System Timing Modification for the MITS®/Altair™ 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® 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™ 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 280μs to 389μs.

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 355μs to 425μs (389μs NOM ±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 140μs to 214μs.

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 195μs to 230μs (214μs NOM ±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.

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 enccuntered, 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.

```
100 CLEAR 137*34
110 PRINT:PRINT"DISK SELF COPY"
120 ' GET TO TRACK ZERO
130 OUT8,0
140 IF (INP(8)AND 64) <> 0 THEN WAIT8,2,2:OUT9,2:GOTO140
150 ' DO IT FOR ALL 77 TRACKS
160 FORT=0TO76
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$(S) THEN PRINT"REREAD T";T;"S";S:GOTO 200
230 NEXT S
240 PRINT:PRINT"WRITE T";T
250 FOR S=0 TO 31 ' WRITE NEW TRACK
260 DSKO$A$(S),S
270 NEXT S
280 FOR S=0 TO 31 ' CHECK NEW DATA
290 B$=DSKI$(S)
300 IF A$(S)<>B$ THEN PRINT"REWRITE T";T;"S";S:DSKO$A$(S),S:GOTO 290
310 NEXT S
320 ' GOTO NEW TRACK
330 ERASE A$
340 IF T=76 THEN 360
350 WAIT 8,2,2:OUT 9,1
360 NEXT T
370 CLEAR 200
380 PRINT:PRINT"THAT SHOULD DO IT"
390 END
```

C. Single Drive DOS Diskette Rewrite Procedure

By Drew Einhorn

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

Observe the 40µs low pulse representing the Index BURST. This low pulse is typically 4µs 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.

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

Channel 1 should show the Read Clear pulse (140µs old, 214µs 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 100µs after the beginning of the Sector. The data

field starts with the Sync Bit 250 to 350μs (old timing) or 350 to 500μs (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 (250-400μs) 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.

More on the KCACR

By Doug Jones

The announcement of the new MITS® KCACR board (Kansas City Audio Cassette Recording) for their MITS/ Altair™ 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?

Machine Language to BASIC Converter

By Richard Ranger

An annoying but necessary step in using the machine language interface, DEFUSR, in MITS®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 DIMT2(15):FØRY=0T015:T2(Y)=-1:NEXTY:PRINT"*";:LINEINPUTB$
40 IFB$=""THENCLEAR200:END
50 IF LEN(B$)>3 THEN C$=RIGHT$(B$,LEN(B$)-3) ELSE C$=B$
60 B$=LEFT$(B$,3)
70 IFB$="DAT"THEN680
80 IFB$="CØP"THEN870
90 IFB$="LIS"THEN800
100 IF B$="CNV" THEN 1040
110 IFB$="DIR"THENF=-1:GØTØ270
120 IF B$="SRT" THEN F=0:DIMA$(255):GØTØ270
130 IFB$<>"INI"THENPRINT"ERR":GØTØ20
140 GØSUB 760
150 A$=STRING$(137,0):MID$(A$,136,1)=CHR$(255)
160 FØRT=6TØ76
170     FØR S=0 TØ 31
180     MID$(A$,1,2)=CHR$(T)+CHR$((S*17)AND31)
190     GØSUB 600:DSKØ$ A$,S
200 NEXT S,T
210 T=70:GØSUB 600 'DIRECTØRY TRACK
220 A$=CHR$(70)+CHR$(0)+CHR$(0)+CHR$(128)+CHR$(127)+CHR$(0)
230 A$=A$+CHR$(0)+CHR$(255)+STRING$(127,0)+CHR$(255)
240 DSKØ$A$,0
```

Continued on page 28

Easy Floppy Disk Alignment Check - *continued*

```
250 PRINT:PRINT"DØNE"
260 GØTØ20
270 GØSUB760:ØPEN"Ø",1,"·····RR",A
280 PRINT#1,1:CLØSE1:KILL"·····RR",A
290 PRINT
300 PRINT"DIRECTØRY DISK";A
310 PRINT:I=0
320 FØRS=0TØ31
330 A$=DSKI$(17*SAND31)
340 A$=LEFT$(A$,135)
350 A$=RIGHT$(A$,128)
360 FØR T=0 TØ 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 ØR I=0 THEN PRINT:GØTØ 20
500 IF I=1 THEN 560
510 SW=0
520 FØR J=0 TØ I-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 FØR J=0 TØ I-1
570 PRINT A$(J)
580 NEXT
590 PRINT:GØTØ20
600 IFT2(A)<>-1THEN640
610 IF(INP(8)AND64)=0THENT2(A)=0:GØTØ640
620 WAIT8,2,2:ØUT9,2
630 GØTØ610
640 IFT2(A)=TTHENRETURN
650 D=1:IFT2(A)>TTHEND=2
660 WAIT8,2,2:ØUT9,D:T2(A)=T2(A)-2*(D-1.5)
670 GØTØ640
680 INPUT"TRACK";T:IF T<0 THEN 20 ELSE INPUT"SECTØR";S
690 GØSUB760:GØSUB600
700 A$=DSKI$(S):FØRI=0TØLEN(A$)-1
710   T1$=ØCT$(ASC(RIGHT$(A$,LEN(A$)-I)))
720   T2$=LEFT$("  000",5-LEN(T1$))+T1$: PRINT T2$;
730   IF I MØD 8=7 THEN PRINT
740 NEXT I: PRINT
750 GØTØ 680
760 A=VAL(C$)
770 IFA<ØØRA>15THENPRINT"ERR":GØTØ20
780 ØUT8,128:ØUT8,A
790 RETURN
800 GØSUB760
810 C$=RIGHT$(C$,LEN(C$)-1+(A>9)):IFASC(C$)<>4054THENPRINT"ERR":GØTØ20
820 C$=RIGHT$(C$,LEN(C$)-1)
830 ØPEN"I",1,C$,A
840 IFEØF(1)THENCLØSE1:GØTØ20
850 LINEINPUT#1,A$
860 PRINTA$:GØTØ840
870 GØSUB760:B=A
880 C$=RIGHT$(C$,LEN(C$)-1+(A>9)):IFASC(C$)<>4054THENPRINT"ERR":GØTØ20
890 C$=RIGHT$(C$,LEN(C$)-1):GØSUB760:C=A
900 PRINT"FRØM ";B;" TØ ";C;
910 INPUTA$:IFASC(A$)<>ASC("Y")THEN20
920 FØRT=0TØ76
930 ØUT8,128:ØUT8,C
940 A=C:GØSUB600:ØUT8,128:ØUT8,B:A=B:GØSUB600
950 FØRS=0TØ31
960 ØUT8,128:ØUT8,B:B$=DSKI$(S)
970 F$=DSKI$(S):IFF$<>B$THENPRINT"REREAD":GØTØ960
980 ØUT8,128:ØUT8,C
990 DSKØ$B$,S:C$=DSKI$(S):IFC$<>B$THENPRINT"REWRITE":GØTØ950
1000 NEXTS
1010 NEXTT
1020 PRINT"DØNE"
1030 GØTØ20
1040 GØSUB 760   'ENABLE DISK
1050 FØR T=6 TØ 76
1060   GØSUB 600   'PØSITIØN TØ TRACK T
1070   FØR S=0 TØ 31
1080     A$=DSKI$(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       DSKØ$ A$,S
1120   NEXT S
1130 NEXT T: GØTØ 20
ØK
```

Machine Language to BASIC Converter - *continued from page 27*

```
CHANGE LINE 100 TO READ:
        100 T$=K$+P$+A$+S$+X$+O$+P$+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 reconnassance. He is currently studying at the University of New Mexico for a degree in Electrical Engineering.