his own company called Kencom Corp. in Arizona. In addition to this, he freelances as an author, computer programmer, broadcast engineer, and television system consultant.



Jumper	First End	Second End
1	Hole to right of cut 7 near SC17	Hole above pin 11, IC K (8212)
2	P-3	Top pad of C4
3	P-4	Lower pad of R41
4	P-14	Upper of two holes located between SC5 and SC6
5	P-10	N-9
6	Top pad of C5	Hole between R101 and CPU
7	Lower pad of C5	N-1
8	Top pad of R43	Top pad of R32
9	P-12	Wrap around pin 12 of CPU
10	Hole slightly to left (top view) of SC9	Wrap around pin 5 of IC Q'
11	Hole in +5V bus between SC8 and SC9	Wrap around pin 16 of IC Q'
12	N-6	Wrap around pin 11 of IC Q'
13	N-8	Wrap around pin 10 of IC Q'

NOTE. Same as Table 1.

Table 3. Jumper Locations

# A BASIC Memory Test

By Dave Culbertson

Rather than test a newly built memory board for your computer by using the supplied memory test routine written in machine or assembly language, have you often simply loaded BASIC, loaded a program, and hoped that it would work? Many of us are guilty of this. However, the problem can be solved by using this simple program that is visual in its manner of testing your memory. The program does not run at blinding speed, but it continually displays what is developing, which should help computer users who do not understand machine or assembly language programming.

This program offers two unique capabilities. Firstly, the program does not stop when it encounters an error. It simply prints the address and the error. Secondly, it is possible to test an address where no memory exists. Again, the address and the error will be printed.

I have written this program in MITS® 4.0 BASIC, but it can be modified for other BASIC languages. Only standard BASIC commands have been selected. One should understand where memory usage occurs in the computer before using this program (Table 1). It is assumed that you presently have 8K bytes of memory operational and that you have just completed a new 8K byte board that you would like to test. First, complete all of the manufacturers' recommended electrical tests. If you do not have the equipment, a local school or computer store should be able to do these tests for you. Next, use this program to test the operational mode of your memory board.

Since you only have 8K bytes of operational memory, you will normally be strapping the memory board for address 8192. But, in this case, temporarily strap the board to a higher address, such as 24576. This would be the sixth such 4K byte address assignment. Remember to restrap the board to the proper address when your test is complete. This program will fit within your 8K byte board, along with the 8K BASIC and the stack. It is necessary to strap the memory board to a non-consecutive address, because the stack will move to the end of your new memory board if you do not separate the memory. If this happens,

you will not be able to test this area. To attempt a test of this type will disturb the BASIC interpreter, and this will prevent some, or all, of the BASIC commands from performing their normal functions. If the program crashes as just described, you will need to reload BASIC and reload the program.

Once you have strapped your memory to address 24576 and are ready to begin execution, run the program. You will be asked for the starting and the finishing locations of the area to test. The computer will then ask for the complete or partial test. I suggest that you run the partial test first, since this runs faster. However, it only checks to see if one number can be written into each location. Try entering the test word #0 (zero). If all is well, the computer will print out the address and the contents of this address. If there is an error, the program will print the address, the test word #, the resulting #, and the word "ERROR". The original contents of this address will be restored to this location.

The test will continue in this manner until complete. I use a teletype to retain a hard copy record of the errors. The good locations are printed on my video terminal to save paper. If you do not have two terminals, change line #150 to read "150 REM" and change line #170 to read "170 REM". These changes will print all data on one terminal. It is necessary to use the partial test with the test word #0 (zero) to determine if the new board will accept all low inputs to be written into each location.

Next, rerun the test using the partial mode and test word #255. This is the reverse condition that tests for all high input (all 1's) to be written into memory. If you have an error and want a complete analysis, run the program using the complete mode. The program will try to write all combinations (0 to 255) into each memory location. The complete routine takes about four seconds per location. Errors are shown as they are in the partial test, except the address will only be printed once if an error is found. The bad combinations of the location will be printed with the resulting word #. The complete analysis may appear confusing initially, but you should be able to analyze the area

of trouble by comparing your result with the knowledge of your memory board.

Many types of memory boards are available, so the results of this test may vary. It is important to know the organization of the memory chips in your new memory board. Some boards use an organization of 1,000 times 1, which means that the chip has 1,000 locations (addresses) that can store a "0" (low) or a "1" (high) in each of these locations. This type of board will require 32 memory chips in order to provide 8K bytes of storage. This is a popular method presently used on static memory boards.

The dynamic memory boards now use fewer chips, since more locations have been put into each chip. If you have a 4K byte dynamic board, you will find only eight chips. The popular organization among this type of chip is 4,096 times 1. The memory boards store the information into the chips by assigning a value to each of the eight chips per address that are used. When an address is selected and you would like to read the information at that location, the output is from these eight chips. If one or more of these chips is defective or if a location within any of the chips is bad, an error will be printed at this address when this program is run. The value of each of the eight chips is shown in Table 2. Assume that the storage of information within each chip has something (1 or high) or nothing (0 or low) as its only variations. If the chip has something (1 or high), add its value (Table 2) to the other chips with something (1 or high) in them. Thus, when an address is read, you are really seeing the combined output of eight chips. If one of these chips having a large organization is defective, many addresses will be affected. For example, if zeros (0) are stored into chips 0 through 6 and if ones (1) are stored into chip #7, use only the chip #7 value as the contents of this memory location. The print-out would be 128. If zero (0) is stored into chips 1,3,4,5, and 7 and if one (1) is stored into chips 0,2, and 6, use only the values of chips #0 = 1, #2 = 4, and #6 = 64. Add these values (1 + 4 + 64 = 69). The number 69 has been stored at this location. This is the method used to store numbers in memory of an eight-chip memory

Address	Description
0 to 6457	This area is used for the BASIC Interpreter.
6457 to 7676	This area is used for the "BASIC MEMORY TEST" program.
7677 to 8192	This area is the overhead and stack area.

This table assumes a MITS/Altair™ computer with 8K consecutive memory.

Table 1

Memory Chip #	Decimal Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

This table assumes 8 memory chips used per bank.

Table 2

bank. Using this method, the computer permits any number from 0 to 255 to be stored. The computer is not able to use a larger number.

If the program has been run, you are now ready to analyze the errors provided for you by the program. It is assumed that the partial program was run and that the "0" (zero) test word was satisfactory. The error occurred when the "255" test word was run, so one or more of the eight chips must be bad. Assume that the error occurred at address 24576 and continued until the program reached address 25576. Also assume that, in all of these addresses, the program showed 191 when you tried to write the test word "255". If 191 is subtracted from 255, the result is the number 64, the value of chip #6. It is now known that chip #6 is either dead or that it is not receiving the correct voltages or signals from the

board. A complex problem has been resolved by locating the area of the trouble. Even if you decide not to repair the board yourself, the repair technician's time, trouble, and expense will be lowered.

Every program has its limitations, and this program is no exception. If your memory board has addressing difficulties, three problems may occur. Firstly, this program may not detect any error. Secondly, your BASIC may bomb when you run your regular program; or, thirdly, random changes may be detected in your regular program. This last condition could also be due to a memory chip that malfunctions intermittently. This is the most difficult problem to find—in which case, happy hunting!

The given program run shows an actual problem. In this example, I

removed chip #6 from my memory board and performed the BASIC MEMORY TEST on it. The board will not pass the partial test with the word #0 (zero), but it will pass the partial test with the word #255. When the complete test program is run, it indicates the number combinations that cannot be written properly into the memory. The decimal combinations that read correctly are not printed out in the complete test. If you suspect the memory as the cause of a problem you are having, this type of program can save a great deal of time and trouble.

*Program on page 33*

### About The Author

*Dave Culbertson graduated from the Springfield Technical Institute and is currently the Vice-President of Custom Electronics, Inc. in Massachusetts.*

## FDOS-III: The Latest from Pertec Computer Corporation

FDOS-III, a powerful new Floppy Disk Operating System for microcomputers, is one of Pertec Computer Corporation's newest additions to their iCOM® product line.

FDOS-III offers the maximum in flexibility and power with its relocatable assembler for Z-80 and 8080 code. All its console communications are either in decimal or hex, thereby simplifying program development. The "BATCH" command allows automatic chain operations, and the system includes an optional operator prompt feature for variable input requirements. Data is stored and

recognized by FDOS-III, and it can use all available disk storage capacity.

The new FDOS-III is available for any iCOM Floppy Disk System operating on the 8080 or the Z-80. The FDOS-III is fully compatible with programs written under iCOM's FDOS-II and allows immediate use of any existing iCOM-compatible programs. The single command operations of FDOS-III give the user disk-to-disk program editing and assembling, disk-to-memory program loading, disk-to-punch device transfer, reader-to-disk transfer, disk-to-disk transfer, named files, and many

other features.

FDOS-III also has relocatable driver modules that provide easy access to files, thus maximizing data handling flexibility. The storage area on each diskette is available for any number of files of lengths ranging from a single sector to an entire diskette. The files may contain program source data, program object data, or user-generated data.

Files are specified by a 1-5 character file name, and any number of files may be merged to create a new file. Any file may be renamed or may be deleted (FDOS repacks the disk-



# Tic Tac Toe Modification

By John Trautschold

ettes automatically at the operator's option to make the deleted file space available). Also, files may be tagged with attributes (i.e., a file may be declared permanent, not allowing it to be inadvertently deleted).

The resident FDOS-III is conveniently contained in a 1K PROM located on the plug-in interface card. The FDOS-III also contains its own powerful disk-resident assembler and editor. The microcomputer's monitor remains intact, thus retaining all existing non-FDOS operations. A typical edit/assembly sequence requires only a few minutes to accomplish, and a string-oriented text editor greatly simplifies file or program modification.

"FDOS-III provides one of the most powerful and complete development packages available anywhere," claims T. E. ("Gene") Smith, Division Vice-President and General Manager of PCC's Microsystems Division. "When used with any of iCOM's family of Floppy Disk Systems and compatible plug-in interfaces, FDOS-III provides an easy-to-use, reliable, fast, and extremely efficient capability for auxiliary program and data storage.

"Using the iCOM program development package, time is reduced by a factor of 20 to 100 compared to cassette or teletype. In sum, FDOS-III, together with iCOM floppies, brings new speed, convenience, and capability to users' development tasks," Smith stated.

Commands available with FDOS-III include Copy, Alloc, Batch, Delet, Pack, Delpk (Delet and Pack functions in a single command), Edit, View, List, Libo, Durnp, Load, Merge, Print, Renam, Run, Link, and Exit.

Also included are two new commands, ASMB and SYSGN. ASMB, in Z-80 or 8080 code, assembles the contents of a source file and directs the object output to the destination file. SYSGN allows the user to store I/O information in sectors on a system diskette for use by FDOS-III, thus minimizing the effort needed to bring FDOS-III up on a custom-configured machine.

FDOS-III is being marketed as part of the PCC Microsystems Division's iCOM Microperipherals® product line. It is available from any of the more than 70 iCOM dealerships nation-wide.

The "Tic Tac Toe" software articles from the August 1977 edition of Computer Notes was very interesting—and frustrating, to say the least! When I loaded the BASIC program, I discovered that, no matter how hard I tried, I could not beat the computer! The program was written in such a way as to make the computer unbeatable; the best that could be achieved was a tie (as was mentioned in the article). Even if the program could have been beaten, there was no logic included permitting the program to jump to the "Player Wins" subroutine at line number 1220. I have recently made some modifications to correct this problem as well as some that now make it possible, but still difficult, to win.

I have eliminated the lines that establish the initial move for the

computer, because these always defaulted the computer to start in the center square (which is nearly unbeatable as an initial move), as well as in square 1-3 (upper right square on the board). To replace these eliminated lines (190 and 200), I have written a random number subroutine that randomly places the computer's first move in an empty square. After the first random move, the other moves follow according to the programmer's logic. To repair the problem of having no logic to determine if the player has won, I have added a complete subroutine that is called in the new line number 225. Line 1500 is the location of the subroutine.

The following is a list of the new lines to be inserted into the program for proper operation:

```
190 D = INT(RND(1)*10/3)
191 IF D = 0 OR D > 3 GOTO 190
192 E = INT(RND(2)*10/3)
193 IF E = 0 OR E > 3 GOTO 192
194 IF C(D,E) = 0 THEN C(D,E) = 3:
      C$(D,E) = "C":GOTO 210
195 IF C(D,E) <> 0 THEN 190
225 GOTO 1500
1500 IF C(1,1) = 1 AND C(1,2) = 1 AND C(1,3) = 1 GOTO 1220
1510 IF C(1,1) = 1 AND C(2,2) = 1 AND C(3,3) = 1 GOTO 1220
1520 IF C(1,1) = 1 AND C(2,1) = 1 AND C(3,1) = 1 GOTO 1220
1530 IF C(1,2) = 1 AND C(2,2) = 1 AND C(3,2) = 1 GOTO 1220
1540 IF C(2,1) = 1 AND C(2,2) = 1 AND C(2,3) = 1 GOTO 1220
1550 IF C(1,3) = 1 AND C(2,2) = 1 AND C(3,1) = 1 GOTO 1220
1560 IF C(1,3) = 1 AND C(2,3) = 1 AND C(3,3) = 1 GOTO 1220
1570 IF C(3,1) = 1 AND C(3,2) = 1 AND C(3,3) = 1 GOTO 1220
1580 GOTO 230
```

This concludes the modifications to the program. I hope that others will enjoy this program as I have.

## About The Author

*John Trautschold has worked for five years in television electronics and engineering. He received his engineering degree from the University of Wisconsin in Milwaukee, and he enjoys working with computers both at home and on the job. It has been three years since he first acquired his MITS Altair 8800.*



# Practical Programming, Part II

By Gary Runyan

*This series is produced by the MITS<sup>®</sup> Computing Services Department, and the articles contain useful ideas for programming MITS BASIC. "Practical Programming, Part I" appeared in the November 1977 issue of Computer Notes, and it discussed the solution to the problem of line counting.*

CTRL-A, a feature of MITS<sup>®</sup>BASIC, has become a powerful programming aid, due to an undocumented feature discovered by Donald Fitchhorn of MITS. If CTRL-A is typed immediately after EDITing a program line, the edited line is returned as a command to be edited. Thus, CTRL-A can be used to shuffle program lines, break apart multiple statement lines, and isolate program errors.

A program line can be shuffled from one place in the program to another by typing the following sequence:

- 1) EDIT xxxx<CR>(xxxx = old line #)
- 2) Q
- 3) CTRL-A
- 4) lyyyy<CR> (yyyy = new line #)
- 5) xxxx<CR>

This moves the line that was at xxxx to line yyyy and deletes line xxxx. The original line can be retained by not executing Step 5. If a new line is needed that is slightly different from the old line, ESCAPE can be typed in place of CR (carriage return) as the last character in Step 4. The editor can then be used to modify the line before placing it at yyyy.

CTRL-A can be used to break a multi-statement program line into two program lines without retyping either of the new lines. A copy of the original line is made using the above procedures for copying a line. Then, the K EDIT command is used to remove the first half of the line from one copy, and the H EDIT command is used to remove the second half of the line from the other copy. For example, to change:

```
600 LPRINTA:PRINTB
```

to:

```
600 LPRINTA:IF X<0THENGOSUB500  
605 PRINTB
```

one would type:

```
EDIT600<CR>  
Q
```

```
CTRL-A  
I605<ESC>2KP<CR>  
EDIT600<CR>  
2SPHIF X<0 THEN GOSUB500<CR>
```

To isolate a syntax error that has been encountered while a program is running, type a Q to exit EDITing without losing the program variables. Typing CTRL-A then restores the program line for execution as a command. The command line can be modified at will without destroying all the program variables and then executed to test the modifications. Colons can be replaced by single quotes in a multiple statement line to isolate the statement with the syntax error. Obvious errors can be corrected and tested immediately. For example, if the line:

```
50 A = 5:PRINT A:A = A + #7:PRINT A
```

is encountered while a program is running, the following will isolate the error, correct it, and continue the program:

```
SYNTAX ERROR IN 50  
OK  
50 (Type: Q)  
OK  
(Type: CTRL-A)  
!(Type: S:C'<CR>)  
OK  
!(Type: CTRL-A)  
!(Type: S'C:S'C' <CR>)  
5  
OK  
(Type: CTRL-A)  
!(Type: S'C:S'C' <CR>)  
5  
SYNTAX ERROR  
OK  
(Type: CTRL-A)  
!(Type: S#C3S'C: <CR>)  
5  
42  
OK  
(Type: GOTO60)
```

After the syntax error is successfully corrected, one executes a GOTO command (if the corrected line did not branch back in) to continue program execution. Continuing after correcting is a good habit to adopt. Other bugs are found without completely rerunning the program. If variable values are clobbered before an error is success-

fully corrected, the programmer must decide if it is better to rerun or to restore values (using direct commands) before continuing with a GOTO.

CTRL-A can be used to isolate ILLEGAL FUNCTION CALL, TYPE MISMATCH, and other errors, as well as syntax errors. One simply types:

```
EDIT [number of line in question]  
Q  
CNTRL-A
```

to gain control of the line in question.

Once a programmer begins using CTRL-A after exit from EDIT, he will find that his whole set toward debugging has changed. Rather than following the old batch system approach of guessing corrections from the listing and rerunning, one will begin using the computer to resolve the bugs. Time lost to and ulcers caused by debugging will be considerably reduced.

Initially, in the joy of a new-found tool, you will use the "don't-leave-the-terminal-until-the-bug-is-resolved" approach in excess. Eventually, after several wild goose chases, you will begin to discriminate between when to sit back and really study the listing and when to poke-around on the terminal.

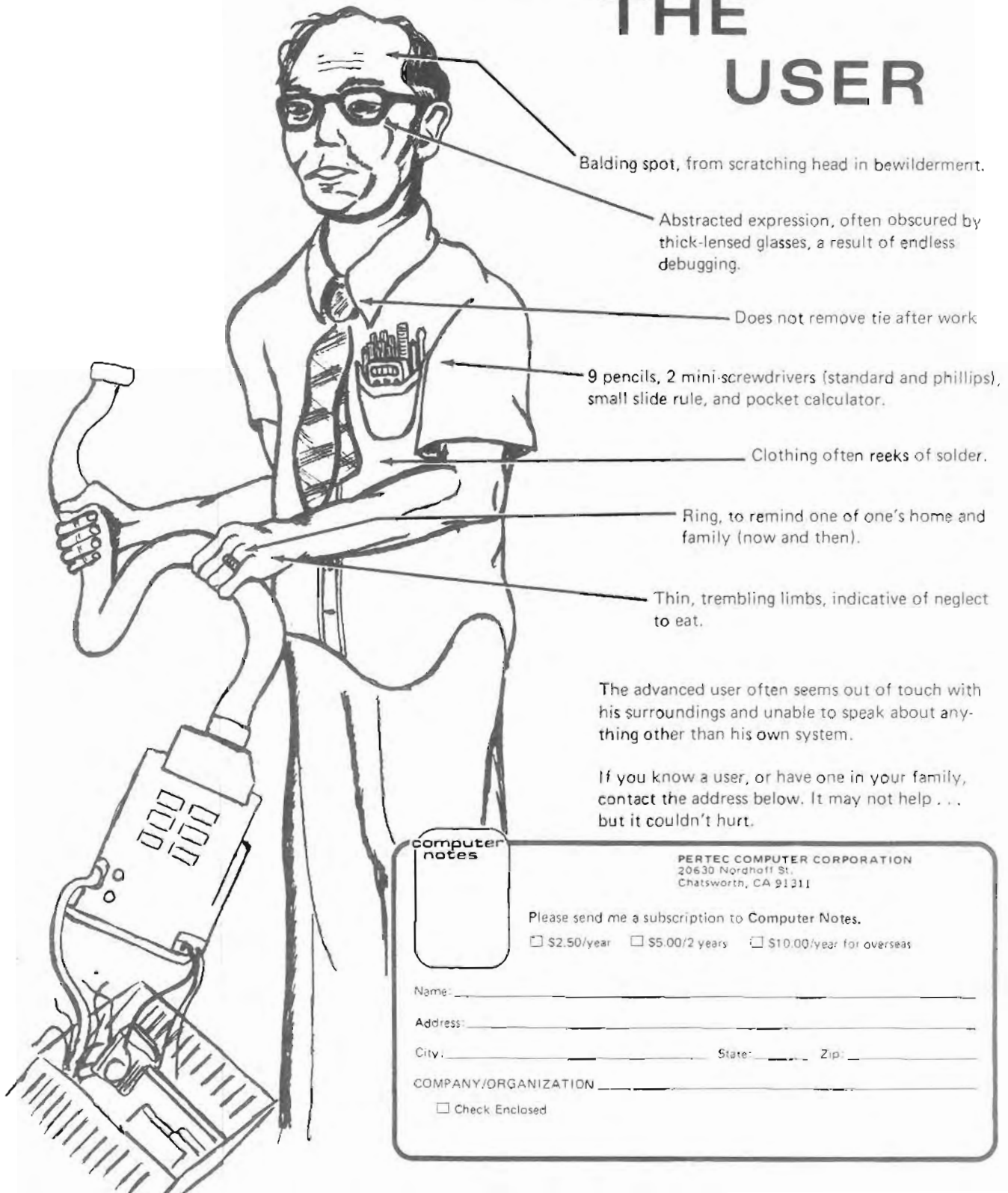
Some additional poke-around hints are:

1. Use CTRL-A to execute lines that print the values of variables.
2. Tack lines that will print variable values onto the end of the program to save constant retyping.
3. Edit extra STOPS into the program to establish poke around points.
4. Edit in extra PRINT commands to monitor the evolution of variable values.
5. Use TRON and TROFF.
6. Edit in GOTOs to skip around undesired outputting, or go directly to problem areas.

## About The Author

*Gary Runyan is the Director of Computing Services and has been a MITS employee for three years. He has worked in the data processing field for six years, and he holds a Bachelor's degree in Electrical Engineering from New Mexico State University.*

# KNOW THE USER



Balding spot, from scratching head in bewilderment.

Abstracted expression, often obscured by thick-lensed glasses, a result of endless debugging.

Does not remove tie after work

9 pencils, 2 mini-screwdrivers (standard and phillips), small slide rule, and pocket calculator.

Clothing often reeks of solder.

Ring, to remind one of one's home and family (now and then).

Thin, trembling limbs, indicative of neglect to eat.

The advanced user often seems out of touch with his surroundings and unable to speak about anything other than his own system.

If you know a user, or have one in your family, contact the address below. It may not help . . . but it couldn't hurt.

computer  
notes

PERTEC COMPUTER CORPORATION  
20630 Nordhoff St.  
Chatsworth, CA 91311

Please send me a subscription to Computer Notes.

\$2.50/year    \$5.00/2 years    \$10.00/year for overseas

Name: \_\_\_\_\_

Address: \_\_\_\_\_

City: \_\_\_\_\_ State: \_\_\_\_\_ Zip: \_\_\_\_\_

COMPANY/ORGANIZATION \_\_\_\_\_

Check Enclosed

# Modifying MITS<sup>®</sup> BASIC for ASCII I/O

By John Palmer

I am a certified electronics technician, but most of my past experience has been in radio and television. Consequently, my MITS<sup>®</sup>/Altair<sup>™</sup> 8800 has been an exciting challenge, and my efforts have been aided by the information in Computer Notes. Many of the CN readers' comments indicate that there are always newcomers to the trade looking for information on how to do elementary tasks, such as making ASCII recordings on cassette, so permit me to relate to you a few pointers that I have learned. On

On modifying MITS BASIC for ASCII I/O:

Hardware: 8800 with 16K, ACR, 2S10, and Model 33 TTY

Software: MITS 8K BASIC, Version 4.0, January 1977

I thought I might be the last person to learn to enter BASIC's I/O and to change a few memory locations to permit an ASCII program listing (source code) to be either output or input on a storage device other than paper tape (a paper tape punch/reader tends to be very expensive!).

Several problems that users of MITS BASIC might encounter can be solved by modifying the I/O routines in such a way that the ACR cassette interface replaces the terminal. If the user has Extended BASIC, the console feature will transfer I/O to the cassette. Questions will arise if the program was written in some version of BASIC that does not have the console command.

The following describes how and why I make ASCII recordings in MITS BASIC. A very simple batch to 8K BASIC, Version 4.0, will place ASCII characters onto the cassette when I use my Model 33 teletype. After loading the program, using CLOAD, I then type:

```
POKE 1362,211:POKE 1363,7
```

Upon hitting the return key, BASIC then pokes these two locations in the output routine, and what goes to the teletype printer will go to the ACR. The MITS Software Library has more information on how to do both input and output using the ACR and 4K BASIC (which has no provision for CSAVE and CLOAD).

But this simple method is only for 8K BASIC, Version 4.0. Before trying this, be sure you have the same

Version of BASIC. Either the locations are different, or there are not several empty locations in the output routine. Note that those two locations are needed for the MITS 4P10 board, but not needed otherwise.

Doing input is slightly more involved, but there are three reasons for troubling yourself.

1. The output of one program may be needed as input to another.
2. Cassette input will transfer a program from one version of BASIC to another. For example, if you key in StarTrek in 8K BASIC, you will find that you cannot load it into the current version of Extended BASIC.
3. Some types of errors due to poor recording can best be corrected by making a new recording in ASCII and then using the new recording as input. BASIC will put all lines in proper order, provided that the input speed is not too fast (put in nulls, just to be sure).

To input an ASCII recording from the cassette, one must either use the POKE command or must stop the microcomputer and alter memory locations with the front panel controls.

I am presenting a partial listing of the routines that are used in MITS 8K BASIC, Version 4.0, for terminal input and output. Note that output is first.

Split-Octal Address	OCTAL Data	Code or Purpose	Changes Needed for ACR Cassette Interface
---------------------	------------	-----------------	---

Here is part of the output:

005	116	361	POP PSW	
005	117	323	OUT	
005	120	021	Data Port	
005	121	385	Push PSW	
005	122	000	NOP	323
005	123	000	NOP	007
005	124	361	Pop PSW	
005	125	311	Return	

Next is the input:

005	126	333	In	
005	127	020	Status Port	006
005	130	346	ANI	
005	131	001	MASK BIT	
005	132	312	JZ	302 JNZ
005	133	126	STARTING	
005	134	005	ADDRESS	
005	136	021	DATA PORT	007
005	137	346	ANI	

The output precedes the input, because both are 'called' routines that are called from somewhere inside BASIC. Furthermore, the front panel will produce the same result as the POKE command.

The following commands will transfer input to the MITS ACR interface using 8K BASIC, Version 4.0:

```
POKE 1367,6:POKE 1370,194:
POKE 1374,7
```

Before hitting RETURN, begin playback of a recording that was made in ASCII mode.

To have BASIC return control to the keyboard, either use the front panel to restore the original input routine or play a tape that was previously prepared. Here is how to prepare the 'change-over' tape.

1. POKE the two empty locations in 8K, Version 4.0, as shown earlier in the article.
2. Type: NULL 3
3. Put a spare tape into the cassette and begin recording (allow 15 seconds for the leader).
4. Hit RETURN two or three times.
5. Type: POKE 1367,16:  
POKE 1370,202:POKE 1374,17
6. Hit RETURN several times.

If a typing mistake is made, you must begin again.

The cassette will now have the instructions needed to restore control

to your keyboard (this is for a keyboard that uses the MITS 2S10 I/O interface). Set aside your 'change-over' tape.

When recording, it is good practice to use at least three nulls to prevent the tape from advancing ahead of BASIC (when playing a tape, use NULL 0).

If you wish to merge two programs, be sure that the two programs have different line numbers. First, input the program with lower line numbers. Otherwise, BASIC must do too much housekeeping, and it will fall behind. MITS BASIC presently has no provision for merging files or programs using CSAVE and CLOAD, and the use of CSAVE and CLOAD is much faster than ASCII.

When using a poor quality tape, a line number may become garbled. When loaded into 8K BASIC, such a recording may cause trouble. The following illustrates this:

```
LIST
10 REM.....
.....
970 A = A + 4
980 IF A 12 THEN 450
999 END
57 &NH SJ%FORBV MID$ = :A15,DLRO F,EIFM93
57 &NH SJ%FORBV MID$ = :A15,DLRO F,EIFM93
57 &NH SJ%FORBV MID$ = :A15,DLRO F,EIFM93
```

Where is line 57 from? Why is it at the end of the program? And why does it repeat on and on and on.....?

Any attempt to erase line 57 will prove to be futile. Aside from peeking inside the program buffer and trying to erase the bad number, the only way to cure this program is to dump it as an ASCII listing.

To make an ASCII recording of an existing program, do the following:

1. POKE locations 1362 and 1363 with 211 and 7.
2. NULL 3.
3. Type the following (don't hit RETURN yet):  
PRINT:PRINT:PRINT:LIST
4. Start the recorder, type a few spaces, and hit RETURN.

While BASIC is listing the program on the printer, the same ASCII characters are being recorded on cassette. A standard teletype runs at 110 baud, yet the ACR interface is normally 300 baud, which presents no

real problem. The cassette will have some verry loooong stop bits, but it will playback adequately.

Be sure to leave a long leader at the start and the end to prevent 'garbage' from being fed into your computer's input routine. The spaces and nulls at the beginning will purge the ACR buffer. First, play the tape, and, when the spaces begin, start your computer input. If you have not already done so, it is advised to modify the cassette machine to hear the playback while the patch cord is in place. Try 47 ohms across the mini-jack contacts.

For the more experienced, all of this may be very elementary. But, for those users like myself, I hope I have been of some help. Incidentally, "NEW" may be used when you don't want to merge programs.

## About The Author

*Patrick Delaney is currently working as an instructor of Digital Electronics at the Rhode Island School of Electronics. He graduated from the University of Rhode Island in 1970 with a B.S.E.E. and is now developing tutorial programs for the MITS/Altair 8800 computer.*

# MITS<sup>®</sup> Newest Business System

The MITS<sup>®</sup> 300 Business System is one of Pertec Computer Corporation's major additions to their already extensive product line.

The MITS 300 is a microcomputer-based system that is complete with all necessary hardware and software. It is available in two configurations, one with a hard disk (the MITS 300/55) and the other using two floppy disks (the MITS 300/25). The fully integrated business system provides capabilities for word processing, inventory control, and accounting functions, which include a general ledger, accounts payable, accounts receivable, and payroll.

"Customers now can buy a totally integrated system from a single supplier," says T.E. Smith, Division Vice-President and General Manager. "We provide both hardware and soft-

ware and can assume responsibility for the entire system. Also, service facilities are available through the PCC Service Division. And we are able to provide extensive dealer support in installing and starting up each application."

Both configurations of the MITS 300 Business System incorporate a MITS/Altair<sup>™</sup> 880b turn-key mainframe with 64K of Dynamic RAM, 1K of PROM, and serial input/output interface. Also included is a MITS/Altair B-100 CRT terminal with a 12-inch, non-glare monitor. The CRT displays 24 lines with 80 characters per line and has a memory page of 1920 characters. The MITS/Altair C-700 line printer, which is also part of the basic configuration, is capable of a bi-directional operation that allows the printed horizontal movement for

seeking the nearest margin of the next line. The C-700 prints 60 characters per second and 26 lines per minute.

Each configuration, comprised of the mainframe, a CRT terminal, and a line printer, also includes either a hard disk or two floppy disks, a controller, and BASIC language software. A MITS/Altair A08 Accounting Package and an Inventory Management Software Package, although not included, are available both with the hard and the floppy disk systems at additional costs.

The MITS 300 Business System is being marketed as part of PCC's MITS product line. It is available at the more than 40 MITS Computer Centers across the continent and by way of PCC's Microsystems Division directly on an OEM basis.



PERTEC COMPUTER CORPORATION's new MITS 300 Business System is comprised of a mainframe, a CRT terminal on a desk, and a line printer on a pedestal. Pictured here is the MITS 300/55, which is the hard disk, rather than the floppy disk, system.

## Favors

Captain Charles P. Connolly is a new MITS® user and would like to ask for your help. He is interested in contacting anyone using BASIC to solve substitution cryptograms. He would be particularly interested if MITS BASIC is being used, but any BASIC without MAT statements will do nicely. Please write to Capt. Connolly at the following address:

2701 Park Center Drive  
Apt. B-501  
Alexandria, Va. 22302

## Book Review

Presented here is a review of Dr. C. William Engel's recently published book entitled *Stimulating Simulations*. The small paperback book is written in MITS® 8K BASIC 3.2 [the programs will also work with all higher versions of BASIC] and contains ten rather unusual simulations written for the enjoyment of the computer hobbyist.

Dr. Engel is a Professor of Mathematics Education at the University of South Florida in Tampa. His book sells for \$5 per copy and \$3 each for orders of ten or more. Send orders, comments, or questions to:

Dr. C. William Engel  
P.O. Box 16612  
Tampa, Florida 33687

### A Review of

### STIMULATING SIMULATIONS: Ten Unique Programs in BASIC

The excitement of deep sea fishing, the intrigue of a jewel robbery, and the challenge of piloting a space ship on a mercy mission are three of ten simulations you can experience with your computer. The interaction between computer and player is a challenging one that forces the player to make logical decisions in order to succeed or, sometimes, survive.

These ten simulations can be found in a clearly-written, well-documented, 64-page book called *Stimulating Simulations*. Although the ideas are fairly sophisticated, the programs are relatively short (from 40 to 100 lines of BASIC). Each program includes a

scenario, a sample run, a flowchart, a listing of the variables, and suggested modifications.

This book is a good starting point for the computer hobbyist who wishes to explore the use of the small computer in simulating real events. A brief description of each program is given below.

#### "Art Auction" (48 lines)

One buys and sells paintings to make a maximum profit. This is a fast simulation and does not require extra materials.

#### "Monster Chase" (48 lines)

A monster is chasing a victim in a cage. The victim must elude the monster for ten moves to survive. This is a fairly quick simulation that does not require too much thought.

#### "Lost Treasure" (74 lines)

A map of an island that contains treasure is presented. The adventurer travels over different terrain with a compass that is not very accurate in an attempt to find the treasure. This is a short simulation that requires about fifteen moves. A map is provided.

#### "Gone Fishing" (83 lines)

The object is to catch a large number of fish during a fishing trip. Half of the catch spoils if the time limit is exceeded, or if time is lost in a storm. In addition, the boat sinks if it is guided off the map. There are also sea gulls and sharks to avoid. A chart is needed to keep track of good fishing spots.

#### "Space Flight" (68 lines)

The task is to deliver medical supplies to a distant planet while

trying to stay on course without running out of fuel. Graph paper is required to plot the course.

#### "Forest Fire" (77 lines)

The object is to subdue a forest fire with chemicals and backfires. Because the output is a 9X9 grid, a fast baud rate to the terminal is desirable. The success of a firefighter is based on the time needed to control the fire and completely extinguish it.

#### "Nautical Navigation" (70 lines)

This simulation requires the navigation of a sailboat to three different islands, using a radio direction finder. The wind direction is an important variable. Graph paper, protractor, and ruler are needed to plot the course.

#### "Business Management" (92 lines)

In this simulation, raw materials are bought, and finished products are produced and sold. The cost of materials and production and the selling price vary each month. The objective is to maximize the profits. No extra materials are required.

#### "Rare Birds" (75 lines)

This is a bird watching simulation. The object is to identify as many different birds as possible. A record of those identified is helpful, and a bird-watching chart is provided.


#### "Diamond Thief" (83 lines)

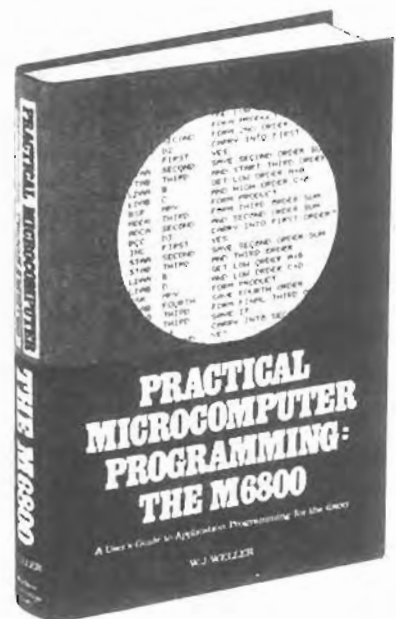
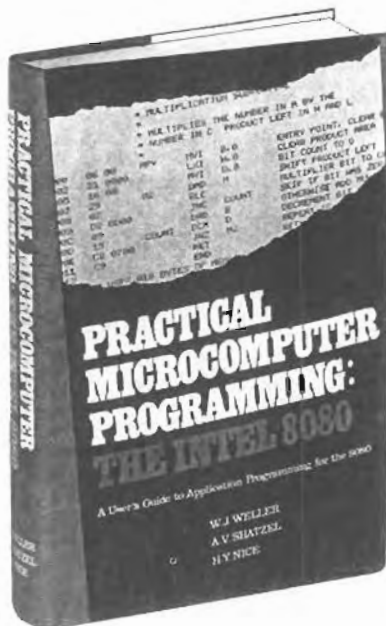
One assumes the role of a detective in this simulation. A thief has just stolen a diamond from a museum. Five suspects must be questioned to determine the thief. A floor plan of the museum and a chart indicating suspects and times are provided.

# If you need real results from your 8080 or 6800 based system

## Then scan this list of topics . . .

- binary arithmetic
- logical operations
- organization of a computer
- referencing memory
- carry and overflow
- multiple precision arithmetic
- loops
- shifting
- software multiplication and division
- number scaling
- floating point arithmetic
- stack pointer usage
- subroutines
- table and array handling
- number base conversions
- BCD arithmetic
- trigonometry
- random number generation
- programming of the 6820 PIA
- programmed input/output
- control of complex peripherals
- programming with interrupts
- a software time of day clock
- multiple interval timers in software
- data transmission under interrupt control
- polling
- debugging techniques
- patching a binary program
- full source listing of a debug program . . .

Order now . . . Start getting real results from your 8080 or 6800 based systems. 



Every one of these topics and many, many more are discussed in the **Practical Microcomputer Programming** books. In chapter after chapter and scores of formal program examples, the basic skills of assembly language programming are developed step by step. The examples are real and have been tested and proven. They run, and more important, they teach. If you're tired of generalities, reproductions of manufacturers data sheets and books with examples that don't run, then there is only *one* place to go, the **Practical Microcomputer Programming** series from Northern Technology Books. At \$21.95 each they are the best bargain in programming information available anywhere.

**Northern Technology Books** Box 62, Evanston, IL 60204

- Practical Microcomputer Programming: The Intel 8080 \$21.95
- Practical Microcomputer Programming: The 6800 \$21.95

- check on US bank enclosed
- money order enclosed

Illinois residents add \$1.10 state sales tax.

Foreign orders add air mail postage if desired (8 kg).

Please type or print

Name \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

Prepaid orders only

# 10,000 Visit MINI/MICRO '77

By Marsha Sutton

The 1977 MINI/MICRO trade show was held in Anaheim, California on December 6—8, and the organizers say it was quite a success. Total attendance for the show was 9,917, falling just short of the projected 10,000. The attendance figure includes 300 booth personnel, representing nearly 180 companies from across the nation.

The show was open for three full days, during which time guests could view the exhibits as well as attend the technical sessions. The 20 sessions consisted of 90 speakers, and the program presentations ranged from formal papers to panel discussions. Topics included such areas as how to begin a new company, small business systems, microcomputers to help save energy, and trends in mini-micro software, small disk memories, CRT terminals, and printer development.

Organizers of the conference were pleased with the guests at the show, claiming many prominent visitors from Japan, Canada, and several from Europe. Included among the guests was LEON RUSSELL, the popular rock performer. He appeared on the second day of the show, looking very conspicuous in his sunglasses and cowboy hat. When asked if he intended to purchase a home computer some day, he replied that he already owns a small system (although he would not reveal the type). He did say that his applications include bookkeeping and synthesizing of music. He was not, however, using his system for composition, amplification, or production of sound effects, which are some of the latest innovative musical applications for microcomputers.

Pertec Computer Corporation appeared at the show in full force with two booths, one for the Microsystems Division and the other representing the Pertec Division. The Pertec Division booth displayed magnetic tape transports and fixed, cartridge, and flexible disk drives in an attractive booth design. The Microsystems Division (MSD) booth was also an impressive display of both MITS® and iCOM® Products.

PCC's Microsystems Division presented several new products at the Anaheim show, all aimed at enhancing and supporting the existing product

line. One of these products is the MITS 300, a microcomputer-based integrated business system. The MITS 300 is available in two configurations, both of which are supplied with complete hardware and software.

Visitors at the MSD booth were also introduced to iCOM's Attaché™

microcomputer. The Attaché is a desktop computer that is built around the 8080 MPU. Its basic configuration includes a CPU board, keyboard, video board, and turnkey monitor board.

MSD has also recently introduced the FDOS-III, which is iCOM's new Floppy Disk Operating System for



*The MSD booth, with MITS Business System admirers on the left and onlookers of the time-sharing BASIC demonstration*



*MINI/MICRO in full swing, with PCC's Pertec Division booth at the end of the aisle*





*The new MITS Business System*



*ICOM's latest addition—the Attaché*

microcomputers. Compatible with FDOS-III is DEBBi™ (Disk Extended BASIC by iCOM), a comprehensive BASIC language system that is easy to use and offers expanded capabilities. Demonstrations of MITS' Time-Sharing BASIC were given regularly all three days, attracting a large number of people to the booth. A variety of other MSD products was also on display for the guests of the show.

The MINI/MICRO '78' show will be held in Philadelphia on April 18-20, and 40 percent booth space is already reserved. The conference organizers are anticipating another successful show for 1978 and are projecting increased interest and attendance for the future as microcomputers reduce in price and gain in popularity.



*A crowd around the MITS OEM Products display*

# Introducing the Compact Attaché™ Computer



*The Attaché is an attractive desktop computer that was recently introduced by PERTEC COMPUTER CORPORATION's Microsystems Division.*

Pertec Computer Corporation's Microsystems Division recently introduced a powerful desktop computer called the Attaché™. The Attaché weighs 25 pounds and is built around the 8080 MPU. Its basic configuration includes a CPU board, video board, turnkey monitor board, and a full 64-character alphanumeric ASCII keyboard.

Standard features of the Attaché include Light Emitting Diode (LED) indicators for on/off and systems status, a reset switch for return to the PROM monitor, and a monitor PROM that controls computer operation from the keyboard. Also standard is a video output jack for providing full upper and lower case character generation, 16 lines of 64 characters each, and a choice of black on white or white on black character display with cursor control.

The Attaché's circuitry uses the S-100 bus configuration with a 10-slot

board capability. Also standard with the system on the turnkey board is 1K RAM with extra sockets for three 256-byte PROMs. An Audio Cassette Recorder (ACR) SIO board is another of the Attaché's standard features, as is a 16K Dynamic RAM Memory Board that uses less than three Watts of power and has an access time of 350 nanoseconds.

In addition to its list of standard features, the Attaché also offers high reliability due to forced air cooling over the vertically mounted cards. Its power supply provides 10V at 10A (regulated to 5V on boards) with pre-regulated plus/minus 18V at 2A. The Attaché also features greater possible expansion, because only three of the ten slots are used by required boards (the CPU, video, and turnkey monitor), leaving seven slots for expansion.

Floppy disk systems and software, including ICOM's® FD3712 Dual Disk

Desk Top IBM-formatted system or the FD2411 Microfloppy with interface supported by FDOOS-III and DEBBIT™ (Disk Extended BASIC by ICOM), are available as options for the Attaché. Other options include an audio cassette recorder (KCACR) board, 110-9600 baud RS232 port, 16K byte memory board expansion for up to 64K of usable RAM, a 16K BASIC ROM board with autojump start, and CSave and CLoad cassette routines that are included in BASIC. A ten-key pad for high-speed data entries in business or statistical applications and plug-in compatibility for many versatile S-100 boards are additional options.

The Attaché is contained in a stylish white cameo case and is priced below competitive systems. The Attaché business computer is available at the more than 40 MITS Computer Centers across the continent.

# Machine Language to BASIC Converter

By Richard Ranger

An annoying but necessary step in using the machine language interface, DEFUSR, in MITS<sup>®</sup>BASIC is the conversion of the machine language program into POKE statements within the calling BASIC program. Using the following program, MITS BASIC users may utilize the machine language subroutines to enhance the capabilities of their computers.

Machine language subroutines that can be interfaced to BASIC through the use of DEFUSR have been written for a number of different functions, from multi-precision addition to fast analog to digital conversion and storage. A few of these programs have appeared in *Computer Notes*, while others are scattered throughout the operation and checkout procedures of various manuals for MITS peripherals. Generally, memory size is limited during initialization. The machine language program is placed above this initialization limit, so that any operation within this subroutine will not affect BASIC. This routine is normally accessed using the DEFUSR function of MITS BASIC, and, since the syntax for this statement varies from version to version, you should refer to the manual to find the correct syntax for calling the DEFUSR function subroutine.

The purpose of this program is to eliminate the need to toggle in the machine language subroutine each time a new routine is used. Without this program, it would be necessary to toggle in the subroutine before calling it with any BASIC program or to convert each octal location and instruction to decimal and then into a statement of the form:

POKE (address), (instruction).

Using the following procedure, the machine can write its own BASIC program that contains all the necessary POKES to duplicate the machine language subroutine. By running this POKE program, the machine language subroutine is quickly POKEd into position before it is needed by the main or calling program.

If you are using disk BASIC, proceed according to the following instructions. First, bring up BASIC, initializing with at least one sequential file and limiting its size so that your particular machine language program will reside in its appropriate location (usually above the BASIC interpreter).

You must either toggle in the machine language or use any method available to enter the machine language program initially, so the converter program will be able to use the PEEK function of BASIC to acquire the data. After this has been accomplished, LOAD the converter program, and RUN it. At this time, you will be required to enter the beginning and ending locations of the machine language program (in decimal) and a temporary file name for the POKE program. The converter will begin PEEKing the locations containing the machine language routine and will create a string comprised of a line number, the characters "POKE", ",", ":", ",", the address, and the contents of the PEEKed location. This string of characters is then written on the disk in ASCII under the temporary file name AND. AND may be merged with any other program which does not contain the same line numbers.

This method of creating machine language subroutines that can be interfaced with BASIC allows you to write several different routines, merge their corresponding POKE programs into a larger BASIC program, and call them much the same as BASIC subroutines are called.

If you do not have a disk but still require the use of machine language subroutines, the temporary POKE program must be written in ASCII but placed on a medium other than floppy

disk. This problem may be resolved in two different ways, depending upon whether you have access to a teletype with a paper tape punch and reader or if you are limited to a cassette recorder and mag tape.

If you do have access to a teletype, load the machine language program as before and delete lines 30, 35, and 140 from the converter program. Line 110 of the converter must be changed to read: 110 PRINT T\$. Enter the converter program, make all the necessary changes, type RUN, turn on the paper tape punch, and type a carriage return. The computer will then print the POKE program on paper tape. After this has been done, this ASCII paper tape may be merged with the main BASIC program by loading the main program and then reading in the paper tape program through the paper tape reader. Again, note that the line numbers of the POKE program and the main program must be different.

If you do not have access to a teletype or a floppy disk, your POKE program must be saved in ASCII on a cassette recorder. To accomplish this, load the machine language as before and be sure that BASIC has been initialized with a "C" when WANT SIN-COS-TAN was asked. (This write-up assumes that the reader is using a version of BASIC that incorporates the CONSOLE command.)

Delete lines 30 and 35, and change or add the following lines accordingly:

*Continued on page 28*

## Easy Floppy Disk Alignment Check - continued from page 7

```
10 PRINT:PRINT"PIP - VER 4.0"
20 CLEAR 0:X=FRE(0)-1500:IF X<0 THEN CLEAR 600 ELSE IF X>32000 THEN
CLEAR 32000 ELSE CLEAR X
30 DIM T2(15):F0RY=0:T015:T2(Y)=-1:NEXT Y:PRINT"*":LINEINPUT B5
40 IF B5="" THEN CLEAR 200:END
50 IF LEN(B5)>3 THEN C5=RIGHT$(B5,LEN(B5)-3) ELSE C5=B5
60 B5=LEFT$(B5,3)
70 IF B5="DAT" THEN 680
80 IF B5="COP" THEN 870
90 IF B5="LIS" THEN 800
100 IF B5="CNV" THEN 1040
110 IF B5="DIR" THEN F=-1:G0T0270
120 IF B5="SRT" THEN F=0:DIMA$(255):G0T0270
130 IF B5<>"INI" THEN PRINT"ERR":G0T020
140 G0SUB 760
150 A5=STRING$(137,0):MID$(A5,136,1)=CHR$(255)
160 F0RT=6T076
170   F0R S=0 T0 31
180     MID$(A5,1,2)=CHR$(T)+CHR$(S*17)AND31)
190   G0SUB 600:DSK0$ A5,S
200 NEXT S,T
210 T=70:G0SUB 600 'DIRECTORY TRACK
220 A5=CHR$(70)+CHR$(0)+CHR$(0)+CHR$(128)+CHR$(127)+CHR$(0)
230 A5=A5+CHR$(0)+CHR$(255)+STRING$(127,0)+CHR$(255)
240 DSK0$A5,0
```

*Continued on page 28*

```

250 PRINT:PRINT"DONE"
260 GOTO20
270 GOSUB760:OPEN"0",1,".....RR",A
280 PRINT#1,1:CLOSE1:KILL".....RR",A
290 PRINT
300 PRINT"DIPECTORY DISK":A
310 PRINT:I=0
320 FORS=0T031
330 A$=DSK1$(17*SAND31)
340 A$=LEFT$(A$,135)
350 A$=RIGHT$(A$,128)
360 FOR T=0 T0 7
370 B$=LEFT$(A$, (T+1)*16)
380 B$=RIGHT$(B$,16)
390 N$=LEFT$(B$,8)
400 B$=RIGHT$(B$,8)
410 X=ASC(B$):B$=RIGHT$(B$,7):Y=ASC(B$)
420 B$=RIGHT$(B$,6):Z=ASC(B$)
430 IFASC(N$)=0THEN470
440 IFASC(N$)=255THEN490
450 R$="S":IFZ<>2THENR$="R"
460 IF F THENPRINTN$;" ";R$;" ";X;" ";Y ELSE A$(I)=N$+" "+
R$+" "+STR$(X)+" " +STR$(Y):I=I+1
470 NEXTT
480 NEXTS
490 IF F OR I=0 THEN PRINT:GOTO 20
500 IF I=1 THEN 560
510 SW=0
520 FOR J=0 T0 1-2
530 IF A$(J)>A$(J+1) THEN SWAP A$(J),A$(J+1):SW=-1
540 NEXT
550 IF SW THEN 510
560 FOR J=0 T0 1-1
570 PRINT A$(J)
580 NEXT
590 PRINT:GOTO20
600 IFT2(A)<>-1THEN640
610 IF(INP(8)AND64)=0THEN T2(A)=0:GOTO640
620 WAIT8,2,2:OUT9,2
630 GOTO610
640 IFT2(A)=1THENRETURN
650 D=1:IFT2(A)>1THEN D=2
660 WAIT8,2,2:OUT9,D:T2(A)=T2(A)-2*(D-1.5)
670 GOTO640
680 INPUT"TRACK":T:IF T<0 THEN 20 ELSE INPUT"SECTOR":S
690 GOSUB760:GOSUB600
700 A$=DSK1$(S):FORI=0T0LEN(A$)-1
710 T1$=MID$(ASC(RIGHT$(A$,LEN(A$)-I))
720 T2$=LEFT$( " 000",5-LEN(T1$))+T1$:PRINT T2$
730 IF I MOD 8=7 THEN PRINT
740 NEXT I:PRINT
750 GOTO 680
760 A=VAL(C$)
770 IFA<0ORA>15THENPRINT"ERR":GOTO20
780 OUT8,128:OUT8,A
790 RETURN
800 GOSUB760
810 C$=RIGHT$(C$,LEN(C$)-1+(A>9)):IFASC(C$)<>4054THENPRINT"ERR":GOTO20
820 C$=RIGHT$(C$,LEN(C$)-1)
830 OPEN"1",1,C$,A
840 IFEOF(1)THENCLOSE1:GOTO20
850 LINEINPUT#1,A$
860 PRINTA$:GOTO840
870 GOSUB760:B=A
880 C$=RIGHT$(C$,LEN(C$)-1+(A>9)):IFASC(C$)<>4054THENPRINT"ERR":GOTO20
890 C$=RIGHT$(C$,LEN(C$)-1):GOSUB760:C=A
900 PRINT"FROM ";B;" TO ";C;
910 INPUTA$:IFASC(A$)<>ASC("Y")THEN20
920 FORT=0T076
930 OUT8,128:OUT8,C
940 A=C:GOSUB600:OUT8,128:OUT8,B:A=B:GOSUB600
950 FORS=0T031
960 OUT8,128:OUT8,B:B$=DSK1$(S)
970 F$=DSK1$(S):IFF$<>B$THENPRINT"REREAD":GOTO960
980 OUT8,128:OUT8,C
990 DSK0$B$,S:C$=DSK1$(S):IFC$<>B$THENPRINT"REWRITE":GOTO960
1000 NEXTS
1010 NEXTT
1020 PRINT"DONE"
1030 GOTO20
1040 GOSUB 760 'ENABLE DISK
1050 FOR T=6 T0 76
1060 GOSUB 600 'POSITION TO TRACK T
1070 FOR S=0 T0 31
1080 A$=DSK1$(S):IF ASC(MID$(A$,3,1))<>0 THEN 1120
1090 IF MID$(A$,136,1)=CHR$(255) THEN 1120
1100 MID$(A$,136,1)=CHR$(255)
1110 DSK0$A$,S
1120 NEXT S
1130 NEXT T:GOTO 20
0K

```

```

CHANGE LINE 100 TO READ:
100 T$=K$+F$+A$+S$+X$+D$+P$+B$+S$+Y$+O$+P$+C$+S$+Z$+CHR$(13)
CHANGE LINE 110 TO READ:
110 FOR J=1 TO LEN(T$):PR=ASC(MID$(T$,J,1))
ADD LINE 112:
112 WAIT 6,128,128:OUT7,PR:NEXT J
CHANGE LINE 140 TO READ:
140 REM THIS IS THE CONSOLE COMMAND FOR A 2SIO
ADD THE FOLLOWING LINES:
142 T$="CONSOLE 16,0"+CHR$(13)
144 FOR I=1 TO 100:REM A SLIGHT DELAY
146 NEXT I
148 FOR K=1 TO LEN(T$)
150 PR=ASC(MID$(T$,K,1))
152 WAIT 6,128,128:OUT7,PR
154 NEXT K

```

At this time, start the cassette recorder (record mode), and, after a few seconds, type RUN, followed by a carriage return. The added parts of the program will allow the computer to place the POKE program on cassette tape and will follow it with a CONSOLE command to the main terminal in use. If an I/O card other than a 2SIO is used for this terminal, line 142 must be changed in accordance with the appropriate console register setting for that particular I/O card (see page 34 of your BASIC manual). After the POKE program has been made on cassette, it may be merged with the main BASIC program by first LOADING the main program into the computer, then typing CONSOLE 6, 3, followed by a carriage return. The computer will now take in data from the input port #7, and, when all of the POKE program has been entered, it will CONSOLE back to the main terminal. (Note again that the line numbers of the POKE program must be different from the line numbers of the main program.)

In all of the procedures just outlined, the entire program, main BASIC plus the POKE program, may be saved together as one main program after they are both in the computer's text buffer. The unmodified conversion program set up for disk BASIC users is also given in this article.

Program on Page 29

## About The Author

Richard Ranger, a MITS engineering technician, is a Navy veteran who worked in airborne reconnaissance. He is currently studying at the University of New Mexico for a degree in Electrical Engineering.

Machine Language to BASIC Converter - continued

```

5 CLEAR 500
10 INPUT "START LOCATION";TRT
20 INPUT "STOP LOCATION";STP
30 LINE INPUT "FILE NAME";N$
35 OPEN "O",1,N$,0
40 K=10
50 FOR I=TRT TO STP STEP 3
60 X$=STR$(PEEK(I));Y$=STR$(PEEK(I+1));Z$=STR$(PEEK(I+2))
70 A$=STR$(I);B$=STR$(I+1);C$=STR$(I+2)
80 K$=STR$(K)
90 P$="POKE" S$=",";":O$="."
100 T$=K$+P$+A$+B$+S$+X$+O$+Y$+B$+S$+V$+O$+P$+C$+S$+Z$
110 PRINT #1,T$
120 K=K+10
130 NEXT I
140 CLOSE 1
    
```

More on the KCACR - continued from page 8

```

FEF5$
NAM PUNKCR
OPT NOG
OUTCH EQU $FDF5
OUT2H EQU $FDE3
CRLF EQU $FFAB
STACK EQU $3FFF
ORG $00F3
FCB $FF
ORG $4000
    
```

\* ENTRY LINE FOR BASIC V1.0 R3.2

```

LDX #1AB2
BRA START
    
```

\* ENTRY LINE FOR EDITOR R1.0

```

LDX #3090B
BRA START
    
```

\* ENTRY LINE FOR ASSEMBLER/EDITOR R1.0

```

LDX #51C81
START STX HERE
    
```

```

LDS #STACK
BSR LEDTRL
LDX #0
STX BEGADR
LDX #5E6
BSR PUN
LDX #5180
STX BEGADR
FCB $CE
HERE FCB 0,0
BSR PUN
LDX #EOF
BSR PMESS
BSR LEDTRL
JMP CRLF
LEDTRL CLR A
CLR B
LEDI JSR OUTCH
DEC A
BNE LEDI
RTS
PUN STX LASADR
PUNO LDX #FORM
BSR PMESS
LDA A LASADR+1
SUB A BEGADR+1
LDA B LASADR
SBC B BEGADR
BNE PUN2
CMP A #15
BCS PUN3
PUN2 LDA A #15
PUN3 STA A NUMBYT
ADD A #4
JSR OUT2H
INX
BSR PNCH2
BSR PNCH2
LDX BEGADR
PUN4 BSR PNCH2
DEC NUMBYT
BPL PUN4
STX BEGADR
COM A
JSR OUT2H
DEX
    
```

```

CPX LASADR
BNE PUNO
RTS
SENDIT JSR OUTCH
INX
PMESS LDA B X
BPL SENDIT
RTS
PNCH2 LDA B X
ABA
PSH A
TBA
JSR OUT2H
PUL A
INX
RTS
FORM FCB $D,$A,'S','I,$FF
BEGADR RMB 2
LASADR RMB 2
NUMBYT RMB 1
EOF FCB $D,$A,'S','9,$FF
ORG $00F3
FCB $A3
END
    
```

00001			NAM	PUNKCR	
00002			OPT	NOG	
00003	FDF5	OUTCH	EQU	\$FDF5	
00004	FDE3	OUT2H	EQU	\$FDE3	
00005	FFAB	CRLF	EQU	\$FFAB	
00006	3FFF	STACK	EQU	\$3FFF	
00007	00F3		ORG	\$00F3	
00008	00F3	FF	FCB	\$FF	
00009	4000		ORG	\$4000	
00010			* ENTRY LINE FOR BASIC V1.0 R3.2		
00011	4000	CE 1AB2	LDX	#1AB2	
00012	4003	20 08	BRA	START	
00013			* ENTRY LINE FOR EDITOR R1.0		
00014	4005	CE 090B	LDX	#090B	
00015	4008	20 03	BRA	START	
00016			* ENTRY LINE FOR ASSEMBLER/EDITOR R1.0		
00017	400A	CE 1C81	LDX	#1C81	
00018	400D	FF 4027	START	STX	HERE
00019	4010	8E 3FFF	LDS	#STACK	
00020	4013	8D 20	BSR	LEDTRL	
00021	4015	CE 0000	LDX	#0	
00022	4018	FF 4098	STX	BEGADR	
00023	401B	CE 00E6	LDX	#5E6	
00024	401E	8D 1E	BSR	PUN	
00025	4020	CE 0100	LDX	#100	
00026	4023	FF 4098	STX	BEGADR	
00027	4026	CE	FCB	\$CE	
00028	4027	00	HERE	FCB	0,0
00029	4029	8D 13	BSR	PUN	
00030	402B	CE 409D	LDX	#EOF	
00031	402E	8D 53	BSR	PMESS	
00032	4030	8D 03	BSR	LEDTRL	
00033	4032	7E FFAB	JMP	CRLF	
00034	4035	4F	LEDTRL	CLR A	
00035	4036	5F		CLR B	
00036	4037	BD FDF5	LEDI	JSR	OUTCH
00037	403A	4A		DEC A	
00038	403B	26 FA		BNE	LEDI
00039	403D	39		RTS	
00040	403E	FF 409A	PUN	STX	LASADR
00041	4041	CE 4093	PUNO	LDX	#FORM
00042	4044	8D 3D	BSR	PMESS	
00043	4046	B6 409B	LDA A	LASADR+1	
00044	4049	B0 4099	SUB A	BEGADR+1	
00045	404C	F6 409A	LDA B	LASADR	
00046	404F	F2 4098	SBC B	BEGADR	
00047	4052	26 04		BNE	PUN2
00048	4054	81 10		CMP A	#16
00049	4056	25 02		BCS	PUN3
00050	4058	86 0F	PUN2	LDA A	#15
00051	405A	87 409C	PUN3	STA A	NUMBYT
00052	405D	8B 04		ADD A	#4
00053	405F	8D FDE3		JSR	OUT2H
00054	4062	08		INX	
00055	4063	8D 23		BSR	PNCH2
00056	4065	8D 21		BSR	PNCH2
00057	4067	FE 4098		LDX	BEGADR
00058	406A	8D 1C	PUN4	BSR	PNCH2
00059	406C	7A 409C		DEC	NUMBYT
00060	406F	2A F9		BPL	PUN4
00061	4071	FF 4098		STX	BEGADR
00062	4074	43		COM A	
00063	4075	BD FDE3		JSR	OUT2H
00064	4078	09		DEX	
00065	4079	BC 409A		CPX	LASADR
00066	407C	26 C3		BNE	PUNO
00067	407E	39		RTS	
00068	407F	8D FDF5	SENDIT	JSR	OUTCH
00069	4082	08		INX	
00070	4083	E6 00	PMESS	LDA B	X
00071	4085	2A F8		BPL	SENDIT
00072	4087	39		RTS	
00073	4088	E6 00	PNCH2	LDA B	X
00074	408A	18		ABA	
00075	408B	36		PSH A	
00076	408C	17		TBA	
00077	408D	BD FDE3		JSR	OUT2H
00078	4090	32		PUL A	
00079	4091	08		INX	
00080	4092	39		RTS	
00081	4093	0D	FORM	FCB	\$D,\$A,'S','I,\$FF
00082	4098	0002	BEGADR	RMB	:2
00083	409A	0002	LASADR	RMB	:2
00084	409C	0001	NUMBYT	RMB	:1
00085	409D	0D	EOF	FCB	\$D,\$A,'S','9,\$FF
00086	00F3		ORG		\$00F3
00087	00F3	03		FCB	\$03
00088			END		

TOTAL ERRORS 00000

ENTER PASS

Continued on page 30

More on the KCACR - continued

```

S00B000050594E4B43522020E1
SI0400F3FF09
SI1E400CE1AB22008CE09082003CE1C81FF40279E3FFF8D20CE0000FFA098BE5
SI1E401BCE00E68D1ECE0100FF4098CE00008D13CE409D8D538D037EFFAB4F01
SI1E40365FBDFDF54A26FA39FF409ACE40938D3DB64098B004099F6409AF2405A
SI1E405198260481102502860FB7409C(B04BD)FDE3088D238D21FE40983D1C9D
SI1E405C7A409C2AF9FF409843BDFE309BC409A26C339BDFDF508E6002AF885
SI14408739E600133617BDFDE3320839000A5331FFF3
SI04009D000A5339FF78
SI0400F30305
S9030000FC
    
```

KCACR MONITOR  
INVERSE ASSEMBLY BY DLJ

```

***** IN ROUTINE *****
FD00 8D 60      BSR ($60) $FD62  GO POLE FOR CHARACTER
FD02 C0 53      SUB B #'S        IS IT THE LETTER 'S'
FD04 26 FA      BNE ($FA) $FD00  YES, GO BACK
FD06 8D 5A      BSR ($5A) $FD62  POLE FOR NEXT CHARACTER
FD08 C1 39      CMP B #'9        IS IT A '9'
FD0A 27 62      BEQ ($62) $FD6E  IF YES, DONE
FD0C C1 31      CMP B #'1        IS IT A '1'
FD0E 26 F0      BNE ($F0) $FD00  BACK TO START IF NOT
FD10 4F         CLR A            ZERO CHECKSUM
FD11 8D 38      BSR ($38) $FD4B  GET A BYTE
FD13 C0 02      SUB B #$02       ADJUST BYTE COUNT
FD15 D7 F9      STA B $F9        STORE AT BYTECT
FD17 8D 40      BSR ($40) $FD59  GET ADDRESS
FD19 8D 30      BSR ($30) $FD4B  GET DATA BYTE
FD1B 7A 00F9    DEC $00F9        DECREMENT BYTE COUNT
FD1E 27 09      BEQ ($09) $FD29  IF ZERO DONE
FD20 E7 00      STA B $00,X     STORE IT
FD22 E1 00      CMP B $00,X     MEMORY OK?
FD24 26 09      BNE ($09) $FD2F  BRANCH IF NOT
FD26 08         INX             BUMP POINTER
FD27 20 F0      BRA ($F0) $FD19  BACK FOR NEXT CHARACTER
FD29 4C         INC A           INCREMENT CHECKSUM
FD2A 27 D4      BEQ ($D4) $FD00  BRANCH IF ZERO, ALL OK
FD2C C6 43      LDA B #'C       LOAD IN 'C' FOR CHECKSUM ERROR
FD2E 8C         FCB $8C        CPX SKIP
FD2F C6 4D      LDA B #'M       LOAD IN 'M' FOR MEMORY ERROR
FD31 BD FF81    JSR $FF81       DUMP TO OUTCH
FD34 20 FB      BRA ($FB) $FD31  LOOP BACK AGAIN

***** IN HEX *****
FD36 8D 2A      BSR ($2A) $FD62  POLE FOR CHARACTER
FD38 C0 30      SUB B #$30       STRIP ASCII
FD3A 2B F0      BMI ($F0) $FD2C  STOP IF NOT VALID HEX
FD3C C1 09      CMP B #$09       NOT HEX
FD3E 2F 0A      BLE ($0A) $FD4A  NOT HEX
FD40 C1 11      CMP B #$11       NOT HEX
FD42 2B E8      BMI ($E8) $FD2C  NOT HEX
FD44 C1 16      CMP B #$16       NOT HEX
FD46 2E EA      BGT ($EA) $FD2C  NOT HEX
FD48 C0 07      SUB B #$07       GET BCD VALUE
FD4A 39         RTS            RETURN

***** BYTE *****
FD4B 8D E9      BSR ($E9) $FD36  GET A CHARACTER
FD4D 58         ASL B           SHIFT TO HIGH 4 BITS
FD4E 58         ASL B
FD4F 58         ASL B
FD50 58         ASL B
FD51 D7 F8      STA B $F8        STORE IT TEMP
FD53 8D E1      BSR ($E1) $FD36  GET 2ND HEX DIGIT
FD55 DB F8      ADD B $F8        COMBINE DIGITS TO GET BYTE
FD57 1B        ABA            ADD TO CHECKSUM
FD58 39         RTS            RETURN

***** BADDR *****
FD59 8D F0      BSR ($F0) $FD4B  GET HALF OF ADDRESS
FD5B D7 FA      STA B $FA        STORE IT
FD5D 8D EC      BSR ($EC) $FD4B  GET REST OF ADDRESS
FD5F 7E FF68    JMP $FF68        JMP MONITOR AND COMPLETE

***** INCH *****
FD62 F6 F010    LDA B $F010     POLE KCACR FOR FLAG
FD65 56         ROR B           ROTATE INTO B
FD66 25 FA      BCS ($FA) $FD62  BACK AGAIN IF SET
FD68 F6 F011    LDA B $F011     LOAD IN CHARACTER
FD6B C4 7F      AND B #$7F      STRIP ASCII
FD6D 39         RTS            RETURN
FD6E 20 52      BRA ($52) $FDC2  MONITOR RETURN
FD70 0D         FCB $D,$A,'S',$B1  FORM CR/LF/S/-1
    
```

More on the KCACR - continued

```

***** OUT ROUTINE *****
FD74 8D 62 BSR ($62) $FDD8 GET HIGH-ORDER ADDRESS
FD76 DF FD STX $FD STORE IT
FD78 8D 5E BSR ($5E) $FDD8 GET LOW-ORDER ADDRESS
FD7A DF F4 STX $F4 STORE IT
FD7C 8D 47 BSR ($47) $FDC5 GO PUNCH LEADER
FD7E CE FD6F LDX #$FD6F LOAD FORM POINTER
FD81 08 INX BUMP POINTER
FD82 E6 00 LDA B $00,X LOAD CHARACTER
FD84 8D 6F BSR ($6F) $FDF5 GO PUNCH IT
FD86 2A F9 BPL ($F9) $FD81 BACK FOR MORE
FD88 96 F5 LDA A $F5 SUBTRACT LOW ORDER BYTES
FD8A 90 FE SUB A $FE
FD8C D6 F4 LDA B $F4 SUBTRACT HIGH ORDER BYTES
FD8E D2 FD SBC B $FD
FD90 26 04 BNE ($04) $FD96 LOTS MORE TO PUNCH
FD92 81 0E CMP A #$0E LESS THAN 15 TO PUNCH
FD94 25 02 BCS ($02) $FD98 BRANCH IF DONE
FD96 86 0D LDA A #$0D NO, SO PUNCH 15
FD98 97 FF STA A $FF STORE A BUFFER-NUMBYT
FD9A 8B 04 ADD A #$04 ADJUST # BYTES
FD9C 8D 45 BSR ($45) $FDE3 PUNCH 2HEX
FD9E CE 00FD LDX #$00FD LOAD BEGADR POINTER
FDA1 8D 2B BSR ($2B) $FDCE PUNCH 2
FDA3 8D 29 BSR ($29) $FDCE PUNCH 2
FDA5 DE FD LDX $FD LOAD BEGADR
FDA7 8D 25 BSR ($25) $FDCE PUNCH DATA
FDA9 7A 00FF DEC $00FF DEC NUMBYT
FDAC 2A F9 BPL ($F9) $FDA7 BACK IF NOT DONE
FAE DF FD STX $FD STORE ADDRESS
FDB0 43 COM A COMPLIMENT CHECKSUM
FDB1 8D 30 BSR ($30) $FDE3 PUNCH 2HEX
FDB3 09 DEX DECRIMENT ADDRESS
FDB4 9C F4 CPX $F4
FDB6 26 C6 BNE ($C6) $FD7E BACK IF NOT DONE
FDB8 C6 53 LDA B #'S LOAD 'S'
FDBA 8D 39 BSR ($39) $FDF5 PUNCH IT
FDBC C6 39 LDA B #'9 LOAD '9'
FDBE 8D 35 BSR ($35) $FDF5 PUNCH IT
FDC0 8D 03 BSR ($03) $FDC5 PUNCH LEADER
FDC2 7E FFAB JMP $FFAB BACK TO MONITOR CRLF
***** LEADER *****
FDC5 86 28 LDA A #$28 LOAD LOOP COUNT
FDC7 5F CLR B CLEAR FOR NULLS
FDC8 8D 2B BSR ($2B) $FDF5 GO PUNCH
FDCA 4A DEC A DECRIMENT LOOP
FDCB 26 FB BNE ($FB) $FDC8 BACK IF NOT DONE
FDCD 39 RTS RETURN
FDCE E6 00 LDA B $00,X GET POINTED CHACTER
FDD0 1B ABA ADD TO CHECKSUM
FDD1 36 PSH A SAVE IT
FDD2 17 TBA TRANSFER
FDD3 8D 0E BSR ($0E) $FDE3 PUNCH IT
FDD5 32 PUL A RETURN CHECKSUM
FDD6 08 INX BUMP ADDRESS
FDD7 39 RTS RETURN
***** ADDRESS *****
FDD8 8D FF82 JSR $FF82 SEND OUT A SPACE
FDDB C6 3F LDA B #'? LOAD A '?'
FDDD 8D FF81 JSR $FF81 TYPE IT
FDE0 7E FF62 JMP $FF62 JMP BADDR IN MONITOR
***** OUT2H *****
FDE3 16 TAB COPY BYTE TO B
FDE4 54 LSR B SHIFT RIGHT
FDE5 54 LSR B
FDE6 54 LSR B
FDE7 54 LSR B
FDE8 8D 01 BSR ($01) $FDEB OUTPUT FIRST DIGIT
FDEA 16 TAB BYTE INTO B
FDEB C4 0F AND B #$0F GET RID OF LEFT DIGIT
***** OUT4R *****
FDED CB 30 ADD B #$30 MAKE IT ASCII
FDEF C1 39 CMP B #'9 IS IT A NUMBER?
FDF1 23 02 BLS ($02) $FDF5
FDF3 CB 07 ADD B #$07 IF ITS A LETTER ADD 7
***** OUTCH *****
FDF5 37 PSH B SAVE CHARACTER
FDF6 F6 F010 LDA B $F010 KCACR CLEAR?
FDF9 2B FB BMI ($FB) $FDF6 BACK IF NOT
FDFB 33 PUL B REGAIN CHARACTER
FDFF F7 F011 STA B $F011 OUT TO KCACR
FDF5 39 RTS RETURN

```

LIST

```

5 CLEAR 250
10 PRINT"HI! I'M A COMPUTER. MY NAME IS HAL."
20 INPUT"WHAT'S YOURS(TYPE YOUR NAME AND HIT THE RETURN KEY)";A$
30 PRINT"WELL ";A$;" A COMPUTER CAN DO A LOT OF THINGS. FOR INSTANCE,"
40 PRINT"WE ARE A SUPER CALCULATOR. LET'S TRY ONE. WE'LL TRY AN EASY"
50 PRINT"ONE FIRST. WOULD YOU LIKE TO ADD, SUBTRACT, MULTIPLY, DIVIDE"
60 PRINT"OR FIND A SQUARE OR SQUARE ROOT? (TYPE YOUR CHOICE AND HIT"
70 PRINT"RETURN)"
90 INPUT B$
100 IF B$="DIVIDE"THEN160
110 IF B$="MULTIPLY"THEN230
120 IF B$="ADD"THEN260
130 IF B$="SUBTRACT"THEN290
132 IF B$="SQUARE ROOT"THEN650
134 IF B$="SQUARE"THEN670
140 PRINT"I'M SORRY, I DON'T UNDERSTAND ";B$;". PLEASE USE ADD,"
150 PRINT"SUBTRACT, MULTIPLY, DIVIDE, SQUARE ROOT OR SQUARE.":GOTO 90
160 PRINT"FIRST THE NUMBER YOU ARE DIVIDING."
165 PRINT"NOT OVER 16 DIGITS, PLEASE.":INPUT A#
170 INPUT"NOW THE DIVISOR";B#:C#=A#/B#
180 PRINT"THE ANSWER IS";C#
190 INPUT"TRY ANOTHER (YES OR NO)";CS
200 IF LEFT$(CS,1)="Y"THEN90
210 IF LEFT$(CS,1)="N"THEN320
220 PRINT"I'M SORRY, I DON'T UNDERSTAND ";C$;". PLEASE USE YES OR NO.":
GOTO190
230 INPUT"THE FIRST NUMBER (NOT OVER 16 DIGITS)";A#
240 INPUT"THE SECOND";B#:C#=A#*B#:GOTO180
260 INPUT"THE FIRST NUMBER (NOT OVER 16 DIGITS)";A#
270 INPUT"THE SECOND";B#:C#=A#+B#:GOTO180
290 INPUT"THE NUMBER YOU ARE SUBTRACTING FROM (NOT OVER 16 DIGITS)";A#
300 INPUT"THE NUMBER YOU ARE SUBTRACTING";B#:C#=A#-B#:GOTO180
320 PRINT"HAD ENOUGH ARITHMATIC ";A$;" HUH? OF COURSE I CAN DO MORE"
330 PRINT"COMPLICATED MATH, TOO. BUT ENOUGH OF THAT. TELL YOU WHAT."
340 PRINT"TYPE ME A SENTENCE.":C=0
350 LINE INPUT B$
360 PRINT"NOW I'LL TELL YOU HOW MANY THERE ARE OF ANY LETTER IN THE"
370 PRINT"SENTENCE."
375 INPUT"WHAT LETTER SHOULD I COUNT";C$
380 IF LEN(CS)>1 THEN PRINT"ONLY ONE CHARACTER, PLEASE.":GOTO375
385 IF CS=>"A" AND CS<="Z"THEN 395
390 PRINT"PLEASE, A LETTER.":GOTO375
395 FOR X=1 TO LEN(B$):IF MID$(B$,X,1)=C$THEN C=C+1
396 NEXT
400 PRINT"THERE ARE";C;" ";C$;"'S IN ";B$
460 PRINT"HOW BOUT THEM APPLES? NOW LET'S PLAY A SIMPLE NUMBER"
470 PRINT"GUESSING GAME. I'LL CHOOSE A NUMBER BETWEEN 1 AND 100."
480 PRINT"YOU TELL ME WHAT YOU THINK IT IS. I'LL TELL YOU IF"
490 PRINT"YOU ARE TOO HIGH OR TOO LOW OR CORRECT."
520 A=INT(99*RND(1)+1):C=0
530 PRINT"OK, I'VE GOT A NUMBER."
540 INPUT"YOUR GUESS";B
550 IF B>ATHENPRINT"TOO HIGH":C=C+1:GOTO540
560 IF B<ATHENPRINT"TOO LOW":C=C+1:GOTO540
570 PRINT"YOU GUESSED IT - IN";C;" TRIES!"
580 INPUT"TRY AGAIN";B$
590 IF LEFT$(B$,1)="Y"THEN 520
600 IF LEFT$(B$,1)="N"THEN 640
610 PRINT"SORRY, I DON'T UNDERSTAND ";B$;". PLEASE USE YES OR NO.":GOTO5
80
640 PRINT"NOW, WASN'T THAT MARVELOUS ";A$;"? AND SO ENDS"
645 PRINT"THE DEMONSTRATION.":END
650 INPUT"THE NUMBER YOU WISH TO FIND THE ROOT OF";A#
660 C#=SQR(A#):GOTO180
670 INPUT"THE NUMBER YOU WISH TO SQUARE";A#
680 C#=A#*A#:GOTO 180
OK

```



A BASIC Memory Test - continued from page 16

```

RUN
STARTING ADDRESS? 28672
FINISHING ADDRESS? 28675
COMPLETE OR PARTIAL ANALYSIS (1=COM,0=PART.)? 0
TEST WORD #? 0
28672      0          64          ERROR
          28672      64
          28673      0          64          ERROR
          28673      64
          28674      0          64          ERROR
          28674      64
          28675      0          64          ERROR
          28675      64
OK
    
```

```

RUN
STARTING ADDRESS? 28672
FINISHING ADDRESS? 28675
COMPLETE OR PARTIAL ANALYSIS (1=COM,0=PART.)? 0
TEST WORD #? 255
28672      64
28673      64
28674      64
28675      64
OK
    
```

```

RUN
STARTING ADDRESS? 28672
FINISHING ADDRESS? 28672
COMPLETE OR PARTIAL ANALYSIS (1=COM,0=PART.)? 1
28672      0          64          ERROR
          1          65          ERROR
          2          66          ERROR
          3          67          ERROR
          4          68          ERROR
          5          69          ERROR
          6          70          ERROR
          7          71          ERROR
          8          72          ERROR
          9          73          ERROR
         10         74          ERROR
         11         75          ERROR
         12         76          ERROR
         13         77          ERROR
         14         78          ERROR
         15         79          ERROR
         16         80          ERROR
         17         81          ERROR
         18         82          ERROR
         19         83          ERROR
         20         84          ERROR
         21         85          ERROR
         22         86          ERROR
         23         87          ERROR
         24         88          ERROR
    
```

```

25          89          ERROR
26          90          ERROR
27          91          ERROR
28          92          ERROR
29          93          ERROR
30          94          ERROR
31          95          ERROR
32          96          ERROR
33          97          ERROR
34          98          ERROR
35          99          ERROR
36         100          ERROR
37         101          ERROR
38         102          ERROR
39         103          ERROR
40         104          ERROR
41         105          ERROR
42         106          ERROR
43         107          ERROR
44         108          ERROR
45         109          ERROR
46         110          ERROR
47         111          ERROR
48         112          ERROR
49         113          ERROR
50         114          ERROR
51         115          ERROR
52         116          ERROR
53         117          ERROR
54         118          ERROR
55         119          ERROR
56         120          ERROR
57         121          ERROR
58         122          ERROR
59         123          ERROR
60         124          ERROR
61         125          ERROR
62         126          ERROR
63         127          ERROR
128         192          ERROR
129         193          ERROR
130         194          ERROR
131         195          ERROR
132         196          ERROR
133         197          ERROR
    
```

Continued on page 34

A BASIC Memory Test - continued

134	193	ERROR	179	243	ERROR
135	199	ERROR	180	244	ERROR
136	200	ERROR	181	245	ERROR
137	201	ERROR	182	246	ERROR
138	202	ERROR	183	247	ERROR
139	203	ERROR	184	248	ERROR
140	204	ERROR	185	249	ERROR
141	205	ERROR	186	250	ERROR
142	206	ERROR	187	251	ERROR
143	207	ERROR	188	252	ERROR
144	208	ERROR	189	253	ERROR
145	209	ERROR	190	254	ERROR
146	210	ERROR	191	255	ERROR
147	211	ERROR	28672	64	
148	212	ERROR	OK		
149	213	ERROR			
150	214	ERROR			
151	215	ERROR			
152	216	ERROR			
153	217	ERROR			
154	218	ERROR			
155	219	ERROR			
156	220	ERROR			
157	221	ERROR			
158	222	ERROR			
159	223	ERROR			
160	224	ERROR			
161	225	ERROR			
162	226	ERROR			
163	227	ERROR			
164	228	ERROR			
165	229	ERROR			
166	230	ERROR			
167	231	ERROR			
168	232	ERROR			
169	233	ERROR			
170	234	ERROR			
171	235	ERROR			
172	236	ERROR			
173	237	ERROR			
174	238	ERROR			
175	239	ERROR			
176	240	ERROR			
177	241	ERROR			
178	242	ERROR			

A = starting address  
 F = finishing address  
 E = complete or partial analysis flag  
 Z = test word  
 C = confirms test word Z written and read from memory  
 B = value of contents of A  
 D = error flag

LIST

```

1 REM **** A BASIC MEMORY TEST ****
2 REM WRITTEN BY DAVID C. CULBERTSON
10 INPUT"STARTING ADDRESS";A:INPUT"FINISHING ADDRESS";F
11 INPUT"COMPLETE OR PARTIAL ANALYSIS (1=COM,0=PART.)";E
12 IF E=0 THEN INPUT"TEST WORD #";Z
13 IF A>F THEN GOSUB 200:INPUT"NEW FINISHING ADDRESS";F:GOTO 13
14 IF Z>255 OR Z<0 THEN GOSUB 300:INPUT"NEW TEST WORD#";Z:GOTO 14
15 IF A=>32768 THEN 30
20 B=PEEK(A)
21 IF E=0 AND A=>3193 THEN GOSUB 50:GOTO 23
22 IF A>8193 THEN GOSUB 100
23 PRINT A,D
24 A=A+1
25 IF A=>32768 THEN 30
26 IF A>F THEN END
27 GOTO 20
30 A=A-(65536):F=F-(65536)
31 E=PEEK(A)
32 IF E=0 THEN GOSUB 50:GOTO 34
33 GOSUB 100
34 PRINT A+65536,B
35 A=A+1
36 IF A=>F THEN END
37 GOTO 31
50 POKE A,Z:C=PEEK(A)
51 IF C=Z THEN 193
52 GOTO 150
100 FOR Z=0 TO 255
110 POKE A,Z:D=PEEK(A)
120 IF D=Z THEN 140
130 IF C=>Z THEN GOSUB 150
140 NEXT Z
145 GOTO 190
150 POKE 1352,15:POKE 1360,19
155 IF B=A THEN 165
159 IF A=>0 AND A<32768 THEN PRINT A,Z,C,"ERROR":PRINT:GOTO 161
160 PRINT A+65536,Z,C,"ERROR":PRINT
161 D=A:GOTO 170
165 PRINT" ",Z,C,"ERROR":PRINT
170 POKE 1352,16:POKE 1360,17
180 RETURN
190 POKE A,B:RETURN
200 PRINT"FINISHING ADDRESS TOO LOW.PLEASE ENTER ";:RETURN
300 PRINT"TEST WORD # TOO LOW.PLEASE INPUT ";:RETURN
OK
  
```



## If your company can afford a pick-up, you can afford your own computer.

It's time to think of a computer like any other cost saving business tool. And the computer every business can afford is the MITS™ 300.

MITS will monitor your inventory, make out your payroll, do your scheduling, ordering, accounts receivable. All the jobs that used to take weeks, days or hours to do, now can be done in a few minutes.

And believe it or not, the MITS 300 microcomputer system is easier to operate than a pick-up truck. Most people can learn its typewriter-like keyboard and BASIC language in a couple of hours using a self-teaching package that makes computer operation simple and easy.

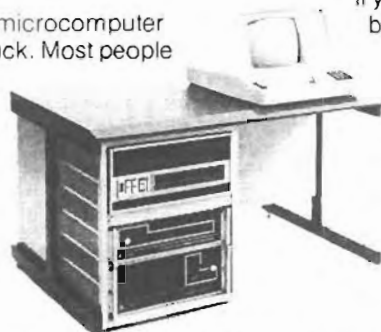
All this, plus a full line of add-on modular hardware, operating software and pre-programmed applications software, makes MITS the ideal microcomputer

for a growing business. After all, it's made by the people who made the first microcomputer.

So put the affordable microcomputer in your office, alongside your typewriters and other office equipment, and see what you've been missing. You'll wonder how you ever ran your business without it.

If you're big enough to be in business, you're big enough for a MITS.

Come in today for a demonstration.



**mits™**

Products of  Perlec Computer Corporation

Built and backed by Perlec Computer Corporation,  
(world's leading independent producer of  
computer peripheral equipment and distributed  
processing and data entry systems.)

**get your MITS up!**

©1978 Perlec Computer Corporation. MITS is a Trademark of Perlec Computer Corporation.

# Visit Your Nearest MITS® Dealer

## ARIZONA

Altair Computer Center  
4941 E. 29th St.  
Tucson, Arizona 85711  
(602) 748-7363

Altair Computer Center  
3815 North Third St.  
Phoenix, Arizona 85012  
(602) 266-1141

## ARKANSAS

JFK Electronics  
3702 JFK Blvd.  
N. Little Rock, Arkansas 72116  
(501) 753-1414

## CALIFORNIA

Computer Kits  
1044 University Ave.  
Berkeley, Calif. 94710  
(415) 845-5300

The Computer Store  
820 Broadway  
Santa Monica, Calif. 90401  
(213) 451-0713

## COLORADO

Gateway Electronics  
2839 W. 44th Ave.  
Denver, Colorado 80211  
(303) 458-5444

Sound-Tronix  
900 Ninth Ave.  
Greeley, Colorado 80631  
(303) 353-1588

Sound-Tronix  
3271 Dillon Dr. Pueblo Mall  
Pueblo, Colorado 81008  
(303) 545-1097

Sound-Tronix  
215 Foothills Pkwy  
Foothills Fashion Mall  
Fort Collins, Colorado 80521  
(303) 221-1700

## FLORIDA

Altair Computer Center of Miami  
7208 N. W. 56th St.  
Miami, Fla. 33166  
(305) 887-7408

Altair Computer Center of  
Orlando  
6220 S. Orange Blossom Trail  
Suite 602  
Orlando, Florida 32809  
(305) 851-0913

## GEORGIA

The Computer Systemcenter  
3330 Piedmont Rd. N.E.  
Atlanta, Ga. 30305  
(404) 231-1691

## ILLINOIS

Chicago Computer Store  
517 Talcott Rd.  
Park Ridge, Illinois 60068  
(312) 823-2388

Chicago Computer Store  
919 N. Sheridan Rd.  
Peoria, Illinois 61614  
(309) 692-7704

Chicago Computer Store  
1 Illinois Center Concourse  
111 E. Wacker Dr.  
Chicago, Illinois 60601

## KANSAS

Advanced Micro Systems Inc.  
5209 W. 94 Terrace  
Prairie Village, Kansas 66207  
(913) 648-0600

## WEST VIRGINIA AND KENTUCKY

The Computer Store  
Suite 5  
Municipal Pkg. Bldg.  
Charleston, W. Virginia 25301  
(304) 345-1360

## MASSACHUSETTS

Mits Computer Center  
36 Cambridge St.  
Burlington, Mass. 01803  
(617) 272-1162

## MICHIGAN

Computer Store of Detroit  
505-507 West 11 Mile Rd.  
Madison Heights, Michigan  
48071  
(313) 545-2225

The Computer Store of  
Ann Arbor  
310 E. Washington St.  
Ann Arbor, Michigan 48104  
(313) 995-7616

## MINNESOTA

The Computer Room  
3938 Beau D. Rue Dr.  
Eagan, Minn. 55122  
(612) 452-2567

## MISSOURI

Gateway Electronics of St. Louis  
8123-25 Page Blvd.  
St. Louis, Mo. 63130  
(314) 427-6116

## NEBRASKA

Altair Computer Center  
611 North 27th St. #9  
Lincoln, Nebraska 68503  
(402) 474-2800

## NEW MEXICO

Computer Shack  
3120 San Mateo N.E.  
Albuquerque, New Mexico 87110  
(505) 883-8282

## NEW YORK

The Computer Store of New York  
55 West 39th St.  
New York, New York 10018  
(212) 221-1404

Micro Systems Store, Inc.  
269 Osborne Rd.  
Albany, New York 12211  
(518) 459-6140

Simplified Business Methods  
19 Rector St.  
New York, New York 10006  
(212) 943-4130

## NORTH CAROLINA

Computer Stores of Carolina  
1808 E. Independence Blvd.  
Charlotte, N. C. 28205  
(704) 334-0242

## OHIO

Altair Computer Center  
5252 North Dixie Drive  
Dayton, Ohio 45414  
(513) 274-1149

Altair Computer Center  
26715 Brook Park Extension  
No. Olmsted, Ohio 44070  
(216) 734-6266

The Computer Store of Toledo  
8 Hillwyck St.  
Toledo, Ohio 43615

## OKLAHOMA

Altair Computer Center  
110 The Annex  
5345 East 41st St.  
Tulsa, Oklahoma 74135  
(918) 664-4564

## OREGON

Altair Computer Center  
8105 S. W. Nimbus Ave.  
Beaverton, Oregon 97005  
(503) 644-2314

## PENNSYLVANIA

Microcomputer Systems, Inc.  
243 West Chocolate Rd.  
Hershey, Pa. 17033  
(717) 533-5880

## TEXAS

Swift Computers, Inc.  
3208 Beltline Rd.  
Suite 206  
Dallas, Texas 75234  
(214) 241-4088

Swift Computers, Suite 145  
6333 Camp Bowie Blvd.  
Ft. Worth, Texas 76116

Altair Computer Center  
7302 Harwin Dr.  
Suite 206  
Houston, Texas 77036  
(713) 780-8981

Altair Computer Center  
3205-A 34th St.  
Lubbock, Texas 79410

## UTAH

Microcosm Inc.  
534 West 9460 South  
South Sandy, Utah 84070  
(801) 566-1322

## VIRGINIA

Computer-To-Go  
1905 Westmoreland St.  
Richmond, Va. 23230  
(804) 355-5773

Megabyte Computer Assoc.  
700 Stony Point, Suite 7  
Newtown Rd.  
Northolk, Va. 23502  
(804) 461-3079

Microsystems Computer Corp.  
Century Mall—Crystal City  
2341 S. Jefferson Davis Hwy  
Arlington, Va. 22202  
(703) 979-5566

## WASHINGTON

Pasco Computer Store  
6704 Argent Rd.  
Pasco, Washington 99301  
(509) 547-9014

Altair Computer Center  
14100 N. E. 20th St.  
Bellevue, Wash. 98007  
(206) 641-8800

## WISCONSIN

Chicago Computer Store  
285 West Northland Ave.  
Appleton, Wisconsin 54911  
(414) 731-9559

## CANADA

The Computer Place  
186 Queen St. West  
Toronto, Ont. M5V 1Z1  
(416) 548-0262  
Telex 0622 634

**PCC** PERTEC  
COMPUTER  
CORPORATION  
MICROSYSTEMS DIVISION

20630 Nordhoff Street  
Chatsworth, California 91311

3

Bulk Rate  
U.S. Postage  
PAID  
Permit No. 26306  
Los Angeles, CA

JOHN BUIK  
1117-5137 ST  
MINNISHA WILSONSIN  
55140