Micro
Technology
Unlimited

K-1008-2L PATCHES TO MICROSOFT BASIC

FOR 6502 PROCESSORS

FEBRUARY 1, 1979

K-1008-2 BASIC PATCHES USER'S MANUAL

The K-1008-2 BASIC Patches software package allows the MTU K-1008 Visible Memory to be used as a terminal display and graphics output device with BASIC. It is designed to work with Microsoft BASIC for the KIM (it is compatible with the Synertek SYM and the Rockwell AIM as well). In order to use the package, the user must first obtain a copy of "Microsoft 9 Digit BASIC" which has been assembled starting at address 2000_{16}. Microsoft 9 Digit BASIC is available from Johnson Computer (Box 523, Medina, OH 44256)

This software package consists of 3 machine language programs and a demonstration program written in BASIC. The first program consists of a text display routine, a set of plotting routines, a routine that "pokes" patches into BASIC, and a dispatch routine. This program is loaded immediately after the BASIC interpreter at address 4261 and extends up through address 49D7.

The second and third programs are keyboard handler routines that can be used in place of a serial teletype keyboard. The first is written for an unencoded keyboard that is available from Jameco Electronics (1021 Howard Ave., San Carlos, CA 94070). The second is written for nearly any kind of parallel encoded ASCII keyboard with a 7-bit plus strobe output. Either routine implements all of the control codes recognized by BASIC correctly, something that is not possible with a teletype keyboard. Both programs start at address 0200.

The fourth program is a BASIC demonstration program that shows off the graphics capabilities of the system and verifies that it is working properly.

These are recorded on the enclosed cassette first in Hypertape format then in standard KIM format.

Prog. #	ID	Address	Description
1	01	4261-49D7	Text, graphics, patches, and dispatcher
2	02	0200-03E1	Unencoded keyboard routine
3	03	0200-02BC	ASCII encoded keyboard routine
4	04	49DA-55FB	Demonstration program in BASIC
5-8	Same as program 1-4 except in standard speed KIM format		

Required Hardware Configuration

1. Standard KIM-1 (see note 1 below for SYM-1 or AIM-65)
2. Model K-1008 Visible Memory addressed from C000-DFFF
3. Model K-1016 or equivalent 16K memory addressed from 2000-5FFF
4. Parallel keyboard recommended, serial teletype keyboard is acceptable (see note 2 below)

Note 1. To prevent conflicts with on-board ROM in the SYM and AIM, the address of the Visible Memory will have to be changed. Store the new page address of the VM in location 4263.

Note 2. The following locations must be modified to restore the serial keyboard handler that comes with BASIC:

427B	5A
4280	1E
4285	A9
428A	01
428F	~~2C~~ 4C

Loading Instructions

1. Reset the KIM and ready it for cassette input.

2. Load in the Microsoft 9 Digit BASIC cassette supplied by Johnson Computer.

3. Load in file 01 from the K-1008-2 cassette supplied in this package.

4. If an unencoded keyboard is used, load in file 02 from the K-1008-2 cassette.
   If an ASCII encoded keyboard is used, load in file 03.
   If a serial teletype keyboard is used, make the changes listed in note 2 on the previous page.

5. If any changes were made to MTU software, dump the updated MTU program onto another tape to save patching effort the next time BASIC is loaded.

6. Begin execution at location 4261. The display connected to the Visible Memory should clear and the message MEMORY SIZE? should appear.

7. Type a carriage return. The message TERMINAL WIDTH? should appear.

8. type 53 and then a carriage return. The message 5670 BYTES FREE followed by the copyright statement should appear. If more than 16K of continuous memory is installed the number of bytes free will be greater.

9. Type LOAD which causes the KIM to wait for cassette input. Play program 4 on the K-1008-2 cassette. The KIM display should light with 0000 4C following a successful load.

10. Press GO to re-enter BASIC at the warm start location. BASIC should respond by typing READY

11. You may list the entire program by typing LIST 0-9999 and carriage return. To temporarily stop the listing, hold down the Control key and type S. To resume listing hold down Control and type Q. To terminate the listing hold down Control and type C. If a teletype keyboard is being used, any key will terminate the listing.

12. Run the demonstration program by typing RUN followed by a carriage return. The program will run for approximately 1.5 hours with a long pause between each demonstration so that the screen can be examined. Most of the time is spent in the prime number mosaic demonstration. An infinite loop has been programmed following the prime number mosiac so Control/C will be necessary to interrupt the program and return to BASIC.

13. The demo program may now be modified as desired or the user can write his own graphics programs according to the following instructions.

14. Note that the cold start location (4261) can be used at any time to completely re-initialize BASIC. The patches made for a different VM address or a serial keyboard will be retained however.

15. The trigonometric routines are always retained when using the K-1008-2 BASIC Patches.

Use of the K-1008-2 Plotting Routines

The graphics routines supplied with the K-1008-2 package are capable of rapid clearing of the screen, plotting and erasing points, plotting and erasing vectors, and readback of points. For plotting purposes, the Visible Memory screen consists of an array of dots 200 dots high by 320 dots wide. Each dot is called a pixel and represents one bit in the Visible Memory. If the bit is a one, the pixel shows as a bright dot; if a zero, the pixel is black. A graphics image is formed by selectively turning pixels on and off in the desired pattern. Although the POKE function of BASIC could be used to create images directly according to the programming instructions given in the Visible Memory manual, plotting would be extremely slow. The machine language graphics routines in the K-1008-2 package perform the plotting functions hundreds of times faster and are more convenient to use.

An X-Y coordinate system is used to identify points on the VM screen. X and Y must always be zero or positive which means that the entire screen appears in the first quadrant. The allowable range for X is 0 through 319 and the allowable range for Y is 0 through 199. If coordinates outside the allowable range are used, the graphics routines will convert them to values in the allowable range by repeated subtraction of 320 (X) or 200 (Y). To plot, erase, or read a point, only a single X,Y pair is needed. To plot or erase a line, two X,Y pairs are needed, one for each endpoint. The following BASIC statement is required in every graphics BASIC program to identify the coordinates to the machine language plotting routines:

```
1 X1%=0: Y1%=0: X2%=0: Y2%=0
```

The statement number 1 insures that this statement is executed first whenever a RUN[1] command is given to BASIC. This causes the integer variables X1%, Y1%, X2%, and Y2% to be placed first in the variable table where the machine language plotting routines can easily find them.

The USR function of BASIC is used to actually call the plotting routines into action. The argument used with the USR function determines which plotting function is performed. These are listed below:

```
USR(0)  Clear the screen
USR(1)  Plot a white point at X1%,Y1%
USR(2)  Plot a white line from X1%,Y1% to X2%,Y2%
USR(3)  Erase the point at X1%,Y1%
USR(4)  Erase the line running from X1%,Y1% to X2%,Y2%
USR(5)  Returns the color of the point at X1%,Y1%  black=0, white=1
```

Note that USR(x) is a function subprogram, not a statement. A convenient method of using it to plot is to code the BASIC statement: Z=USR(x) where x is the argument corresponding to the desired plot function and Z is a dummy variable. The line plot and erase routines copy X2% into X1% and Y2% into Y1% when they execute. This allows a chain of end-to-end lines to be plotted or erased by simply changing X2% and Y2% for each successive endpoint after the first.

1. Use of RUN (statement number) will not work correctly because the coordinate definition statement will not be executed. Instead the statement: 2 GOTO (statement number) should be entered and the plain RUN command used.

The following program segments are examples of how the graphics routines are used to perform fundamental plotting operations (be sure to define the coordinates as outlined previously):

1. To clear the screen before plotting:

```
10 Z=USR(0)
```

2. To plot a point at X=160 Y=100 (the center of the screen)

```
10 X1%=160
20 Y1%=100
30 Z=USR(1)
```

3. To plot a line from X=20 Y=30 to X=113 Y=165

```
10 X1%=20
20 Y1%=30
30 X2%=113
40 Y2%=165
50 Z=USR(2)
```

After statement 50 is executed, X1%=X2%=113 and Y1%=Y2%=165.

4. To erase the point at X=180 Y=32

```
10 X1%=180
20 Y1%=32
30 Z=USR(3)
```

5. To erase a line running from X=78 Y=73 to X=13 Y=19

```
10 X1%=78
20 Y1%=73
30 X2%=13
40 Y2%=19
50 Z=USR(4)
```

After statement 50 is executed, X1%=X2%=13 and Y1%=Y2%=19.

6. To read the color of the pixel at X=100 Y=50 into the variable A

```
10 X1%=100
20 Y1%=50
30 A=USR(5)
```

The demonstration program should be consulted for other examples of plotting.

Use of the K-1008-2 Text Display Routines

The text display capability built into the K-1008-2 package can be used to annotate the graphic images created by the plotting routines. Normal PRINT statements are used to create the text so the secret to successful use is positioning the text in the desired locations on the screen.

The text display routine, SDTXT, keeps two variables of its own which identify the location of the text cursor on the screen. The character number is stored in location E4 (228 decimal) and varies from 0 for the left screen edge to 52 decimal for the right screen edge. The line number is kept in location E5 (229 decimal) and varies from 0 for the top line to 21 decimal for the bottom line. BASIC also has its own character number which is stored in location 16 (22 decimal) and ranges from 0 to 52 for a terminal width of 53. Normally BASIC's character number and SDTXT's character number agree. Every carriage-return/line-feed issued by BASIC sets both character numbers to zero and increments SDTXT's line number. When the line number tries to go beyond 21 the screen contents are moved upward by 9 raster lines instead.

Putting text at arbitrary locations on the screen basically amounts to POKEing the desired character and line numbers into memory at 228 and 229 respectively. The text is then generated with print statements. The coordinates of the center of a character at character position C and line number L are: X=6*C+2 Y=195-9*L; C=(X-2)/6 L=(195-Y)/9. Characters extend 2 pixels either side of center widthwise and 3 pixels either side of center heightwise. A semicolon terminator should be used after each element printed to prevent BASIC from following it with a carriage return. Also, BASIC's character number at location 22 should be reset to zero before the accumulated output exceeds 53 characters or else BASIC will insert a carriage-return/line-feed anyway. Also be aware that when numbers are printed with the semicolon terminator that a blank is printed following the number and that positive numbers are preceeded by a blank.

There is one additional complication. The cursor displayed by SDTXT is a software cursor and arbitrarily changing the line and character numbers will foul up its proper handling. Therefore before changing the line or character numbers, the cursor should be cleared by executing the statement: Z=USR(6). After the line and character numbers are changed but before any PRINT statements, the cursor should be inserted by executing the statement: Z=USR(7). After all labels and captions are printed, the cursor may be cleared if desired. Return to BASIC's command mode will automatically restore the cursor for normal interactive text output. If possible, text printing should be done before any plotting.

For example, if the caption "Market Index" is desired to start at X=70 Y=180 the following BASIC statements should be coded:

```
10 Z=USR(6)
20 POKE 228,11
30 POKE 229,2
40 POKE 22,0
50 Z=USR(7)
60 PRINT "Market Index";
```

Character number 11 and line number 2 are closest to the desired starting point of X=70 Y=180. Note that lower case letters are available and may be part of a literal field with no problems. The demonstration program can be consulted for additional examples of text output.

Demonstration Program Documentation

The BASIC demonstration program supplied with the K-1008-2 software package is designed to illustrate the use of the plotting and text display functions. It is intended to be easy to read and understand rather than illustrate techniques for program compression and speed enhancement. The program is composed of five different demonstrations that execute in sequence with a long pause between each demonstration. The fifth ends with an infinite loop which must be interrupted to return to BASIC.

The first program illustrates point plotting by drawing a circle with 250 individual dots. The parametric equations: X=COS(A) and Y=SIN(A) are used to generate X,Y pairs as a function of the variable, A. Note that scaling of X and Y, which vary between -1 and +1, is necessary. Although it does not happen in the demonstration, if Y became exactly 1.0 then Y1% would become 200 which is outside the 0-199 range of Y1%.

The second program illustrates vector plotting by creating a very beautiful 31 point star. Since the string of endpoints is connected, the line drawing routine's property of updating X1% and Y1% is utilized to advantage. However the first point is a special case. To handle the first point, a variable called FP is initially set to 1. As each new endpoint is computed, the value of FP is interrogated. If it is found to be non-zero, the endpoints are forced to be equal which effectively moves the "pen" without drawing a line from where it was. After the first point is plotted, FP is set to zero thus allowing vectors to be drawn between all successive points.

The third program illustrates selective erasure of previously plotted lines. Points for the same 31 point star are computed but USR(4) is used to erase the lines computed. Note that when two lines cross and one of them is erased that a small gap is left in the other line. This is a fundamental problem of all stored image (as opposed to refresh vector) graphic displays.

The fourth program illustrates how a fully labelled and captioned graph can be produced. First the Y axis calibration labels are produced with a FOR loop and and print statements. Note that if the FOR loop had been written: FOR Y=-1 TO 1 STEP .2 that after 10 iterations Y would not be precisely 0 because of roundoff error in decimal fraction to binary floating point conversion. Thus rather than 0 being printed, something like -1.16415322E-10 would be printed instead. The captions are printed next. BASIC's character number is reset to zero once to prevent a spurious carriage-return/line-feed. Then the axes themselves are plotted with calibration marks for the Y axis. Finally the Fourier synthesis of the sound waveform of a particular organ pipe is plotted.

The last program demonstrates the ability to read data back from the Visible Memory. It also shows that visualization of number sequences can lead to new insights about the sequences. In the demonstration the sieve of Eratosthenes is used to plot the prime numbers from 3 to 132,001. Each pixel represents an odd integer starting with 3 in the lower left corner of the screen. The sieve method starts with all pixels set to one. Then all of the odd multiples of 3 up to 132,001 are computed and the corresponding pixels are reset to zero. Then a search for the next pixel beyond 3 which is still a one is performed and all of its odd multiples are set to zero and so on. This continues until 363, which is approximately equal to the square root of 132001 is tried. At this point, pixels remaining on the screen correspond to prime numbers. Do the prime numbers appear to be randomly placed? Is there a decrease in prime number density as the numbers get larger? Approximately what percentage of the odd integers are prime? The answers to these questions are immediately apparent when viewing the screen and may be supprising.

Notes on the Text and Graphics Routines

The heart of the K-1008-2 BASIC Patches software package is the VMBAS program. This program is file 01 on the cassette and is loaded immediately following BASIC into locations 4261 through 49D7. After loading, the cold start location (INIT) 4261 is executed. The main job of the cold start routine is to automatically patch certain locations in BASIC. These patches alter the operation of BASIC as follows:

1. USRLOC is altered to point to the graphics dispatch routine.
2. The call to KIM's teletype input routine is altered to point to an internal input routine (ANKBX) which calls a parallel keyboard routine in location 0200 and echos the input text to SDTXT.
3. The call for testing for control/C is altered to point to an internal routine (CNTLCX) which in turn calls an improved control/C test routine in location 0203.
4. Patches BASIC so that program storage starts at location 49D8 instead of 4261.
5. Patches BASIC so that the question about keeping the trigonometric routines is bypassed.
6. Clears the screen, inserts a cursor at character 0 line 0 and enters BASIC's initialization routine.

The graphics dispatch routine (DISPCH) is entered whenever the USR function is used in a statement. Its job is to look at the value of the argument and jump to the corresponding graphics routine. If the argument is out of range, an immediate return is taken. However the contents of 4316 and 4317 may be changed to jump to another machine language program instead if the argument is outside the range of 0 through 7.

Before dispatching to a graphics routine, the first 4 variables in BASIC's variable table are transferred to page zero locations for easy access by the graphics routines. After the transfer they are range checked and corrected if necessary by successive subtraction of the maximum value+1 if. After returning from a graphics routine, these page zero locations are copied back into BASIC's variable table. The four variables are assumed to be integer variables and are assumed to be stored in the following order: X1, Y1, X2, Y2. All of the routines return a value for the USR function but only argument 5 (read pixel) returns a predictable value.

USR function arguments 6 and 7 merely link to CSRCLR and CSRSET respectively in SDTXT. The character and line numbers utilized by SDTXT are checked for validity and corrected if necessary every time SDTXT is called.

Notes on the Keyboard Routines

Because of severe limitations with teletype input to KIM BASIC, the K-1008-2 BASIC Patches Package includes two parallel keyboard input subroutines. Besides, who wants to use a noisy teletype when the Visible Memory is doing all of BASIC's printing? Teletype input may still be selected however by putting the KIM in teletype mode which causes both keyboard routines to call the TTY input routine in the KIM monitor. Both keyboard routines use a transfer vector. Location 0200 contains a jump to the actual keyboard input subroutine and location 0203 contains a jump to the control/C test routine.

The keyboard routine in file 02 in conjunction with an unencoded keyboard is the least cost approach to adding a parallel keyboard to the KIM-1. In addition, port A is left completely free for other uses such as operating the MTU K-1002 8-Bit Audio System. An article reprint describing the theory and construction details of the keyboard is included. For best results with the file 02 keyboard routine the following additions to the keyboard matrix described in the article should be made:

1. The Shift Lock key should be connected between row 3 and column 3. This key should be unlocked while using the KIM monitor to avoid possible interference with the display.
2. Germanium diodes (type 1N270 is best) should be placed in series with the shift key, shift lock key, repeat key, and control key to eliminate a possible "phantom key" effect. The cathode (end with the band) should connect to the column lines.

For maximum usefulness with BASIC (and all other keyboard applications as well) the shift lock functions as an "upper case only" (caps lock) mode key. When active, all letters will be forced to upper case but the numbers and special characters will be unaffected. This is important because a bug in BASIC prevents recognition of statements and commands entered in lower case. In fact, a quite reasonable word processing system can be set up using the strings facility of BASIC and the lower case capability of the keyboard and Visible Memory display.

The test for control/C routine performs several functions. BASIC calls this routine periodically while printing and while running programs. When entered, it first tests for the control and C keys being pressed simultaneously. If that combination is seen the carry flag is set and an immediate return is taken. This causes BASIC to stop what it was doing and print a BREAK message. If control/C is not seen then control/O is tested. If this condition is true, BASIC's "suppress output" flag at location 0014 is toggled. The effect is to "flush" all output until the flag is turned back off by another control/O. Extra code is required to insure that the flag is toggled only once each time control/O is pressed. If control S is seen a loop is entered which waits until control/Q is seen. The effect is to suspend execution of the BASIC program until control/Q is pressed. If none of these special control functions are seen, an immediate return is taken with the carry flag off.

File 03 contains a similar but much shorter keyboard routine for parallel ASCII encoded keyboards. The keyboard should be connected to port A with the 7 ASCII data bits connected to bits 0 through 6. The key pressed or strobe signal should be connected to bit 7. The data is assumed to be in true form and the strobe is assumed to be active when it is a logic one although either or both polarities may be altered by changing the mask byte in location 02BA. All functions are similar to those of the unencoded keyboard with the exception of the caps lock feature. CNTL/R is used to turn caps lock on and CNTL/T is used to turn it off. Note that proper operation of the control/C routine with a pulse strobe keyboard requires a register to hold the keycode between keystrokes. This is a standard feature of keyboards using an LSI encoder chip. Also note that the strobe pulse must be at least 12 microseconds long to be seen reliably.

```
1 X1%=0: Y1%=0: X2%=0: Y2%=0:
2 REM PREVIOUS STATEMENT REQUIRED TO DEFINE
3 REM GRAPHIC COORDINATES
10 REM CLEAR THE SCREEN
11 Z=USR(0)
100 REM DEMONSTRATION OF POINT PLOT
110 REM PLOT A CIRCLE IN DEAD CENTER OF SCREEN USING
120 REM 250 POINTS
130 FOR I=0 TO 250
140 A=6.28318*I/250
150 X1%=100*COS(A)+160
160 Y1%=100*SIN(A)+100
170 Z=USR(1)
180 NEXT I
190 GOSUB 9000
200 REM DEMONSTRATION OF VECTOR PLOT
210 Z=USR(0): REM CLEAR SCREEN
220 FP=1: REM SET FIRST POINT FLAG
230 FOR I=0 TO 31
240 A=13*I*6.2831828/31
250 X2%=150*COS(A)+160
260 Y2%=100*SIN(A)+100
270 IF FP<>1 THEN GOTO 290
280 X1%=X2%: Y1%=Y2%: FP=0
290 Z=USR(2)
300 NEXT I
310 GOSUB 9000
400 REM DEMONSTRATION OF VECTOR ERASE
410 FP=1
420 FOR I=0 TO 31
430 A=13*I*6.2831828/31
440 X2%=150*COS(A)+160
450 Y2%=100*SIN(A)+100
460 IF FP<>1 THEN GOTO 480
470 X1%=X2%: Y1%=Y2%: FP=0
480 Z=USR(4)
490 NEXT I
500 GOSUB 9000
600 REM DEMONSTRATION OF AXIS PLOT, LABEL, AND TITLE
610 Z=USR(0)
620 REM INSERT Y AXIS LABELLING FIRST
630 FOR Y=-10 TO 10 STEP 2
640 REM REPOSITION TEXT CURSOR
650 Z=USR(6)
660 POKE 228,0: POKE 229,(-Y+10)
670 Z=USR(7)
680 PRINT Y/10;: REM PRINT Y AXIS LABEL
690 NEXT Y
700 REM PRINT X AXIS CAPTION
710 Z=USR(6): POKE 228,49: POKE 229,10: Z=USR(7)
720 PRINT "Time";
730 REM PRINT X AXIS CAPTION AND FIGURE CAPTION
740 Z=USR(6): POKE 228,0: POKE 229,21: Z=USR(7)
741 POKE 22,0: REM RESET BASIC'S CHAR POINTER TO 0
750 PRINT "Amplitude";
760 PRINT "     Waveform of Great Diapason C4 16FT";
770 Z=USR(6)
800 REM PLOT X AND Y AXES
810 X1%=20: X2%=294: Y1%=105: Y2%=105: REM HOR AXIS
820 Z=USR(2)
830 X1%=20: X2%=20: Y1%=11: Y2%=199: REM VERT AXIS
840 Z=USR(2)
```

```
900 REM PLOT TIC MARKS ON Y AXIS
910 FOR Y=-1 TO 1 STEP .2
920 X1%=18: X2%=20
930 Y1%=15+90*(Y+1): Y2%=Y1%
940 Z=USR(2)
950 NEXT Y
1000 REM PLOT THE WAVEFORM USING VECTORS
1010 FP=1
1020 XF=270/(4*3.14159): REM X SCALE FACTOR
1030 YF=60: REM Y SCALE FACTOR
1040 FOR X=0 TO 4*3.14159 STEP 4*3.14159/270
1050 Y=SIN(X)+.49*SIN(2*X+3.9)+.3*SIN(3*X+5.81)
1060 Y=Y+.24*SIN(4*X+3.8)+.18*SIN(5*X+.97)
1070 Y=Y+.12*SIN(6*X+4.3)+.04*SIN(7*X+3.54)
1080 Y=Y+.07*SIN(8*X+.87)+.03*SIN(9*X+5.3)
1090 X2%=20+XF*X: Y2%=105+YF*Y
1100 IF FP<>1 THEN GOTO 1120
1110 X1%=X2%: Y1%=Y2%: FP=0
1120 Z=USR(2)
1130 NEXT X
1140 GOSUB 9000
2000 REM SIEVE OF ERATOSTHENES DEMONSTRATION
2001 REM THIS PROGRAM FINDS ALL OF THE PRIME NUMBERS
2002 REM FROM 3 TO 128001 USING THE VISIBLE MEMORY.
2003 REM EACH PIXEL ON THE SCREEN REPRESENTS AN ODD
2004 REM INTEGER STARTING WITH 3 IN THE LOWER LEFT
2005 REM CORNER.  THE PROGRAM FIRST TURNS ALL PIXELS
2006 REM ON AND THEN TURNS THOSE OFF THAT DO NOT
2007 REM REPRESENT PRIME NUMBERS.  THOSE THAT ARE
2008 REM LEFT ON AFTER EXECUTION ARE PRIME.  IS THE
2009 REM RESULTING PATTERN RANDOM?  ARE THE PRIME
2010 REM NUMBERS UNIFORMLY DISTRIBUTED?  THE ABILITY
2011 REM TO READ BACK FROM THE VISIBLE MEMORY IS
2012 REM USED IN THIS PROGRAM.
2100 Z=USR(0)
2110 REM QUICKLY TURN ALL PIXELS ON
2120 FOR I=0 TO 199
2130 X1%=0: X2%=319: Y1%=I: Y2%=I: Z=USR(2)
2140 NEXT I
2200 FOR I=3 TO SQR(128001) STEP 2
2210 N=I
2220 GOSUB 8000
2230 IF USR(5)=0 THEN GOTO 2300
2240 FOR J=3 TO 128001 STEP 2
2250 N=I*J
2260 IF N>128001 THEN GOTO 2300
2270 GOSUB 8000
2280 Z=USR(3)
2290 NEXT J
2300 NEXT I
2310 GOTO 2310: REM WAIT FOREVER
8000 REM FUNCTION TO CONVERT ODD INTEGER TO X,Y
8010 N1=(N-3)/2
8020 N2=N1/320
8030 X1%=N1-INT(N2)*320
8040 Y1%=N2
8050 RETURN
9000 REM DELAY ROUTINE TO HOLD IMAGE IN SCREEN
9010 FOR D=1 TO 10000
9020 NEXT D
9030 RETURN
9999 END
```

```
                .PAGE 'DOCUMENTATION'
;               *****MODIFIED FOR KIM BASIC*****
;               THIS PACKAGE ALLOWS THE VISIBLE MEMORY TO BE USED WITH MICRO-
;               SOFT BASIC AS TERMINAL DISPLAY DEVICE AND A GRAPHICS DISPLAY
;               DEVICE.  A SLIGHTLY MODIFIED VERSION OF SDTXT IS USED FOR TEXT
;               DISPLAY AND AN ABBREVIATED VERSION OF THE GRAPHICS PACKAGE IS
;               USED FOR GRAPHICS.

;               INTERFACE WITH BASIC IS AT TWO LEVELS.  THE CALL TO THE KIM
;               TTY PRINT ROUTINE IS REPLACED BY A CALL TO SDTXT WHICH MEANS
;               THAT ALL PRINTED OUTPUT FROM BASIC GOES TO THE SCREEN.  THE
;               KEYBOARD ROUTINE SUPPLIED BY THE USER SHOULD ALSO CALL SDTXT
;               SO THAT TYPED INPUT APPEARS ON THE SCREEN AS WELL.  INTERFACE
;               TO THE GRAPHICS ROUTINES IS THROUGH THE USR FUNCTION AND THE
;               VARIABLE STORAGE AREA IN BASIC.  THE ARGUMENT OF THE USR
;               FUNCTION CALL SELECTS WHICH GRAPHICS ROUTINE IS TO BE USED.
;               THE COORDINATE DATA USED BY THE GRAPHICS FUNCTIONS IS ASSUMED
;               TO BE IN THE FIRST 4 ENTRIES OF THE VARIABLE TABLE AND IS
;               ASSUMED TO BE INTEGER DATA.  TO ESTABLISH THE COORDINATE NAMES
;               AND INSURE THAT THEY ARE STORED FIRST IN THE VARIABLE TABLE,
;               THE FOLLOWING BASIC STATEMENT MUST BE CODED AS PART OF THE
;               USER'S PROGRAM:

;                    1 X1%=0; Y1%=0; X2%=0; Y2%=0

;               THE STATEMENT NUMBER 1 INSURES THAT IT WILL BE EXECUTED FIRST
;               AND THAT X1%, Y1%, X2%, AND Y2% WILL APPEAR FIRST IN THE
;               VARIABLE TABLE AND IN THE CORRECT ORDER.  THE % SIGNS AFTER
;               THE VARIABLE NAMES INDICATE THAT THEY ARE INTEGER VARIABLES
;               AND MUST BE USED.  THE ACTUAL NAME MAY BE CHANGED BUT
;               CONFUSION IS MINIMIZED BY USING THE NAMES GIVEN.  THE ORIGIN
;               OF THE COORDINATE SYSTEM IS THE LOWER LEFT CORNER OF THE
;               SCREEN.  THE ALLOWABLE RANGE OF X IS 0 TO 319 INCLUSIVE AND
;               THE Y RANGE IS 0 TO 199 INCLUSIVE.  OUT OF RANGE COORDINATES
;               WILL BE CORRECTED BY COMPUTING THEIR VALUE MODULO THE MAXIMUM
;               VALUE PLUS 1.  THE MODULUS COMPUTATION IS PRIMITIVE AND MAY BE
;               SLOW HOWEVER.  IF THE GRAPHICS ROUTINE MODIFIES ANY OF THE
;               COORDINATES, THEY WILL BE MODIFIED IN BASIC'S VARIABLE TABLE
;               AS WELL.

;               THE USR FUNCTION CODES ARE AS BELOW:

;                    0  CLEAR THE SCREEN AND SET THE TEXT CURSOR AT UPPER LEFT
;                       CORNER OF THE SCREEN
;                    1  POINT PLOT X1,Y1 WHITE DOT  X1, Y1 NOT CHANGED
;                    2  LINE PLOT FROM X1,Y1 TO X2,Y2  WHITE LINE
;                       X2 COPIED INTO X1 AND Y2 COPIED INTO Y1 UPON RETURN
;                    3  POINT PLOT X1,Y1 BLACK DOT  X1, Y1 NOT CHANGED
;                    4  LINE PLOT FROM X1,Y1 TO X2,Y2  BLACK LINE (ERASE)
;                       X2 COPIED INTO X1 AND Y2 COPIED INTO Y1 UPON RETURN
;                    5  READ POINT AT X1,Y1  VALUE OF USR FUNCTION ON RETURN IS
;                       THE STATE OF THE POINT  0=BLACK  1=WHITE
;                    6  CLEAR THE TEXT CURSOR FROM THE SCREEN
;                    7  SET THE TEXT CURSOR ON THE SCREEN
```

```
;        FOR TEXT OUTPUT ANYWHERE ON THE SCREEN THE POKE FUN(
;        BE USED TO DIRECTLY ALTER THE CURSOR POSITION.  THE
;        NUMBER IS KEPT IN LOCATION 228 (10) AND THE LINE NUM
;        IS KEPT IN LOCATION 229.  THE CHARACTER NUMBER RANGE
;        TO 52 INCLUSIVE AND THE LINE NUMBER RANGES FROM 0 TO
;        INCLUSIVE.  THE CHARACTER MATRIX IS 5 BY 7 IN A CHAR
;        CELL OF 6 BY 9.  LINE 0 CHARACTER 0 IS THE UPPER LEF
;        OF THE SCREEN AND COVERS X COORDINATES OF 0 TO 6 AND
;        COORDINATES OF 191-199 INCLUSIVE.
;        OUT OF RANGE LINE OR CHARACTER NUMBERS WILL BE CORRE
;        AS WITH POINT COORDINATES.

;        NOTE THAT THE TEXT CURSOR SHOULD BE CLEARED FROM THE
;        BEFORE MOVING IT WITH POKES AND SHOULD BE SET AFTER
;        STANDARD BASIC PRINT STATEMENTS CAN BE USED FOR PLOT
;        IF SEMICOLONS ARE USED TO SUPPRESS CARRIAGE RETURN/L
;        NOTE THAT IF A CARRIAGE RETURN/LINE FEED IS PRINTED
;        LINE NUMBER IS 21 THAT THE ENTIRE DISPLAY, GRAPHICS
;        WILL BE SCROLLED UPWARD 9 SCAN LINES.
```

```
                        .PAGE   ' EQUATES AND STORAGE'

               ;        GENERAL EQUATES

0140           NX       =       320            ; NUMBER OF BITS IN A ROW
00C8           NY       =       200            ; NUMBER OF ROWS
FA00           NPIX     =       NX*NY          ; NUMBER OF PIXELS
1F40           NLOC     =       8000           ; NUMBER OF VISIBLE LOCATIONS
0009           CHHI     =       9              ; CHARACTER WINDOW HEIGHT
0006           CHWID    =       6              ; CHARACTER WINDOW WIDTH
0035           NCHR     =       320/CHWID      ; NUMBER OF CHARACTERS PER LINE
0016           NLIN     =       NLOC/40/CHHI   ; NUMBER OF TEXT LINES
1D88           NSCRL    =       NLIN-1*CHHI*40 ; NUMBER OF LOCATIONS TO SCROLL
01B8           NCLR     =       NLOC-NSCRL     ; NUMBER OF LOCATIONS TO CLEAR AFTER SCROL
0200           ANKB     =       X'0200         ; LOCATION OF KEYBOARD ROUTINE
0203           CNTLC    =       X'0203         ; LOCATION OF TEST FOR CONTROL/C ROUTINE


               ;        PAGE 0 STORAGE         THIS IS THE ONLY RAM STORAGE USED BY THIS
               ;                               PROGRAM
0000                    .=      X'E3           ; START BASE PAGE STORAGE 3 PAST END OF
                                               ; BASIC AREA

               ;        PERMANENT STORAGE THAT MUST NOT BE WIPED OUT BY EXITING TO
               ;        THE KIM MONITOR

00E3           VMORG:   .=.+    1              ; FIRST PAGE NUMBER OF VISIBLE MEMORY
00E4           CSRX:    .=.+    1              ; TEXT CURSOR CHARACTER NUMBER
00E5           CSRY:    .=.+    1              ; TEXT CURSOR LINE NUMBER

               ;        TEMPORARY STORAGE THAT MAY BE WIPED OUT BY EXITING TO THE KIM
               ;        MONITOR

00E6           X1CORD:  .=.+    2              ; COPY OF BASIC'S X1 COORDINATE
00E8           Y1CORD:  .=.+    2              ; COPY OF BASIC'S Y1 COORDINATE
00EA           X2CORD:  .=.+    2              ; COPY OF BASIC'S X2 COORDINATE
00EC           Y2CORD:  .=.+    2              ; COPY OF BASIC'S X2 COORDINATE
00EE           TEMP:    .=.+    2              ; TEMPORARY STORAGE
00F0           BTPT:    .=.+    1              ; BIT POINTER WITHIN BYTE
00F1                    .=.+    1              ; DO NOT USE KIM'S STATUS SAVE BYTE!!!
00F2           ADP1:    .=.+    2              ; ADDRESS POINTER 1
00F4           ADP2:    .=.+    2              ; ADDRESS POINTER 2
00F6           DELTAX:  .=.+    2              ; DELTA X FOR LINE DRAW
00F8           DELTAY:  .=.+    2              ; DELTA Y FOR LINE DRAW
00FA           ACC:     .=.+    2              ; ACCUMULATOR FOR LINE DRAW
00FC           XDIR:    .=.+    1              ; X MOVEMENT DIRECTION, ZERO=+
00FD           YDIR:    .=.+    1              ; Y MOVEMENT DIRECTION, ZERO=+
00FE           XCHFLG:  .=.+    1              ; EXCHANGE X AND Y FLAG, EXCHANGE IF NOT 0
00FF           COLOR:   .=.+    1              ; COLOR OF LINE DRAWN -1=WHITE
00EE           DCNT1    =       TEMP           ; DOUBLE PRECISION COUNTER
00FE           MRGT1    =       XCHFLG         ; TEMPORARY STORAGE FOR MERGE
```

```
                               .PAGE   ' INITIALIZATION ROUTINE'

     0100                      .=      X'4261        ; START IMMEDIATELY BEYOND BASIC

     4261 D8         INIT:     CLD                   ; CLEAR DECIMAL MODE
                                                     ; DON'T CARE WHERE THE STACK IS RIGHT
     4262 A9C0                 LDA     #X'C0         ; INITIALIZE THE LOCATION OF THE VISII
     4264 85E3                 STA     VMORG         ; MEMORY
     4266 A9CB                 LDA     #DISPCH&X'FF  ; SET USRLOC TO GO TO GRAPHICS DISPAT(
     4268 8D4020               STA     X'2040        ; ROUTINE
     426B A942                 LDA     #DISPCH/256
     426D 8D4120               STA     X'2041
     4270 A937                 LDA     #SDTXT&X'FF   ; SET BASIC PRINT CALL TO GO TO SDTXT
     4272 8D522A               STA     X'2A52
     4275 A945                 LDA     #SDTXT/256
     4277 8D532A               STA     X'2A53
     427A A9B7                 LDA     #ANKBX&X'FF   ; SET BASIC KEYBOARD CALL TO GO TO ANK
     427C 8D5724               STA     X'2457
     427F A942                 LDA     #ANKBX/256
     4281 8D5824               STA     X'2458
     4284 A94C                 LDA     #X'4C         ; SET BASIC TEST CONTROL/C CALL TO GO
     4286 8DDA26               STA     X'26DA        ; CNTLCX
     4289 A9BE                 LDA     #CNTLCX&X'FF
     428B 8DDB26               STA     X'26DB
     428E A942                 LDA     #CNTLCX/256
     4290 8DDC26               STA     X'26DC
     4293 A9D9                 LDA     #END+1&X'FF   ; ADJUST BEGINNING OF BASIC PROGRAM A
     4295 8DCE40               STA     X'40CE        ; TO SKIP OVER GRAPHICS PACKAGE
     4298 A949                 LDA     #END+1/256
     429A 8DD040               STA     X'40D0
     429D A207                 LDX     #7            ; MOVE 7 BYTES INTO BASIC WHICH CAUSES
     429F BDB042     INIT1:    LDA     INTCOD,X      ; QUESTION REGARDING TRIG FUNCTIONS TO BE
     42A2 9D3641               STA     X'4136,X      ; SKIPPED AND PRESERVES BASIC'S INITI
     42A5 CA                   DEX                   ; ROUTINE
     42A6 10F7                 BPL     INIT1
     42A8 A90C                 LDA     #X'0C         ; CLEAR THE SCREEN AND PUT THE CURSOR
     42AA 203745               JSR     SDTXT         ; CHARACTER 0 LINE 0
     42AD 4C6540               JMP     X'4065        ; ENTER BASIC


     42B0 A2D9       INTCOD:   LDX     #END+1&X'FF   ; INITIALIZATION CODE TO POKE INTO BASIC
     42B2 A049                 LDY     #END+1/256    ; INITIALIZATION ROUTINE
     42B4 4C8341               JMP     X'4183
```

```
                                .PAGE  ' INPUT ROUTINE'
                        ;       KEYBOARD ROUTINE - CALL USER'S KEYBOARD ROUTINE, ECHO BACK THE
                        ;       CHARACTER TYPED ON THE SCREEN, AND RETURN.

     42B7 200002        ANKBX:  JSR     ANKB            ; GO TO KEYBOARD ROUTINE
     42BA 203745                JSR     SDTXT           ; ECHO THE CHARACTER ON THE SCREEN
     42BD 60                    RTS                     ; RETURN

                        ;       CONTROL/C TEST ROUTINE - CALL USER'S CONTROL/C TEST ROUTINE
                        ;       USER'S ROUTINE SHOULD RETURN WITH CARRY SET IF CONTROL/C IS
                        ;       SEEN AND RETURN WITH CARRY CLEAR IF NOT SEEN.  USER'S ROUTINE
                        ;       MAY ALSO IMPLEMENT THE CONTROL/O FUNCTION AND XOFF-XON (CNTL/S
                        ;       AND CNTL/Q)

     42BE 200302        CNTLCX: JSR     CNTLC           ; GO TO CONTROL C ROUTINE
     42C1 B005                  BCS     CTCYES          ; JUMP IF CONTROL/C SEEN
     42C3 A901          CTCNO:  LDA     #1              ; "NO" RETURN, LOAD 1 INTO A
     42C5 C902                  CMP     #2              ; SET THE NEGATIVE INDICATOR
     42C7 60                    RTS                     ; RETURN
     42C8 4CE126        CTCYES: JMP     X'26E1          ; GO TO YES RETURN IN BASIC
```

```
                                 .PAGE  ' DISPATCH ROUTINE FROM A USR CALL'
                      ;          DISPATCH ROUTINE FROM A USR CALL
                      ;          THIS ROUTINE LOOKS AT THE ARGUMENT OF THE USR FUNCTION CALL
                      ;          AND DISPATCHES TO THE PROPER GRAPHICS ROUTINE.
                      ;          IT ALSO COPIES THE COORDINATES FROM THE VARIABLE AREA IN BASIC
                      ;          TO PAGE 0 LOCATIONS BEFORE EXECUTING A GRAPHICS ROUTINE AND
                      ;          COPIES THEM BACK AFTER EXECUTING A GRAPHICS ROUTINE

42CB A002             DISPCH:    LDY    #2             ; SET UP TO MOVE 4 COORDINATES TO PAGE 0
42CD A200                        LDX    #0
42CF B17A             DISPC1:    LDA    (X'007A),Y     ; GET HIGH BYTE OF AN INTEGER VARIABLE
42D1 95E7                        STA    X1CORD+1,X     ; STORE IT IN PAGE 0
42D3 C8                          INY
42D4 B17A                        LDA    (X'007A),Y     ; GET LOW BYTE OF THE VARIABLE
42D6 95E6                        STA    X1CORD,X       ; STORE IT IN PAGE 0
42D8 98                          TYA                   ; ADD 6 TO Y TO POINT TO NEXT VARIABLE
42D9 18                          CLC                   ; IN BASIC'S VARIABLE TABLE
42DA 6906                        ADC    #6
42DC A8                          TAY
42DD E8                          INX                   ; ADD 2 TO X TO POINT TO NEXT VARIABLE IN
42DE E8                          INX                   ; PAGE 0
42DF E008                        CPX    #8             ; TEST IF MOVE IS COMPLETE
42E1 D0EC                        BNE    DISPC1         ; CONTINUE IF NOT
42E3 20EB43                      JSR    CKCRD          ; TEST IF COORDINATES ARE IN RANGE AND
                                                       ; CORRECT IF NECESSARY

                      ;          GET ARGUMENT OF USR CALL, CHECK FOR ALLOWABLE RANGE, AND
                      ;          DISPATCH TO CORRECT GRAPHICS ROUTINE

42E6 201843                      JSR    GETARG         ; GET LOW ARGUMENT IN A AND HIGH ARGUMENT
                                                       ; IN Y
42E9 A5B1                        LDA    X'B1           ; TEST FOR LEGAL ARGUMENT
                                                       ; UPPER BYTE MUST BE ZERO
42EB D028                        BNE    ILLEGL         ; GO RETURN IF ILLEGAL ARGUMENT
42ED A5B2                        LDA    X'B2           ; TEST FOR RANGE OF 0 TO 7 INCLUSIVE IN
42EF C907                        CMP    #7             ; LOWER BYTE AND
42F1 B022                        BCS    ILLEGL         ; GO RETURN IF NOT IN RANGE
42F3 201E43                      JSR    VCTJSR         ; DO A VECTOR JSR TO THE CORRESPONDING
                                                       ; GRAPHICS ROUTINE
42F6 A8                          TAY                   ; RETURN FUNCTION VALUE TO BASIC
42F7 A900                        LDA    #0
42F9 201B43                      JSR    PUTARG

                      ;          MOVE THE COORDINATES BACK TO BASIC

42FC A002                        LDY    #2             ; SET UP TO MOVE 4 COORDINATES BACK
42FE A200                        LDX    #0
4300 B5E7             DISPC2:    LDA    X1CORD+1,X     ; GET HIGH BYTE OF AN INTEGER VARIABLE
4302 917A                        STA    (X'007A),Y     ; STORE IT BACK IN BASIC
4304 C8                          INY
4305 B5E6                        LDA    X1CORD,X       ; GET LOW BYTE OF THE VARIABLE
4307 917A                        STA    (X'007A),Y     ; STORE IT BACK
4309 98                          TYA                   ; ADD 6 TO Y TO POINT TO NEXT VARIABLE
430A 18                          CLC                   ; IN BASIC'S VARIABLE TABLE
430B 6906                        ADC    #6
```

```
430D A8                   TAY
430E E8                   INX                  ; ADD 2 TO X TO POINT TO NEXT VARIABLE IN
430F E8                   INX                  ; PAGE 0
4310 E008                 CPX     #8           ; TEST IF MOVE IS COMPLETE
4312 D0EC                 BNE     DISPC2       ; CONTINUE IF NOT
4314 60         DISPC3:   RTS                  ; RETURN TO BASIC
4315 4C1443     ILLEGL:   JMP     DISPC3       ; IMMEDIATE RETURN ON ILLEGAL ARGUMENT
                                               ; CAN BE CHANGED TO GO TO ANOTHER USR
                                               ; ROUTINE
4318 6C0600     GETARG:   JMP     (X'0006)     ; GO TO GET ARGUMENT FUNCTION IN BASIC
431B 6C0800     PUTARG:   JMP     (X'0008)     ; GO TO PUT ARGUMENT FUNCTION IN BASIC
431E 0A         VCTJSR:   ASLA                 ; DOUBLE THE ARGUMENT VALUE
431F AA                   TAX                  ; USE AS INDEX INTO DISPATCH TABLE
4320 BD2A43               LDA     DSPTAB+1,X   ; TRANSFER TABLE ENTRY TO THE STACK
4323 48                   PHA
4324 BD2943               LDA     DSPTAB,X
4327 48                   PHA
4328 60                   RTS                  ; JUMP TO THE ADDRESS ON THE TOP OF THE
                                               ; STACK
                                               ; RETURNS TO THE CALLER OF THIS ROUTINE
4329 3843       DSPTAB:   .WORD   CLEAR-1      ; 0  ADDRESS OF CLEAR SCREEN ROUTINE
432B A543                 .WORD   STPIX-1      ; 1  ADDRESS OF SET PIXEL ROUTINE
432D 2D44                 .WORD   DRAW-1       ; 2  ADDRESS OF DRAW LINE ROUTINE
432F B443                 .WORD   CLPIX-1      ; 3  ADDRESS OF CLEAR PIXEL ROUTINE
4331 2944                 .WORD   ERASE-1      ; 4  ADDRESS OF ERASE LINE ROUTINE
4333 C343                 .WORD   RDPIX-1      ; 5  ADDRESS OF READ PIXEL ROUTINE
4335 2746                 .WORD   CSRCLR-1     ; 6  ADDRESS OF CLEAR CURSOR ROUTINE
4337 1C46                 .WORD   CSRSET-1     ; 7  ADDRESS OF INSERT CURSOR ROUTINE
```

```
                        .PAGE 'DOCUMENTATION OF ABBREVIATED GRAPHICS PACKAGE'

                ;       THIS PACKAGE PROVIDES FUNDAMENTAL GRAPHICS ORIENTED
                ;       SUBROUTINES NEEDED FOR EFFECTIVE USE OF THE VISIBLE MEMORY AS
                ;       A GRAPHIC DISPLAY DEVICE WITH MICROSOFT BASIC.  THE ROUTINES
                ;       INCLUDED ARE AS FOLLOWS:
                ;
                ;          CLEAR - CLEARS THE ENTIRE VISIBLE MEMORY AS DEFINED BY
                ;                  NPIX/8
                ;          PIXADR- RETURNS BYTE AND BIT ADDRESS OF PIXEL AT X1CORD,
                ;                  Y1CORD
                ;          CKCRD - PERFORM A RANGE CHECK ON ALL COORDINATES
                ;          STPIX - SET PIXEL AT X1CORD,Y1CORD TO A ONE (WHITE DOT)
                ;          CLPIX - CLEAR PIXEL AT X1CORD,Y1CORD TO ZERO (BLACK DOT)
                ;          RDPIX - COPY THE STATE OF THE PIXEL AT X1CORD,Y1CORD INTO
                ;                  THE ACCUMULATOR
                ;          DRAW -  DRAW THE BEST STRAIGHT LINE FROM X1CORD,Y1CORD
                ;                  TO X2CORD,Y2CORD.  X2CORD,Y2CORD COPIED TO
                ;                  X1CORD,Y1CORD AFTER DRAWING
                ;          ERASE - SAME AS DRAW EXCEPT A BLACK LINE IS DRAWN
                ;
                ;       ALL SUBROUTINES DEPEND ON ONE OR TWO PAIRS OF COORDINATES.
                ;       EACH COORDINATE IS A DOUBLE PRECISION, UNSIGNED NUMBER WITH
                ;       THE LOW BYTE FIRST (I.E.  LIKE MEMORY ADDRESSES IN THE 6502)
                ;       THE ORIGIN OF THE COORDINATE SYSTEM IS AT THE LOWER LEFT
                ;       CORNER OF THE SCREEN THEREFORE THE ENITRE SCREEN IS IN THE
                ;       FIRST QUADRANT.  ALLOWABLE RANGE OF THE X COORDINATE IS 0 TO
                ;       319 (DECIMAL) AND THE RANGE OF THE Y COORDINATE IS 0 TO 199.

```



```
                        .PAGE  'CLEAR ENTIRE SCREEN ROUTINE'
                ;       CLEAR ENTIRE SCREEN ROUTTNE
                ;       USES BOTH INDICES AND ADP1

4339 A000       CLEAR:  LDY     #0              ; INITIALIZE ADDRESS POINTER
433B 84F4               STY     ADP2            ; AND ZERO INDEX Y
433D A5E3               LDA     VMORG
433F 85F5               STA     ADP2+1
4341 A91F               LDA     #NPIX/8/256     ; SET COUNT OF BYTES TO CLEAR
4343 85EF               STA     DCNT1+1
4345 A940               LDA     #NPIX/8&X'FF
4347 85EE               STA     DCNT1
4349 4C1A47             JMP     FCLR            ; GO DO CLEAR AND RETURN

```

```
                             .PAGE  'PIXADR - BYTE AND BIT ADDRESS OF A PIXEL'
                   ;         PIXADR - FIND THE BYTE ADDRESS AND BIT NUMBER OF PIXEL AT
                   ;                  X1CORD,Y1CORD
                   ;         PUTS BYTE ADDRESS IN ADP1 AND BIT NUMBER (BIT 0 IS LEFTMOST)
                   ;         IN BTPT.
                   ;         DOES NOT CHECK MAGNITUDE OF COORDINATES FOR MAXIMUM SPEED
                   ;         PRESERVES X AND Y REGISTERS, DESTROYS A
                   ;         BYTE ADDRESS = VMORG*256+(199-Y1CORD)*40+INT(XCORD/8)
                   ;         BIT ADDRESS = REM(XCORD/8)
                   ;         OPTIMIZED FOR SPEED THEREFORE CALLS TO A DOUBLE SHIFT ROUTINE
                   ;         ARE NOT DONE

434C A5E6      PIXADR:  LDA      X1CORD        ; COMPUTE BIT ADDRESS FIRST
434E 85F2               STA      ADP1          ; ALSO TRANSFER X1CORD TO ADP1
4350 2907               AND      #X'07         ; WHICH IS SIMPLY THE LOW 3 BITS OF X
4352 85F0               STA      BTPT
4354 A5E7               LDA      X1CORD+1      ; FINISH TRANSFERRING X1CORD TO ADP1
4356 85F3               STA      ADP1+1
4358 46F3               LSR      ADP1+1        ; DOUBLE SHIFT ADP1 RIGHT 3 TO GET
435A 66F2               ROR      ADP1          ; INT(XCORD/8)
435C 46F3               LSR      ADP1+1
435E 66F2               ROR      ADP1
4360 46F3               LSR      ADP1+1
4362 66F2               ROR      ADP1
4364 A9C7               LDA      #199          ; TRANSFER (199-Y1CORD) TO ADP2
4366 38                 SEC                    ; AND TEMPORARY STORAGE
4367 E5E8               SBC      Y1CORD
4369 85F4               STA      ADP2
436B 85EE               STA      TEMP
436D A900               LDA      #0
436F E5E9               SBC      Y1CORD+1
4371 85F5               STA      ADP2+1
4373 85EF               STA      TEMP+1
4375 06F4               ASL      ADP2          ; COMPUTE 40*(199-Y1CORD)
4377 26F5               ROL      ADP2+1        ;  2*(199-Y1CORD)
4379 06F4               ASL      ADP2
437B 26F5               ROL      ADP2+1        ;  4*(199-Y1CORD)
437D A5F4               LDA      ADP2          ;  ADD IN TEMPORARY SAVE OF (199-Y1CORD)
437F 18                 CLC                    ;  TO MAKE 5*(199-Y1CORD)
4380 65EE               ADC      TEMP
4382 85F4               STA      ADP2
4384 A5F5               LDA      ADP2+1
4386 65EF               ADC      TEMP+1
4388 85F5               STA      ADP2+1        ;  5*(199-Y1CORD)
438A 06F4               ASL      ADP2          ;  10*(199-Y1CORD)
438C 26F5               ROL      ADP2+1
438E 06F4               ASL      ADP2          ;  20*(199-Y1CORD)
4390 26F5               ROL      ADP2+1
4392 06F4               ASL      ADP2          ;  40*(199-Y1CORD)
4394 26F5               ROL      ADP2+1
4396 A5F4               LDA      ADP2          ; ADD IN INT(X1CORD/8) COMPUTED EARLIER
4398 18                 CLC
4399 65F2               ADC      ADP1
439B 85F2               STA      ADP1
439D A5F5               LDA      ADP2+1
439F 65F3               ADC      ADP1+1
43A1 65E3               ADC      VMORG         ; ADD IN VMORG*256
43A3 85F3               STA      ADP1+1        ; FINAL RESULT
43A5 60                 RTS                    ; RETURN
```

```
                                .PAGE  'INDIVIDUAL PIXEL SUBROUTINES'
                      ;         STPIX - SETS THE PIXEL AT X1CORD,Y1CORD TO A ONE (WHITE DOT)
                      ;         DOES NOT ALTER X1CORD OR Y1CORD
                      ;         ASSUMES IN RANGE CORRDINATES
43A6 204C43   STPIX:  JSR     PIXADR        ; GET BYTE ADDRESS AND BIT NUMBER OF PIXE
                                            ; INTO ADP1
43A9 A4F0             LDY     BTPT          ; GET BIT NUMBER IN Y
43AB B9D543           LDA     MSKTB1,Y      ; GET A BYTE WITH THAT BIT =1, OTHERS =0
43AE A000             LDY     #0            ; ZERO Y
43B0 11F2             ORA     (ADP1),Y      ; COMBINE THE BIT WITH THE ADDRESSED VM
43B2 91F2             STA     (ADP1),Y      ; BYTE
43B4 60               RTS                   ; RETURN
                      ;         CLPIX - CLEARS THE PIXEL AT X1CORD,Y1CORD TO A ZERO (BLACK DO
                      ;         DOES NOT ALTER X1CORD OR Y1CORD
                      ;         ASSUMES IN RANGE COORDINATES
43B5 204C43   CLPIX:  JSR     PIXADR        ; GET BYTE ADDRESS AND BIT NUMBER OF PIXE
                                            ; INTO ADP1
43B8 A4F0             LDY     BTPT          ; GET BIT NUMBER IN Y
43BA B9DD43           LDA     MSKTB2,Y      ; GET A BYTE WITH THAT BIT =0, OTHERS =1
43BD A000             LDY     #0            ; ZERO Y
43BF 31F2             AND     (ADP1),Y      ; REMOVE THE BIT FROM THE ADDRESSED VM
43C1 91F2             STA     (ADP1),Y      ; BYTE
43C3 60               RTS                   ; AND RETURN
                      ;         RDPIX - READS THE PIXEL AT X1CORD,Y1CORD AND SETS A TO ALL
                      ;         ZEROES IF IT IS A ZERO OR TO ONE IF IT IS A ONE.
                      ;         LOW BYTE OF ADP1 IS EQUAL TO A ON RETURN
                      ;         DOES NOT ALTER X1CORD OR Y1CORD
                      ;         ASSUMES IN RANGE CORRDINATES
43C4 204C43   RDPIX:  JSR     PIXADR        ; GET BYTE AND BIT ADDRESS OF PIXEL
43C7 A000             LDY     #0            ; GET ADDRESSED BYTE FROM VM
43C9 B1F2             LDA     (ADP1),Y
43CB A4F0             LDY     BTPT          ; GET BIT NUMBER IN Y
43CD 39D543           AND     MSKTB1,Y      ; CLEAR ALL BUT ADDRESSED BIT
43D0 F002             BEQ     RDPIX1        ; SKIP AHEAD IF IT WAS A ZERO
43D2 A901             LDA     #X'01         ; SET TO 01 IF IT WAS A ONE
43D4 60       RDPIX1: RTS                   ; RETURN
                      ;         MASK TABLES FOR INDIVIDUAL PIXEL SUBROUTINES
                      ;         MSKTB1 IS A TABLE OF 1 BITS CORRESPONDING TO BIT NUMBERS
                      ;         MSKTB2 IS A TABLE OF 0 BITS CORRESPONDING TO BIT NUMBERS
43D5 80402010 MSKTB1: .BYTE  X'80,X'40,X'20,X'10
43D9 08040201         .BYTE  X'08,X'04,X'02,X'01
43DD 7FBFDFEF MSKTB2: .BYTE  X'7F,X'BF,X'DF,X'EF
43E1 F7FBFDFE         .BYTE  X'F7,X'FB,X'FD,X'FE
                      ;         WRPIX - SETS THE PIXEL AT X1CORD,Y1CORD ACCORDING TO THE STA
                      ;         OF BIT 0 (RIGHTMOST) OF A
                      ;         DOES NOT ALTER X1CORD OR Y1CORD
                      ;         ASSUMES IN RANGE CORRDINATES
```

VMBAS   BASIC/VM PATCHES
INDIVIDUAL PIXEL SUBROUTINES

```
43E5 2901      WRPIX:  AND    #X'01          ; TEST LOW BIT OF A
43E7 F0CC              BEQ    CLPIX          ; GO WRITE A ZERO IF IT IS ZERO
43E9 D0BB              BNE    STPIX          ; OTHERWISE WRITE A ONE
```

VMBAS   BASIC/VM PATCHES
COORDINATE CHECK AND CORRECT ROUTINE

```
                       .PAGE  'COORDINATE CHECK AND CORRECT ROUTINE'
               ;       CHECKS ALL COORDINATES TO VERIFY THAT THEY ARE IN THE
               ;       PROPER RANGE.  IF NOT, THEY ARE REPLACED BY A VALUE
               ;       MODULO THE MAXIMUM VALUE+1.
               ;       NOTE THAT THESE ROUTINES CAN BE VERY SLOW WHEN CORRECTIONS ARE
               ;       NECESSARY BECAUSE A BRUTE FORCE DIVISON ROUTINE IS USED TO
               ;       COMPUTE THE MODULUS.

43EB A200      CKCRD:  LDX    #X1CORD-X1CORD ; CHECK X1CORD
43ED A000              LDY    #XLIMIT-LIMTAB
43EF 200344            JSR    CK
43F2 A204              LDX    #X2CORD-X1CORD ; CHECK X2CORD
43F4 200344            JSR    CK
43F7 A202              LDX    #Y1CORD-X1CORD ; CHECK Y1CORD
43F9 A002              LDY    #YLIMIT-LIMTAB
43FB 200344            JSR    CK
43FE A606              LDX    Y2CORD-X1CORD  ; CHECK Y2CORD
4400 4C0344            JMP    CK             ; AND RETURN


4403 B5E7      CK:     LDA    X1CORD+1,X     ; CHECK UPPER BYTE
4405 D92744            CMP    LIMTAB+1,Y     ; AGAINST UPPER BYTE OF LIMIT
4408 901B              BCC    CK4            ; OK IF LESS THAN UPPER BYTE OF LIMIT
440A F012              BEQ    CK3            ; GO CHECK LOWER BYTE IF EQUAL TO
                                             ; UPPER BYTE OF LIMIT
440C B5E6      CK2:    LDA    X1CORD,X       ; SUBTRACT THE LIMIT
440E 38                SEC                   ; LOWER BYTE FIRST
440F F92644            SBC    LIMTAB,Y
4412 95E6              STA    X1CORD,X
4414 B5E7              LDA    X1CORD+1,X
4416 F92744            SBC    LIMTAB+1,Y
4419 95E7              STA    X1CORD+1,X
441B 4C0344            JMP    CK             ; AND THEN GO CHECK RANGE AGAIN
441E B5E6      CK3:    LDA    X1CORD,X       ; CHECK LOWER BYTE OF X
4420 D92644            CMP    LIMTAB,Y
4423 B0E7              BCS    CK2            ; GO ADJUST IF TOO LARGE
4425 60        CK4:    RTS                   ; RETURN


               LIMTAB:                       ; TABLE OF LIMITS
4426 4001      XLIMIT: .WORD  NX
4428 C800      YLIMIT: .WORD  NY
```

```
                                  .PAGE   'LINE DRAWING ROUTINES'
                       ;          DRAW - DRAW THE BEST STRAIGHT LINE FROM X1CORD,Y1CORD TO
                       ;          X2CORD,Y2CORD.
                       ;          X2CORD,Y2CORD COPIED TO X1CORD,Y1CORD AFTER DRAWING
                       ;          USES AN ALGORITHM THAT REQUIRES NO MULTIPLICATION OR DIVIS

 442A A900             ERASE:     LDA     #X'00         ; SET LINE COLOR TO BLACK
 442C F002                        BEQ     DRAW1         ; GO DRAW THE LINE

 442E A9FF             DRAW:      LDA     #X'FF         ; SET LINE COLOR TO WHITE
 4430 85FF             DRAW1:     STA     COLOR

                       ;          COMPUTE SIGN AND MAGNITUDE OF DELTA X = X2-X1
                       ;          PUT MAGNITUDE IN DELTAX AND SIGN IN XDIR

 4432 A900                        LDA     #0            ; FIRST ZERO XDIR
 4434 85FC                        STA     XDIR
 4436 A5EA                        LDA     X2CORD        ; NEXT COMPUTE TWOS COMPLEMENT DIFFERE
 4438 38                          SEC
 4439 E5E6                        SBC     X1CORD
 443B 85F6                        STA     DELTAX
 443D A5EB                        LDA     X2CORD+1
 443F E5E7                        SBC     X1CORD+1
 4441 85F7                        STA     DELTAX+1
 4443 100F                        BPL     DRAW2         ; SKIP AHEAD IF DIFFERENCE IS POSITIVE
 4445 C6FC                        DEC     XDIR          ; SET XDIR TO -1
 4447 38                          SEC                   ; NEGATE DELTAX
 4448 A900                        LDA     #0
 444A E5F6                        SBC     DELTAX
 444C 85F6                        STA     DELTAX
 444E A900                        LDA     #0
 4450 E5F7                        SBC     DELTAX+1
 4452 85F7                        STA     DELTAX+1

                       ;          COMPUTE SIGN AND MAGNITUDE OF DELTA Y = Y2-Y1
                       ;          PUT MAGNITUDE IN DELTAY AND SIGN IN YDIR

 4454 A900             DRAW2:     LDA     #0            ; FIRST ZERO YDIR
 4456 85FD                        STA     YDIR
 4458 A5EC                        LDA     Y2CORD        ; NEXT COMPUTE TWOS COMPLEMENT DIFFERE
 445A 38                          SEC
 445B E5E8                        SBC     Y1CORD
 445D 85F8                        STA     DELTAY
 445F A5ED                        LDA     Y2CORD+1
 4461 E5E9                        SBC     Y1CORD+1
 4463 85F9                        STA     DELTAY+1
 4465 100F                        BPL     DRAW3         ; SKIP AHEAD IF DIFFERENCE IS POSITIVE
 4467 C6FD                        DEC     YDIR          ; SET YDIR TO -1
 4469 38                          SEC                   ; NEGATE DELTAX
 446A A900                        LDA     #0
 446C E5F8                        SBC     DELTAY
 446E 85F8                        STA     DELTAY
 4470 A900                        LDA     #0
 4472 E5F9                        SBC     DELTAY+1
 4474 85F9                        STA     DELTAY+1
```

```

                    ;       DETERMINE IF DELTAY IS LARGER THAN DELTAX
                    ;       IF SO, EXCHANGE DELTAY AND DELTAX AND SET XCHFLG NONZERO
                    ;       ALSO INITIALIZE ACC TO DELTAX
                    ;       PUT A DOT AT THE INITIAL DENPOINT

4476 A900           DRAW3:  LDA    #0           ; FIRST ZERO XCHFLG
4478 85FE                   STA    XCHFLG
447A A5F8                   LDA    DELTAY       ; COMPARE DELTAY WITH DELTAX
447C 38                     SEC
447D E5F6                   SBC    DELTAX
447F A5F9                   LDA    DELTAY+1
4481 E5F7                   SBC    DELTAX+1
4483 9012                   BCC    DRAW4        ; SKIP EXCHANGE IF DELTAX IS GREATER THAN
                                                ; DELTAY
4485 A6F8                   LDX    DELTAY       ; EXCHANGE DELTAX AND DELTAY
4487 A5F6                   LDA    DELTAX
4489 85F8                   STA    DELTAY
448B 86F6                   STX    DELTAX
448D A6F9                   LDX    DELTAY+1
448F A5F7                   LDA    DELTAX+1
4491 85F9                   STA    DELTAY+1
4493 86F7                   STX    DELTAX+1
4495 C6FE                   DEC    XCHFLG       ; SET XCHFLG TO -1
4497 A5F6           DRAW4:  LDA    DELTAX       ; INITIALIZE ACC TO DELTAX
4499 85FA                   STA    ACC
449B A5F7                   LDA    DELTAX+1
449D 85FB                   STA    ACC+1
449F A5FF                   LDA    COLOR        ; PUT A DOT AT THE INITIAL ENDPOINT
44A1 20E543                 JSR    WRPIX        ; X1CORD,Y1CORD

                    ;       HEAD OF MAIN DRAWING LOOP
                    ;       TEST IF DONE

44A4 A5FE           DRAW45: LDA    XCHFLG       ; TEST IF X AND Y EXCHANGED
44A6 D00E                   BNE    DRAW5        ; JUMP AHEAD IF SO
44A8 A5E6                   LDA    X1CORD       ; TEST FOR X1CORD=X2CORD
44AA C5EA                   CMP    X2CORD
44AC D015                   BNE    DRAW7        ; GO FOR ANOTHER ITERATION IF NOT
44AE A5E7                   LDA    X1CORD+1
44B0 C5EB                   CMP    X2CORD+1
44B2 D00F                   BNE    DRAW7        ; GO FOR ANOTHER ITERATION IF NOT
44B4 F00C                   BEQ    DRAW6        ; GO RETURN IF SO
44B6 A5E8           DRAW5:  LDA    Y1CORD       ; TEST FOR Y1CORD=Y2CORD
44B8 C5EC                   CMP    Y2CORD
44BA D007                   BNE    DRAW7        ; GO FOR ANOTHER INTERATION IF NOT
44BC A5E9                   LDA    Y1CORD+1
44BE C5ED                   CMP    Y2CORD+1
44C0 D001                   BNE    DRAW7        ; GO FOR ANOTHER INTERATION IF NOT
44C2 60             DRAW6:  RTS                 ; RETURN

                    ;       DO A CLACULATION TO DETERMINE IF ONE OR BOTH AXES ARE TO BE
                    ;       BUMPED (INCREMENTED OR DECREMENTED ACCORDING TO XDIR AND YDIR)
                    ;       AND DO THE BUMPING

```

```
44C3 A5FE     DRAW7:   LDA   XCHFLG     ; TEST IF X AND Y EXCHANGED
44C5 D006              BNE   DRAW8      ; JUMP IF SO
44C7 200F45            JSR   BMPX       ; BUMP X IF NOT
44CA 4CD044            JMP   DRAW9
44CD 202345   DRAW8:   JSR   BMPY       ; BUMP Y IF SO
44D0 20F344   DRAW9:   JSR   SBDY       ; SUBTRACT DY FROM ACC TWICE
44D3 20F344            JSR   SBDY
44D6 1013              BPL   DRAW12     ; SKIP AHEAD IF ACC IS NOT NEGATIVE
44D8 A5FE              LDA   XCHFLG     ; TEST IF X AND Y EXCHANGED
44DA D006              BNE   DRAW10     ; JUMP IF SO
44DC 202345            JSR   BMPY       ; BUMP Y IF NOT
44DF 4CE544            JMP   DRAW11
44E2 200F45   DRAW10:  JSR   BMPX       ; BUMP X IF SO
44E5 200145   DRAW11:  JSR   ADDX       ; ADD DX TO ACC TWICE
44E8 200145            JSR   ADDX
44EB A5FF     DRAW12:  LDA   COLOR      ; OUTPUT THE NEW POINT
44ED 20E543            JSR   WRPIX
44F0 4CA444            JMP   DRAW45     ; GO TEST IF DONE
```

```
                                .PAGE  'SUBROUTINES FOR DRAW'
                        ;       SUBROUTINES FOR DRAW

    44F3 A5FA   SBDY:   LDA     ACC             ; SUBTRACT DELTAY FROM ACC AND PUT RESULT
    44F5 38             SEC                     ; IN ACC
    44F6 E5F8           SBC     DELTAY
    44F8 85FA           STA     ACC
    44FA A5FB           LDA     ACC+1
    44FC E5F9           SBC     DELTAY+1
    44FE 85FB           STA     ACC+1
    4500 60             RTS


    4501 A5FA   ADDX:   LDA     ACC             ; ADD DELTAX TO ACC AND PUT RESULT IN ACC
    4503 18             CLC
    4504 65F6           ADC     DELTAX
    4506 85FA           STA     ACC
    4508 A5FB           LDA     ACC+1
    450A 65F7           ADC     DELTAX+1
    450C 85FB           STA     ACC+1
    450E 60             RTS


    450F A5FC   BMPX:   LDA     XDIR            ; BUMP X1CORD BY +1 OR -1 ACCORDING TO
    4511 D007           BNE     BMPX2           ; XDIR
    4513 E6E6           INC     X1CORD          ; DOUBLE INCREMENT X1CORD IF XDIR=0
    4515 D002           BNE     BMPX1
    4517 E6E7           INC     X1CORD+1
    4519 60     BMPX1:  RTS
    451A A5E6   BMPX2:  LDA     X1CORD          ; DOUBLE DECREMENT X1CORD IF XDIR≠0
    451C D002           BNE     BMPX3
    451E C6E7           DEC     X1CORD+1
    4520 C6E6   BMPX3:  DEC     X1CORD
    4522 60             RTS


    4523 A5FD   BMPY:   LDA     YDIR            ; BUMP Y1CORD BY +1 OR -1 ACCORDING TO
    4525 D007           BNE     BMPY2           ; YDIR
    4527 E6E8           INC     Y1CORD          ; DOUBLE INCREMENT Y1CORD IF YDIR=0
    4529 D002           BNE     BMPY1
    452B E6E9           INC     Y1CORD+1
    452D 60     BMPY1:  RTS
    452E A5E8   BMPY2:  LDA     Y1CORD          ; DOUBLE DECREMENT Y1CORD IF YDIR≠0
    4530 D002           BNE     BMPY3
    4532 C6E9           DEC     Y1CORD+1
    4534 C6E8   BMPY3:  DEC     Y1CORD
    4536 60             RTS

```

```
                          .PAGE  'SIMPLIFIED TEXT DISPLAY FOR BASIC'
                  ;       THIS SUBROUTINE TURNS THE VISABLE MEMORY INTO A DATA DISPLAY
                  ;       TERMINAL (GLASS TELETYPE).
                  ;       CHARACTER SET IS 96 FULL ASCII UPPER AND LOWER CASE.
                  ;       CHARACTER MATRIX IS 5 BY 7 SET INTO A 6 BY 9 RECTANGLE.
                  ;       LOWER CASE IS REPRESENTED AS SMALL (5 BY 5) CAPITALS.
                  ;       SCREEN CAPACITY IS 22 LINES OF 53 CHARACTERS
                  ;       CURSOR IS A NON-BLINKING UNDERLINE.
                  ;       CONTROL CODES RECOGNIZED:

                  ;       CR      X'OD          SETS CURSOR TO LEFT SCREEN EDGE
                  ;       LF      X'OA          MOVES CURSOR DOWN ONE LINE, SCROLLS
                  ;                             DISPLAY UP ONE LINE IF ALREADY ON BOTTOM
                  ;                             LINE
                  ;       BACK ARROW X'5F       MOVES CURSOR ONE CHARACTER LEFT, DOES
                  ;                             NOTHING IF ALREADY AT LEFT SCREEN EDGE
                  ;       FF      X'OC          CLEARS SCREEN AND PUTS CURSOR AT TOP LEF
                  ;                             OF SCREEN, SHOULD BE CALLED FOR
                  ;                             INITIALIZATION

                  ;       ALL OTHER CONTROL CODES IGNORED.
                  ;       ENTER WITH CHARACTER TO BE DISPLAYED IN A.
                  ;       CSRX SHOULD CONTAIN THE CHARACTER NUMBER
                  ;       CSRY SHOULD CONTAIN THE LINE NUMBER
                  ;       CSRX AND CSRY ARE CHECK FOR IN RANGE VALUES AND CORRECTED IF
                  ;       NECESSARY

                  ;       ************************************************************
                  ;       *
                  ;       *        VMORG MUST BE SET BEFORE CALLING SDTXT
                  ;       *
                  ;       ************************************************************

4537 48           SDTXT:  PHA                   ; SAVE INPUT
4538 208746               JSR     CKCUSR        ; CHECK AND CORRECT CURSOR SETTING
453B A900                 LDA     #0            ; CLEAR UPPER ADP2
453D 85F5                 STA     ADP2+1
453F 68                   PLA                   ; GET INPUT BACK
4540 48                   PHA                   ; BUT LEAVE IT ON THE STACK
4541 297F                 AND     #X'7F         ; INSURE 7 BIT ASCII INPUT
4543 38                   SEC
4544 E920                 SBC     #X'20         ; TEST IF A CONTROL CHARACTER
4546 3049                 BMI     SDTX10        ; JUMP IF SO
4548 C93F                 CMP     #X'5F-X'20    ; TEST IF BACK ARROW (UNDERLINE)
454A F045                 BEQ     SDTX10        ; JUMP IF SO

                  ;       CALCULATE TABLE ADDRESS FOR CHAR SHAPE AND PUT IT INTO ADP1

454C 85F4         SDTXT1: STA     ADP2          ; SAVE CHARACTER CODE IN ADP2
454E 203346               JSR     SADP2L        ; COMPUTE 8*CHARACTER CODE IN ADP2
4551 203346               JSR     SADP2L
4554 203346               JSR     SADP2L
4557 49FF                 EOR     #X'FF         ; NEGATE CHARACTER CODE
4559 38                   SEC                   ; SUBTRACT CHARACTER CODE FROM ADP2 AND
455A 65F4                 ADC     ADP2          ; PUT RESULT IN ADP1 FOR A FINAL RESULT
```

```
455C 85F2                 STA   ADP1          ; 7*CHARACTER CODE
455E A5F5                 LDA   ADP2+1
4560 69FF                 ADC   #X'FF
4562 85F3                 STA   ADP1+1
4564 A5F2                 LDA   ADP1          ; ADD IN ORIGIN OF CHARACTER TABLE
4566 18                   CLC
4567 6938                 ADC   #CHTB&X'FF
4569 85F2                 STA   ADP1
456B A5F3                 LDA   ADP1+1
456D 6947                 ADC   #CHTB/256
456F 85F3                 STA   ADP1+1        ; ADP1 NOW HAS ADDRESS OF TOP ROW OF
                                              ; CHARACTER SHAPE
                ;         COMPUTE BYTE AND BIT ADDRESS OF FIRST SCAN LINE OF
                ;         CHARACTER AT CURSOR POSITION
4571 204446               JSR   CSRTAD        ; COMPUTE BYTE AND BIT ADDRESSES OF FIRST
                                              ; SCAN LINE OF CHARACTER AT CURSOR POS.
                ;         SCAN OUT THE 7 CHARACTER ROWS
4574 A000                 LDY   #0            ; INITIALIZE Y INDEX=FONT TABLE POINTER
4576 B1F2       SDTX2:    LDA   (ADP1),Y      ; GET A DOT ROW FROM THE FONT TABLE
4578 20A246               JSR   MERGE         ; MERGE IT WITH GRAPHIC MEMORY AT (ADP2)
457B 203846               JSR   DN1SCN        ; ADD 40 TO ADP2 TO MOVE DOWN ONE SCAN
                                              ; LINE IN GRAPHIC MEMORY
457E C8                   INY                 ; BUMP UP POINTER INTO FONT TABLE
457F C007                 CPY   #7            ; TEST IF DONE
4581 D0F3                 BNE   SDTX2         ; GO DO NEXT SCAN LINE IF NOT
4583 A5E4                 LDA   CSRX          ; DO A CURSOR RIGHT
4585 C934                 CMP   #NCHR-1       ; TEST IF LAST CHARACTER ON THE LINE
4587 1005                 BPL   SDTX3         ; SKIP CURSOR RIGHT IF SO
4589 202846               JSR   CSRCLR        ; CLEAR OLD CURSOR
458C E6E4                 INC   CSRX          ; MOVE CURSOR ONE POSITION RIGHT
458E 4C0746     SDTX3:    JMP   SDTXRT        ; GO INSERT CURSOR, RESTORE REGISTERS,
                                              ; AND RETURN
                ;         INTERPRET CONTROL CODES
4591 C9ED       SDTX10:   CMP   #X'0D-X'20    ; TEST IF CR
4593 F00F                 BEQ   SDTXCR        ; JUMP IF SO
4595 C9EA                 CMP   #X'0A-X'20    ; TEST IF LF
4597 F02F                 BEQ   SDTXLF        ; JUMP IF SO
4599 C93F                 CMP   #X'5F-X'20    ; TEST IF BACK ARROW (UNDERLINE)
459B F011                 BEQ   SDTXCL        ; JUMP IF SO
459D C9EC                 CMP   #X'0C-X'20    ; TEST IF FF
459F F01B                 BEQ   SDTXFF        ; JUMP IF SO
45A1 4C0746               JMP   SDTXRT        ; GO RETURN IF UNRECOGNIZABLE CONTROL
45A4 202846     SDTXCR:   JSR   CSRCLR        ; CARRIAGE RETURN, FIRST CLEAR CURSOR
45A7 A900                 LDA   #0            ; ZERO CURSOR HORIZONTAL POSITION
45A9 85E4                 STA   CSRX
45AB 4C0746               JMP   SDTXRT        ; GO SET CURSOR AND RETURN
45AE 202846     SDTXCL:   JSR   CSRCLR        ; CURSOR LEFT, FIRST CLEAR CURSOR
45B1 A5E4                 LDA   CSRX          ; GET CURSOR HORIZONTAL POSITION
```

```
45B3 C900              CMP   #0            ; TEST IF AGAINST LEFT EDGE
45B5 F002              BEQ   SDTX20        ; SKIP UPDATE IF SO
45B7 C6E4              DEC   CSRX          ; OTHERWISE DECREMENT CURSOR X POSITI
45B9 4C0746   SDTX20:  JMP   SDTXRT        ; GO SET CURSOR AND RETURN

45BC 203943   SDTXFF:  JSR   CLEAR         ; CLEAR THE SCREEN
45BF A900              LDA   #0
45C1 85E4              STA   CSRX          ; PUT CURSOR IN UPPER LEFT CORNER
45C3 85E5              STA   CSRY
45C5 4C0746            JMP   SDTXRT        ; GO SET CURSOR AND RETURN

45C8 202846   SDTXLF:  JSR   CSRCLR        ; LINE FEED, FIRST CLEAR CURSOR
45CB A5E5              LDA   CSRY          ; GET CURRENT LINE POSITION
45CD C915              CMP   #NLIN-1       ; TEST IF AT BOTTOM OF SCREEN
45CF 1004              BPL   SDTX40        ; GO SCROLL IF SO
45D1 E6E5              INC   CSRY          ; INCREMENT LINE NUMBER IF NOT AT BOT
45D3 D032              BNE   SDTXRT        ; GO INSERT CURSOR AND RETURN
45D5 A900     SDTX40:  LDA   #0            ; SET UP ADDRESS POINTERS FOR MOVE
45D7 85F4              STA   ADP2          ; ADP1 = SOURCE FOR MOVE = FIRST BYTE
45D9 A5E3              LDA   VMORG         ; SECOND LINE OF TEXT
45DB 85F5              STA   ADP2+1        ; ADP2 = DESTINATION FOR MOVE = FIRST
45DD 18                CLC                 ; IN VISIBLE MEMORY
45DE 6901              ADC   #CHHI*40/256
45E0 85F3              STA   ADP1+1
45E2 A968              LDA   #CHHI*40&X'FF
45E4 85F2              STA   ADP1
45E6 A988              LDA   #NSCRL&X'FF   ; SET NUMBER OF LOCATIONS TO MOVE
45E8 85EE              STA   DCNT1         ; LOW PART
45EA A91D              LDA   #NSCRL/256    ; HIGH PART
45EC 85EF              STA   DCNT1+1
45EE 20EE46            JSR   FMOVE         ; EXECUTE MOVE USING AN OPTIMIZED, HI
                                           ; SPEED MEMORY MOVE ROUTINE

                                           ; CLEAR LAST LINE OF TEXT
45F1 A988              LDA   #NLIN-1*CHHI*40&X'FF   ; SET ADDRESS POINTER
45F3 85F4              STA   ADP2          ; LOW BYTE
45F5 A91D              LDA   #NLIN-1*CHHI*40/256
45F7 18                CLC
45F8 65E3              ADC   VMORG
45FA 85F5              STA   ADP2+1        ; HIGH BYTE
45FC A9B8              LDA   #NCLR&X'FF    ; SET LOW BYTE OF CLEAR COUNT
45FE 85EE              STA   DCNT1
4600 A901              LDA   #NCLR/256     ; SET HIGH BYTE OF CLEAR COUNT
4602 85EF              STA   DCNT1+1
4604 201A47            JSR   FCLR          ; CLEAR THE DESIGNATED AREA

                                           ; NO EFFECTIVE CHANGE IN CURSOR POSI

4607 201D46   SDTXRT:  JSR   CSRSET        ; RETURN SEQUENCE, INSERT CURSOR
460A 68                PLA                 ; RESTORE INPUT FROM THE STACK
460B 60                RTS                 ; RETURN
```

```
                                .PAGE  'SUBROUTINES FOR SDTXT'
                    ;           COMPUTE ADDRESS OF BYTE CONTAINING LAST SCAN LINE OF
                    ;           CHARACTER AT CURSOR POSITION
                    ;           ADDRESS = CSRTAD+(CHHI-1)*40   SINCE CHHI IS A CONSTANT 9,
                    ;           (CHHI-1)*40=320
                    ;           BTPT HOLDS BIT ADDRESS, 0=LEFTMOST

4600C 204446        CSRBAD: JSR    CSRTAD       ; COMPUTE ADDRESS OF TOP OF CHARACTER CELL
                                                ; FIRST
460F A5F4                   LDA    ADP2         ; ADD 320 TO RESULT = 8 SCAN LINES
4611 18                     CLC
4612 6940                   ADC    #320&X'FF
4614 85F4                   STA    ADP2
4616 A5F5                   LDA    ADP2+1
4618 6901                   ADC    #320/256
461A 85F5                   STA    ADP2+1
461C 60                     RTS

                    ;           SET CURSOR AT CURRENT POSITION

461D 208746         CSRSET: JSR    CKCUSR       ; VERIFY LEGAL CURSOR COORDINATES
4620 200C46                 JSR    CSRBAD       ; GET BYTE AND BIT ADDRESS OF CURSOR
4623 A9F8                   LDA    #X'F8        ; DATA = UNDERLINE CURSOR
4625 4CA246         CSRST1: JMP    MERGE        ; MERGE CURSOR WITH GRAPHIC MEMORY
                                                ; AND RETURN

                    ;           CLEAR CURSOR AT CURRENT POSITION

4628 208746         CSRCLR: JSR    CKCUSR       ; VERIFY LEGAL CURSOR COORDINATES
462B 200C46                 JSR    CSRBAD       ; GET BYTE AND BIT ADDRESS OF CURSOR
462E A900                   LDA    #0           ; DATA = BLANK DOT ROW
4630 4CA246                 JMP    MERGE        ; REMOVE DOT ROW FROM GRAPHIC MEMORY
                                                ; AND RETURN

                    ;           SHIFT ADP2 LEFT ONE BIT POSITION

4633 06F4           SADP2L: ASL    ADP2
4635 26F5                   ROL    ADP2+1
4637 60                     RTS

                    ;           MOVE DOWN ONE SCAN LINE      DOUBLE ADDS 40 TO ADP2

4638 A5F4           DN1SCN: LDA    ADP2         ; ADD 40 TO LOW BYTE
463A 18                     CLC
463B 6928                   ADC    #40
463D 85F4                   STA    ADP2
463F 9002                   BCC    DN1SC1       ; EXTEND CARRY INTO UPPER BYTE
4641 E6F5                   INC    ADP2+1
4643 60             DN1SC1: RTS                 ; RETURN

                    ;           COMPUTE BYTE ADDRESS CONTAINING FIRST SCAN LINE OF
                    ;           CHARACTER AT CURSOR POSITION AND PUT IN ADP2
                    ;           BIT ADDRESS (BIT 0 IS LEFTMOST) AT BTPT
                    ;           BYTE ADDRESS =VMORG*256+CHHI*40*CSRY+INT(CSRX*6/8)
                    ;           SINCE CHHI IS A CONSTANT 9, THEN CHHI*40=360
```

```
                     ;        BIT ADDRESS=REM(CSRX*5/8)

4644 A900     CSRTAD: LDA     #0          ; ZERO UPPER ADP2
4646 85F5             STA     ADP2+1
4648 A5E5             LDA     CSRY        ; FIRST COMPUTE 360*CSRY
464A 0A               ASLA                ;   COMPUTE 9*CSRY DIRECTLY IN A
464B 0A               ASLA
464C 0A               ASLA
464D 65E5             ADC     CSRY
464F 85F4             STA     ADP2        ;   STORE 9*CSRY IN LOWER ADP2
4651 203346           JSR     SADP2L      ;   18*CSRY IN ADP2
4654 203346           JSR     SADP2L      ;   36*CSRY IN ADP2
4657 65F4             ADC     ADP2        ;   ADD IN 9*CSRY TO MAKE 45*CSRY
4659 85F4             STA     ADP2
465B A900             LDA     #0
465D 65F5             ADC     ADP2+1
465F 85F5             STA     ADP2+1      ;   45*CSRY IN ADP2
4661 203346           JSR     SADP2L      ;   90*CSRY IN ADP2
4664 203346           JSR     SADP2L      ;   180*CSRY IN ADP2
4667 203346           JSR     SADP2L      ;   360*CSRY IN ADP2
466A A5E4             LDA     CSRX        ; NEXT COMPUTE 6*CSRX WHICH IS A 9 BIT
466C 0A               ASLA                ; VALUE
466D 65E4             ADC     CSRX
466F 0A               ASLA
4670 85F0             STA     BTPT        ;   SAVE RESULT TEMPORARILY
4672 6A               RORA                ;   DIVIDE BY 8 AND TRUNCATE FOR INT
4673 4A               LSRA                ;   FUNCTION
4674 4A               LSRA                ;   NOW HAVE INT(CSRX*6/8)
4675 18               CLC                 ; DOUBLE ADD TO ADP2
4676 65F4             ADC     ADP2
4678 85F4             STA     ADP2
467A A5F5             LDA     ADP2+1
467C 65E3             ADC     VMORG       ; ADD IN VMORG*256
467E 85F5             STA     ADP2+1      ; FINISHED WITH ADP2
4680 A5F0             LDA     BTPT        ; COMPUTE REM(CSRX*6/8) WHICH IS LOW 3
4682 2907             AND     #7          ; BITS OF CSRX*6
4684 85F0             STA     BTPT        ; KEEP IN BTPT
4686 60               RTS                 ; FINISHED

                     ;        CHECK CSRX AND CSRY FOR LEGAL VALUES.  IF ILLEGAL, COMPUTE
                     ;        THEIR VALUE MOD THEIR MAXIMUM VALUE

4687 A5E4     CKCUSR: LDA     CSRX        ; GET CHARACTER NUMBER
4689 C935             CMP     #NCHR       ; COMPARE WITH MAXIMUM CHARACTER NUMBER
468B 9007             BCC     CKCSR1      ; JUMP AHEAD IF OK
468D E935             SBC     #NCHR       ; SUBTRACT MAXIMUM FROM IT IF TOO BIG
468F 85E4             STA     CSRX        ; SAVE UPDATED
4691 4C8746           JMP     CKCUSR      ; GO TRY AGAIN
4694 A5E5     CKCSR1: LDA     CSRY        ; GET LINE NUMBER
4696 C916             CMP     #NLIN       ; COMPARE WITH MAXIMUM LINE NUMBER
4698 9007             BCC     CKCSR2      ; GO RETURN IF OK
469A E916             SBC     #NLIN       ; SUBTRACT MAXIMUM FROM IT IF TOO BIG
469C 85E5             STA     CSRY        ; SAVE UPDATED
469E 4C9446           JMP     CKCSR1      ; GO TRY AGAIN
46A1 60       CKCSR2: RTS                 ; RETURN
```

```
                        ;       MERGE A ROW OF 5 DOTS WITH GRAPHIC MEMORY STARTING AT BYTE
                        ;       ADDRESS AND BIT NUMBER IN ADP2 AND BTPT
                        ;       5 DOTS TO MERGE LEFT JUSTIFIED IN A
                        ;       PRESERVES X AND Y
46A2 85FE       MERGE:  STA     MRGT1           ; SAVE INPUT DATA
46A4 98                 TYA                     ; SAVE Y
46A5 48                 PHA
46A6 A4F0               LDY     BTPT            ; OPEN UP A 5 BIT WINDOW IN GRAPHIC MEMORY
46A8 B9DE46             LDA     MERGT,Y         ; LEFT BITS
46AB A000               LDY     #0              ; ZERO Y
46AD 31F4               AND     (ADP2),Y
46AF 91F4               STA     (ADP2),Y
46B1 A4F0               LDY     BTPT
46B3 B9E646             LDA     MERGT+8,Y       ; RIGHT BITS
46B6 A001               LDY     #1
46B8 31F4               AND     (ADP2),Y
46BA 91F4               STA     (ADP2),Y
46BC A5FE               LDA     MRGT1           ; SHIFT DATA RIGHT TO LINE UP LEFTMOST
46BE A4F0               LDY     BTPT            ; DATA BIT WITH LEFTMOST GRAPHIC FIELD
46C0 F004               BEQ     MERGE2          ; SHIFT BTPT TIMES
46C2 4A         MERGE1: LSRA
46C3 88                 DEY
46C4 D0FC               BNE     MERGE1
46C6 11F4       MERGE2: ORA     (ADP2),Y        ; OVERLAY WITH GRAPHIC MEMORY
46C8 91F4               STA     (ADP2),Y
46CA A908               LDA     #8              ; SHIFT DATA LEFT TO LINE UP RIGHTMOST
46CC 38                 SEC                     ; DATA BIT WITH RIGHTMOST GRAPHIC FIELD
46CD E5F0               SBC     BTPT            ; SHIFT (8-BTPT) TIMES
46CF A8                 TAY
46D0 A5FE               LDA     MRGT1
46D2 0A         MERGE3: ASLA
46D3 88                 DEY
46D4 D0FC               BNE     MERGE3
46D6 C8                 INY
46D7 11F4               ORA     (ADP2),Y        ; OVERLAY WITH GRAPHIC MEMORY
46D9 91F4               STA     (ADP2),Y
46DB 68                 PLA                     ; RESTORE Y
46DC A8                 TAY
46DD 60                 RTS                     ; RETURN
46DE 0783C1E0   MERGT:  .BYTE   X'07,X'83,X'C1,X'E0  ; TABLE OF MASKS FOR OPENING UP
46E2 F0F8FCFE           .BYTE   X'F0,X'F8,X'FC,X'FE  ; A 5 BIT WINDOW ANYWHERE
46E6 FFFFFFFF           .BYTE   X'FF,X'FF,X'FF,X'FF  ; IN GRAPHIC MEMORY
46EA 7F3F1F0F           .BYTE   X'7F,X'3F,X'1F,X'0F
                        ;       FAST MEMORY MOVE ROUTINE
                        ;       ENTER WITH SOURCE ADDRESS IN ADPT1 AND DESTINATION ADDRESS IN
                        ;       ADPT2 AND MOVE COUNT (DOUBLE PRECISION) IN DCNT1.
                        ;       MOVE PROCEEDS FROM LOW TO HIGH ADDRESSES AT APPROXIMATELY 16US
                        ;       PER BYTE.
                        ;       EXIT WITH ADDRESS POINTERS AND COUNT IN UNKNOWN STATE.
                        ;       PRESERVES X AND Y REGISTERS.
```

```
46EE 8A          FMOVE:   TXA                    ; SAVE X AND Y ON THE STACK
46EF 48                   PHA
46F0 98                   TYA
46F1 48                   PHA
46F2 C6EF        FMOVE1:  DEC     DCNT1+1        ; TEST IF LESS THAN 256 LEFT TO MC
46F4 3015                 BMI     FMOVE3         ; JUMP TO FINAL MOVE IF SO
46F6 A000                 LDY     #0             ; MOVE A BLOCK OF 256 BYTES QUICKL
46F8 B1F2        FMOVE2:  LDA     (ADP1),Y       ; TWO BYTES AT A TIME
46FA 91F4                 STA     (ADP2),Y
46FC C8                   INY
46FD B1F2                 LDA     (ADP1),Y
46FF 91F4                 STA     (ADP2),Y
4701 C8                   INY
4702 D0F4                 BNE     FMOVE2         ; CONTINUE UNTIL DONE
4704 E6F3                 INC     ADP1+1         ; BUMP ADDRESS POINTERS TO NEXT PA
4706 E6F5                 INC     ADP2+1
4708 4CF246               JMP     FMOVE1         ; GO MOVE NEXT PAGE
470B A6EE        FMOVE3:  LDX     DCNT1          ; GET REMAINING BYTE COUNT INTO X
470D B1F2        FMOVE4:  LDA     (ADP1),Y       ; MOVE A BYTE
470F 91F4                 STA     (ADP2),Y
4711 C8                   INY
4712 CA                   DEX
4713 D0F8                 BNE     FMOVE4         ; CONTINUE UNTIL DONE
4715 68                   PLA                    ; RESTORE INDEX REGISTERS
4716 A8                   TAY
4717 68                   PLA
4718 AA                   TAX
4719 60                   RTS                    ; AND RETURN

                 ;        FAST MEMORY CLEAR ROUTINE
                 ;        ENTER WITH ADDRESS OF BLOCK TO CLEAR IN ADP2 AND CLEAR
                 ;        IN DCNT1.
                 ;        EXIT WITH ADDRESS POINTERS AND COUNT IN UNKNOWN STATE
                 ;        PRESERVES X AND Y REGISTERS

471A 98          FCLR:    TYA                    ; SAVE Y
471B 48                   PHA
471C A000        FCLR1:   LDY     #0
471E C6EF                 DEC     DCNT1+1        ; TEST IF LESS THAN 256 LEFT TO MC
4720 300B                 BMI     FCLR3          ; JUMP TO FINAL CLEAR IF SO
4722 98                   TYA                    ; CLEAR A BLOCK OF 256 QUICKLY
4723 91F4        FCLR2:   STA     (ADP2),Y       ; CLEAR A BYTE
4725 C8                   INY
4726 D0FB                 BNE     FCLR2
4728 E6F5                 INC     ADP2+1         ; BUMP ADDRESS POINTER TO NEXT PAC
472A 4C1C47               JMP     FCLR1          ; GO CLEAR NEXT PAGE
472D 98          FCLR3:   TYA                    ; CLEAR REMAINING PARTIAL PAGE
472E 91F4        FCLR4:   STA     (ADP2),Y
4730 C8                   INY
4731 C6EE                 DEC     DCNT1
4733 D0F9                 BNE     FCLR4
4735 68                   PLA                    ; RESTORE Y
4736 A8                   TAY
4737 60                   RTS                    ; RETURN
```

```
                                .PAGE    'CHARACTER FONT TABLE'
                        ;       CHARACTER FONT TABLE
                        ;       ENTRIES IN ORDER STARTING AT ASCII BLANK
                        ;       96 ENTRIES
                        ;       EACH ENTRY CONTAINS 7 BYTES
                        ;       7 BYTES ARE CHARACTER MATRIX, TOP ROW FIRST, LEFTMOST DOT
                        ;       IS LEFTMOST IN BYTE
                        ;       LOWER CASE FONT IS SMALL UPPER CASE, 5 BY 5 MATRIX

4738 000000             CHTB:   .BYTE       X'00,X'00,X'00     ; BLANK
473B 00000000                   .BYTE  X'00,X'00,X'00,X'00
473F 202020                     .BYTE       X'20,X'20,X'20     ; !
4742 20200020                   .BYTE  X'20,X'20,X'00,X'20
4746 505050                     .BYTE       X'50,X'50,X'50     ; "
4749 00000000                   .BYTE  X'00,X'00,X'00,X'00
474D 5050F8                     .BYTE       X'50,X'50,X'F8     ; #
4750 50F85050                   .BYTE  X'50,X'F8,X'50,X'50
4754 2078A0                     .BYTE       X'20,X'78,X'A0     ; $
4757 7028F020                   .BYTE  X'70,X'28,X'F0,X'20
475B C8C810                     .BYTE       X'C8,X'C8,X'10     ; %
475E 20409898                   .BYTE  X'20,X'40,X'98,X'98
4762 40A0A0                     .BYTE       X'40,X'A0,X'A0     ; &
4765 40A89068                   .BYTE  X'40,X'A8,X'90,X'68
4769 303030                     .BYTE       X'30,X'30,X'30     ; '
476C 00000000                   .BYTE  X'00,X'00,X'00,X'00
4770 204040                     .BYTE       X'20,X'40,X'40     ; (
4773 40404020                   .BYTE  X'40,X'40,X'40,X'20
4777 201010                     .BYTE       X'20,X'10,X'10     ; )
477A 10101020                   .BYTE  X'10,X'10,X'10,X'20
477E 20A870                     .BYTE       X'20,X'A8,X'70     ; *
4781 2070A820                   .BYTE  X'20,X'70,X'A8,X'20
4785 002020                     .BYTE       X'00,X'20,X'20     ; +
4788 F8202000                   .BYTE  X'F8,X'20,X'20,X'00
478C 000000                     .BYTE       X'00,X'00,X'00     ; ,
478F 30301020                   .BYTE  X'30,X'30,X'10,X'20
4793 000000                     .BYTE       X'00,X'00,X'00     ; -
4796 F8000000                   .BYTE  X'F8,X'00,X'00,X'00
479A 000000                     .BYTE       X'00,X'00,X'00     ; .
479D 00003030                   .BYTE  X'00,X'00,X'30,X'30
47A1 080810                     .BYTE       X'08,X'08,X'10     ; /
47A4 20408080                   .BYTE  X'20,X'40,X'80,X'80
47A8 609090                     .BYTE       X'60,X'90,X'90     ; 0
47AB 90909060                   .BYTE  X'90,X'90,X'90,X'60
47AF 206020                     .BYTE       X'20,X'60,X'20     ; 1
47B2 20202070                   .BYTE  X'20,X'20,X'20,X'70
47B6 708810                     .BYTE       X'70,X'88,X'10     ; 2
47B9 204080F8                   .BYTE  X'20,X'40,X'80,X'F8
47BD 708808                     .BYTE       X'70,X'88,X'08     ; 3
47C0 30088870                   .BYTE  X'30,X'08,X'88,X'70
47C4 103050                     .BYTE       X'10,X'30,X'50     ; 4
47C7 90F81010                   .BYTE  X'90,X'F8,X'10,X'10
47CB F880F0                     .BYTE       X'F8,X'80,X'F0     ; 5
47CE 080808F0                   .BYTE  X'08,X'08,X'08,X'F0
47D2 708080                     .BYTE       X'70,X'80,X'80     ; 6
47D5 F0888870                   .BYTE  X'F0,X'88,X'88,X'70
```

```
47D9 F80810             .BYTE        X'F8,X'08,X'10     ; 7
47DC 20408080           .BYTE   X'20,X'40,X'80,X'80
47E0 708888             .BYTE        X'70,X'88,X'88     ; 8
47E3 70888870           .BYTE   X'70,X'88,X'88,X'70
47E7 708888             .BYTE        X'70,X'88,X'88     ; 9
47EA 78080870           .BYTE   X'78,X'08,X'08,X'70
47EE 303000             .BYTE        X'30,X'30,X'00     ; :
47F1 00003030           .BYTE   X'00,X'00,X'30,X'30
47F5 303000             .BYTE        X'30,X'30,X'00     ; ;
47F8 30301020           .BYTE   X'30,X'30,X'10,X'20
47FC 102040             .BYTE        X'10,X'20,X'40     ; LESS THAN
47FF 80402010           .BYTE   X'80,X'40,X'20,X'10
4803 0000F8             .BYTE        X'00,X'00,X'F8     ; =
4806 00F80000           .BYTE   X'00,X'F8,X'00,X'00
480A 402010             .BYTE        X'40,X'20,X'10     ; GREATER THAN
480D 08102040           .BYTE   X'08,X'10,X'20,X'40
4811 708808             .BYTE        X'70,X'88,X'08     ; ?
4814 10200020           .BYTE   X'10,X'20,X'00,X'20
4818 708808             .BYTE        X'70,X'88,X'08     ; @
481B 68A8A8D0           .BYTE   X'68,X'A8,X'A8,X'D0
481F 205088             .BYTE        X'20,X'50,X'88     ; A
4822 88F88888           .BYTE   X'88,X'F8,X'88,X'88
4826 F04848             .BYTE        X'F0,X'48,X'48     ; B
4829 704848F0           .BYTE   X'70,X'48,X'48,X'F0
482D 708880             .BYTE        X'70,X'88,X'80     ; C
4830 80808870           .BYTE   X'80,X'80,X'88,X'70
4834 F04848             .BYTE        X'F0,X'48,X'48     ; D
4837 484848F0           .BYTE   X'48,X'48,X'48,X'F0
483B F88080             .BYTE        X'F8,X'80,X'80     ; E
483E F08080F8           .BYTE   X'F0,X'80,X'80,X'F8
4842 F88080             .BYTE        X'F8,X'80,X'80     ; F
4845 F0808080           .BYTE   X'F0,X'80,X'80,X'80
4849 708880             .BYTE        X'70,X'88,X'80     ; G
484C B8888870           .BYTE   X'B8,X'88,X'88,X'70
4850 888888             .BYTE        X'88,X'88,X'88     ; H
4853 F8888888           .BYTE   X'F8,X'88,X'88,X'88
4857 702020             .BYTE        X'70,X'20,X'20     ; I
485A 20202070           .BYTE   X'20,X'20,X'20,X'70
485E 381010             .BYTE        X'38,X'10,X'10     ; J
4861 10109060           .BYTE   X'10,X'10,X'90,X'60
4865 8890A0             .BYTE        X'88,X'90,X'A0     ; K
4868 C0A09088           .BYTE   X'C0,X'A0,X'90,X'88
486C 808080             .BYTE        X'80,X'80,X'80     ; L
486F 808080F8           .BYTE   X'80,X'80,X'80,X'F8
4873 88D8A8             .BYTE        X'88,X'D8,X'A8     ; M
4876 A8888888           .BYTE   X'A8,X'88,X'88,X'88
487A 8888C8             .BYTE        X'88,X'88,X'C8     ; N
487D A8988888           .BYTE   X'A8,X'98,X'88,X'88
4881 708888             .BYTE        X'70,X'88,X'88     ; O
4884 88888870           .BYTE   X'88,X'88,X'88,X'70
4888 F08888             .BYTE        X'F0,X'88,X'88     ; P
488B F0808080           .BYTE   X'F0,X'80,X'80,X'80
488F 708888             .BYTE        X'70,X'88,X'88     ; Q
4892 88A89068           .BYTE   X'88,X'A8,X'90,X'68
4896 F08888             .BYTE        X'F0,X'88,X'88     ; R
```

```
1145 4899 F0A09088          .BYTE  X'F0,X'A0,X'90,X'88
1146 489D 788080            .BYTE      X'78,X'80,X'80     ; S
1147 48A0 700808F0          .BYTE  X'70,X'08,X'08,X'F0
1148 48A4 F82020            .BYTE      X'F8,X'20,X'20     ; T
1149 48A7 20202020          .BYTE  X'20,X'20,X'20,X'20
1150 48AB 888888            .BYTE      X'88,X'88,X'88     ; U
1151 48AE 88888870          .BYTE  X'88,X'88,X'88,X'70
1152 48B2 888888            .BYTE      X'88,X'88,X'88     ; V
1153 48B5 50502020          .BYTE  X'50,X'50,X'20,X'20
1154 48B9 888888            .BYTE      X'88,X'88,X'88     ; W
1155 48BC A8A8D888          .BYTE  X'A8,X'A8,X'D8,X'88
1156 48C0 888850            .BYTE      X'88,X'88,X'50     ; X
1157 48C3 20508888          .BYTE  X'20,X'50,X'88,X'88
1158 48C7 888850            .BYTE      X'88,X'88,X'50     ; Y
1159 48CA 20202020          .BYTE  X'20,X'20,X'20,X'20
1160 48CE F80810            .BYTE      X'F8,X'08,X'10     ; Z
1161 48D1 204080F8          .BYTE  X'20,X'40,X'80,X'F8
1162 48D5 704040            .BYTE      X'70,X'40,X'40     ; LEFT BRACKET
1163 48D8 40404070          .BYTE  X'40,X'40,X'40,X'70
1164 48DC 808040            .BYTE      X'80,X'80,X'40     ; BACKSLASH
1165 48DF 20100808          .BYTE  X'20,X'10,X'08,X'08
1166 48E3 701010            .BYTE      X'70,X'10,X'10     ; RIGHT BRACKET
1167 48E6 10101070          .BYTE  X'10,X'10,X'10,X'70
1168 48EA 205088            .BYTE      X'20,X'50,X'88     ; CARROT
1169 48ED 00000000          .BYTE  X'00,X'00,X'00,X'00
1170 48F1 000000            .BYTE      X'00,X'00,X'00     ; UNDERLINE
1171 48F4 000000F8          .BYTE  X'00,X'00,X'00,X'F8
1172 48F8 C06030            .BYTE      X'C0,X'60,X'30     ; GRAVE ACCENT
1173 48FB 00000000          .BYTE  X'00,X'00,X'00,X'00
1174 48FF 000020            .BYTE      X'00,X'00,X'20     ; A (LC)
1175 4902 5088F888          .BYTE  X'50,X'88,X'F8,X'88
1176 4906 0000F0            .BYTE      X'00,X'00,X'F0     ; B (LC)
1177 4909 487048F0          .BYTE  X'48,X'70,X'48,X'F0
1178 490D 000078            .BYTE      X'00,X'00,X'78     ; C (LC)
1179 4910 80808078          .BYTE  X'80,X'80,X'80,X'78
1180 4914 0000F0            .BYTE      X'00,X'00,X'F0     ; D (LC)
1181 4917 484848F0          .BYTE  X'48,X'48,X'48,X'F0
1182 491B 0000F8            .BYTE      X'00,X'00,X'F8     ; E (LC)
1183 491E 80E080F8          .BYTE  X'80,X'E0,X'80,X'F8
1184 4922 0000F8            .BYTE      X'00,X'00,X'F8     ; F (LC)
1185 4925 80E08080          .BYTE  X'80,X'E0,X'80,X'80
1186 4929 000078            .BYTE      X'00,X'00,X'78     ; G (LC)
1187 492C 80988878          .BYTE  X'80,X'98,X'88,X'78
1188 4930 000088            .BYTE      X'00,X'00,X'88     ; H (LC)
1189 4933 88F88888          .BYTE  X'88,X'F8,X'88,X'88
1190 4937 000070            .BYTE      X'00,X'00,X'70     ; I (LC)
1191 493A 20202070          .BYTE  X'20,X'20,X'20,X'70
1192 493E 000038            .BYTE      X'00,X'00,X'38     ; J (LC)
1193 4941 10105020          .BYTE  X'10,X'10,X'50,X'20
1194 4945 000090            .BYTE      X'00,X'00,X'90     ; K (LC)
1195 4948 A0C0A090          .BYTE  X'A0,X'C0,X'A0,X'90
1196 494C 000080            .BYTE      X'00,X'00,X'80     ; L (LC)
1197 494F 808080F8          .BYTE  X'80,X'80,X'80,X'F8
1198 4953 000088            .BYTE      X'00,X'00,X'88     ; M (LC)
1199 4956 D8A88888          .BYTE  X'D8,X'A8,X'88,X'88
```

```
4959A 000088              .BYTE       X'00,X'00,X'88    ; N (LC)
495D C8A89888             .BYTE  X'C8,X'A8,X'98,X'88
4961 000070               .BYTE       X'00,X'00,X'70    ; O (LC)
4964 88888870             .BYTE  X'88,X'88,X'88,X'70
4968 0000F0               .BYTE       X'00,X'00,X'F0    ; P (LC)
496B 88F08080             .BYTE  X'88,X'F0,X'80,X'80
496F 000070               .BYTE       X'00,X'00,X'70    ; Q (LC)
4972 88A89068             .BYTE  X'88,X'A8,X'90,X'68
4976 0000F0               .BYTE       X'00,X'00,X'F0    ; R (LC)
4979 88F0A090             .BYTE  X'88,X'F0,X'A0,X'90
497D 000078               .BYTE       X'00,X'00,X'78    ; S (LC)
4980 807008F0             .BYTE  X'80,X'70,X'08,X'F0
4984 0000F8               .BYTE       X'00,X'00,X'F8    ; T (LC)
4987 20202020             .BYTE  X'20,X'20,X'20,X'20
498B 000088               .BYTE       X'00,X'00,X'88    ; U (LC)
498E 88888870             .BYTE  X'88,X'88,X'88,X'70
4992 000088               .BYTE       X'00,X'00,X'88    ; V (LC)
4995 88885020             .BYTE  X'88,X'88,X'50,X'20
4999 000088               .BYTE       X'00,X'00,X'88    ; W (LC)
499C 88A8D888             .BYTE  X'88,X'A8,X'D8,X'88
49A0 000088               .BYTE       X'00,X'00,X'88    ; X (LC)
49A3 50205088             .BYTE  X'50,X'20,X'50,X'88
49A7 000088               .BYTE       X'00,X'00,X'88    ; Y (LC)
49AA 50202020             .BYTE  X'50,X'20,X'20,X'20
49AE 0000F8               .BYTE       X'00,X'00,X'F8    ; Z (LC)
49B1 102040F8             .BYTE  X'10,X'20,X'40,X'F8
49B5 102020               .BYTE       X'10,X'20,X'20    ; LEFT BRACE
49B8 60202010             .BYTE  X'60,X'20,X'20,X'10
49BC 202020               .BYTE       X'20,X'20,X'20    ; VERTICAL BAR
49BF 20202020             .BYTE  X'20,X'20,X'20,X'20
49C3 402020               .BYTE       X'40,X'20,X'20    ; RIGHT BRACE
49C6 30202040             .BYTE  X'30,X'20,X'20,X'40
49CA 10A840               .BYTE       X'10,X'A8,X'40    ; TILDA
49CD 00000000             .BYTE  X'00,X'00,X'00,X'00
49D1 A850A8               .BYTE       X'A8,X'50,X'A8    ; RUBOUT
49D4 50A850A8             .BYTE  X'50,X'A8,X'50,X'A8

0000            END:      .END
 NO ERROR LINES
```

```
                                 .PAGE  'KIM-1 ALPHANUMERIC KEYBOARD SCAN AND ENCODE ROUTINE'
                        ;        *****MODIFIED FOR KIM BASIC*****
                        ;        THIS SUBROUTINE SCANS AN UNENCODED KEYBOARD MATRIX CONNECTED
                        ;        TO THE KIM-1 APPLICATION CONNECTOR.  USER PERIPHERAL PORT B
                        ;        BITS 5 (MSB) THROUGH 2 (LSB) ARE CONNECTED TO A ONE-OF-16
                        ;        DECODER (74154) WHICH DRIVES THE KEYSWITCH COLUMNS.
                        ;        SENSING OF THE ROWS IS BY A PORTION OF THE KIM ON-BOARD
                        ;        KEYBOARD CIRCUITRY WHICH USES SYSTEM PERIPHERAL PORT B BITS
                        ;        0 - 4.
                        ;        WHEN CALLED, THE ROUTINE SITS IN A LOOP WAITING FOR A KEY TO
                        ;        BE PRESSED.  WHEN A KEY IS PRESSED (EXCEPTING SHIFT, CONTROL,
                        ;        REPEAT), THE ROUTINE RETURNS WITH KEY CODE IN ACCUMULATOR.
                        ;        BOTH INDEX REGISTERS ARE RETAINED.
                        ;        THE ROUTINE IMPLEMENTS TRUE 2-KEY ROLLOVER, KEY DEBOUNCING,
                        ;        AND REPEAT TIMING.  ONE RAM LOCATION IS REQUIRED, ITS INITIAL
                        ;        CONTENT IS INSIGNIFICANT.
                        ;        SHIFT LOCK IS SUPPORTED, IT ONLY AFFECTS LETTERS MAKING IT
                        ;        EFFECTIVELY A "CAPS LOCK" KEY.
                        ;        SHIFT LOCK SHOULD BE RELEASED WHEN USING THE KIM MONITOR.
                        ;        GERMANIUM DIODES SHOULD BE WIRED IN SERIES WITH ALL MODE
                        ;        MODIFYING KEYS TO AVOID THE "PHANTOM KEY" EFFECT.  THESE
                        ;        INCLUDE: SHIFT, CONTROL, REPEAT, AND SHIFT LOCK.


1740                    SYSPA    =      X'1740      ; SYSTEM PORT A DATA REGISTER
1741                    SYSPAD   =      X'1741      ; SYSTEM PORT A DIRECTION REGISTER
1702                    USRPB    =      X'1702      ; USER PORT B DATA REGISTER
1703                    USRPBD   =      X'1703      ; USER PORT B DIRECTION REGISTER
0032                    RPTRAT   =      50          ; REPEAT PERIOD, MILLISECONDS
0005                    DBCDLA   =      5           ; DEBOUNCE DELAY, MILLISECONDS

0000                             .=     X'0200      ; PUT INTO KIM RAM UNUSED BY BASIC

0200 4C0602                      JMP    ANKB        ; DISPATCH VECTOR  KEYBOARD ROUTINE
0203 4CF202                      JMP    CNTLC       ; DISPATCH VECTOR  CONTROL/C ROUTINE

0206 98                 ANKB:    TYA                ; SAVE THE INDEX REGISTERS
0207 48                          PHA
0208 8A                          TXA
0209 48                          PHA
020A A901                        LDA    #1          ; TEST KIM TTY/KEYB SWITCH
020C 2C4017                      BIT    X'1740
020F D006                        BNE    ANKB0       ; CONTINUE WITH KEYBOARD IF IN KEYB POSIT.
0211 205A1E                      JSR    X'1E5A      ; GET A TTY CHARACTER IF IN TTY POSITION
0214 4C9802                      JMP    ANKB10      ; GO ECHO IT AND RETURN
0217 AD4117             ANKB0:   LDA    SYSPAD      ; SET UP DATA DIRECTION REGISTERS
021A 29E0                        AND    #X'E0       ; SET SYSTEM PORT A BITS 4-0 TO INPUT
021C 8D4117                      STA    SYSPAD
021F AD0317                      LDA    USRPBD
0222 093C                        ORA    #X'3C       ; SET USER PORT B BITS 5-2 TO OUTPUT
0224 8D0317                      STA    USRPBD
0227 A032                        LDY    #RPTRAT     ; INITIALIZE REPEAT DELAY
0229 A205               ANKB1:   LDX    #DBCDLA     ; INITIALIZE DEBOUNCE DELAY
022B 20AF02             ANKB2:   JSR    WA1MS       ; WAIT 1 MILLISECOND
022E ADE103                      LDA    ANKBT1      ; GET KEY ADDRESS LAST DOWN
```

```
0231 20B602              JSR    KEYTST        ; TEST IF ADDRESSED KEY STILL DOWN
0234 B00C                BCS    ANKB4         ; JUMP IF UP
0236 A931                LDA    #X'31         ; TEST STATE OF REPEAT KEY
0238 20B602              JSR    KEYTST
023B B0EC                BCS    ANKB1         ; LOOP BACK IF REPEAT KEY IS UP
023D 88                  DEY                  ; DECREMENT REPEAT DELAY
023E D0E9                BNE    ANKB1         ; LOOP BACK IF REPEAT DELAY UNEXPIRED
0240 F029                BEQ    ANKB7         ; GO OUTPUT REPEATED CODE
0242 CA         ANKB4:   DEX                  ; DECREMENT DEBOUNCE DELAY
0243 D0E6                BNE    ANKB2         ; GO TEST KEY AGAIN IF NOT EXPIRED

                ;        PREVIOUS KEY IS NOW RELEASED, RESUME SCAN OF KEYBOARD

0245 EEE103     ANKB5:   INC    ANKBT1        ; INCREMENT KEY ADDRESS TO TEST
0248 ADE103              LDA    ANKBT1
024B C93F                CMP    #X'3F         ; SKIP OVER SHIFT
024D F0F6                BEQ    ANKB5
024F C933                CMP    #X'33         ; SKIP OVER CAPS LOCK
0251 F0F2                BEQ    ANKB5
0253 C92E                CMP    #X'2E         ; SKIP OVER CONTROL
0255 F0EE                BEQ    ANKB5
0257 C931                CMP    #X'31         ; SKIP OVER REPEAT
0259 F0EA                BEQ    ANKB5
025B A205                LDX    #DBCDLA       ; INITIALIZE DEBOUNCE DELAY
025D ADE103     ANKB6:   LDA    ANKBT1        ; TEST STATE OF CURRENTLY ADDRESSED KE
0260 20B602              JSR    KEYTST
0263 B0E0                BCS    ANKB5         ; GO TRY NEXT KEY IF THIS ONE IS UP
0265 20AF02              JSR    WA1MS         ; WAIT 1 MILLISECOND IF DOWN
0268 CA                  DEX                  ; DECREMENT DEBOUNCE DELAY
0269 D0F2                BNE    ANKB6         ; GO CHECK KEY AGAIN IF NOT EXPIRED

                ;        TRANSLATE AND OUTPUT A KEY CODE

026B AEE103     ANKB7:   LDX    ANKBT1        ; GET BASIC ASCII CODE FROM TABLE
026E BC4103              LDY    ANKBTB,X      ; INTO INDEX Y
0271 A92E                LDA    #X'2E         ; TEST STATE OF CONTROL KEY
0273 20B602              JSR    KEYTST
0276 B006                BCS    ANKB8         ; SKIP AHEAD IF NOT PRESSED
0278 98                  TYA                  ; CLEAR UPPER THREE BITS OF CODE IF
0279 291F                AND    #X'1F         ; CONTROL PRESSED
027B 4C9802              JMP    ANKB10        ; IGNORE SHIFT AND GO RETURN
027E A93F       ANKB8:   LDA    #X'3F         ; TEST STATE OF SHIFT KEY
0280 20B602              JSR    KEYTST
0283 9010                BCC    ANKB9         ; SKIP AHEAD IF PRESSED
0285 A933                LDA    #X'33         ; TEST STATE OF CAPS LOCK KEY
0287 20B602              JSR    KEYTST
028A 98                  TYA                  ; RETRIEVE PLAIN CODE FROM Y
028B B00B                BCS    ANKB10        ; GO RESTORE REGISTERS AND RETURN IF CA
                                              ; LOCK KEY IS UP
028D C961                CMP    #X'61         ; IF DOWN, TEST IF CODE IS A LETTER
028F 9007                BCC    ANKB10        ; NO, GO RETURN
0291 C97B                CMP    #X'7B
0293 B003                BCS    ANKB10        ; NO, GO RETURN
0295 BD9103     ANKB9:   LDA    ANKBTB+80,X   ; FETCH SHIFTED CODE FROM TABLE
0298 48         ANKB10:  PHA                  ; SAVE CHARACTER CODE
```

KIM-1 ALPHANUMERIC KEYBOARD SCAN AND ENCODE ROUTINE

```
0299 C90F                CMP    #X'0F        ; TEST IF THE CODE IS CNTL/O
029B D006                BNE    ANKB11       ; SKIP IF NOT
029D A514                LDA    X'14         ; TOGGLE OUTPUT ENABLE BIT IN BASIC
029F 45FF                EOR    X'FF
02A1 8514                STA    X'14
02A3 68        ANKB11:   PLA                 ; RESTORE A
02A4 BA                  TSX
02A5 BC0201              LDY    X'102,X      ; RESTORE Y FROM STACK
02A8 9D0201              STA    X'102,X      ; SAVE CHARACTER CODE IN STACK WHERE Y WAS
02AB 68                  PLA                 ; RESTORE X
02AC AA                  TAX
02AD 68                  PLA                 ; RESTORE CHARACTER CODE IN A
02AE 60                  RTS                 ; RETURN
               ;         WAIT FOR ONE MILLISECOND ROUTINE
02AF A9C8      WA1MS:    LDA    #200         ; WAIT FOR APPROXIMATELY 1 MILLISECOND
02B1 E901      WA1MS1:   SBC    #1
02B3 D0FC                BNE    WA1MS1
02B5 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
               ;         RETURNS WITH CARRY FLAG ON IF NOT PRESSED, OFF IF PRESSED
02B6 C950      KEYTST:   CMP    #80          ; TEST IF LEGAL KEY ADDRESS
02B8 9005                BCC    KEYTS1       ; SKIP AHEAD IF SO
02BA A900                LDA    #0           ; SET TO ZERO OTHERWISE
02BC 8DE103              STA    ANKBT1       ; UPDATE ANKBT1
02BF 48        KEYTS1:   PHA                 ; SAVE A ON STACK
02C0 8A                  TXA                 ; SAVE X ON STACK
02C1 48                  PHA
02C2 AD0217              LDA    USRPB        ; CLEAR USER PORT B BITS 2-5
02C5 29C3                AND    #X'C3
02C7 8D0217              STA    USRPB
02CA BA                  TSX                 ; RESTORE KEY ADDRESS FROM STACK
02CB BD0201              LDA    X'102,X
02CE 290F                AND    #X'0F        ; ISOLATE LOW 4 BITS OF KEY ADDRESS
02D0 0A                  ASLA                ; POSITION TO LINE UP WITH BITS 2-5
02D1 0A                  ASLA
02D2 0D0217              ORA    USRPB        ; SEND TO USER PORT B WITHOUT DISTURBING
02D5 8D0217              STA    USRPB        ; OTHER BITS
02D8 BD0201              LDA    X'102,X      ; GET KEY ADDRESS BACK
02DB 4A                  LSRA                ; RIGHT JUSTIFY HIGH 3 BITS
02DC 4A                  LSRA
02DD 4A                  LSRA
02DE 4A                  LSRA
02DF AA                  TAX                 ; USE AS AN INDEX INTO MASK TABLE
02E0 AD4017              LDA    SYSPA        ; GET SYSTEM PORT A STATUS
02E3 3DED02              AND    MSKTAB,X     ; SELECT BIT TO TEST AND SET CARRY FLAG
02E6 18                  CLC                 ; ACCORDINGLY
02E7 E900                SBC    #0
02E9 68                  PLA                 ; RESTORE X FROM STACK
```

```
02EA AA                         TAX
02EB 68                         PLA                  ; RESTORE A FROM STACK
02EC 60                         RTS                  ; RETURN
02ED 01020408  MSKTAB:  .BYTE  X'01,X'02,X'04,X'08    ; MASK TABLE FOR KEYTST
02F1 10                 .BYTE  X'10
               ;        TEST FOR CONTROL/C ROUTINE
               ;        RETURNS WITH CARRY SET IF CONTROL AND C KEYS DOWN, RETURNS
               ;        WITH CARRY OFF IF NOT
               ;        ALSO TESTS IF CONTROL AND O KEY STRUCK, IF SO TOGGLES THE
               ;        CONTROL/O FLAG IN BASIC
               ;        ALSO TEST IF CONTROL AND S KEYS DOWN, IF SO WAITS UNTIL
               ;        CONTROL AND Q KEYS ARE DOWN AND RETURNS
               ;        PRESERVES BOTH INDEX REGISTERS
02F2 AD4117    CNTLC:   LDA    SYSPAD        ; SET UP DATA DIRECTION REGISTERS
02F5 29E0               AND    #X'E0         ; SET SYSTEM PORT A BITS 4-0 TO INPUT
02F7 8D4117             STA    SYSPAD
02FA AD0317             LDA    USRPBD
02FD 093C               ORA    #X'3C         ; SET USER PORT B BITS 5-2 TO OUTPUT
02FF 8D0317             STA    USRPBD
0302 A92E               LDA    #X'2E         ; TEST STATE OF CONTROL KEY
0304 20B602             JSR    KEYTST
0307 B034               BCS    CTLCNO        ; GO TO "NO" RETURN IF NOT PRESSED
0309 A93B               LDA    #X'3B         ; TEST STATE IF "C" KEY
030B 20B602             JSR    KEYTST
030E 902F               BCC    CTLCYS        ; GO TO "YES" RETURN IF PRESSED
0310 A915               LDA    #X'15         ; TEST STATE OF "O" KEY
0312 20B602             JSR    KEYTST
0315 9016               BCC    CTLODN        ; BRANCH IF IT IS DOWN
0317 A914               LDA    #X'14         ; SET ANKBT1 OFF OF O CODE IF NOT SEEN
0319 8DE103             STA    ANKBT1
031C A92C               LDA    #X'2C         ; TEST IF S KEY IS DOWN (CNTL/S = XOFF)
031E 20B602             JSR    KEYTST
0321 B01A               BCS    CTLCNO        ; GO TO CONTROL C FAIL IF NOT
                                             ; IF CNTL S IS SEEN, HANG IN A LOOP UNTIL
                                             ; CONTROL Q IS SEEN  (CNTL/Q = XON)
0323 A91D      CTLA:    LDA    #X'1D         ; TEST "Q" KEY
0325 20B602             JSR    KEYTST
0328 B0F9               BCS    CTLA          ; LOOP UNTIL IT IS SEEN
032A 38                 SEC                  ; WHEN SEEN, EXIT TO CONTROL C FAILURE
032B B010               BCS    CTLCNO        ; WITH CARRY FLAG ON
032D A915      CTLODN:  LDA    #X'15         ; CONTROL O IS DOWN, TEST IF IT WAS DOWN
032F CDE103             CMP    ANKBT1        ; PREVIOUSLY
0332 F009               BEQ    CTLCNO        ; DO NOTHING IF DOWN PREVIOUSLY, GO TO
                                             ; CONTROL C FAIL RETURN
0334 8DE103             STA    ANKBT1        ; SET ANKBT1 TO O CODE
0337 A514               LDA    X'14          ; FLIP OUTPUT CONTROL FLAG WHEN CONTROL O
0339 49FF               EOR    #X'FF         ; IS PRESSED
033B 8514               STA    X'14          ; AND EXECUTE CONTROL C FAIL
033D 18        CTLCNO:  CLC                  ; "NO" RETURN, CLEAR CARRY
033E 60                 RTS                  ; RETURN
```

MAINPR
KIM-1 ALPHANUMERIC KEYBOARD SCAN AND ENCODE ROUTINE

```
033F 38        CTLCYS:  SEC                        ; "YES" RETURN, SET CARRY
0340 60                 RTS                        ; RETURN

               ;        ASCII CHARACTER CODE TRANSLATE TABLE



                                                   ; UNSHIFTED SECTION
0341 5F5E3A2D  ANKBTB:  .BYTE  X'5F,X'5E,X'3A,X'2D  ; BS  CARRET  :  -
0345 30393837           .BYTE  X'30,X'39,X'38,X'37  ; 0  9  8  7
0349 36353433           .BYTE  X'36,X'35,X'34,X'33  ; 6  5  4  3
034D 32311BA0           .BYTE  X'32,X'31,X'1B,X'A0  ; 2  1  ESC  (AUX H)
0351 7F0A5C5B           .BYTE  X'7F,X'0A,X'5C,X'5B  ; DEL  LF  BACKSLASH
0355 706F6975           .BYTE  X'70,X'6F,X'69,X'75  ; P  O  I  U
0359 79747265           .BYTE  X'79,X'74,X'72,X'65  ; Y  T  R  E
035D 777109A1           .BYTE  X'77,X'71,X'09,X'A1  ; W  Q  HT  (AUX L)
0361 060D5D40           .BYTE  X'06,X'0D,X'5D,X'40  ; HEREIS  CR      @
0365 3B6C6B6A           .BYTE  X'3B,X'6C,X'6B,X'6A  ; ;  L  K  J
0369 68676664           .BYTE  X'68,X'67,X'66,X'64  ; H  G  F  D
036D 736100A2           .BYTE  X'73,X'61,X'00,X'A2  ; S  A  CTL  (AUX SHIFT)
0371 00002000           .BYTE  X'00,X'00,X'20,X'00  ; (RIGHT BLANK)  REPAT  SP  LOCK
0375 2F2E2C6D           .BYTE  X'2F,X'2E,X'2C,X'6D  ; /  .  ,  M
0379 6E627663           .BYTE  X'6E,X'62,X'76,X'63  ; N  B  V  C
037D 787A0000           .BYTE  X'78,X'7A,X'00,X'00  ; X  Z  (LEFT BLANK)  SHIFT
0381 80818283           .BYTE  X'80,X'81,X'82,X'83  ; (AUX 0  1  2  3)
0385 84858687           .BYTE  X'84,X'85,X'86,X'87  ; (AUX 4  5  6  7)
0389 88898A8B           .BYTE  X'88,X'89,X'8A,X'8B  ; (AUX 8  9  A  B)
038D 8C8D8E8F           .BYTE  X'8C,X'8D,X'8E,X'8F  ; (AUX C  D  E  F)

                                                   ; SHIFTED SECTION

0391 5F7E2A3D           .BYTE  X'5F,X'7E,X'2A,X'3D  ; BS  TILDA  *  =
0395 30292827           .BYTE  X'30,X'29,X'28,X'27  ; 0  )  (  '
0399 26252423           .BYTE  X'26,X'25,X'24,X'23  ; &  %  $  #
039D 22211BA3           .BYTE  X'22,X'21,X'1B,X'A3  ; "  !  ESC  (AUX H)
03A1 7F0A7C7B           .BYTE  X'7F,X'0A,X'7C,X'7B  ; DEL  LF  VERTBAR
03A5 504F4955           .BYTE  X'50,X'4F,X'49,X'55  ; P  O  I  U
03A9 59545245           .BYTE  X'59,X'54,X'52,X'45  ; Y  T  R  E
03AD 575109A4           .BYTE  X'57,X'51,X'09,X'A4  ; W  Q  HT  (AUX L)
03B1 060D7D60           .BYTE  X'06,X'0D,X'7D,X'60  ; HEREIS  CR      GRAVEACCENT
03B5 2B4C4B4A           .BYTE  X'2B,X'4C,X'4B,X'4A  ; +  L  K  J
03B9 48474644           .BYTE  X'48,X'47,X'46,X'44  ; H  G  F  D
03BD 534100A5           .BYTE  X'53,X'41,X'00,X'A5  ; S  A  CTL  (AUX SHIFT)
03C1 5F002000           .BYTE  X'5F,X'00,X'20,X'00  ; (RIGHT BLANK)  REPAT  SP  LOCK
03C5 3F3E3C4D           .BYTE  X'3F,X'3E,X'3C,X'4D  ; ?  ½  ¼  M
03C9 4E425643           .BYTE  X'4E,X'42,X'56,X'43  ; N  B  V  C
03CD 585A0000           .BYTE  X'58,X'5A,X'00,X'00  ; X  Z  (LEFT BLANK)  SHIFT
03D1 90919293           .BYTE  X'90,X'91,X'92,X'93  ; (AUX 0  1  2  3)
03D5 94959697           .BYTE  X'94,X'95,X'96,X'97  ; (AUX 4  5  6  7)
03D9 98999A9B           .BYTE  X'98,X'99,X'9A,X'9B  ; (AUX 8  9  A  B)
03DD 9C9D9E9F           .BYTE  X'9C,X'9D,X'9E,X'9F  ; (AUX C  D  E  F)

03E1 00        ANKBT1:  .BYTE  0                    ; STORAGE OF CURRENTLY SCANNED
                                                   ; KEY NUMBER

0000                    .END
NO ERROR LINES
```

```
                                .PAGE  'KIM-1 PARALLEL ASCII KEYBOARD ROUTINE'
                       ;        *****MODIFIED FOR KIM BASIC*****
                       ;        THIS SUBROUTINE WAITS FOR A KEY TO BE PRESSED ON A PARALLEL
                       ;        KEYBOARD CONNECTED TO PORT A ON THE KIM-1 APPLICATION
                       ;        CONNECTOR.  IT RETURNS WITH THE ASCII CODE IN THE ACCUMULAT
                       ;        WHEN A KEY IS PRESSED.
                       ;
                       ;        THE KEYBOARD IS ASSUMED TO PRESENT 7 BIT ASCII TO PORT A BI
                       ;        0 (LSB) THROUGH 6 (MSB).  THE STROBE MAY BE EITHER A "KEY
                       ;        PRESSED" LEVEL STROBE OR A PULSED STROBE.
                       ;        PROPER OPERATION OF THE CONTROL/C ROUTINE HOWEVER REQUIRES
                       ;        DATA LATCH IN THE KEYBOARD FOR PROPER OPERATION WITH A PULS
                       ;        STROBE.
                       ;
                       ;        A "CAPS LOCK" FEATURE HAS BEEN INCLUDED.  IF CNTL/R IS PRES
                       ;        CAPS LOCK WILL BE TURNED ON.  IF CNTL/T IS PRESSED CAPS LOC
                       ;        WILL BE TURNED OFF.  WHEN CAPS LOCK IS ON, ALL LOWER CASE
                       ;        LETTERS ARE TRANSLATED TO UPPER CASE; THE NUMBERS AND SPECI
                       ;        CHARACTERS ARE UNAFFECTED.
                       ;
                       ;        TRUE DATA AND POSITIVE-GOING STROBE ARE ASSUMED.  HOWEVER B
                       ;        CHANGING THE DATA/STROBE INVERSION MASK AT MASK, ANY
                       ;        COMBINATION OF TRUE/FALSE POSITIVE/NEGATIVE CAN BE ACCOMODA
                       ;
                       ;        IF THE TTY/KEYBOARD MODE SWITCH ON THE KIM IS IN TTY POSITI
                       ;        INPUT IS TAKEN FROM THE TELETYPE PORT USING KIM'S TTY INPUT
                       ;        ROUTINE.  HOWEVER CNTL/C, CNTL/O, CNTL/S, AND CNTL/Q ARE ST
                       ;        ACTIVATED FROM THE PARALLEL KEYBOARD.


 1700                  USRPA    =      X'1700       ; USER PORT A DATA REGISTER
 1701                  USRPAD   =      X'1701       ; USER PORT A DIRECTION REGISTER

 0000                           .=     X'0200       ; PUT INTO KIM RAM UNUSED BY BASIC

 0200 4C0602                    JMP    ANKB         ; DISPATCH VECTOR  KEYBOARD ROUTINE
 0203 4C6D02                    JMP    CNTLC        ; DISPATCH VECTOR  CONTROL/C ROUTINE

 0206 98               ANKB:    TYA                 ; SAVE THE INDEX REGISTERS
 0207 48                        PHA
 0208 8A                        TXA
 0209 48                        PHA
 020A A901                      LDA    #1           ; TEST KIM TTY/KEYB SWITCH
 020C 2C4017                    BIT    X'1740
 020F D006                      BNE    ANKB0        ; CONTINUE WITH KEYBOARD IF IN KEYB POS
 0211 205A1E                    JSR    X'1E5A       ; GET A TTY CHARACTER IF IN TTY POSITIO
 0214 4C2F02                    JMP    ANKB3        ; GO ECHO IT AND RETURN
 0217 A900             ANKB0:   LDA    #0           ; SET UP DATA DIRECTION REGISTERS
 0219 8D0117                    STA    USRPAD       ; SET USER PORT A FOR INPUT
 021C AD0017           ANKB1:   LDA    USRPA        ; TEST STATUS OF KEY PRESSED BIT
 021F 4DBA02                    EOR    MASK         ; PERFORM SELECTIVE BIT INVERSION
 0222 30F8                      BMI    ANKB1        ; IF PRESSED, WAIT UNTIL RELEASED
 0224 AD0017           ANKB2:   LDA    USRPA        ; WHEN RELEASED, WAIT UNTIL PRESSED AGA
 0227 4DBA02                    EOR    MASK
 022A 10F8                      BPL    ANKB2
```

```
022C 8DBC02           STA   ANKBT1      ; SET LAST STATE OF STROBE
022F 297F    ANKB3:   AND   #X'7F       ; CLEAR OUT STROBE BIT AND
0231 48               PHA               ; SAVE CHARACTER CODE
0232 C90F             CMP   #X'0F       ; TEST IF THE CODE IS CNTL/O
0234 D006             BNE   ANKB4       ; SKIP IF NOT
0236 A514             LDA   X'14        ; TOGGLE OUTPUT ENABLE BIT IN BASIC
0238 49FF             EOR   #X'FF
023A 8514             STA   X'14
023C C912    ANKB4:   CMP   #X'12       ; TEST IF CNTL/R
023E D007             BNE   ANKB5       ; SKIP IF NOT
0240 A9FF             LDA   #X'FF       ; SET CAPS LOCK FLAG IF SO
0242 8DBB02           STA   CAPSLK
0245 D009             BNE   ANKB6
0247 C914    ANKB5:   CMP   #X'14       ; TEST IF CNTL/T
0249 D005             BNE   ANKB6       ; SKIP IF NOT
024B A900             LDA   #0          ; CLEAR CAPS LOCK FLAG IF SO
024D 8DBB02           STA   CAPSLK
0250 ADBB02  ANKB6:   LDA   CAPSLK      ; TEST STATE OF CAPS LOCK FLAG
0253 F00C             BEQ   ANKB11      ; DO NOTHING IF OFF
0255 68               PLA               ; IF ON, TEST IF CODE IS A LOWER CASE
0256 C961             CMP   #X'61       ; LETTER
0258 9006             BCC   ANKB10      ; JUMP IF NOT
025A C97B             CMP   #X'7B
025C B002             BCS   ANKB10      ; JUMP IF NOT
025E 29DF             AND   #X'DF       ; TURN OFF BIT 5 IF A LOWER CASE LETTER
0260 48      ANKB10:  PHA
0261 68      ANKB11:  PLA               ; RESTORE A
0262 BA               TSX
0263 BC0201           LDY   X'102,X     ; RESTORE Y FROM STACK
0266 9D0201           STA   X'102,X     ; SAVE CHARACTER CODE IN STACK WHERE Y WAS
0269 68               PLA               ; RESTORE X
026A AA               TAX
026B 68               PLA               ; RESTORE CHARACTER CODE IN A
026C 60               RTS               ; RETURN
```

```
                                .PAGE  'TEST FOR CONTROL/C ROUTINE'
                        ;       TEST FOR CONTROL/C ROUTINE
                        ;       RETURNS WITH CARRY SET IF CONTROL AND C KEYS DOWN, RETURNS
                        ;       WITH CARRY OFF IF NOT
                        ;       ALSO TESTS IF CONTROL AND O KEY STRUCK, IF SO TOGGLES THE
                        ;       CONTROL/O FLAG IN BASIC
                        ;       ALSO TEST IF CONTROL AND S KEYS DOWN, IF SO WAITS UNTIL
                        ;       CONTROL AND Q KEYS ARE DOWN AND RETURNS
                        ;       PRESERVES BOTH INDEX REGISTERS

 026D A900      CNTLC:  LDA     #0              ; SET UP DATA DIRECTION REGISTER
 026F 8D0117            STA     USRPAD          ; FOR INPUT
 0272 AD0017            LDA     USRPA           ; LOOK AT KEYBOARD DATA IRREGARDLESS O
 0275 4DBA02            EOR     MASK
 0278 297F              AND     #X'7F           ; STATE OF STROBE
 027A C903              CMP     #X'03           ; TEST IF CNTL/C
 027C F034              BEQ     CTLCYS          ; GO TO CNTL/C SUCCESS IF SO
 027E C913              CMP     #X'13           ; TEST IF CNTL/S
 0280 D00E              BNE     CNTLC2          ; JUMP AHEAD IF NOT
 0282 AD0017    CNTLC1: LDA     USRPA           ; IF SO, WAIT UNTIL CNTL/Q IS SEEN
 0285 4DBA02            EOR     MASK
 0288 297F              AND     #X'7F
 028A C911              CMP     #X'11
 028C D0F4              BNE     CNTLC1
 028E F017              BEQ     CTLCNO          ; GO TO CNTL/C FAILURE RETURN
 0290 C90F      CNTLC2: CMP     #X'0F           ; TEST IF CNTL/O
 0292 D013              BNE     CTLCNO          ; GO TO CNTL/C FAILURE RETURN IF NOT
 0294 AD0017            LDA     USRPA           ; COMPARE LAST STATE OF STROBE WITH
 0297 4DBA02            EOR     MASK            ; CURRENT STATE OF STROBE
 029A 4980              EOR     #X'80
 029C 0DBC02            ORA     ANKBT1          ; IF PREVIOUSLY OFF AND NOW ON
 029F 3006              BMI     CTLCNO          ; FLIP THE SUPPRESS OUTPUT FLAG IN BAS
                                                ; OTHERWISE EXECUTE A CTL/C FAILURE RE
 02A1 A514              LDA     X'14            ; FLIP OUTPUT CONTROL FLAG WHEN CONTRC
 02A3 49FF              EOR     #X'FF           ; IS PRESSED
 02A5 8514              STA     X'14            ; AND EXECUTE CONTROL C FAIL
 02A7 AD0017    CTLCNO: LDA     USRPA           ; "NO" RETURN, UPDATE LAST STATE OF ST
 02AA 4DBA02            EOR     MASK
 02AD 8DBC02            STA     ANKBT1
 02B0 18                CLC                     ; CLEAR CARRY
 02B1 60                RTS                     ; RETURN
 02B2 AD0017    CTLCYS: LDA     USRPA           ; "YES" RETURN, UPDATE LAST STATE OF
 02B5 8DBC02            STA     ANKBT1          ; STROBE
 02B8 38                SEC                     ; SET CARRY
 02B9 60                RTS                     ; RETURN

 02BA 00        MASK:   .BYTE   0               ; MASK FOR TRUE DATA AND POSITIVE STRC
 02BB 00        CAPSLK: .BYTE   0               ; CAPS LOCK FLAG, ON IF NON-ZERO
 02BC 00        ANKBT1: .BYTE   0               ; STORAGE FOR CURRENT STATE OF STROBE

 0000                   .END
NO ERROR LINES
```

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 key-switch 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

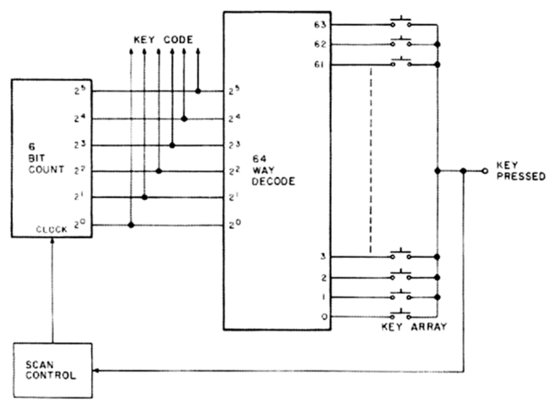


Fig. 1. Basic keyboard scanner.

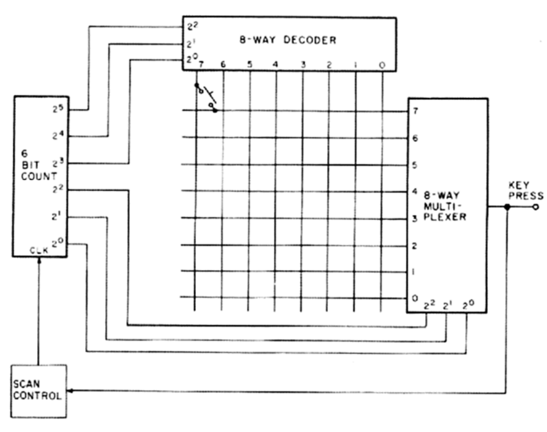
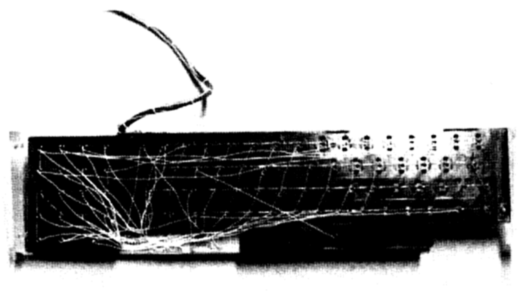


Fig. 2. Matrix keyboard scanner.



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

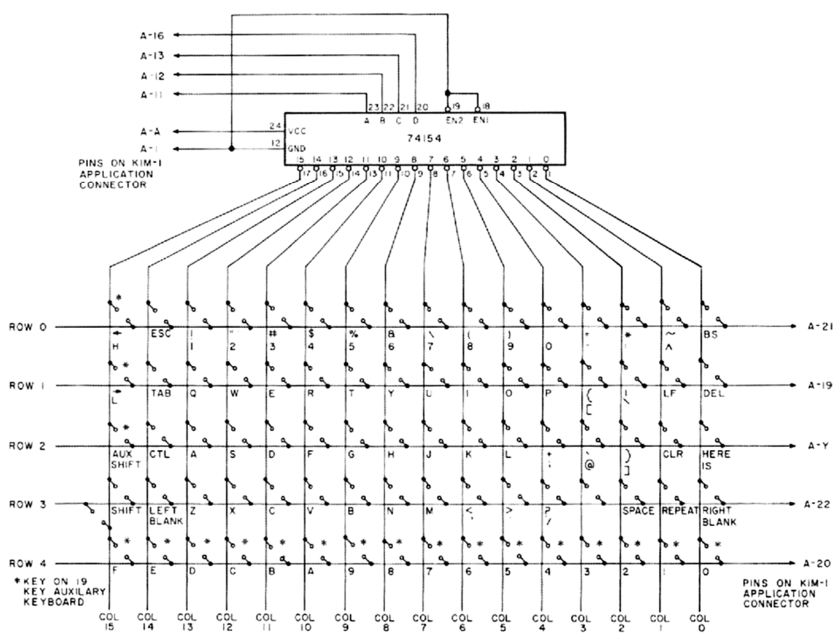


Fig. 3. Complete KIM-1 alphanumeric keyboard interface schematic.

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.

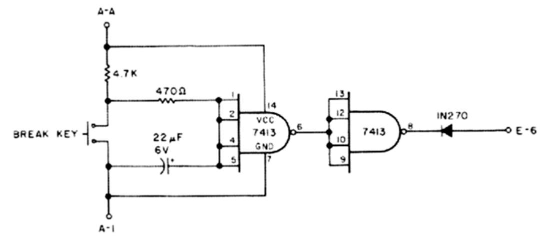


Fig. 4. Optional break-key interface.

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.

```
                        .PAGE  'KIM-1 ALPHANUMERIC KEYBOARD SCAN AND ENCODE ROUTINE'
                ;       THIS SUBROUTINE SCANS AN UNENCODED KEYBOARD MATRIX CONNECTED
                ;       TO THE KIM-1 APPLICATION CONNECTOR.  USER PERIPHERAL PORT B
                ;       BITS 5 (MSB) THROUGH 2 (LSB) ARE CONNECTED TO A ONE-OF-16
                ;       DECODER (74154) WHICH DRIVES THE KEYSWITCH COLUMNS.
                ;       SENSING OF THE ROWS IS BY A PORTION OF THE KIM ON-BOARD
                ;       KEYBOARD CIRCUITRY WHICH USES SYSTEM PERIPHERAL PORT B BITS
                ;       0 - 4.
                ;       WHEN CALLED, THE ROUTINE SITS IN A LOOP WAITING FOR A KEY TO
                ;       BE PRESSED.  WHEN A KEY IS PRESSED (EXCEPTING SHIFT, CONTROL,
                ;       REPEAT), THE ROUTINE RETURNS WITH KEY CODE IN ACCUMULATOR.
                ;       BOTH INDEX REGISTERS ARE RETAINED.
                ;       THE ROUTINE IMPLEMENTS TRUE 2-KEY ROLLOVER, KEY DEBOUNCING,
                ;       AND REPEAT TIMING.  ONE RAM LOCATION IS REQUIRED, ITS INITIAL
                ;       CONTENT IS INSIGNIFICANT.

0000                    .=      X'200       ; START PROGRAM AT LOCATION 0200 (HEX)

1740        SYSPA       =       X'1740      ; SYSTEM PORT A DATA REGISTER
```

```
1741           SYSPAD    =       X'1741          ; SYSTEM PORT A DIRECTION REGISTER
1702           USRPB     =       X'1702          ; USER PORT B DATA REGISTER
1703           USRPBD    =       X'1703          ; USER PORT B DIRECTION REGISTER
0032           RPTRAT    =       50              ; REPEAT PERIOD, MILLISECONDS
0005           DBCDLA    =       5               ; DEBOUNCE DELAY, MILLISECONDS
00EE           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.■

```
0282 C950      KEYTST:  CMP    #80                ; TEST IF LEGAL KEY ADDRESS
0284 9004               BCC    KEYTS1             ; SKIP AHEAD IF SO
0286 A900               LDA    #0                 ; SET TO ZERO OTHERWISE
0288 85EE               STA    ANKBT1             ; UPDATE ANKBT1
028A 48        KEYTS1:  PHA                       ; SAVE A ON STACK
028B 8A                 TXA                       ; SAVE X ON STACK
028C 48                 PHA
028D AD0217             LDA    USRPB              ; CLEAR USER PORT B BITS 2-5
0290 29C3               AND    #X'C3
0292 8D0217             STA    USRPB
0295 BA                 TSX                       ; RESTORE KEY ADDRESS FROM STACK
0296 BD0201             LDA    X'102,X
0299 290F               AND    #X'0F              ; ISOLATE LOW 4 BITS OF KEY ADDRESS
029B 0A                 ASLA                      ; POSITION TO LINE UP WITH BITS 2-5
029C 0A                 ASLA
029D 0D0217             ORA    USRPB              ; SEND TO USER PORT B WITHOUT DISTURBING
02A0 8D0217             STA    USRPB              ; OTHER BITS
02A3 BD0201             LDA    X'102,X            ; GET KEY ADDRESS BACK
02A6 4A                 LSRA                      ; RIGHT JUSTIFY HIGH 3 BITS
02A7 4A                 LSRA
02A8 4A                 LSRA
02A9 4A                 LSRA
02AA AA                 TAX                       ; USE AS AN INDEX INTO MASK TABLE
02AB AD4017             LDA    SYSPA              ; GET SYSTEM PORT A STATUS
02AE 3DB802             AND    MSKTAB,X           ; SELECT BIT TO TEST AND SET CARRY FLAG
02B1 18                 CLC                       ; ACCORDINGLY
02B2 E900               SBC    #0
02B4 68                 PLA                       ; RESTORE X FROM STACK
02B5 AA                 TAX
02B6 68                 PLA                       ; RESTORE A FROM STACK
02B7 60                 RTS                       ; RETURN

02B8 01020408  MSKTAB:  .BYTE  X'01,X'02,X'04,X'08    ; MASK TABLE FOR KEYTST
02BC 10                 .BYTE  X'10

               ;        ASCII CHARACTER CODE TRANSLATE TABLE

                                                      ; UNSHIFTED SECTION
02BD 085E3A2D  ANKBTB:  .BYTE  X'08,X'5E,X'3A,X'2D    ; BS  CARRET  :  -
02C1 30393837           .BYTE  X'30,X'39,X'38,X'37    ; 0  9  8  7
02C5 36353433           .BYTE  X'36,X'35,X'34,X'33    ; 6  5  4  3
02C9 32311BA0           .BYTE  X'32,X'31,X'1B,X'A0    ; 2  1  ESC  (AUX H)
02CD 7F0A5C5B           .BYTE  X'7F,X'0A,X'5C,X'5B    ; DEL  LF  BACKSLASH  [
02D1 706F6975           .BYTE  X'70,X'6F,X'69,X'75    ; P  O  I  U
02D5 79747265           .BYTE  X'79,X'74,X'72,X'65    ; Y  T  R  E
02D9 777109A1           .BYTE  X'77,X'71,X'09,X'A1    ; W  Q  HT  (AUX L)
02DD 060D5D40           .BYTE  X'06,X'0D,X'5D,X'40    ; HEREIS  CR  ]  @
02E1 3B6C6B6A           .BYTE  X'3B,X'6C,X'6B,X'6A    ; ;  L  K  J
02E5 68676664           .BYTE  X'68,X'67,X'66,X'64    ; H  G  F  D
02E9 736100A2           .BYTE  X'73,X'61,X'00,X'A2    ; S  A  CTL  (AUX SHIFT)
02ED 00002000           .BYTE  X'00,X'00,X'20,X'00    ; (RIGHT BLANK)  REPAT  SP
02F1 2F2E2C6D           .BYTE  X'2F,X'2E,X'2C,X'6D    ; /  .  ,  M
02F5 6E627663           .BYTE  X'6E,X'62,X'76,X'63    ; N  B  V  C
02F9 787A0000           .BYTE  X'78,X'7A,X'00,X'00    ; X  Z  (LEFT BLANK)  SHIFT
02FD 80818283           .BYTE  X'80,X'81,X'82,X'83    ; (AUX 0  1  2  3)
0301 84858687           .BYTE  X'84,X'85,X'86,X'87    ; (AUX 4  5  6  7)
0305 88898A8B           .BYTE  X'88,X'89,X'8A,X'8B    ; (AUX 8  9  A  B)
0309 8C8D8E8F           .BYTE  X'8C,X'8D,X'8E,X'8F    ; (AUX C  D  E  F)

                                                      ; SHIFTED SECTION

030D 085E2A3D           .BYTE  X'08,X'5E,X'2A,X'3D    ; BS  CARRET  *  =
0311 30292827           .BYTE  X'30,X'29,X'28,X'27    ; 0  )  (  '
0315 26252423           .BYTE  X'26,X'25,X'24,X'23    ; &  %  $  #
0319 22211BA3           .BYTE  X'22,X'21,X'1B,X'A3    ; "  !  ESC  (AUX H)
031D 7F0A7C7B           .BYTE  X'7F,X'0A,X'7C,X'7B    ; DEL  LF  VERTBAR  {
0321 504F4955           .BYTE  X'50,X'4F,X'49,X'55    ; P  O  I  U
0325 59545245           .BYTE  X'59,X'54,X'52,X'45    ; Y  T  R  E
0329 575109A4           .BYTE  X'57,X'51,X'09,X'A4    ; W  Q  HT  (AUX L)
032D 060D7D60           .BYTE  X'06,X'0D,X'7D,X'60    ; HEREIS  CR  }  GRAVEACCENT
0331 2B4C4B4A           .BYTE  X'2B,X'4C,X'4B,X'4A    ; +  L  K  J
0335 48474644           .BYTE  X'48,X'47,X'46,X'44    ; H  G  F  D
0339 534100A5           .BYTE  X'53,X'41,X'00,X'A5    ; S  A  CTL  (AUX SHIFT)
033D 00002000           .BYTE  X'00,X'00,X'20,X'00    ; (RIGHT BLANK)  REPAT  SP
0341 3F3E3C4D           .BYTE  X'3F,X'3E,X'3C,X'4D    ; ?  >  <  M
0345 4E425643           .BYTE  X'4E,X'42,X'56,X'43    ; N  B  V  C
0349 585A0000           .BYTE  X'58,X'5A,X'00,X'00    ; X  Z  (LEFT BLANK)  SHIFT
034D 90919293           .BYTE  X'90,X'91,X'92,X'93    ; (AUX 0  1  2  3)
0351 94959697           .BYTE  X'94,X'95,X'96,X'97    ; (AUX 4  5  6  7)
0355 98999A9B           .BYTE  X'98,X'99,X'9A,X'9B    ; (AUX 8  9  A  B)
0359 9C9D9E9F           .BYTE  X'9C,X'9D,X'9E,X'9F    ; (AUX C  D  E  F)

0000                    .END
NO ERROR LINES
```