Hal Chamberlin
29 Mead St.
Manchester NH 03104

Software Keyboard Interface

with a pittance of hardware !

I'll bet you're thinking, "Oh sure, another scheme using some obscure surplus keyboard that will be sold out by the time I get around to this project." Not so! This keyboard (manufactured by Datanetics Corp. of Fountain Valley CA) is offered by at least a half-dozen mail-order houses and is a current production item. But at $20 each (the most common price), these outfits are not doing us any favors; their cost is probably less than $10 each. An auxiliary keyboard the same style as the main unit is also available for less than $10 and can be used in this project for function keys, etc.

Why are these keyboards so cheap? The reason certainly is not lack of mechanical or electrical quality. They are unusually rigid one-piece construction of one-sixteenth-inch-thick Bakelite plastic, ribbed into a honeycomb form, with an overall depth of one-half inch. Each cell contains a contact arrangement with no fewer than four parallel contacts mounted inside a rugged plastic plunger. The contacts effectively reduce bounce and insure a long, error-free life. Finally, a keybutton is pressed onto the plunger, sealing the cell from dust and liquids.

One reason for the low cost is the one-piece base casting and cell structure. As I understand, the initial cost of the mold was borne by a huge quantity contract with Digital Equipment Corporation. However, the other reason is that the keyboard is devoid of any encoding electronics. This is the problem whose solution will be addressed in this article.

Besides the keyboard, the only other hardware this project requires is a single 74154 TTL integrated circuit (1-of-16 decoder), which costs less than two dollars, and some wire. Only four of the I/O port bits on the KIM's application connector are used, and even these may be used for other purposes when typing is not actually being done. Standard two-key rollover operation (which will be described later) is provided, and a full uppercase and lowercase ASCII character set is available. Even the repeat key works and has a programmable rate. The auxiliary keyboard is also supported with codes from its keys being identified by having the eighth bit set to a one. Even though some of the KIM's built-in keyboard circuitry is utilized, there is no conflict (with one small exception) between the built-in keypad and the new alphanumeric keyboard. A slight amount of additional circuitry using another IC may be added to have the break key function as an interrupt.

A software routine of approximately 350 bytes does all of the key scanning and code translation. This, in fact, is how the on-board KIM keypad is handled, with the difference being that the scanning software is in the KIM monitor ROM. If a code other than ASCII is desired, such as EBCDIC or Baudot, a translate table in the software may be easily altered. This table can be changed to suit different application programs, such as ASCII for running Tiny BASIC or Baudot for an automated RTTY application. The complete assembled and tested program is given at the end of this article.

Keyboard Scanning Theory

Nearly all keyboards in common use with more than a few keys use some kind of scanning logic to detect keyswitch closures, eliminate contact bounce, and generate unique key codes. In operation, scanning logic sequentially tests the state (up or down) of each individual key in the array. When a key is found in the down position, its code is determined and sent out. In order to avoid the code's being sent out more than once for each key depression, the scanning is stopped while the key is down and resumed when it is released. Typical scanning rates range from 20 to 500 complete scans per second of the approximately 60 keys in an average array.

Besides being a simple and inexpensive method of having a single logic circuit monitor the states of 60 individual keys, scanning also can cope with simultaneous key depressions. When someone is typing at substantial speed it is a common occurrence for more than one key to be down simultaneously. For example, consider rapid typing of the word THE. The T would first be pressed, followed shortly thereafter by a finger of the other hand pressing the H. Next the T would be released and the E would be quickly pressed with another finger of the same hand. Subsequently, the H would be released followed by the E, which completes the triad. A scanning keyboard would actually send the proper THE sequence to the computer, with no additional logic or buffer register required.

In order to understand how this works, let us examine the detailed sequence of events. Initially, no keys are pressed, and the scanning circuitry is running at full speed. When the T is pressed, the scanner eventually finds it, sends the T code and stops. As long as the T is held down, the scanner is stopped and testing the T key. While waiting for the T to be released, the typist presses the H, but the scanner is not aware of it. When the T is

finally released, the scanner takes off again but is immediately stopped when it sees that the H key is down. After sending the H code it waits for the H to be released, and so on.

If the typist is sloppy (or unusually fast) it is possible for even the E key to be pressed before the T is released, resulting in three keys being down simultaneously. In this situation, two keys are pressed while the scanner is waiting for the T key to be released. When scanning is resumed, two keys are down. The scanner will see the one that is closest to T in the scanning sequence and send that code next. The closest key might very well be the E, resulting in an error. This action on multiple key depressions is termed two-key rollover and is found on most computer terminals and other equipment used by casual typists. Some word-processing machines and other equipment used by professional typists have N-key rollover logic, which responds only to the order of key depression, regardless of how many keys are down simultaneously or the order in which they are released. Either special keyswitches or more complex scanning logic can be used to achieve N-key rollover. This keyboard interface is capable of N-key rollover with a more complex scanning program.

The scanning method can also easily take care of keyswitch contact bounce. When a closed contact is found, scanning is stopped, but sending of the code is delayed. If the contact should open during the delay, the closure is ignored and scanning is resumed without sending the code. If the momentary closure was really due to contact bounce, the key will be seen again on the next scan. If the closure is solid for the entire delay time, the code is sent. In addition, noise on contact opening may be rejected by requiring that the contact remain continuously open for a delay period before scanning is resumed. Typical values of debounce delay are one to five milliseconds.

Now, how is scanning circuitry typically implemented? One simple scheme for up to 64 keys would be to have an oscillator drive a 6-bit binary counter. The output of the counter would drive a decoder network having 64 separate outputs. All but one of the decoder outputs would be off, with the one on corresponding to the binary number in the counter. As the counter counts, each of the 64 decoder outputs would be turned on in sequence. For scanning a keyboard, each decoder output would be connected to one side of a keyswitch contact as shown in Fig. 1. The other sides of the contacts would all be connected together. This signal would be a zero except when a keyswitch was closed *and* that particular switch was addressed by the counter and decoder. With proper wiring between the decoder and the switch array, the 6-bit content of the binary counter while it is addressing a closed key can be the actual desired code of that key! Thus encoding is automatic with a scanning keyboard. Unfortunately, the shift and control keys of a typical keyboard complicate coding matters somewhat, but the basic concept is still valid.

Actually the scanning logic and switch wiring can be simplified greatly from the above conceptual model by arranging the keys in a matrix. Taking the same 64-key array, let us wire the keys in a matrix of eight rows and eight columns with a signal wire for each row and column. The contacts of a switch will be wired across each intersection, as shown in Fig. 2. Using the same 6-bit counter, let us connect three of the bits to a one-of-eight decoder and the other three bits to an 8-input multiplexer. A multiplexer is a logic circuit that has several signal inputs, some binary address inputs and one output. In operation, one of the signal inputs is logically connected to the output according to the binary code at the address inputs. The single output of the multiplexer is the addressed-key-closed signal as before. With matrix connection of the keys, the scanning logic grows in proportion to the square root of the number of keys, instead of directly.

As the scanning counter counts, the decoder activates one column of the matrix at a time and the multiplexer sequentially examines each row for a closed switch transferring the column signal over to a row. When a closed switch is found, the counter contains a unique code for the switch as before. Although it is still possible for this code to be the actual desired keycode, the scrambled key layout of a typical keyboard would make the matrix wiring quite messy. Typically a read only memory is used to translate the scramble code from the scanner into the end-use code the computer system needs. This same ROM also takes care of the shift and control keys, which are wired in directly.

Connection to the KIM

All of the previously described functions of scanning hardware can also be easily performed by software, along with an output and an input port. The most straightforward approach to simulate matrix scanning hardware would be to use an 8-bit

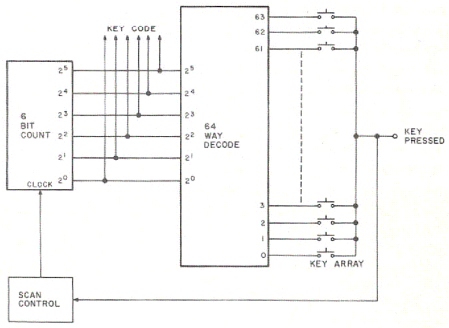


Fig. 1. Basic keyboard scanner.

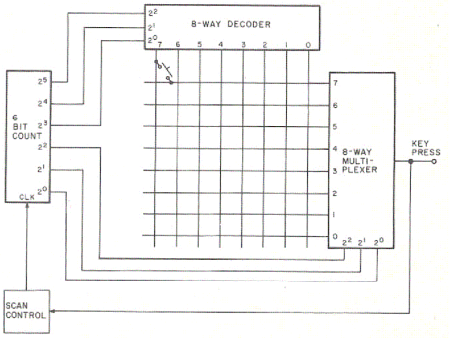
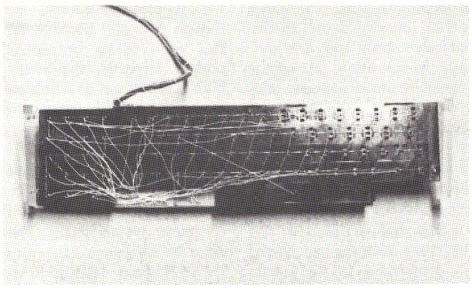


Fig. 2. Matrix keyboard scanner.



Keyboard point-to-point wiring.

output port with software to simulate the one-of-eight decoder and an 8-bit input port with software to simulate the 8-input multiplexer. The counter, of course, would be just a memory location that is incremented to perform the scanning. Unfortunately, in the case of the KIM this would utilize all of the built-in ports and then some.

A look at the KIM manual will reveal that much of the circuitry for the on-board keypay has signals brought out to the application edge connector. In particular, seven bits of an internal input port are available. These are connected internally to the on-board keypad and seven-segment displays, but when the KIM monitor is not running (user program running) they are completely free for use as an input port. Of course, when the monitor *is* in control, these inputs must not be driven by external circuitry, or interference with the keypad and display will result. If this port is connected to the rows of a key matrix and no keys are pressed, then nothing is driving the row wires; they are just hanging. Thus, when using the KIM monitor, one would not expect to be typing on the external keyboard so any interference is completely avoided.

At this point, one could use an 8-bit output port on the KIM to drive the key matrix and handle up to 56 keys without any interfacing circuitry. If you do not need the one full 8-bit port, and a limited character set (some missing symbols) is sufficient for your needs, then this can indeed be done. However, on my system the 8-bit port is connected to a digital-to-analog converter (for playing music) and two of the seven bits on the other port are motor controls for two cassette recorders. This leaves five bits for selecting the column to be scanned. The solution is to use four of these bits and an external 1-of-16 decoder to drive up to 16 columns. Combined with seven rows, up to 112 keys could be scanned.

Fig. 3 shows the connections to the KIM and the matrix hookup of the keys. Note that the optional 19-key keyboard is included. The arrangement of keys in the matrix was chosen mostly for simplicity of wiring, with proper coding taken care of with translation software. The one exception is the wiring of the 0-F keys on the auxiliary keyboard. They are in order with the 0 key in column 0, 1 key in column 1, etc. This would simplify a scanning routine that uses just those 16 keys. The 74154 decoder needs about 35 milliamps of +5 volt power. This should not strain any decent power supply for the KIM, but could be reduced to a mere 10 milliamps if a 74LS154 was substituted.

Note that the two shift keys are both wired into the matrix at row 3, column 15. The key labeled SHIFT on the auxiliary keyboard is intended to be relabeled and used for a less redundant function. The shift lock can be connected across the other two shift keys, but a problem arises in doing so. If it is left in the lock position when using the KIM monitor, there can be interference between the add-on keyboard and the KIM keyboard. If the shift-lock function is desired, and the requirement that it be unlocked before using the monitor is not judged to be bothersome, then the shift-lock key may be wired in.

Wiring the little tabs sticking out of the back of the keyboard should not be difficult. They are stiff enough and long enough to be wire-wrapped, too, if care is taken. Actually, this would be an ideal use of a Vector wiring pencil, which should get the job done in about 30 minutes. If hand wiring and soldering must be done, however, it is permissible to use bare bus wire for the row wiring and insulated wire for the columns. The purist can mount the 74154 IC in a socket on a piece of perfboard, but there is no reason that it cannot be glued to the bottom or side of the keyboard and wired directly.

The little circuit in Fig. 4 can be added to allow the Break key to be used as an interrupt. The KIM board would respond to this key in

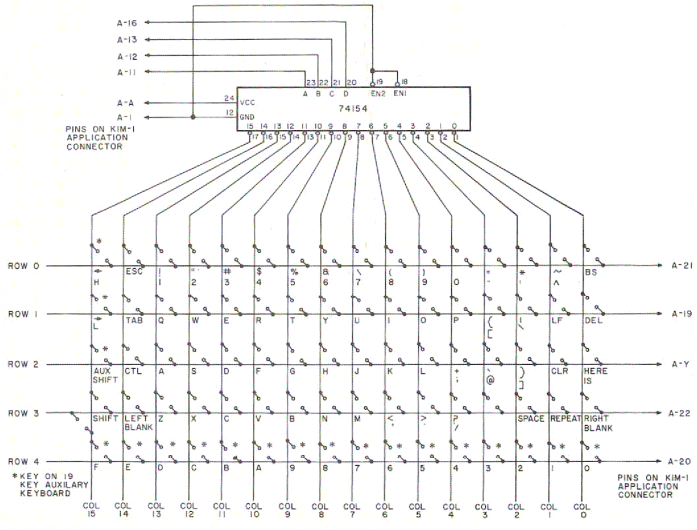


Fig. 3. Complete KIM-1 alphanumeric keyboard interface schematic.

the same manner as the ST key on the built-in keypad and return to the monitor. However, if the nonmaskable interrupt (NMI) vector is changed at 17FA and 17FB, the interrupt could jump to a specific point in the user's program instead. The resistors, capacitor and 7413 Schmitt trigger IC debounce the break key to prevent multiple interrupts. The diode in series with the output simulates an open-collector output so that normal ST key operation is not affected. Preferably, the diode is a germanium type such as a 1N34 or 1N270, but a silicon unit will generally work OK.

Scanning Program

The program in Fig. 5 is the heart of the add-on keyboard system and is responsible for most of its features. Although shown assembled for locations 0200-035C (hexadecimal), it may be modified for execution anywhere by changing those locations marked with an underline in the object-code column. One temporary storage location is required on page 0. Its initial value when the keyboard is first used in a user program is not important, but thereafter it should not be bothered. The routine may be interrupted with no ill effects, but it is not reentrant (that is, it may not be called by an interrupt-service routine if it was itself interrupted) due to the temporary storage location just mentioned. This temporary location is at 00EE (just below the KIM reserved area) in the listing shown but may be easily moved elsewhere.

Using the program is quite simple. It is called as a subroutine whenever a character from the keyboard is needed. The contents of the registers when called are not important. When called, the routine waits until a key is pressed (except for code, shift or repeat). When a key is pressed, its code is loaded into the accumulator and a return taken. For maximum flexibility, the contents of the index registers are not disturbed by the routine.

Before you get into the program logic, perhaps a word should be said about the assembly language. The assembler used to prepare the listing is a modified version of the National Semiconductor IMP-16, which, in turn, is similar to the PACE assembler. In most respects, the syntax conforms to that recommended by MOS Technology. The major difference is that hexadecimal constants are denoted by X' instead of $. The use of a # before a constant or symbol specifies the immediate addressing mode. The assembler automatically distinguishes between zero page and absolute mode addressing according to the numerical magnitude of the address — zero page if between 0000 and 00FF and absolute otherwise. The various indexed and indirect addressing modes are represented in the same way as with the MOS Technology assembler.

The overall logic of the keyboard subroutine closely parallels that described for a hardware keyboard scanner. The first step when it is entered is to save the index registers on the stack. Next, the direction registers for the input and output port bits are set up. Note that only the direction bits for the port bits actually used are changed; the others are left unchanged.

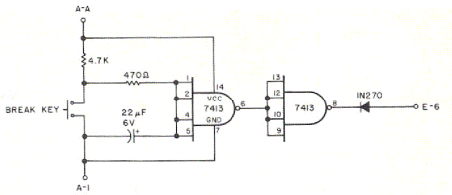


Fig. 4. Optional break-key interface.

When the subroutine is entered, an assumption is made that the last key pressed is still down. This is certainly a valid assumption since a return from the previous invocation of this subroutine occurred immediately when a key was pressed, and it is unlikely that processing of that character by the calling program took very long. ANKBT1 is the temporary storage location mentioned earlier. Functionally, it is equivalent to the counter in a hardware keyboard scanner. It always addresses a key in the matrix, and in this case it points to the key that was last pressed and had its code sent.

Thus, after saving the registers and setting up the ports, a loop is entered in which the keyboard routine is waiting for this last-pressed key to be released. While in this waiting loop, the status of the repeat key is continually interrogated. If the repeat key is continuously down while the last-pressed key is also continuously down for the repeat period, an exit is taken from the loop and the key code is sent again. Note that the repeat period, RPTRAT, is a parameter that may be changed; in this case it is set to 50 milliseconds, giving a moderately fast repeat rate of approximately 20 characters per second.

An internal subroutine, KYTST, is used to actually test the state of a key. It is used by loading the address of the key to be tested into the accumulator, and then calling it. When it returns, the carry flag will be on if the key is up, and off if it is down.

The other exit from this waiting loop, of course, is sensing that the last addressed key has been released. A debounce delay (DBCDLA) is included to insure that the key is interpreted to be up only when it has been continuously up for the debounce delay period. This will prevent noisy contacts from generating multiple characters.

At this point, scanning of the keyboard resumes.

Fig. 5. KIM-1 alphanumeric keyboard scan and encode routine.

```
 1                          .PAGE  'KIM-1 ALPHANUMERIC KEYBOARD SCAN AND ENCODE ROUTINE'
 2                ;         THIS SUBROUTINE SCANS AN UNENCODED KEYBOARD MATRIX CONNECTED
 3                ;         TO THE KIM-1 APPLICATION CONNECTOR.  USER PERIPHERAL PORT B
 4                ;         BITS 5 (MSB) THROUGH 2 (LSB) ARE CONNECTED TO A ONE-OF-16
 5                ;         DECODER (74154) WHICH DRIVES THE KEYSWITCH COLUMNS.
 6                ;         SENSING OF THE ROWS IS BY A PORTION OF THE KIM ON-BOARD
 7                ;         KEYBOARD CIRCUITRY WHICH USES SYSTEM PERIPHERAL PORT B BITS
 8                ;         0 - 4.
 9                ;         WHEN CALLED, THE ROUTINE SITS IN A LOOP WAITING FOR A KEY TO
10                ;         BE PRESSED.  WHEN A KEY IS PRESSED (EXCEPTING SHIFT, CONTROL,
11                ;         REPEAT), THE ROUTINE RETURNS WITH KEY CODE IN ACCUMULATOR.
12                ;         BOTH INDEX REGISTERS ARE RETAINED.
13                ;         THE ROUTINE IMPLEMENTS TRUE 2-KEY ROLLOVER, KEY DEBOUNCING,
14                ;         AND REPEAT TIMING.  ONE RAM LOCATION IS REQUIRED, ITS INITIAL
15                ;         CONTENT IS INSIGNIFICANT.
16
17 0000                          .=     X'200        ; START PROGRAM AT LOCATION 0200 (HEX)
18
19 1740           SYSPA   =      X'1740       ; SYSTEM PORT A DATA REGISTER
```

```
                SYSPAD  =       X'1741          ; SYSTEM PORT A DIRECTION REGISTER
                USRPB   =       X'1702          ; USER PORT B DATA REGISTER
                USRPBD  =       X'1703          ; USER PORT B DIRECTION REGISTER
                RPTRAT  =       50              ; REPEAT PERIOD, MILLISECONDS
                DBCDLA  =       5               ; DEBOUNCE DELAY, MILLISECONDS

                ANKBT1  =       X'EE            ; TEMPORARY STORAGE LOCATION ADDRESS


0200 98         ANKB:   TYA                     ; SAVE THE INDEX REGISTERS
0201 48                 PHA
0202 8A                 TXA
0203 48                 PHA
0204 AD4117             LDA     SYSPAD          ; SET UP DATA DIRECTION REGISTERS
0207 29E0               AND     #X'E0           ; SET SYSTEM PORT A BITS 4-0 TO INPUT
0209 8D4117             STA     SYSPAD
020C AD0317             LDA     USRPBD
020F 093C               ORA     #X'3C           ; SET USER PORT B BITS 5-2 TO OUTPUT
0211 8D0317             STA     USRPBD
0214 A032               LDY     #RPTRAT         ; INITIALIZE REPEAT DELAY
0216 A205       ANKB1:  LDX     #DBCDLA         ; INITIALIZE DEBOUNCE DELAY
0218 207B02     ANKB2:  JSR     WA1MS           ; WAIT 1 MILLISECOND
021B A5EE               LDA     ANKBT1          ; GET KEY ADDRESS LAST DOWN
021D 208202             JSR     KEYTST          ; TEST IF ADDRESSED KEY STILL DOWN
0220 B00C               BCS     ANKB4           ; JUMP IF UP
0222 A931               LDA     #X'31           ; TEST STATE OF REPEAT KEY
0224 208202             JSR     KEYTST
0227 B0ED               BCS     ANKB1           ; LOOP BACK IF REPEAT KEY IS UP
0229 88                 DEY                     ; DECREMENT REPEAT DELAY
022A D0EA               BNE     ANKB1           ; LOOP BACK IF REPEAT DELAY UNEXPIRED
022C F022               BEQ     ANKB7           ; GO OUTPUT REPEATED CODE
022E CA         ANKB4:  DEX                     ; DECREMENT DEBOUNCE DELAY
022F D0E7               BNE     ANKB2           ; GO TEST KEY AGAIN IF NOT EXPIRED

                ;       PREVIOUS KEY IS NOW RELEASED, RESUME SCAN OF KEYBOARD

0231 E6EE       ANKB5:  INC     ANKBT1          ; INCREMENT KEY ADDRESS TO TEST
0233 A5EE               LDA     ANKBT1
0235 C93F               CMP     #X'3F           ; SKIP OVER SHIFT
0237 F0F8               BEQ     ANKB5
0239 C92E               CMP     #X'2E           ; SKIP OVER CONTROL
023B F0F4               BEQ     ANKB5
023D C931               CMP     #X'31           ; SKIP OVER REPEAT
023F F0F0               BEQ     ANKB5
0241 A205               LDX     #DBCDLA         ; INITIALIZE DEBOUNCE DELAY
0243 A5EE       ANKB6:  LDA     ANKBT1          ; TEST STATE OF CURRENTLY ADDRESSED KEY
0245 208202             JSR     KEYTST
0248 B0E7               BCS     ANKB5           ; GO TRY NEXT KEY IF THIS ONE IS UP
024A 207B02             JSR     WA1MS           ; WAIT 1 MILLISECOND IF DOWN
024D CA                 DEX                     ; DECREMENT DEBOUNCE DELAY
024E D0F3               BNE     ANKB6           ; GO CHECK KEY AGAIN IF NOT EXPIRED

                ;       TRANSLATE AND OUTPUT A KEY CODE

0250 A6EE       ANKB7:  LDX     ANKBT1          ; GET BASIC ASCII CODE FROM TABLE
0252 BCBD02             LDY     ANKBTB,X        ; INTO INDEX Y
0255 A92E               LDA     #X'2E           ; TEST STATE OF CONTROL KEY
0257 208202             JSR     KEYTST
025A B006               BCS     ANKB8           ; SKIP AHEAD IF NOT PRESSED
025C 98                 TYA                     ; CLEAR UPPER THREE BITS OF CODE IF
025D 291F               AND     #X'1F           ; CONTROL PRESSED
025F 4C7002             JMP     ANKB10          ; IGNORE SHIFT AND GO RETURN
0262 A93F       ANKB8:  LDA     #X'3F           ; TEST STATE OF SHIFT KEY
0264 208202             JSR     KEYTST
0267 9004               BCC     ANKB9           ; SKIP AHEAD IF PRESSED
0269 98                 TYA                     ; RETRIEVE PLAIN CODE FROM Y
026A 4C7002             JMP     ANKB10          ; GO RESTORE REGISTERS AND RETURN
026D BD0D03     ANKB9:  LDA     ANKBTB+80,X     ; FETCH SHIFTED CODE FROM TABLE
0270 BA         ANKB10: TSX
0271 BC0201             LDY     X'102,X         ; RESTORE Y FROM STACK
0274 9D0201             STA     X'102,X         ; SAVE CHARACTER CODE IN STACK WHERE Y WAS
0277 68                 PLA                     ; RESTORE X
0278 AA                 TAX
0279 68                 PLA                     ; RESTORE CHARACTER CODE IN A
027A 60                 RTS                     ; RETURN

                ;       WAIT FOR ONE MILLISECOND ROUTINE

027B A9C8       WA1MS:  LDA     #200            ; WAIT FOR APPROXIMATELY 1 MILLISECOND
027D E901       WA1MS1: SBC     #1
027F D0FC               BNE     WA1MS1
0281 60                 RTS

                ;       KEY STATE TEST ROUTINE
                ;       ENTER WITH ADDRESS OF KEY TO TEST IN ACCUMULATOR
                ;       LEAVES BOTH INDEX REGISTERS ALONE
                ;       SETS ANKBT1 TO ZERO IF ILLEGAL KEY ADDRESS AND TESTS KEY ZERO
```

Scanning is accomplished by incrementing ANKBT1 and calling KEYTST to look at the state of the newly addressed key. Note that the shift, code and repeat keys are specifically skipped in the scan sequence. Also note that another function of KEYTST is to detect an illegal key address and set ANKBT1 to zero if an illegal address occurs. Such an illegal address would normally occur after testing the last key in sequence, so the forced reset to zero would start another scanning cycle. If a key is found depressed, another loop is entered that verifies that it is continuously depressed for the debounce delay interval before it is declared to be really pressed.

Once a newly pressed key has been found (or the conditions for a repeated character have been satisfied), the key code must be generated. First, the current key address in ANKBT1 is translated into a plain unshifted character code by using it as an index into the first part of the code table. Next, the state of the control key is tested. If it is down, only the lower five bits of the translated code are retained, and an exit is taken. If control is up, then the shift key is tested. If it, too, is up, an exit is taken. If the shift key is down, however, the code is retranslated using the second part of the code table. Note that with a code like ASCII, with logical bit pairing (unshifted and shifted codes differ by only one bit), the second half of the code table might be replaced with a little more programming to make the adjustments necessary on shifted characters.

Finally, the two index registers are restored and a return taken. Note that some playing around with the stack was necessary to preserve the character code in A while the other registers were restored.

The key state test routine, KEYTST, takes a key address in A and tests if the corresponding key is pressed. After checking for a valid key

address, and correcting it if not, the lower four bits of the address are sent to the port bits that have the 1-of-16 column decoder connected to them. These four port bits are updated without affecting any of the other bits on the same port. After the column address is sent out, the remaining three upper bits of the key address are used to access a "mask table," which selects one of the five significant row input bits to test. Then the input port that senses the five rows is read and tested against the mask. The zero or nonzero result is transferred to the carry flag, which won't be destroyed during the register restore sequence.

The code translate table is divided into two parts. The first is for unshifted codes; the second is for shifted codes. The characters are in matrix-wise order, starting with row 0, column 0, going through the columns on row 0, proceeding to row 1, and so forth, ending with row 4, column 15. The table given is for ASCII on the main keyboard. The blank or oddly marked keys are assigned to useful ASCII control codes such as CR for the key marked CLR. The 0-F keys of the auxiliary keyboard become 80-8F for lowercase and 90-9F for uppercase. The remaining three auxiliary keys are assigned codes A0-A5. The table may be changed freely to reflect the user's choice of convenient control codes or to accommodate a completely different character code.

Building this keyboard interface for the KIM should prove to be a worthwhile one-evening project. Besides saving a substantial amount of money, it serves as a good learning tool and an excellent example of how software can substitute for hardware, offer a lot of extra features and still be easy to use. The basic concepts can be easily applied to expanding other low-cost microcomputer trainer boards.■

```
108
109
110
111 0282 C950      KEYTST:  CMP    #80          ; TEST IF LEGAL KEY ADDRESS
112 0284 9004               BCC    KEYTS1       ; SKIP AHEAD IF SO
113 0286 A900               LDA    #0           ; SET TO ZERO OTHERWISE
114 0288 85EE               STA    ANKBT1       ; UPDATE ANKBT1
115 028A 48       KEYTS1:  PHA                 ; SAVE A ON STACK
116 028B 8A                TXA                 ; SAVE X ON STACK
117 028C 48                PHA
118 028D AD0217            LDA    USRPB        ; CLEAR USER PORT B BITS 2-5
119 0290 29C3              AND    #X'C3
120 0292 8D0217            STA    USRPB
121 0295 BA                TSX                 ; RESTORE KEY ADDRESS FROM STACK
122 0296 BD0201            LDA    X'102,X
123 0299 290F              AND    #X'0F        ; ISOLATE LOW 4 BITS OF KEY ADDRESS
124 029B 0A                ASLA                ; POSITION TO LINE UP WITH BITS 2-5
125 029C 0A                ASLA
126 029D 0D0217            ORA    USRPB        ; SEND TO USER PORT B WITHOUT DISTURBING
127 02A0 8D0217            STA    USRPB        ; OTHER BITS
128 02A3 BD0201            LDA    X'102,X      ; GET KEY ADDRESS BACK
129 02A6 4A                LSRA                ; RIGHT JUSTIFY HIGH 3 BITS
130 02A7 4A                LSRA
131 02A8 4A                LSRA
132 02A9 4A                LSRA
133 02AA AA                TAX                 ; USE AS AN INDEX INTO MASK TABLE
134 02AB AD4017            LDA    SYSPA        ; GET SYSTEM PORT A STATUS
135 02AE 3DB802            AND    MSKTAB,X     ; SELECT BIT TO TEST AND SET CARRY FLAG
136 02B1 18                CLC                 ; ACCORDINGLY
137 02B2 E900              SBC    #0
138 02B4 68                PLA                 ; RESTORE X FROM STACK
139 02B5 AA                TAX
140 02B6 68                PLA                 ; RESTORE A FROM STACK
141 02B7 60                RTS                 ; RETURN
142
143 02B8 01020408 MSKTAB:  .BYTE  X'01,X'02,X'04,X'08    ; MASK TABLE FOR KEYTST
144 02BC 10                .BYTE  X'10
145
146                ;       ASCII CHARACTER CODE TRANSLATE TABLE
147
148
149                                                      ; UNSHIFTED SECTION
150 02BD 085E3A2D ANKBTB:  .BYTE  X'08,X'5E,X'3A,X'2D    ; BS  CARRET  :  -
151 02C1 30393837          .BYTE  X'30,X'39,X'38,X'37    ; 0  9  8  7
152 02C5 36353433          .BYTE  X'36,X'35,X'34,X'33    ; 6  5  4  3
153 02C9 32311BA0          .BYTE  X'32,X'31,X'1B,X'A0    ; 2  1  ESC  (AUX H)
154 02CD 7F0A5C5B          .BYTE  X'7F,X'0A,X'5C,X'5B    ; DEL  LF  BACKSLASH  [
155 02D1 706F6975          .BYTE  X'70,X'6F,X'69,X'75    ; P  O  I  U
156 02D5 79747265          .BYTE  X'79,X'74,X'72,X'65    ; Y  T  R  E
157 02D9 777109A1          .BYTE  X'77,X'71,X'09,X'A1    ; W  Q  HT  (AUX L)
158 02DD 060D5D40          .BYTE  X'06,X'0D,X'5D,X'40    ; HEREIS  CR  ]  @
159 02E1 3B6C6B6A          .BYTE  X'3B,X'6C,X'6B,X'6A    ; ;  L  K  J
160 02E5 68676664          .BYTE  X'68,X'67,X'66,X'64    ; H  G  F  D
161 02E9 736100A2          .BYTE  X'73,X'61,X'00,X'A2    ; S  A  CTL  (AUX SHIFT)
162 02ED 00002000          .BYTE  X'00,X'00,X'20,X'00    ; (RIGHT BLANK)  REPAT  SP
163 02F1 2F2E2C6D          .BYTE  X'2F,X'2E,X'2C,X'6D    ; /  .  ,  M
164 02F5 6E627663          .BYTE  X'6E,X'62,X'76,X'63    ; N  B  V  C
165 02F9 787A0000          .BYTE  X'78,X'7A,X'00,X'00    ; X  Z  (LEFT BLANK)  SHIFT
166 02FD 80818283          .BYTE  X'80,X'81,X'82,X'83    ; (AUX 0  1  2  3)
167 0301 84858687          .BYTE  X'84,X'85,X'86,X'87    ; (AUX 4  5  6  7)
168 0305 88898A8B          .BYTE  X'88,X'89,X'8A,X'8B    ; (AUX 8  9  A  B)
169 0309 8C8D8E8F          .BYTE  X'8C,X'8D,X'8E,X'8F    ; (AUX C  D  E  F)
170
171                                                      ; SHIFTED SECTION
172
173 030D 085E2A3D          .BYTE  X'08,X'5E,X'2A,X'3D    ; BS  CARRET  *  =
174 0311 30292827          .BYTE  X'30,X'29,X'28,X'27    ; 0  )  (  '
175 0315 26252423          .BYTE  X'26,X'25,X'24,X'23    ; &  %  $  #
176 0319 22211BA3          .BYTE  X'22,X'21,X'1B,X'A3    ; "  !  ESC  (AUX H)
177 031D 7F0A7C7B          .BYTE  X'7F,X'0A,X'7C,X'7B    ; DEL  LF  VERTBAR  {
178 0321 504F4955          .BYTE  X'50,X'4F,X'49,X'55    ; P  O  I  U
179 0325 59545245          .BYTE  X'59,X'54,X'52,X'45    ; Y  T  R  E
180 0329 575109A4          .BYTE  X'57,X'51,X'09,X'A4    ; W  Q  HT  (AUX L)
181 032D 060D7D60          .BYTE  X'06,X'0D,X'7D,X'60    ; HEREIS  CR  }  GRAVEACCENT
182 0331 2B4C4B4A          .BYTE  X'2B,X'4C,X'4B,X'4A    ; +  L  K  J
183 0335 48474644          .BYTE  X'48,X'47,X'46,X'44    ; H  G  F  D
184 0339 534100A5          .BYTE  X'53,X'41,X'00,X'A5    ; S  A  CTL  (AUX SHIFT)
185 033D 00002000          .BYTE  X'00,X'00,X'20,X'00    ; (RIGHT BLANK)  REPAT  SP
186 0341 3F3E3C4D          .BYTE  X'3F,X'3E,X'3C,X'4D    ; ?  >  <  M
187 0345 4B425643          .BYTE  X'4B,X'42,X'56,X'43    ; N  B  V  C
188 0349 585A0000          .BYTE  X'58,X'5A,X'00,X'00    ; X  Z  (LEFT BLANK)  SHIFT
189 034D 90919293          .BYTE  X'90,X'91,X'92,X'93    ; (AUX 0  1  2  3)
190 0351 94959697          .BYTE  X'94,X'95,X'96,X'97    ; (AUX 4  5  6  7)
191 0355 98999A9B          .BYTE  X'98,X'99,X'9A,X'9B    ; (AUX 8  9  A  B)
192 0359 9C9D9E9F          .BYTE  X'9C,X'9D,X'9E,X'9F    ; (AUX C  D  E  F)
193
194 0000                   .END
NO ERROR LINES
```