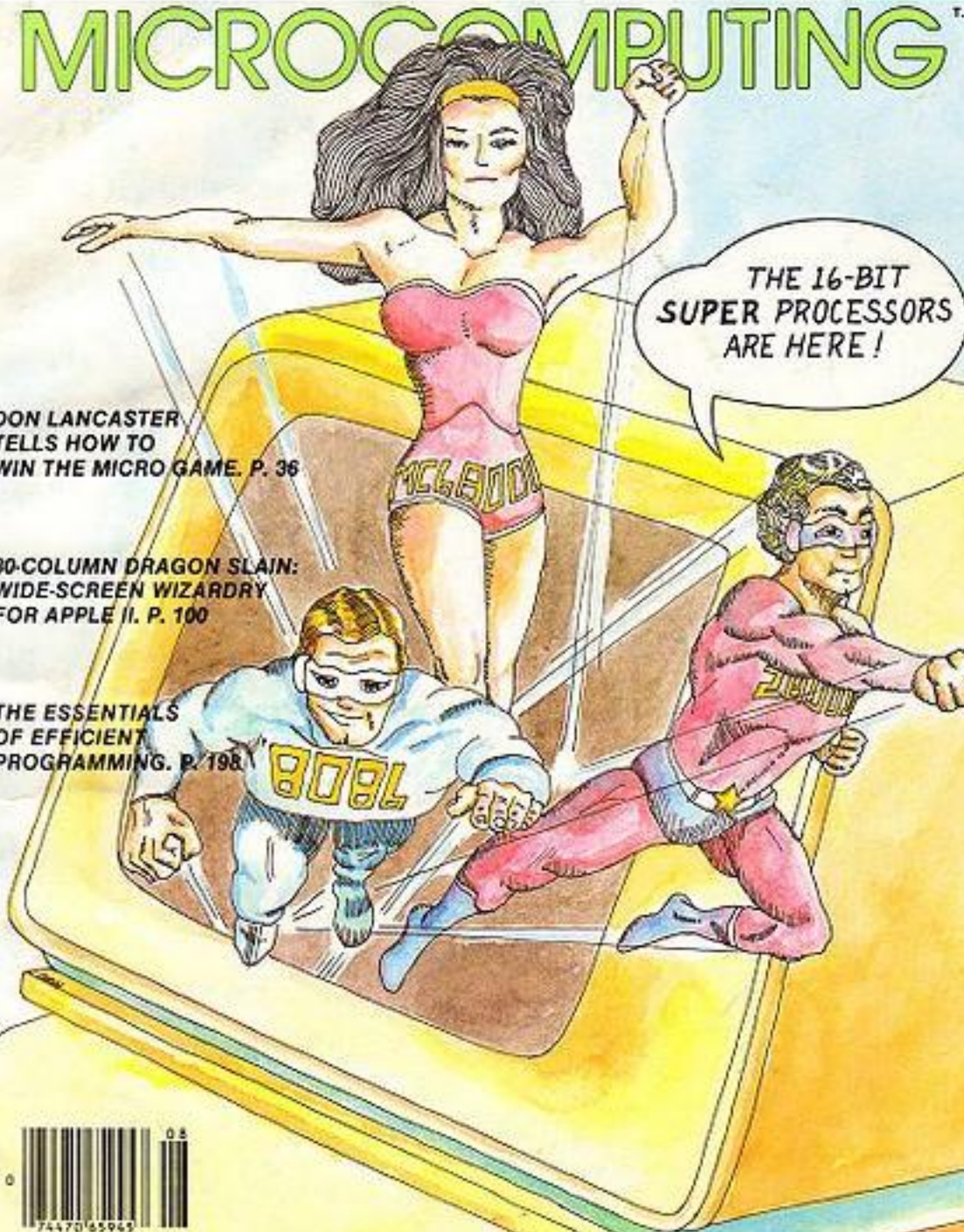


kilobaud

MICROCOMPUTING ^{T.M.}



THE 16-BIT
SUPER PROCESSORS
ARE HERE!

**DON LANCASTER
TELLS HOW TO
WIN THE MICRO GAME. P. 36**

**80-COLUMN DRAGON SLAIN:
WIDE-SCREEN WIZARDRY
FOR APPLE II. P. 100**

**THE ESSENTIALS
OF EFFICIENT
PROGRAMMING. P. 198**



Level II BASIC On a Z-80 System

Although the author used Radio Shack's three-ROM BASIC, the two-ROM version should work as well.

Richard J. Uschold
80 Woodview Dr.
Port Orange, FL 32019

Since I have been a dedicated hardware hacker for many years, I just had to build my own computer. I started designing at Christmas in 1976. By September 1977 I had my computer basically working, and by Christmas 1977 it was working in BASIC. It was a 2K Tiny BASIC interpreter, but it was better than nothing.

After about a year of using my Tiny BASIC, I decided I was

ready for a real BASIC. Since I had chosen the Z-80 microprocessor for my computer, I could use any BASIC written for the 8080 or the Z-80.

There were a number of BASICs available that required from 8K to 24K of memory at prices from \$50 to several hundred dollars. I really liked the idea of having the BASIC in ROM so that I wouldn't have to load it from tape every time, which seemed to take forever. (Even with my 2400 baud cassette interface, programs longer than 4K become annoying!) This

meant I had to either use EPROMs or buy the BASIC already in ROM. The EPROMs would cost upwards of \$80, plus the price of the BASIC.

There was only one BASIC offered in ROM that I knew of, although I had heard rumors of another one coming soon. The rumors have since become fact, and Livermore BASIC is now available on an 8K byte ROM for \$95. I bought the other one, Radio Shack's Level II BASIC, for \$89.10. (Several companies offer ten percent off Radio Shack's original \$99 price. Radio Shack has since raised the price to \$120.)

Radio Shack's Level II BASIC has another significant advantage—software availability. Since it is the most popular microcomputer around today, it has much software designed for it. Also, many programs not originally written for it are being offered in compatible forms (for example, the CP/M disk operating system and the Electric Pencil).

In this article, I will describe how I interfaced the Level II ROMs to my computer, even though my hardware bears little resemblance to that of the TRS-80. I will also give some hints to those computerists whose hardware doesn't resemble mine either!

Preliminary Work

Before I bought the Level II ROMs, I did some preliminary investigation, which included re-reading articles that described

the TRS-80 hardware and software. I also bought and read the "TRS-80 Microcomputer Technical Reference Handbook" published by Radio Shack. All of this material provided several important pieces of information.

First, the TVT was a more or less standard type of memory-mapped interface, which, I figured, should present no problems.

Second, the keyboard was an unorthodox arrangement with the key matrix directly mapped in memory (see Fig. 1). I figured I could write a program to take ASCII data from my keyboard and calculate the required memory bits to set so that the ROM could find the bits in memory and convert them back to ASCII (a kludge, but it worked!).

Third, the cassette interface was software timed and would require a different clock rate on my processor or else some software patches to get the timing right.

Finally, and perhaps most importantly, the ROMs were located in memory at address 0000H. This meant I would have to move my monitor, which was now there, to another address. I moved it to F000H. This required a reset vector other than 0000H to initialize to the monitor.

The circuit I used was described in the September 1977 *Kilobaud* ("Using an Invisible PROM," p. 106, by Jack Regula). My version is in Fig. 2. I spent the next month or so rewriting and improving my monitor. When I had it just right, I put it in

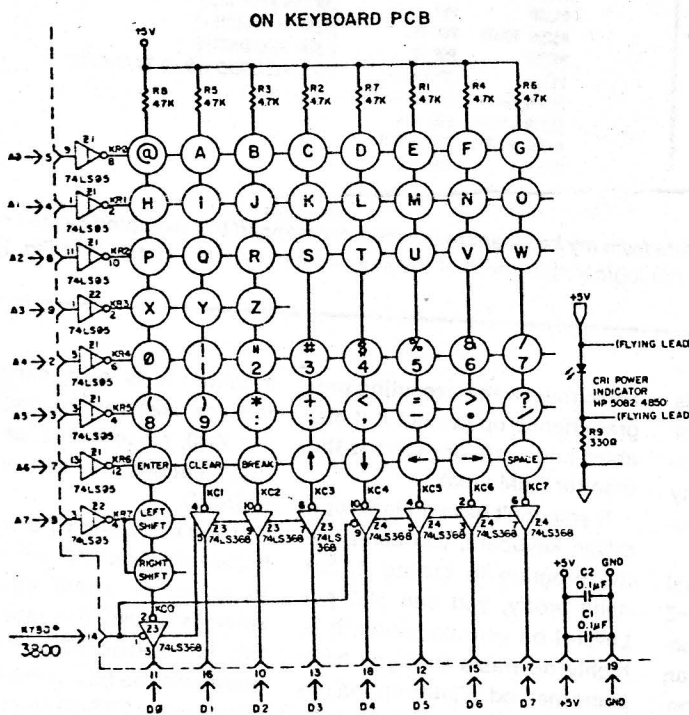


Fig. 1. TRS-80 keyboard connected to the address and data buses. (Reprinted from the "TRS-80 Technical Reference Manual," courtesy Radio Shack.)


```

00100 ; TRS-80 KEYBOARD SIMULATOR INTERRUPT ROUTINE
00110 ; BY RICHARD J. USCHOLD
00120   ORG 0F76H
00130 TKEINT  PUSH HL      ; SAVE REGISTERS
00140   PUSH AF
00150   PUSH BC
00160   LD R,30H      ; MASK KERO AND LOWER INTS
00170   OUT (INTSK),A  ; FOR MY SYSTEM ONLY
00180   EI
00190   XOR A          ; CLEAR A
00200   LD B,B         ; LOOP COUNT
00210   LD HL,3001H   ; KERO MEMORY ADDRESS
00220 CLRLOP  LD (HL),A  ; CLEAR KERO ADDRESS
00230   RLC L         ; GENERATE NEXT ADDRESS
00240   DJNZ CLRLOP
00250   LD L,0FFH     ; KEY PRESSED BYTE
00260   LD (HL),A    ; CLEAR IT
00270   IN A,(KERO)  ; GET DATA FROM KEYBOARD
00280   LD C,A       ; SAVE DATA
00290   BIT 7,A      ; CHECK ALL SHIFT BIT
00300   JR Z,NASHFT  ; NOT ALL SHIFT
00310   LD A,1        ; YES-SHIFT BIT DATA
00320   LD L,80H    ; SHIFT ADDRESS
00330   LD (HL),A   ; SET SHIFT BIT
00340   RES 7,C      ; CLEAR ALL SHIFT BIT
00350   LD A,C       ; GET DATA
00360 NASHFT CP LFBKRT ; UPPER LIMIT CHARACTER
00370   JR NC,VALID  ; ONLY ONE VALID ABOVE THIS
00380   CP ATSN      ; CHECK IF ALPHABETIC
00390   JR C,NORLPH
00400   RRCA         ; /2 - YES, GENERATE
00410   RRCA         ; /4 - ADDRESS BIT
00420   RRCA         ; /8
00430   AND 03      ; MASK ALL BUT TWO BITS
00440   INC A        ; ADJUST COUNT TRUE
00450   LD B,A       ; SET UP LOOP COUNT
00460   XOR A        ; CLEAR A
00470   SCF         ; CARRY TO BE SHIFTED IN
00480 GENADR  RLA     ; GENERATE ADDRESS BIT
00490   DJNZ GENADR
00500   LD L,A        ; SAVE ADDRESS
00510 DATA  LD R,C   ; RESTORE ASCII DATA
00520   AND 07       ; MASK ALL BUT THREE BITS
00530   LD B,A       ; MOVE TO COUNTER
00540 DATAL  XOR A    ; CLEAR A
00550   SCF         ; CARRY TO BE SHIFTED IN
00560   INC B        ; INK COUNT TRUE
00570 GENADR  RLA     ; GENERATE DATA BIT
00580   DJNZ GENADR
00590   LD (HL),A    ; SET BIT IN MEMORY
00600   LD L,0FFH     ; KEY PRESSED BYTE
00610   LD (HL),A    ; SET KEY PRESSED BIT
00620 NVALID POP BC   ; RESTORE REGISTER
00630   JP KBINT1   ; FINISH BY DOING NORMAL
00640   ; KEYBOARD INTERRUPT ROUTINE
00650 NORLPH CP '!'  ; IS IT CONTROL?
00660   JR C,DATAL  ; YES
00670   BIT 3,A     ; NUMERIC OR SPECIAL?
00680   JR NZ,NORLPH
00690   BIT 4,A     ; CHECK IF SHIFT
00700   LD B,10H    ; SAVE ADDRESS BIT
00710   JR NZ,NASHFT
00720 SHFT  LD L,80H ; SHIFT ADDRESS
00730   LD A,1      ; SHIFT BIT

```

```

00750 NASHFT LD L,B  ; SET ADDRESS BIT
00760   JR DATA
00770 NONUM  LD B,20H ; SAVE ADDRESS BIT
00780   AND 14H     ; CHECK IF SHIFT
00790   JR Z,SHFT
00800   XOR 14H     ; CHECK IF SHIFT
00810   JR Z,SHFT
00820   JR NUSHT
00830 VALID  CP RUBOUT ; THIS IS BACK ARROW KEY
00840   JR NZ,NVALID
00850 CNTRL  LD HL,CTRLB ; TABLE ADDRESS
00860   LD BC,8      ; LOOP COUNT
00870   CPIR        ; SEARCH TABLE FOR MATCH
00880   JR NZ,NVALID ; NOT FOUND
00890   LD HL,3840H ; CONTROL BIT ADDRESS
00900   LD B,C       ; LOAD LOOP COUNT
00910   JR DATAL    ; COMPUTE BIT AND FINISH
00920 CTRLB  DEFB SPACE
00930   DEFB RTAROW
00940   DEFB RUBOUT
00950   DEFB LF
00960   DEFB UPAROW
00970   DEFB ESC
00980   DEFB ENA
00990   DEFB CR
01000 INTSK  EQU 13H
01010 KBRD  EQU 4
01020 LFBKRT EQU 58H
01030 RUBOUT EQU 7FH
01040 SPACE EQU 20H
01050 RTAROW EQU 9
01060 LF     EQU 10
01070 UPAROW EQU 08H
01080 ESC   EQU 1BH
01090 ENA   EQU 5
01100 CR    EQU 0DH
01110 ATSN  EQU 40H
01115 ; -----
01120 ; THIS IS THE NORMAL KEYBOARD INTERRUPT SERVICE ROUTINE
01130   ORG 0F83CH
01140 KBDINT  PUSH HL
01150   PUSH AF
01160 KBINT1  LD HL,KERDAT ; SAVE ADDRESS FOR DATA
01170   IN A,(KBRD) ; GET DATA
01180   LD (HL),A   ; SAVE IT
01190   CP CTRLZ   ; TO RETURN TO MONITOR
01200   LD A,10H   ; ENABLE ALL INTERRUPTS
01210   OUT (INTSK),A
01220   JR Z,TOMON ; IT WAS CONTROL Z
01230   DEC HL     ; POINT TO STATUS WORD
01240   SET 0,(HL) ; SET KERO FLAG
01250   POP AF
01260   POP HL
01270   EI        ; ENABLE INTERRUPTS
01280   RET       ; RETURN FROM INTERRUPT
01290 TOMON  POP AF  ; RESTORE REGISTERS
01300   POP HL   ; FOR SAVE ROUTINE
01310   JP ROSAVE ; SAVE REGISTERS AND GO TO MONITOR
01320 KERDAT EQU 0F83AH
01330 CTRLZ  EQU 1AH
01340 ROSAVE EQU 0F839H
01350   END

```

Listing 1. TRS-80 Keyboard Simulator program converts the ASCII data from my keyboard to the memory-mapped bits expected by the Level II BASIC ROM. Program is simpler than it might have been due to the logical placement of the keys in the keyboard matrix (see Fig. 1).

EPROM, and I ordered the Level II ROMs.

Getting Ready

While waiting for the ROMs to arrive, I wrote a couple of programs to simulate the TRS-80 hardware, and I made a couple of hardware modifications to my computer in those areas that could not be readily done with software. The first program, in Listing 1, simulated the TRS-80

memory-mapped keyboard. This program is an interrupt driver that must be used as such. The program exits by jumping to my normal keyboard interrupt routine.

As you can see, the normal routine checks for a control-Z character and jumps to the monitor if it detects one. This is an invaluable feature of my monitor. This allows me to always jump back to the monitor if for

some reason the executing program hangs up (except if it disables interrupts or destroys the monitor RAM area).

If you don't have an interrupt-driven keyboard, you can't use the program in Listing 1, but don't worry, you can still put Level II on your computer. It is highly desirable that you have some method of interrupting the computer, saving the registers, etc., and jumping back to your

monitor. It is also necessary that you use interrupt mode 2 on the Z-80, since the other interrupt locations are used by Level II BASIC.

If you use Listing 1 with most keyboards, you will not be able to enter the same character twice in a row! The reason for this is because when the program sets the bits in memory to simulate the TRS-80 keyboard, it never resets the bits until the

```

00100 ;BASIC INITIALIZATION ROUTINES
00110 ;RICHARD J. USCHOLD
00120 PCG EQU 1CH
00130 SPACE EQU 0F82FH
00140 CHIN EQU 0F859H
00150 JPIGRT EQU 0F5CFH
00160 VIDVEC EQU 401EH
00170 INITVT EQU 0FD00H
00180 TVT EQU 14H
00190 PRTEC EQU 4026H
00200 LINEPP EQU 4023H
00210 KB0VEC EQU 4016H
00220 JNP4 EQU 0FC59H
00230 KB0ST EQU 0F039H
00240 RUBOUT EQU 7FH
00250 BS EQU 8
00260 ENQ EQU 5
00270 CHRCHT EQU 4026H
00280 TYP0UT EQU 0F068H
00290 OK EQU 0F849H
00300 LZVID EQU 33H
00310 TRNDLR EQU 4012H
00320 INTMSK EQU 13H
00330 BOOT EQU 69FH
00340 SVPC EQU 0F051H
00350 DELAY EQU 60H
00360 CRET EQU 0CH
00370 TRSCRS EQU 0FFH
00380 VIDJMP EQU 0F068H
00390 UPAROM EQU 5EH
00400 NMIVEC EQU 66H
00410 ;GENERATE TRS-80 GRAPHICS
00420 ORG 0F5E9H
00430 BASIC IN R,(PCG+2) ;DISABLE WRITE PROTECT ON
00440 ;PROGRAMMABLE CHARACTER GENERATOR
00450 LD HL,33FH ;LAST PRG CHR ADDRESS
00460 LD C,0FFH ;DATA FOR LAST CHARACTER
00470 LD D,40H ;64 CHARACTER COUNT
00480 MAINLP LD E,4 ;4 DOT ROWS PER CHR COUNT
00490 CHARLP LD B,2 ;SHIFT LOOP COUNT
00500 SHFTLP RLC C ;GET DATA TO CARRY
00510 RRA ;ROTATE CARRY TO ACC
00520 SRA A ;COPY BIT TO FOUR PLACES
00530 SRA A
00540 SRA A
00550 DJNZ SHFTLP ;DO NEXT FOUR BITS
00560 LD B,4 ;4 LINES PER DOT ROW COUNT
00570 DOTL0P LD (HL),A ;LOAD DATA TO PRG CHR
00580 DEC HL ;BUMP TO NEXT ADDRESS
00590 DJNZ DOTL0P ;DO 4 LINES
00600 DEC E ;CHARLOOP COUNTER
00610 JR NZ,CHARLP
00620 LD A,C ;GET DATA FOR NEXT ROW
00630 SUB 41H ;GENERATE NEXT DOT ROW
00640 LD C,A ;SAVE NEXT DOT ROW
00650 DEC D ;MAIN LOOP COUNTER
00660 JR NZ,MAINLP
00670 IN A,(PCG+3) ;PROTECT MEMORY
00680 ;BASIC COMMAND DECODE - THE NEXT TWO LINES ARE PARTICULAR TO MY MONITOR [
00690 CALL SPACE ;TYPES A SPACE
00700 CALL CHIN ;GET A CHARACTER FROM KEYBOARD AND ECHO IT
00710 CP 'C' ;FOR CONTINUE
00720 JR Z,RETBAS ;GO BACK TO BASIC
00730 CP 'I' ;FOR INITIALIZE
00740 JP Z,0001 ;INITIALIZE BASIC
00750 CP 'R' ;FOR RESET
00760 JR NZ,JPIGRT ;ILLEGAL CHARACTER
00770 ;THIS ROUTINE PRINTS ILLEGAL CHARACTER MESSAGE AND RETURNS TO THE MONITOR FOR THE NEXT COMMAND
00780 JP NMIVEC ;TRS-80 RESET SWITCH
00790 NOP
00800 NOP
00810 NOP
00820 NOP
00830 RETBAS LD HL,VIDPCH ;INIT VID10 PATCH
00840 LD (VIDVEC),HL ;CHANGE TRS VECTOR
00850 CALL INITVT ;THIS SETS UP MY TVT AND
00860 ;CLEARS THE SCREEN. THIS IS PARTICULAR TO MY TVT AS IS THE NEXT LINE
00870 LD A,0CEH ;NO SCROLL CURSER OFF
00880 OUT (TVT+2),A
00890 ;THIS NEXT SECTION SETS UP A JUMP ADDRESS SO I CAN SWITCH BETWEEN THE NORMAL SPACE
00900 ;COMPRESSION CODES OR 64 MORE PROGRAMMABLE CHARACTERS
00910 LD HL,VIDJMP ;JUMP TO 0A96 DOES SPACE
00920 LD (HL),0C3H ;COMPRESSION CODES. JUMP
00930 INC HL ;TO 0A7D DOES PROGRAMMABLE
00940 LD (HL),0A6H ;CHARACTERS. FROM BASIC.
00950 INC HL ;POKE -3988,125 FOR PRG
00960 LD (HL),04H ;CHRS. POKE -3988,166
00970 ;FOR TABS
00980 LD HL,TRSPRT ;PRINTER DRIVER
00990 LD (PRTEC),HL
01000 LD HL,KB0SUB ;KEYBOARD SUBSTITUTE DRIVER
01010 LD (KB0VEC),HL ;CHANGE TRS VECTOR
01020 LD A,57
01030 LD (LINEPP),A ;PRINTER LINES PER PAGE
01040 NOP
01050 JP JNP4 ;THIS IS THE SECTION IN
01060 ;MY MONITOR WHICH RESTORES THE REGISTERS AND RETURNS TO THE MAIN
01070 ;PROGRAM - AS FROM A CONTROL Z INTERRUPT.

```

Listing 2. The first part of this program generates the bit patterns necessary to program my programmable character generator so it simulates the TRS-80 graphics. The second part sets up my computer so it is compatible with the Level II BASIC ROM.

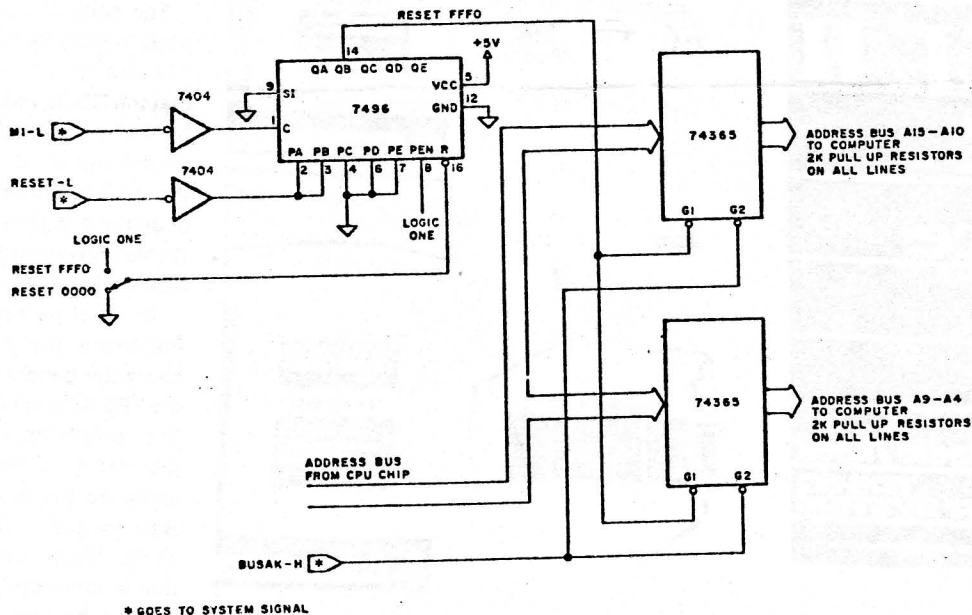


Fig. 2. Alternate reset vector circuit. Address FFF0 must be in PROM and contain a three-byte jump instruction to the start of the monitor.

next key is hit. If the next key is the same as the last one, the same bits will be set and the ROM will think you have not released the key yet!

There are several solutions to this problem. I modified my keyboard so it gives a second data strobe when a key is released. This will strobe in a null, and the program will clear the memory when the key is released. Another solution is to hit any key on the keyboard that is not encoded by the program. This will clear the memory and leave it that way. This is only necessary if you wish to hit the same character twice in a row.

Actually, I don't really recommend you use this program. I am only describing it since it is the way I started this project. Later,

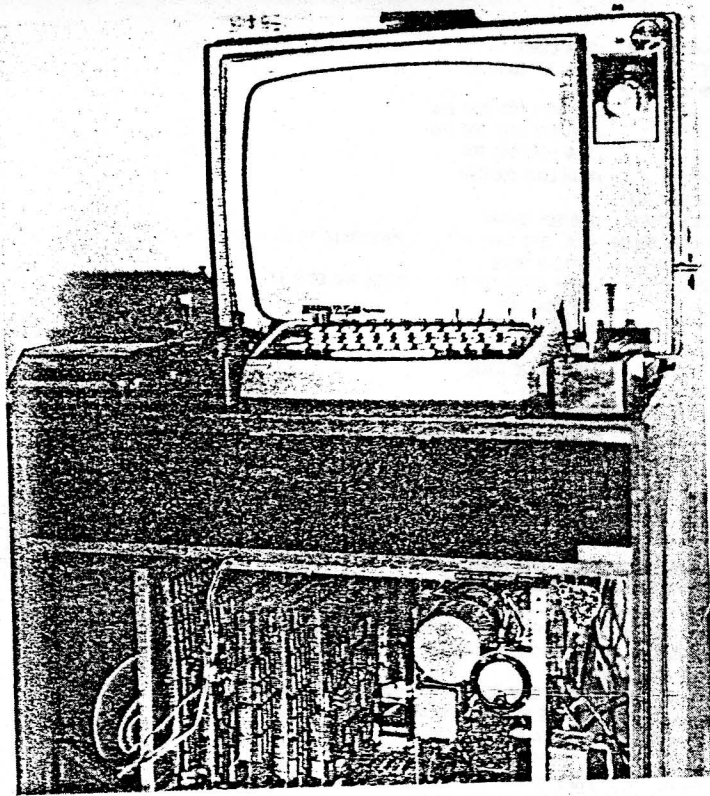


Photo 2. My completely home-brew system. The first board contains the TVT and programmable character generator. Board 2 has my front panel logic, the interrupt logic, EPROM programmer, two serial ports and the cassette interface, which supports Kansas City Standard, Tarbell, PE2400 Radio Shack Level II and CUTS with a slight mod. The third board contains the Z-80 CPU chip, 10K of static RAM, 3K EPROM, the clock switch and "No Memory" interrupt circuit. Board 4 is a 12K static RAM board. The fifth board contains a joystick interface, Level II ROMs, alternate reset circuit, floppy disk interface, real-time clock and sockets for 32K of dynamic RAM.

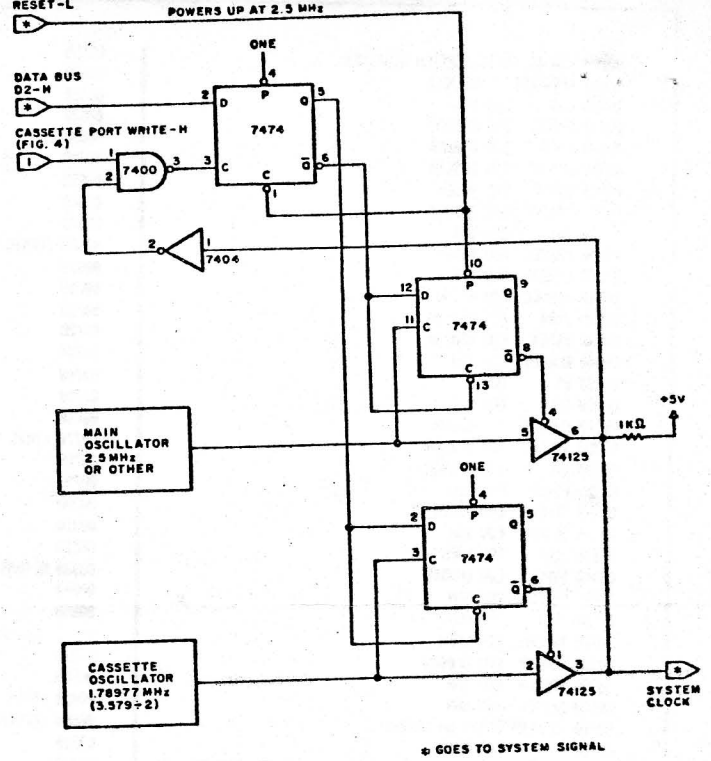


Fig. 3. Clock switch circuit automatically switches the clock from the normal frequency (2.5 MHz on my system) to 1.79 MHz when I/O port FF is written with bit 2 set. It switches back when port FF bit 2 is reset. This bit is the TRS-80 cassette motor control bit.

I'll tell you what you should use and what I am now using.

Another noteworthy feature about this program is the shift. The TRS-80 keyboard program

generates lowercase characters if the shift key is pushed with a regular key. It also generates special control characters when the shift is pushed with the arrow keys.

I handled this by using the eighth bit as the shift bit. My keyboard has an extra key that sets the eighth bit when pushed. Most keyboards don't have this.

The second program I wrote while waiting for the ROMs is an initialization of my system so that the ROMs will think they are hooked up to a TRS-80. Listing 2 essentially is the program, although it is a little bit different. I changed it slightly after I got the ROMs and learned a few things I didn't originally know.

The first part of the program initializes my programmable character generator to simulate the TRS-80 graphics characters. The programmable character generator is essentially the same as the one described in *Byte* magazine (May and June 1978). There are 128 programmable characters that can be printed by sending the codes 80H-FFH to the video driver or directly loading these codes in



Photo 1. Level II kit. ROMs have been removed from the circuit board. (Photos by Michael Tabellion).

TVT memory area. The TRS-80 has 64 graphics characters having codes 80H-BFH. These corresponding characters are generated by the program.

The next section, command decode, checks for one of three options: initialize, continue or reset. The initialize section jumps to the ROM so it can initialize the Level II RAM area as it requires. The reset jumps to the ROM, where the reset button on the TRS-80 would send it. This is used when the Level II hangs up and you do not wish to destroy the BASIC program in memory.

On my system, I type a control-Z to get back to the monitor and then BR. B is the BASIC command in my monitor that jumps to the program I am now describing. R is the reset option. The continue option initializes a few more things, which I'll describe later, restores the registers and continues where it was interrupted (usually by a control-Z). I frequently use this to save BASIC programs with my 2400 baud cassette interface rather than use Level II's 500 baud cassette interface.

I made several hardware mods to accommodate the Level II ROMs. The simplest was to move my RAM, EPROM and TVT RAM to the proper locations. The TRS-80 hardware manual has a memory map, so this was no real problem. The other two mods were a bit more involved. Both of these mods are for the cassette interface.

The first one (Fig. 3) changes the clock speed during the cassette operation. Normally my computer runs at its rated speed of 2.5 MHz; during a cassette operation, the speed is reduced to 1.7898 MHz. This is about one percent higher than the TRS-80 clock and is more than close enough when you consider the tolerance of the cassette machine.

The required clock rate is one-eighth the rate of my TVT clock, so I didn't require another oscillator. The required clock is also one-half the color burst frequency. There are inexpensive crystals available that you can use; 3.579 MHz color burst crystals cost less than \$2.

The other changes are more directly related to the cassette interface itself (Fig. 4). The output circuit is little more than a couple of latches and a few resistors. I also added some Tri-state buffers so I could use the same cable as my 2400 baud interface. The first input circuit I tried is simpler than what the TRS-80 has, with three fewer op amps and many fewer resistors and capacitors. The idea was to

change the input circuit to be used with my 2400 baud interface as little as possible.

Well, I was finally ready for the ROMs, which would not arrive for over a month.

The ROMs Arrive

After calling the company twice, asking where my order was, I finally received the ROMs, which came on a small circuit board with a 24-pin jumper cable

cable. No instructions came with the kit; however, the handbook shows a schematic of the circuit board (Fig. 5). There are also other items, including an unprogrammed DIP header and a resistor, in the kit (see Photo 1). The DIP header alters the ROM decode in the TRS-80; I'm not sure what the resistor is used for. Anyway, I didn't use either of these.

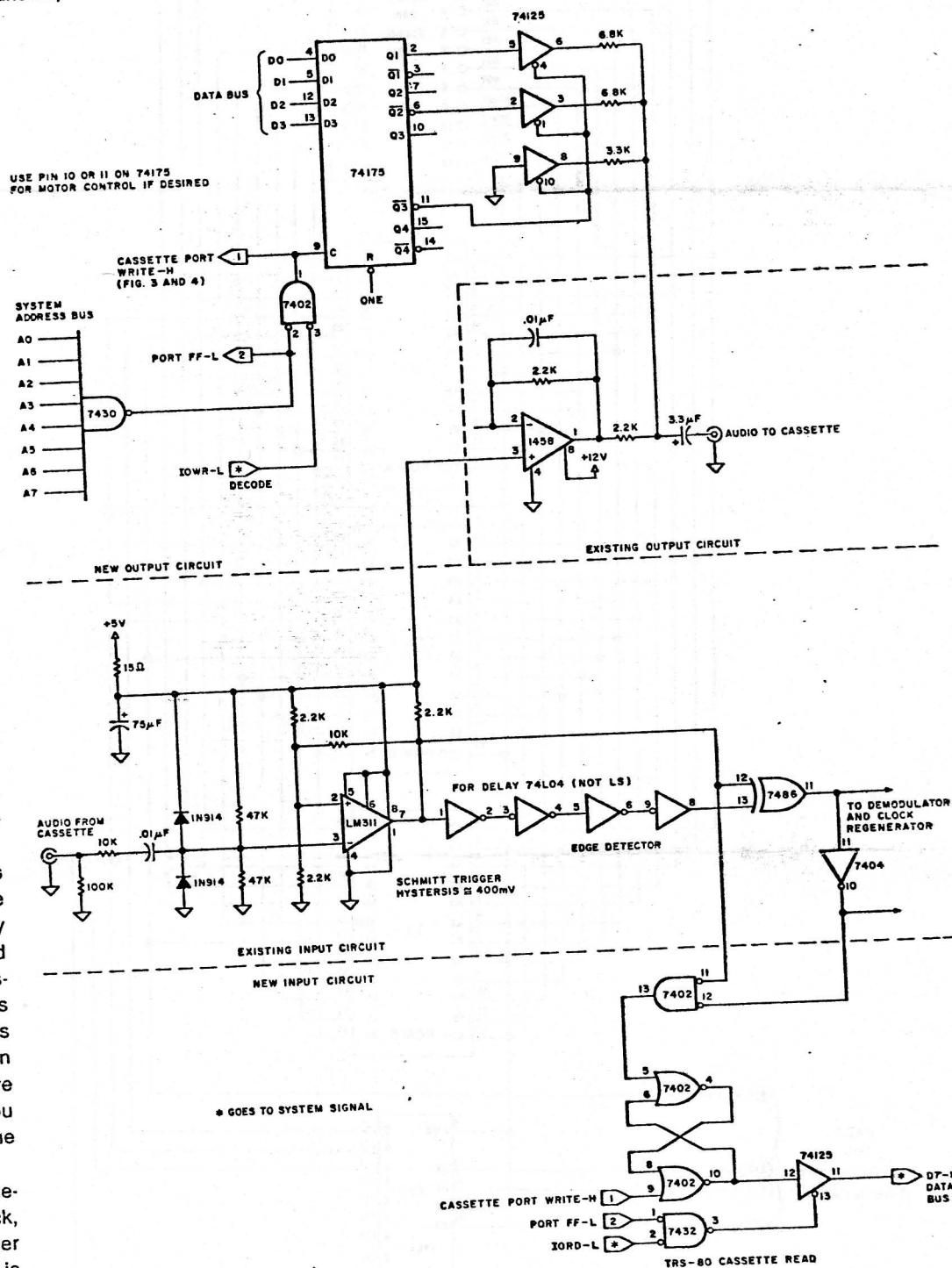


Fig. 4. Cassette output circuit similar to the TRS-80. I added the Tri-state buffers and changed the resistor values a bit so I could wire it directly to my existing output circuit. You can use bit 2 for cassette motor control if you wish.

Also included are three pre-recorded cassettes with some very brief instructions on how to use them. One cassette contains Blackjack and Backgammon. The other two cassettes are for conversion of Level I programs and data to Level II for-

mat. I haven't had a need for these two yet, though I have used the games a few times. Finally, there is the "Level II Reference Manual," along with errata sheets, containing useful information.

The small circuit board didn't

seem to fit anywhere in my system, so I wired up three sockets and just removed the ROMs. A friend had given me a poor copy of a copy containing a hex dump of the ROMs and partial disassembly of the initialization portion of the program. The

first thing I did was to check the first few bytes in each ROM. They matched! Next, I ran off a hex dump of my own so I could read it without straining my eyes.

There was one more thing I wanted to do before I actually tried to execute the program contained in the ROMs. From all the information I had acquired, I knew that the TRS-80 used interrupts only when it had the expansion interface connected. Also, it only used interrupt mode 1 on the Z-80 chip. Since my system would only work if I used interrupt mode 2, I searched the ROMs for any instructions that affected the interrupts. There were two: a disable interrupts at 0000H and an enable interrupts at 06E4H.

The enable interrupt instruction is actually the interrupt service routine, which is moved to RAM during the initialization. The routine merely enables interrupts and returns. This is modified when interrupts are needed. What all this boils down to is that I shouldn't have any problems with my interrupt-driven keyboard as long as I start the ROM at 0001H.

The Big Moment

So, I tried it. The screen cleared, and a short message appeared in the upper-left corner. It said, "ξξξπ√ υκ→ζ?" My computer was talking to me in Greek! There was obviously some incompatibility between the TRS-80 video driver and my TVT. The Level II manual tells me that the computer is supposed to say, "MEMORY SIZE?" Anyway, I responded with a "32000," which appeared on the screen just as I typed it.

Hmmmm, my keyboard kludge was working alright and the numerals printed correctly; but the alphabet was in Greek! I hit the carriage return. Nothing happened for a moment, then another couple lines of Greek appeared.

You may be wondering where the Greek was coming from. Well, that is an easy one. The character generator ROM I bought for my TVT has Greek characters and some special math symbols where the control

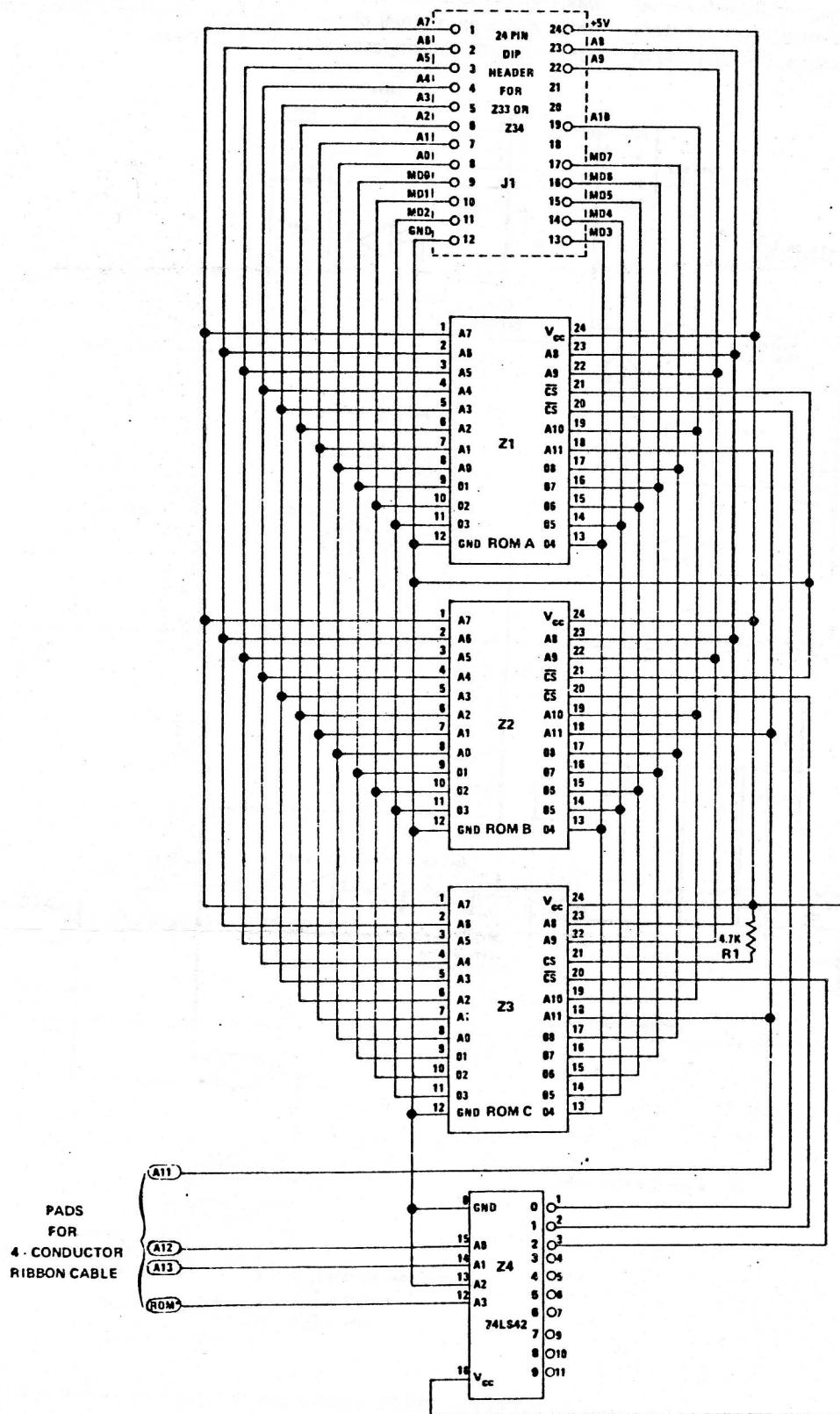


Fig. 5. Level II BASIC schematic. (Reprinted from the "TRS-80 Technical Reference Handbook," courtesy Radio Shack.)

16421

16429

16463
16464
16466
16476
16478

16512

16633
16638

16870

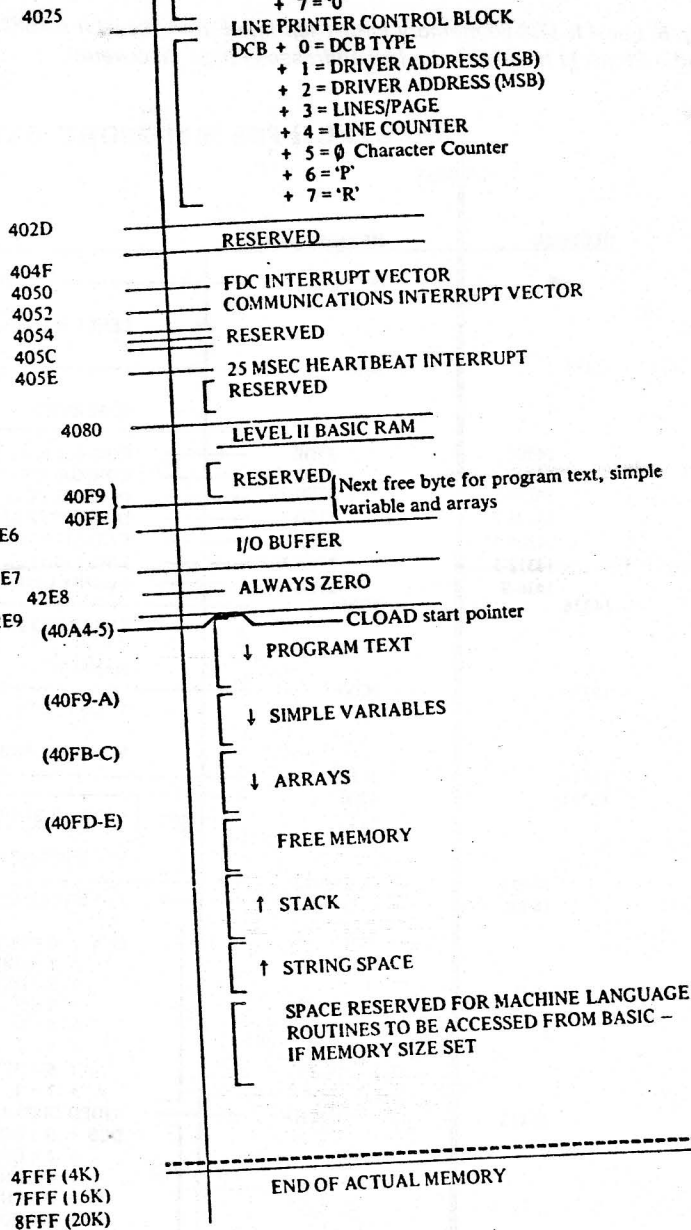
17127

17129

Pointer
Addresses

(16548-9)
(16633-4)
(16635-6)
(16637-8)

20479 (4K)
32767 (16K)
- 28673 = 36863 (20K)



the entry point of my video driver. I was then able to determine that the data was always in register C; my driver required the data in register A. I patched this in and tried again.

Now I was getting data, but everything was on the same line! There were only carriage returns and no line feeds! It seems the TRS-80 video driver automatically generates a line feed when it gets a carriage return. As it turns out, my video driver generates a carriage return if it gets a line feed! So I checked for carriage returns and converted them to the line feeds and tried again.

Now that was much better!

Everything seemed to work. Well... almost everything. The clear screen function did not work. I know this used to work when everything was in Greek. Referring again to the Level II manual, I noticed they have a table that describes all of the control codes that are implemented (Table 1).

I had two choices: modify my video driver to handle all of the control codes or try to see if I could patch their video driver so it would work. Half out of curiosity as to what they were doing and why it worked (on a TRS-80) and half because I didn't really feel like rewriting my driver, I disassembled their driver.

As I had guessed earlier, they are converting both upper and lowercase letters to control codes. The question is, "Why do they do this and how come it works?" The answer is in the hardware manual. It seems they thought it would be less expensive to use only seven bits of information in the video RAM instead of eight. They use one bit to select graphics characters or regular characters. That leaves six bits for the ASCII code.

But the ASCII code is a seven-bit code; how can that work? They cheat a little. The seventh ASCII bit is generated with a NOR gate from two other bits. This means that if they sent an

video RAM, it would be as a numeral or a special character. So they had to convert lowercase to uppercase. It was probably simpler to convert both upper and lowercase letters to control codes than to just change lowercase to uppercase.

Anyway, as far as they were concerned, that particular bit didn't really matter because it was not even in the RAM! Personally, I think they should have spent the extra buck on one more memory chip, then they could have had both upper and lowercase on the computer.

The final solution I came up with was to duplicate the first dozen instructions of their driver and then skip over the section that screws up the characters and jump back to their driver. The total patch is about 40 bytes.

Listing 3 shows that I have included two more small patches to the driver. The first changes the up-arrow code from 5B (which prints a left bracket()) to 1C, so it prints an up arrow on my TVT. Radio Shack mentions in the Level II manual that some TRS-80s may print the up-arrow as a left bracket. The second allows me to bypass the space-compression codes and print 64 more of my programmable characters instead. This is accomplished by poking one byte in a memory location.

The Cassette Interface

Having gotten the video driver working made me feel very confident. I was now ready to attack the cassette interface. I placed the Blackjack tape supplied with the Level II kit in the recorder (a Radio Shack CTR-40) and typed CLOAD. I have a small tape controller box, which enables me to hear the data while the computer is reading it. This is convenient because you can tell the difference in the sound of the actual data and the leader tone on the tape.

I turned on the recorder and hit the return key. One nice thing about the TRS-80 cassette driver is that two asterisks flash in the upper-right corner of the screen when the computer is reading data. The asterisks first appear

characters would normally be. Most video drivers don't actually send control characters to the video RAM; rather, they decode them and take the appropriate action. For some strange reason, the TRS-80 video driver was changing the normal alphabetic codes to control codes before sending them to the video RAM.

The First Program (in Greek)

I know that some people think that programming computers is like talking in Greek, but this is ridiculous! The Level II manual has a short program in the back which will display all of the graphics characters. I typed the program into my computer... in Greek! I changed it slightly, so it would print all characters not including the control codes. After I finished typing it, I listed it. Since I can't read Greek, I couldn't tell if I had it right or not, but at least the list command worked.

Next I typed "τχo;" that's RUN, for those of you who don't know Greek. Characters flashed by on the screen, and scrolled off before I could read them. I ran it again, but I halted the computer before everything disappeared. The special characters and numerals looked good. Then there were two sets of Greek characters where the uppercase and lowercase should be. Next came the graphics characters, which looked all right.

Fig. 6. Level II TRS-80 memory map. (Reprinted from "Level II BASIC Reference Manual," courtesy Radio Shack.) I have added a few addresses I have discovered.

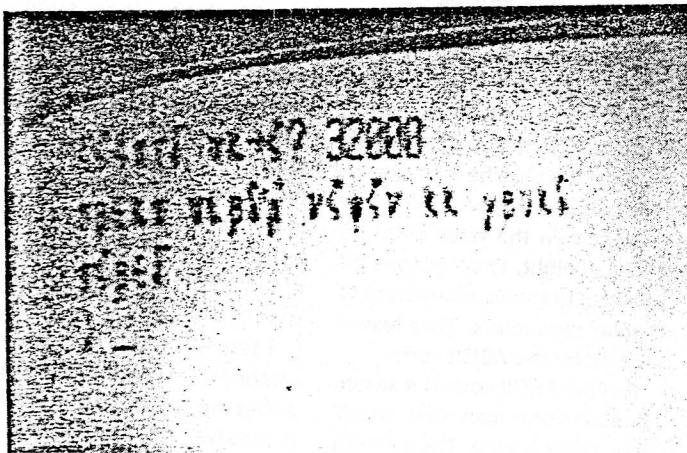
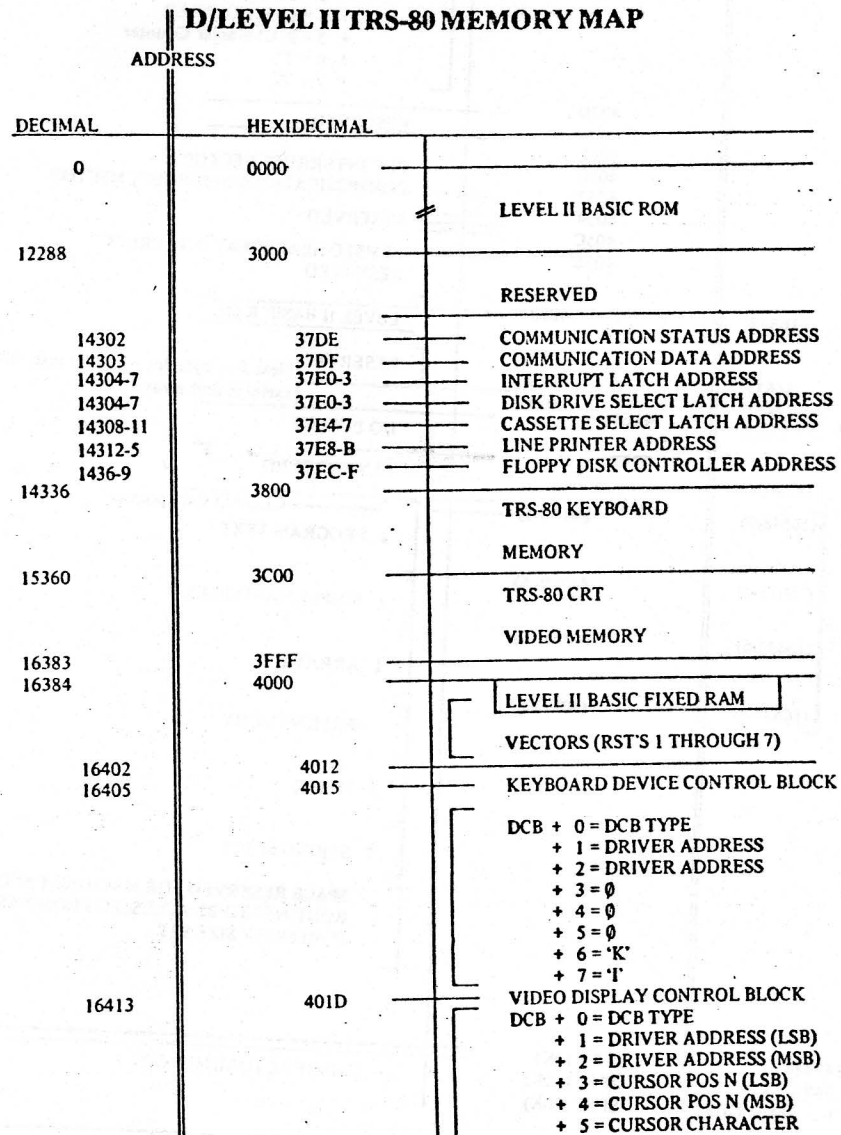


Photo 3. Initial run of Level II BASIC. Translation:
 MEMORY SIZE? 32000
 RADIO SHACK LEVEL II BASIC
 READY
 >

Finally, there were all of those spaces, as everything scrolled off the screen. The Level II manual has a good explanation for the scrolling phenomenon. The codes, C0H to FFH, are space-compression codes for 0-63 spaces. So, by printing all of those codes, I had printed about 2000 spaces to the screen. I changed the program so it did not print the space-compression codes and ran it again. This time it didn't scroll off the screen.

Video Driver Patch

I remembered something I had seen in the Level II manual, which showed a memory map,

which had a detailed description of some of the RAM locations used by the Level II BASIC. I was interested in a short section of 25 RAM locations containing three device control blocks. There were control blocks for the keyboard, the video display and the line printer. As you can see from Fig. 6, among other things, each block contains a driver address.

Now I figured all I had to do was to change the driver address to my own video driver, and I would be in business. I tried it. Nothing! I guessed that they used a different register to transfer the data byte. With this in mind, I set up a breakpoint at

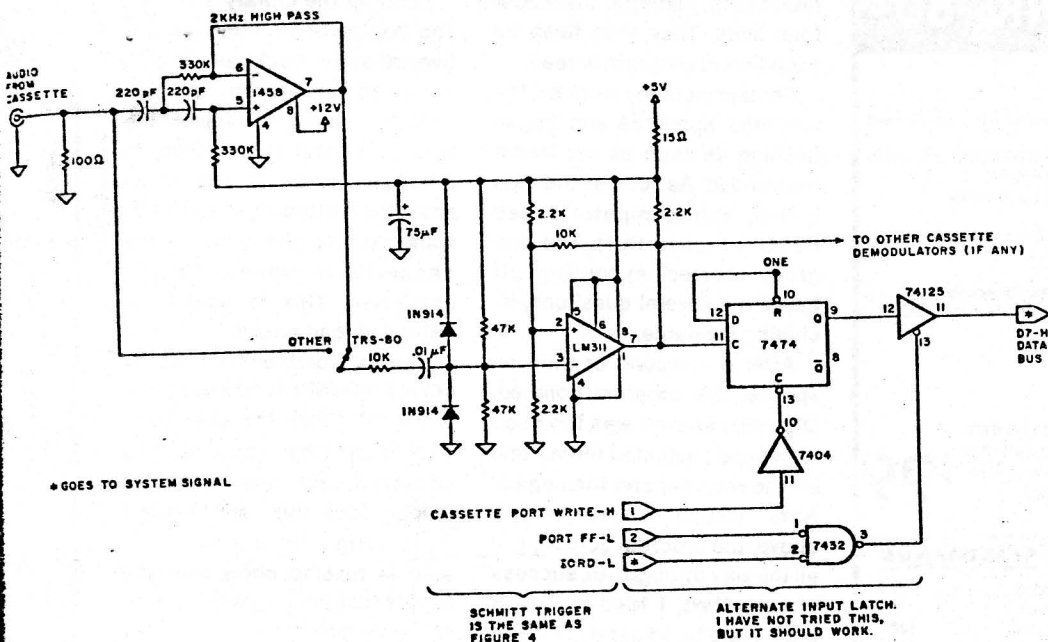


Fig. 7. Cassette input circuit I am now using. The 2 kHz high-pass filter is switched in to read Radio Shack tapes. The Schmitt trigger section is the same as in Fig. 4. The input latch is simpler than that shown in Fig. 4.

computer doesn't immediately respond with READY; if READY occurs before the data ends; or if the asterisks do not flash. If the asterisks flash slowly or erratically, the load may be bad. This clue takes some getting used to since the flash rate is not the same for all programs. You have to get a feel for how the asterisks normally flash.

If any of these symptoms occur, you will have to reload the program. Several of these problems cause the computer to

hang up. A reset must then be issued to get back to BASIC. During the next few weeks, I tried all of my 100 programs. I found that some of the tapes read fairly well, while others were very poor. These tapes have the same programs recorded on both sides as a backup: I found that I couldn't read some programs at all; I could read only one side correctly on some tapes; and I could read both copies on others. I tried reading some of these programs on a

real TRS-80, and some that I couldn't read worked. Since my input circuit was considerably simpler than the one they use, I breadboarded their circuit and tried it. It worked much better. The volume setting was less critical, but it was still more sensitive than I would have liked. With some experimenting, I found that I only needed the high-pass filter section of their interface. Since the TRS-80 tape format was so much improved with the filter, I tried it on my 2400 baud interface. It bombed. My interface became totally useless with the active filter.

The reason I attribute to this

seeming inconsistency is that the Radio Shack recording method is an amplitude modulation scheme, while my interface is a phase modulation scheme. The active filter adds too much phase distortion for my interface to work properly.

The final circuit I implemented for my cassette interface is shown in Fig. 7. The switch is to select Radio Shack or other recording methods. I'm not really sure if my circuit is more or less reliable than Radio Shack's, but my circuit seems adequate. Most of the tapes read through with two or fewer volume adjustments. Some don't need any adjustments. I don't use my Radio Shack interface to save programs anyway, since my 2400 baud interface is nearly five times faster.

One feature of the Radio Shack cassette interface I haven't built is the motor control circuit. I've been using my cassette interface for a year and a half, and I don't think a motor control is necessary. I do use the motor control signal to change the clock frequency and to enable the output circuit though. This works very well.

Keyboard and Printer Patches

I decided to get rid of that keyboard kludge I was using. I wrote the short driver in Listing 4. This program simply checks the keyboard status bit and either returns a null if it is not set or returns the character. It also checks for and changes two characters that were different on my keyboard than what the

```

01080 ;VIDIO DRIVER PATCH - PRINTS UPPER AND LOWER CASE
01090 VIDOPCH LD L,(IX+3) ;GET CURSOR POINTER
01100 LD H,(IX+4) ;GET CURSOR POINTER
01110 JP C,0499H ;I'M NOT SURE WHAT THIS IS
01120 LD A,(IX+5) ;GET CURSOR CHARACTER
01130 OR A
01140 JR Z,PATCH1
01150 LD (HL),A
01160 PATCH1 LD A,C ;GET CHARACTER
01170 ;THE FOLLOWING FEW LINES ADJUST THE UP ARROW CODE FROM THE
01180 ;TRS-80 CODE TO THE EQUIVALENT CODE ON THE CHARACTER GENERATOR I
01190 ;HAVE, WHICH IS MCM6571A
01200 CP UPARROW ;THIS IS THE UP ARROW CODE
01210 JR NZ,PATCH2 ;IS NOT UP ARROW
01220 INC C ;YES ADJUST
01230 JP 0467H ;DON'T BYPASS UPPER,LOWER ADJ
01240 PATCH2 CP ' ' ;CONTROL?
01250 JP C,0506H ;YES, DO IT
01260 CP 80H ;GRAPHICS?
01270 JP NC,VIDJMP ;YES, DO IT
01280 JP 0470H ;NO, ALL OTHER

```

Listing 3. Patch to the TRS-80 video driver eliminates the section that converts lowercase and uppercase character codes to control character codes. This permits both upper and lowercase to be printed.

TRS-80 Key	ASCII	Hex	Normal Keyboard
BREAK	SOH	01	CTRL A
←	BKSP	08	CTRL H
→	HT	09	CTRL I
↓	LF	0A	CTRL J
↑	[5B	[
ENTER	CR	0D	RETURN
SHIFT ←	CAN	18	CTRL X
SHIFT →	EM	19	CTRL Y
SHIFT ↓	SUB	1A	CTRL Z
SHIFT ↑	ESC	1B	ESCAPE
CLEAR	VS	1F	

Table 2. Control codes generated by the keyboard driver on the Level II BASIC ROMs. Your keyboard must generate these characters also.

SAVE ON APPLE® AND TRS-80®

NEWDOS/80

Powerful Disk Operating System for the TRS-80® designed for the sophisticated user and professional programmer. NEWDOS/80 is not meant to replace the present version of NEWDOS 2.1 which satisfies most users, but is a carefully planned upward enhancement.

- New BASIC Commands with variable record lengths up to 4095.
- Mix or match drives 35, 40, 77, 80TK.
- Security boot-up for BASIC or machine code application programs.
- Improved editing commands.
- Enhanced RENUMBER that allows relocation.
- Device handling for routing to display and printer simultaneously.
- CDE function: striking of C, D, and E keys allows user to enter a mini-DOS.
- Compatible with NEWDOS and TRSDOS 2.3.
- Superzap 3.0 and 2.1 utilities.

NEW DOS FOR APPLE® "APEX"

The complete APEX package with operating system, assembler, editor and user manuals. The package also includes a complete set of utilities to maintain files on single or multiple drive systems. (Specify 5 inch Apple disk or 8 inch disk.)

\$99

RELATED SOFTWARE

XPLO \$79
FOCAL™ \$59

TRS-80® SOFTWARE

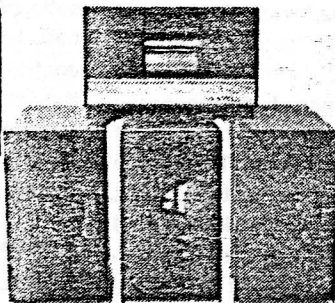
NEW DOS+ 35 track \$99
AJA Word Processor \$89
AJA Business Program \$289
Disk Drive Alignment Program \$109
Radix Data Base Program \$99.95

MOD I "B" DISK SYSTEM

- One SA800R DOS and Cable
- 2 Drive Chassis and Power Supply

\$1095

TRS-80® DISK DRIVES



DISK DRIVE SYSTEM

- 2 Shugart SA400 with power/chassis
- Cable
- Interface 32K
- 1 35-Track DOS+

\$1199

SPECIAL PRICE ONLY

SAVE ON APPLE II 16K FREE MTI MEMORY UPGRADE KIT TO 48K WITH PURCHASE OF APPLE II 16K

(MTI ONLY) **\$1195**



16K RAM MEMORY KIT **\$69**

DISK DRIVE SALE!

\$70 worth of FREE merchandise with purchase of Shugart SA400 with power supply and chassis. The disk that Radio Shack sells for \$499.

SAVE \$200 \$369
TF-1 Perrec FD200, 40 track ... \$389
TF-5 MPI B51, 40 track \$389
TF-70 Micropolis, 77 track \$639
TDH-1 Dual sided, 35 track \$499
MAX Disk 2: 10 Megabyte \$4995

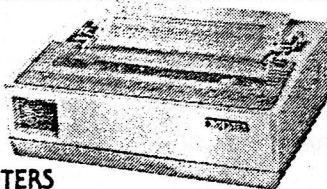
★ BARE DRIVES FOR ANY MICROCOMPUTER ★

Perrec FD200 \$282 FD250 \$359
Shugart SA400 \$279 SA800 \$479
MPI B52 \$349 B51 \$279

®Registered trade mark of Radio Shack and Apple

OKIDATA PRINTER

LIST \$1009
OUR PRICE **\$699**



PRINTERS

Centronics 779 \$1069 ANADEX \$925
Centronics 737 \$939 700-1 \$1195
Centronics 701-1 \$1795 702-2 \$1995
Spinwriter-NEC \$2549
Base 2 Printer 80, 132 col. graphics/tracators \$599



MICROCOMPUTER
TECHNOLOGY
INCORPORATED ✓ 30

3304 W. MacArthur
Santa Ana, CA 92704
(714) 979-9923



ASK FOR FREE
CATALOGUE

Apparat, Inc.

7310 E. Princeton Ave.
Denver, CO 80237
(303) 741-1778

Telex #678401TABIRIN

ALL PRICES CASH DISCOUNTED • FREIGHT FOB FACTORY

when the actual data on the tape starts, just after the leader tone ends. They then flash as each line of program is read.

Somewhat to my surprise, the asterisks appeared and began flashing as soon as the leader tone ended. As soon as the data ended, the computer typed READY. I typed RUN. The program started executing! It asked me several questions, including my name.

After my second or third response, the program bombed. Oh well, I knew it was too good to be true. I adjusted the volume on the recorder and tried again. After several repeats of the above, the program actually ran all the way through. Ah, success at last. Next, I tried making a tape. I had to adjust the volume several times to get it to read back correctly, but this also worked.

The volume setting on the taperecorder is critical. I usually have to adjust it several times before I can get a program to

load correctly.

I bought the Library 100 from The Bottom Shelf, Inc. This is a five-cassette package of 100 assorted programs for the TRS-80. I have to adjust the volume several times even to read programs on the same cassette. According to the hardware manual, the data on the cassette is saved with a checksum. This is useful for detecting load errors.

The only problem is that the Level II cassette loader program does not check the checksum and tell you when a bad load has occurred. My own cassette loader does this, and while I don't have frequent errors, it sure is nice to know that the load is bad before you try to execute the program.

I have discovered several ways to help determine if a load is good or not. The load will be bad if the asterisks appear before or after the point on the tape where the data actually starts; if the data stops and the

Code	Hex	Function
0-7	00-07	None
8	08	Backspaces and erases current character
9	09	None
10-13	0A-0D	Carriage returns
14	0E	Turns on cursor
15	0F	Turns off cursor
16-22	10-16	None
23	17	Converts to 32 character mode
24	18	Backspace ← Cursor
25	19	Advance → Cursor
26	1A	Downward ↓ linefeed
27	1B	Upward ↑ linefeed
28	1C	Home, return cursor to display position(0,0)
29	1D	Move cursor to beginning of line
30	1E	Erases to the end of the line
31	1F	Clear to the end of the frame

Table 1. Control codes decoded by the video driver on the Level II BASIC ROMs. (Reprinted from "Level II BASIC Reference Manual," courtesy Radio Shack.) I have added hex codes.

gram's attention is with an interrupt. If you have an interrupt-driven keyboard, you could use a program such as Listing 1 to simulate the TRS-80 memory-mapped keyboard, as I did at first. Otherwise, you need some other means of interrupting the computer. This could be as simple as a switch to the interrupt line on the Z-80. The interrupt service routine could simply change the keyboard driver address and then return to the Level II program.

There are only two situations where you could get by without any interrupts. If you actually connect your keyboard the same way as Radio Shack did, you wouldn't need interrupts. If you already have a keyboard connected some other way, re-wiring it is probably undesirable. Or, if you have a hardware front panel, you could interrupt the computer that way and change the keyboard driver address. While that is not really very difficult, it is kind of a bother to flip all those switches. My system includes a front panel, and I didn't want to do it that way.

The method I used to interrupt the computer is a bit unusual for a microprocessor. I have a circuit in my computer that generates an interrupt if the computer attempts to read a memory address at which there is no memory installed (see Fig. 8). This interrupt saves all the registers, prints a "No Memory"

message and jumps to my monitor. When the ROM tries to read the keyboard, this interrupt is generated because I don't have any memory there. From here I simply type BC, a monitor command that stands for BASIC Continue.

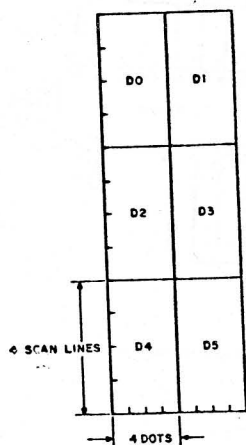
Listing 2 is the program. Its function is very simple—it merely sets up the new driver addresses for the keyboard, TVT and the printer. Then it restores all the registers and returns to where it was interrupted.

TVT Specifics

If your TVT is a memory-mapped device with 16 lines of 64 characters, you should have no problems getting it to work with Level II BASIC. You will have to change its address to 3C00-3FFF. If you don't have a programmable character generator, you will have to modify the TVT to implement the TRS-80 graphics. The modification should consist of only three ICs as shown in Fig. 9.

Fig. 10 shows the graphics-character format. As you can see, each character cell is divided into six blocks. Each block is controlled by one bit in the video memory. The most significant bit determines if a particular character is a graphics character or a regular character. The multiplexers simply steer the bits to the appropriate positions.

This circuit will work for TVTs, which have a character cell con-



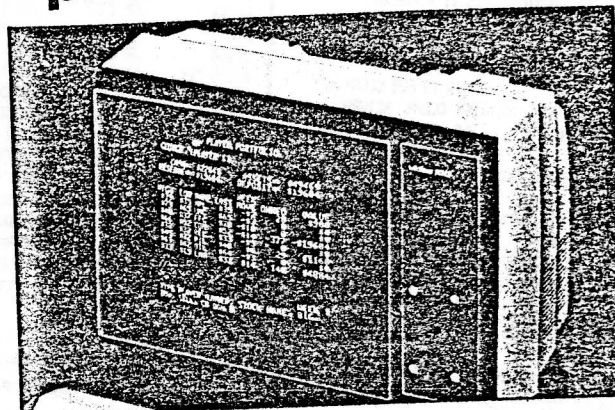
ONE CHARACTER CELL IS 12 SCAN LINES BY 8 DOTS
THIS DRAWING IS APPROXIMATELY TO SCALE.

7 6 5 4 3 2 1 0 VIDEO RAM DATA

Fig. 10. Scale drawing of one character cell shows that each graphics dot is approximately twice as tall as it is wide. The video RAM bits that control each graphics dot are also shown. This format matches the circuit in Fig. 9.

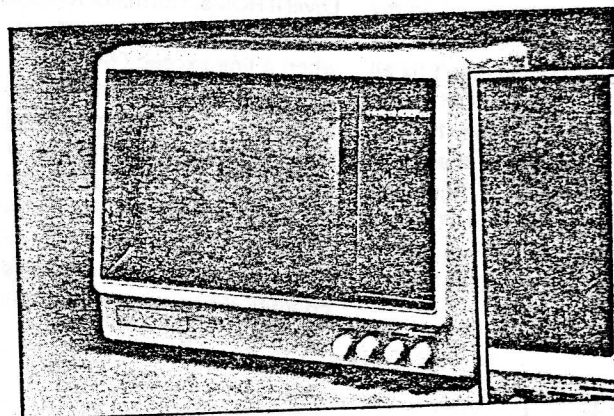
PROFESSIONAL B/W MONITORS

Designed for industry... priced for the home.



video 100 The video 100 computer monitors are ideal for all your personal and business needs. These highly reliable 12" black and white monitors feature a 12 MHz band width and 80 character by 24 line display. Plug-in compatibility with Apple, Atari, Radio Shack, O.S.I., Micro-Term and Exidy make these the perfect text display for almost any system.

UNDER \$160.00



video 100-80 The model 80 features an industrial grade metal cabinet with built-in disk mounting capability and space for an 11" x 14" PC board for custom designed electronics.

The solid state circuitry assures a sharp, stable, and trouble-free picture. The front panel controls include power, contrast, horizontal hold, vertical hold, and brightness. Adjustments for size, video level, and width are located on the rear panel.

UNDER \$200.00

VIDEO 100 AND VIDEO 100-80 SPECIFICATIONS

- 12" diagonal measure display
- Video band width 12 MHz ± 3 DB
- 80 character by 24 line display
- Video 100-80 provides mounting space for mini floppy disk.
- Resolution—Over 700 lines at center horizontally—over 350 lines at center vertically.
- Convenient front panel controls
- Input impedance 75 Ohms
- 90% deflection picture tube display

LEEDEX CORPORATION

2420-E Oakton St. • Arlington Heights, Illinois. 60005 • (312) 364-1180 • TLX: 25-4786

Dealer discount available


```

01290 TRS-80 SUBSTITUTE KEYBOARD DRIVER
01300 KBD5UB LD HL,KBDST ;KEYBOARD STATUS ADDRESS
01310 BIT 0,(HL)
01320 JR NZ,KBSB1 ;YES, ITS READY
01330 XOR A ;NOT READY
01340 RET ;RETURN WITH A NULL
01350 KBSB1 RES 0,(HL) ;RESET STATUS BIT
01360 INC HL ;BUMP POINTER TO DATA
01370 LD A,(HL) ;GET DATA
01380 CP RUBOUT ;IS IT A RUBOUT?
01390 JR NZ,KBSB2 ;NO
01400 LD A,BS ;YES, LOAD BACKSPACE
01410 RET ;AND RETURN
01420 KBSB2 CP ENH ;THIS IS FOR CLEAR KEY
01430 RET NZ ;NOT CLEAR, RETURN
01440 LD A,1FH ;YES, LOAD CLEAR CODE
01450 RET ;AND RETURN
01460 ;PRINTER DRIVER PATCH
01470 TRSPRT LD A,C ;MOVE DATA TO CORRECT REG
01480 LD HL,CHRONT ;GET POINTER
01490 CP 0CH ;FORM FEED
01500 JR Z,FORMF ;INITIALIZE LINE COUNT
01510 CP 0DH ;CARRIAGE RETURN
01520 JR Z,CR ;DO CARRIAGE RETURN
01530 LD A,(HL) ;GET CHAR COUNT

```

```

01540 CP 40H ;=64?
01550 CALL Z,CR ;INSERT A CR IF =64
01560 LD A,C ;GET DATA AGAIN
01570 TRSPRT CALL TYPOT ;OUTPUT CHR TO PRINTER
01580 INC HL ;BUMP CHAR COUNTER
01590 RET ;DONE
01600 CR DEC HL ;POINT TO LINE COUNT
01610 LD A,(HL) ;GET LINE COUNT
01620 DEC HL ;POINT TO LINES PER PAGE
01630 INC A ;BUMP LINE COUNT
01640 CP (HL) ;AT LIMIT?
01650 INC HL ;POINT TO LINE COUNT
01660 JR Z,NXTPAG ;YES, AT LIMIT
01670 CR LD (HL),A ;SAVE NEW LINE COUNT
01680 INC HL ;POINT TO CHAR COUNT
01690 LD (HL),0FFH ;RESET CHAR COUNT
01700 LD A,0DH ;GET A CARRIAGE RETURN
01710 JR TRSPRT ;DO CARRIAGE RETURN
01720 FORMF DEC HL ;POINT TO LINE COUNT
01730 NXTPAG CALL OK ;THIS ROUTINE PRINTS "OK?"
01740 ;AND WAITS FOR A CHARACTER FROM THE KEYBOARD BEFORE CONTINUING
01750 ;THIS ALLOWS ME TO PUT ANOTHER PAGE IN MY PRINTER.
01760 XOR A ;CLEAR A
01770 JR CR ;FINISH BY DOING CR

```

Listing 4. Keyboard driver replaces the one on the Level II ROM. It works in conjunction with my normal keyboard interrupt routine in Listing 1. The printer patch replaces the Level II printer driver.

ROM expected.

Actually, there were more characters that didn't quite match, but I reprogrammed my keyboard encoder EPROM to fix those. The reason I didn't fix all of them on the EPROM is because I would have had to change my monitor that used those characters. I figured it was better this way. Table 2 shows the control characters generated by the TRS-80 keyboard.

The printer patch adds a few features that my driver didn't have but are assumed by the Level II ROMs. The major feature is to add extra carriage returns when a line exceeds 64 characters in length. My first printer patch did not do this, and when I listed BASIC programs that had multiple statement lines longer than 64 characters, the extra characters would not print. I also added a lines-per-page counter. When the line count is

at the limit, the program waits for me to put another page in my printer.

The routine TYPOT in List-

ing 4 converts the ASCII input to the special code required by my modified Olivetti Lexikon 82 typewriter. The form-feed check in the program permits resetting the line counter to zero. This should be used before any new listing, so it starts at the top of a new page. From BASIC, the following line will work: LPRINT CHR\$(12);.

Specific Hardware Requirements

The single most important hardware requirement for your system is the use of interrupts. The reason for this is the way the keyboard is set up. When the Level II ROM initializes its RAM space, it assumes that you have a memory-mapped keyboard. If you don't have a memory-mapped keyboard and you don't have interrupts either, there is no way you can talk to the Level II program, and the computer will be hung up.

The only way to get the pro-

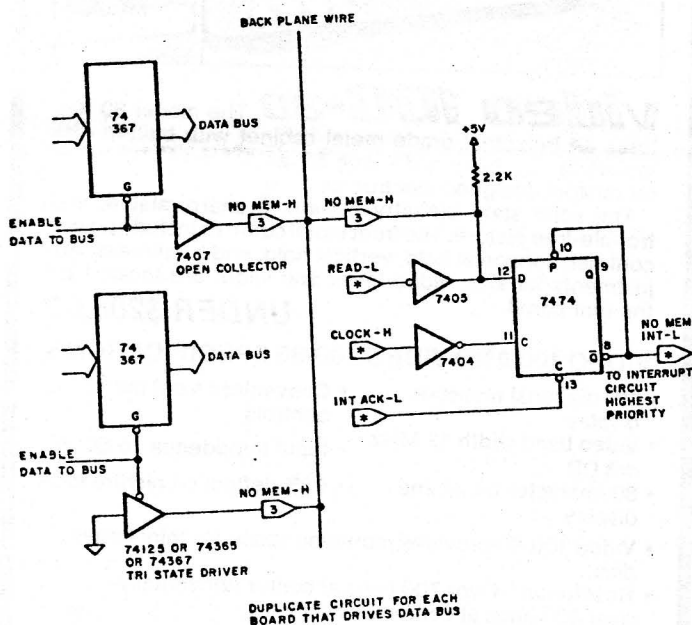


Fig. 8. To use this "No Memory" interrupt circuit, you must add a gate to every board in your system that drives the data bus. This circuit will generate an interrupt every time a nonexistent memory address or input port is read. This should be the highest priority interrupt in your system.

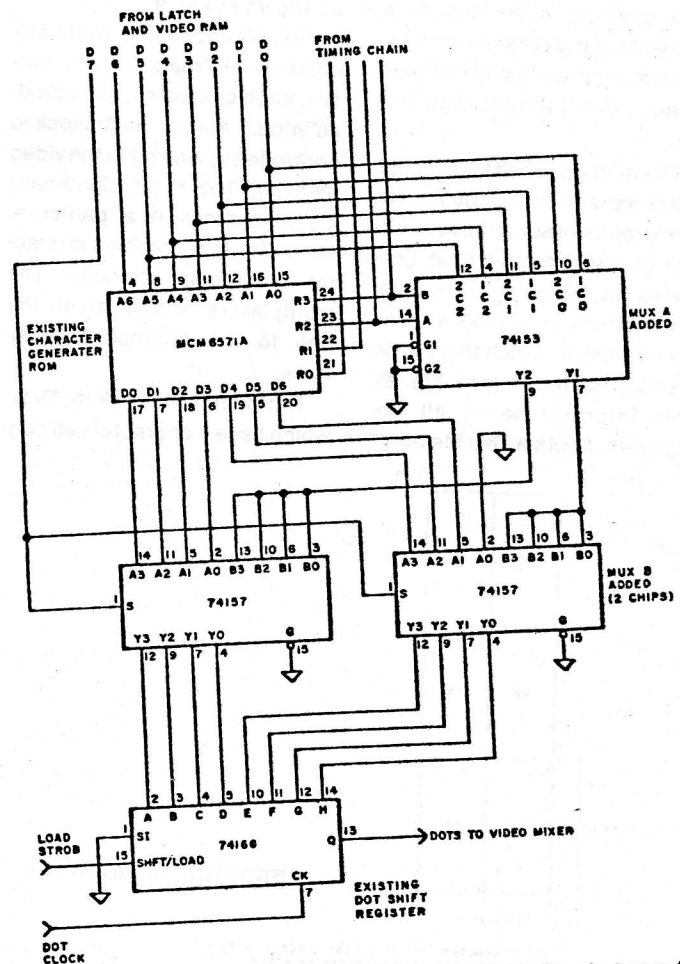


Fig. 9. This simple graphics mod will work for TVTs that use the MCM6571A character generator ROM. If you have a different character generator, the pin numbers will be different, but the circuit will be similar.

sisting of 12 lines of eight dots. If your TVT has 12 lines of six dots, simply tie the two outputs from mux A to each of three inputs on mux B instead of the four shown. If your TVT has a different arrangement of lines and dots, you have several choices.

First, you could stretch or shrink some of the graphics dots so they fill the available lines and dots in the character cell. This may cause some graphics dots to be different sizes than other ones if the total number of lines and dots are not evenly divisible by three and two, respectively.

Second, you could modify your TVT so it has a line count divisible by three and a dot count divisible by two. This is a bit tricky and should be attempted only after you have examined the schematic and understand the timing details of the TVT. The first mod is simpler and doesn't affect the timing, but you should still closely examine the schematic of your TVT before attempting to install the change.

Third, you could forget about the graphics. This is the simplest solution, but since a lot of game programs use the graphics, you may not want to do this. If you never play games, then you don't need the graphics anyway.

I suggest you try the first solution before trying the second. The slightly different size dots will go unnoticed in many applications anyway. My own TVT has a software-selectable character cell size. I can select 13 by 9 or 12 by 8. I normally operate in

the 13 by 9 mode and have found it satisfactory in many graphics applications.

If your video terminal is a completely separate unit from your computer, you obviously don't have a memory-mapped device. This means you can't use any part of the TRS-80 video driver. You will have to either write your own or modify the one you are presently using. The most important thing is to have the control characters respond correctly (see Table 1).

There are a few features in Level II BASIC that won't work with this type of setup. The graphics functions, SET, RESET and POINT, won't work, although you could send the graphics characters to the terminal like any other character. The PRINT@ and POS commands won't work either. Everything else should be fine though.

Your First Run

When you first try to run the Level II BASIC, you may have a different sequence of events than I do, depending on just how your hardware is configured. As you recall, my first run produced Greek characters. I no longer get Greek when I initialize the BASIC ROM. The first thing that appears is a "No Memory" message. This occurs when the ROM attempts to read the keyboard memory. I then type BC (BASIC Continue).

As described earlier, this changes some of the RAM locations just initialized by the ROM and returns to Level II BASIC. From here, my system behaves

just like a TRS-80.

If you don't have a "No Memory" interrupt on your system, and depending on what your TVT does with control characters, your system could produce Greek characters, some strange graphics characters or absolutely nothing. The next display will depend on what you have in the keyboard memory area. If this memory is all zeros, you will only see one line of whatever characters your system is producing. If the memory is all ones (FF hex) or random data, you should see several lines of these characters continuously being written to the TVT and scrolling off the screen.

No matter what you see, you should now hit your interrupt button (control-Z, or whatever) to put you back into monitor. After typing the BASIC Continue command, you should have a blank screen.

The ROM is now waiting for your response to the MEMORY SIZE question, even though you can't see that message. Typing anything should cause it to appear on the screen. Since there may be several unknown characters in the keyboard buffer, you should first delete these with the back-arrow key. When the cursor stops moving back, all characters have been deleted. Now answer the MEMORY SIZE question as you wish. If you hit a carriage return with garbage data, the ROM will ask the MEMORY SIZE question again.

One final note: if, on your system, memory address 37ECh returns anything other

than 00 or FFH when read, the ROM may attempt to boot the disk. I'm not sure exactly what will happen, but it will most likely get hung up and do nothing. If you have no memory at that address, you should be OK, since most systems read FFH or 00 to nonexistent memory.

Conclusions

For someone with a Z-80 microcomputer system who is looking for a good BASIC and would prefer to have it on ROM, Radio Shack's Level II ROM add-on kit for their TRS-80 is a good way to go. The price is reasonable—less than many BASICs that only come on cassette. If you consider the additional cost of EPROMs to put another BASIC on ROM, the Level II BASIC is less expensive than any other I know.

That the TRS-80 is the most popular microcomputer today ensures that there will be more directly compatible software than any one person can use. The ROM also contains a floppy disk bootstrap routine. This allows easy addition of one or more mini-floppy disk drives for a more versatile system. Radio Shack's TRSDOS may not be the best, but at only \$14.95, it certainly is the most inexpensive disk operating system I have ever seen. ■

References

- "TRS-80 Microcomputer Technical Reference Handbook," Radio Shack.
- "LEVEL II BASIC Reference Manual," Radio Shack.
- "TRSDOS & DISK BASIC Reference Manual," Radio Shack.

MOVING?

Let us know 8 weeks in advance so that you won't miss a single issue of *Kilobaud Microcomputing*. Attach old label where indicated and print new address in space provided. Also include your mailing label whenever you write concerning your subscription. It helps us serve you promptly.

- | | |
|---|---|
| <input type="checkbox"/> Address change only | <input type="checkbox"/> Payment enclosed |
| <input type="checkbox"/> Extend subscription | <input type="checkbox"/> Bill me later |
| <input type="checkbox"/> Enter new subscription | |
| <input type="checkbox"/> 1 year \$18.00 | |

If you have no label handy, print OLD address here.

AFFIX LABEL
 Name _____
 Address _____
 City _____ State _____ Zip _____

print NEW address here:

Name _____
 Address _____
 City _____ State _____ Zip _____

Kilobaud Microcomputing, P.O. Box 997, Farmingdale, NY 11737