

MICRO TECHNOLOGY UNLIMITED
GRAPHICS SOFTWARE PACKAGE FOR THE K-1008 VISIBLE MEMORY

The graphics software package for the K-1008 Visible Memory is designed to provide the user with a library of basic graphics oriented subroutines. By incorporating calls to these routines, the user can create and manipulate text and graphic images whose complexity is limited only by the 320 by 200 display matrix size. The graphics and text display subroutines are available only as printed, assembled, and commented program listings since the user is expected to assemble them into his own application programs.

In addition, two self-contained demonstration programs are included. Both of these will run on the bare KIM with no extra hardware other than the K-1008 Visible Memory and video monitor. In many cases, the demonstration programs contain simplified versions of the graphics subroutine package having only enough capability to satisfy the needs of the demonstration. Printed listings of the demo programs are normally included with the graphics software package. The demo programs are also available on a standard KIM cassette for \$5.00.

INCLUSIONS

In this package you should find the following:

1. Printed, assembled, and commented program listings of
 - A. SWIRL demonstration program
 - B. LIFE demonstration program
 - C. SDXTX Simplified text display subroutine, 22 lines 53 char.
 - D. Comprehensive graphics subroutine library containing point and line plotting routines, a character drawing routine, and an ASCII text display routine.
2. Instruction manual which you are now reading
3. Copyright notice

In addition, a standard speed KIM format cassette may be supplied if it was specifically ordered (available only to purchasers of the entire software package for \$5.00). The cassette contains:

1. File 01 (recorded twice) SWIRL demonstration program.
Loads into locations 0000 - 03EC
2. File 02 (recorded twice) LIFE demonstration program.
Loads into locations 0000 - 3FB
3. File 03 (recorded twice) Continuation of LIFE program.
Loads into locations 1780 - 17DC

Note that the demonstration programs assume that the VM occupies addresses from 2000-3FFF. If your system is configured differently, put the first VM page number in 000B for SWIRL and 0000 for LIFE.

A separate package will be available shortly for linking MicroSoft BASIC for the KIM with the text and graphics routines. Using this patch package, the user may utilize the Visible Memory for normal textual communications with BASIC (along with an external keyboard) and for graphic output. Repetitive graphic calculations are handled by the package in machine language thus insuring maximum overall speed.

(1)

RUNNING THE DEMONSTRATION PROGRAMS

I. SWIRL

Swirl is a demonstration program that generates a variety of interesting spiral and spiderweb like patterns on the screen. Two parameters determine the appearance of the pattern and a third either includes or suppresses lines connecting the computed points. The user may set these parameters manually and then have a single pattern computed and held or another routine may be invoked which uses a random number generator to select the parameters thus giving an endless series of different patterns.

The program is based on the differential equation for a circle which tends toward an ellipse when evaluated digitally at a point at a time. As the calculation proceeds, the radius of the circle decreases until it is essentially zero. Since the calculation is point by point, the visual effect on the display can be considerably different from a simple inward spiral.

One may also think of the algorithm as a digital damped sine wave generator or ultimately a digital bandpass filter. The algorithm works on two variables, SIN and COS, which relate to the sine and cosine of an angle. Basically, the program takes the current values of SIN and COS and computes new values of both under the control of two constants. Each time a new SIN,COS pair is computed, it is treated as an X,Y pair and plotted on the Visible Memory screen. Straight lines may or may not connect successive points; both give distinctive patterns.

Two constants control the program, FREQ and DAMP which, of course, relate to the damped sine wave nature of the algorithm. FREQ is a double precision, signed binary fraction. The larger its value, the fewer points per revolution of the circle and therefore the higher the frequency. The relationship between FREQ and points per cycle is roughly linear. A value of +.9999 (7FFF₁₆) gives 6 points per cycle, +.5 (4000₁₆) gives about 12, and so forth. Negative values of FREQ cause the spiral to rotate clockwise rather than counterclockwise. DAMP is also a double precision signed binary fraction but it must be positive for proper operation. If it is negative, the oscillation will build up instead of dying out until the fixed point arithmetic routines overflow creating a garbage display. Normal values of DAMP are very close to 1.0 and the useful range is from approximately 7000 to 7FFF. Smaller values of DAMP produce so few points before the circle collapses to zero that the resulting pattern is diffuse and uninteresting.

To run the program, first load it into KIM memory exactly as it appears in the listing. If the cassette was ordered, load file 01 into memory. If loading was done by hand, check it (goes twice as fast with two people, one calling out the hex and the other reading the listing) and then immediately dump it to cassette. The slightest error in hand loading could cause the program to wipe itself out!

Default values for all of the parameters have been supplied. To see the default pattern, start execution at address 002F (SWIRL). The screen, which was initially semi-random garbage, should be cleared and then a spiderweb-like pattern should be gradually built up over a time span of several seconds. It is complete when the dark area at the center of the screen is completely filled up. The user may return to the KIM monitor with the ST or the reset key at any time even if the pattern is not complete.

(2)

In order to get a feel for the visual effect of the various parameters, first try setting LINES (at address 0000) to 00 and then go to SWIRL again. This time only the vertices of the angled lines that were seen earlier are shown. Although the default FREQ and DAMP parameters were chosen for an appealing display with LINES equal to 1, some very impressive displays indeed are possible with LINES set to 00. For an example, set FREQ to 1102 (0001<02, 0002<11) and DAMP to 7FC0 (0003<C0, 0004<7F) and execute SWIRL again. Interrupt the program execution when the hole in the middle is completely surrounded by a couple of dot depths of solid white. The resulting display, particularly when viewed at a distance in a darkened room, could easily pass for an artist's conception of a Black Hole; an astronomical object which is thought to be matter crushed out of existence by its own gravity!

Returning to the original settings of FREQ, DAMP, and LINES, lets see the effect of changing DAMP. Regenerate the default pattern and fix it in your mind. Then change DAMP from 7E00 to 7F00. This has the effect of cutting the decay rate of the damped sine wave in half. The visual effect is a denser display that decays toward the center more slowly. DAMP may be further increased to 7F80, 7FC0, etc. (set 0006 to 70 to avoid overflow). As DAMP approaches 7FFF, the density of the image becomes so great that the pattern becomes essentially solid white and takes a long time to complete. Conversely, as DAMP is reduced to 7C00, 7800, 7000, etc., the pattern becomes sparser and eventually degrades into an angular spiral. Try some of these values of DAMP with LINES set to zero also.

All of the preceding patterns had very nearly 6 points per revolution of the spiral. The vertices themselves created a spiral pattern as they overlapped and created moire-like effects. Slight changes in FREQ can have a profound effect on the moire aspect of the pattern without a significant effect on the number of points per revolution. Try 7E80, 7F80, and 7FFF for FREQ to see this effect. Many more points per revolution are possible by reducing FREQ. Reduction to 4000, 2000, 1000, and even lower will cause the vertices to become so closely spaced that the effect of a continuous curve (within the resolution constraint of the display) is created. Also note that decreasing FREQ apparently increases the damping causing the spiral to decay after fewer revolutions than before. This effect may be countered by increasing DAMP. For example, if FREQ was reduced in half from, say, 3000 to 1800, then the difference between DAMP and 7FFF should also be reduced in half, say from 7D00 to 7E80. The lower values of FREQ are particularly effective with LINES set to zero. If FREQ is low enough, there will be no visual difference between LINES=1 and LINES=0.

Some combinations of FREQ and DAMP can cause the arithmetic to overflow, that is, SIN or COS may try to reach or exceed 1.0 in magnitude. There is no danger of such an occurrence damaging the program or wiping out memory but the resulting pattern on the screen can be very random looking. Simultaneous high values of FREQ and DAMP will cause the overflow situation. Reducing COSINT to 7000 will prevent the possibility of overflow but will also reduce the image size somewhat. If FREQ is kept less than 4000 or so, COSINT may be increased to 7E00 for a somewhat larger pattern.

Entry into RSWIRL (address 0045) will cause continuous random selection of the parameters and computation of patterns. To insure that the "pattern complete" test functions properly, COSINT should be set to 7000 to prevent the possibility of overflow. The sequence of patterns will not repeat for days!

(3)

II. LIFE

This program is based on the Life cellular automaton algorithm written up in Scientific American magazine several years ago. The basic concept is that of a rectangular array of "cells" that "live" and "die" in discrete time "generations". On the Visible Memory screen, each picture element (pixel or bit position) is a cell location. A live cell is represented as a One bit which shows as a white dot and a dead or missing cell is represented as a Zero which leaves a black area. A generation is the state or configuration of live cells on the screen at a point in time. A set of rules are defined which determines, based on the configuration of live cells in the present generation, which cells live or die in the next generation as well as "births" of new cells where none had existed previously.

The rules of Life are simple. In fact, their very simplicity yet varied and wonderful effect is what makes Life so appealing to many people. The rules are based purely on the eight neighbors (above, below, left of, right of, and the 4 diagonal neighbors) of every cell position. To determine the next generation, the live neighbors of every cell position in the life field are counted. Based on this count and the current state of the central cell, the fate of the central cell is determined. The rules are as follows:

- A. Central cell is alive
 - 1. 0 or 1 live neighbors, the central cell dies of starvation
 - 2. 2 or 3 live neighbors, the central cell lives on
 - 3. 4 or more live neighbors, the central cell dies of overcrowding
- B. Central cell is not alive
 - 1. Fewer than or more than 3 live neighbors, the central cell remains dead
 - 2. Exactly 3 live neighbors, a birth is recorded.

When applying these rules to determine the next generation, the present configuration of live cells is always used. Any births or deaths are recorded separately and do not influence events around the birth or death site until the next generation becomes current. When programming Life, this may be accomplished by making a copy of the Life field as the next generation is formed. In a limited memory machine such as the KIM, buffering of lines of cells is needed to simulate a copy of the field.

The resulting sequence of generations is completely determined by the configuration of the initial colony of cells and is called a life history. Such a history may end in one of several ways. The colony may eventually die out completely leaving no cells on the screen at all. This often happens after several generations of spectacular buildup which suddenly shrink and disintegrate after a few more. A colony may also become stable. This happens when each succeeding generation is exactly like the previous one. Cycles of generations are also possible in which a configuration may go through a cycle of two or more differing configurations only to return to the exact same configuration for another cycle. A variation of the cyclic pattern is one which moves across the screen as it cycles. Finally, a pattern may grow without limit. Initially this was thought to be impossible until a pattern that periodically emits cyclic, traveling patterns was discovered.

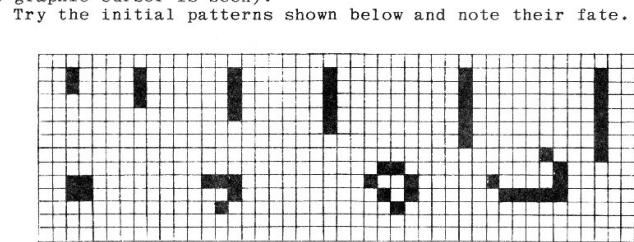
(4)

The Life demonstration program consists of four entry points. INIT (009A) when entered will merely clear the screen and return to the KIM monitor. This is generally necessary before entering a pattern by hand. KYPT (03C7) allows entry of an initial pattern of cells using a graphic cursor and the KIM keypad. Initial patterns may also be entered using the KIM monitor to write directly into the visible memory. Other methods include reading the pattern from cassette tape using the KIM monitor or generating the pattern with another program (such as SWIRL), loading LIFE, and executing it. The entry point LIFE (0100) starts the evolution process. Finally, DEMO will create an appropriate, canned, initial pattern and then execute LIFE to produce an amazingly beautiful life history.

If the reader is not familiar with the Life algorithm and some of the folklore surrounding it, it is instructive to experiment some before executing DEMO (leave it as a surprise!). First load the program from the listing or cassette tape in the same manner as SWIRL. Be sure to load the auxiliary RAM from 1780 to 17DC or KYPT will not function. After loading (and saving on cassette if by hand), execute INIT (009A) to clear the screen. INIT should return to the KIM monitor after the screen is cleared. Next execute KYPT (03C7) (a bug in the program requires that 13 be stored into 0001 before executing KYPT). In the middle of the screen should be a single flashing dot. Note that the dot is off most of the time flashing on for only a short period. This is a signal that the graphic cursor is covering a "dead" cell. Press the + key on the KIM. The flashing should change such that the dot is on most of the time. This signifies that a live cell is being covered. Thus the "+" key is used to set a cell at the current cursor position. Hitting the "F" key will kill the cell under the cursor.

The cursor may be moved horizontally and vertically by hitting the "9" key for up, "1" key for down, "4" for left, and "6" for right. With these movement keys, the + key, and the F key, simple initial patterns may be easily entered or existing patterns may be edited in a limited way. You may notice that the KIM keyboard keys bounce less or none at all using this routine. This is due to a more sophisticated debouncing algorithm than is utilized in the KIM monitor.

Once the desired initial pattern is obtained, the "GO" key may be pressed to start execution of the Life algorithm. Alternatively, KYPT may be interrupted and LIFE may be manually entered at 0100. The succession of generations may be stopped by pressing any keyboard key (except ST or RS) and KYPT will regain control at the conclusion of the current generation (hold the key down until the graphic cursor is seen).



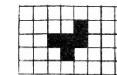
(5)

The patterns that evolve from those on the previous page are fundamental and well known to every Life fan. They are so common in the result of many initial patterns that they have been given descriptive names. See if you can match the following names with the corresponding final patterns: Block, Honeyfarm, Glider, Blinker, Beehive, Lifeboat, Rocketship, Traffic Lights.

Another interesting pastime is to note the life history (number of generations before dying off, becoming stable, or becoming cyclic) of simple lines of dots with 3, 4, ..., 30 ... dots in a line. Sometimes the addition of a single dot in a long string can have a profound effect on the final result. Another possibility is to trace the history of all possible configurations of three live cells, 4 cells, 5 cells, etc. Note that the majority of the possible configurations are redundant because of symmetry, rotation, or mirror images. Also, sparse initial patterns invariably die off in one or two generations because of starvation.

Note that initial patterns should be placed in the center of the screen to allow maximum room for expansion of the colony. If live cells get within one cell width of the matrix boundaries, the next generation is no longer correctly computed. This only applies to the region where the boundary is touched, the remainder of the screen is unaffected.

Finally, before executing DEMO, try the very simple initial pattern below. As it expands and differentiates, it will leave a litter of the fundamental patterns discussed earlier.



To execute DEMO, simply go to 00A1. An initial pattern will be generated and the Life algorithm will be executed on it. When seen, numerous practical applications for Life should present themselves. The initial pattern generated by DEMO may be changed by altering the table of coordinates that starts at LIST (0335). Note that the line drawing routine that connects the endpoints in the list is limited to horizontal, vertical, and 45 degree lines. Other angles are not harmful but will be displayed as a 45 degree segment followed by a 90 degree segment.

(6)

III. USING SDXTXT FOR TEXT DISPLAY ON THE VISIBLE MEMORY

SDXTXT stands for Simplified Display TeXT which is a highly optimized text display subroutine for the Visible Memory graphics display. Within the constraints of structured programming technique and overall programming effort, SDXTXT is optimized for small size and fast execution speed. It is also designed to fit the maximum practical amount of text into the 320 by 200 display matrix without adversely affecting legibility.

Given that the SDXTXT subroutine is resident in memory, either RAM or ROM, it is as easy to generate text on the Visible Memory display as it is with a conventional characters-only display. Note however that SDXTXT and the Visible Memory form an "output only" display device as far as the actual ASCII character codes are concerned. Although bit patterns forming the character shape are readily read from the display memory, the actual ASCII codes cannot be retrieved (unless of course one wishes to write a character recognition program to convert dot patterns to ASCII). Thus an actual text editing application would have to maintain a separate text buffer for the ASCII codes. This is discussed in greater detail later.

The basic display format of SDXTXT is 22 lines of 53 characters per line. Although it would be nice to have a longer line, the majority of low cost character-only displays actually have less capacity than this such as 16 lines of 32 or 40 characters. The characters themselves are formed from a 5 wide by 7 high dot matrix. Lower case characters are represented as small capital letters in a 5 by 5 matrix. Although normal lower case with descenders is readily handled on a graphic display device, additional room must be allowed for the descender thus reducing the number of possible text lines. Lower case shapes without descenders were judged to be more difficult to read than the small caps. The 5 by 7 matrix is positioned in a 6 wide by 9 high "window" to allow space between adjacent characters and lines. Although 25 lines could be displayed if the interline spacing was reduced to one dot, the sacrifice in legibility was judged to be excessive. If the user disagrees with these choices, reassembly of the subroutine with different values (within limits) of CHHI and CHWD and a slight recoding of CSRTAD is sufficient to change them. The character font table is also readily changed to suit individual tastes. If the user wishes to operate in the half screen mode, NLOC should be changed to 4096 and the program reassembled. This will cut the number of lines displayed to 11 but leave the second 4K half of the VM free for other uses.

SDXTXT requires some RAM for parameter and temporary storage. There are three types of storage required. Base page temporary storage **must** be in page zero since the indirect addressing modes require this. Four bytes are required but they need not be preserved between calls to SDXTXT thus they may be used by other programs as well. Four additional bytes of temporary storage may be placed anywhere and also used by other programs. Finally, three bytes are required for the storage of parameters. Since these hold the cursor location and the page number of the VM, they must not be disturbed between calls to SDXTXT unless the user desires to change these parameters. Note that if all RAM storage is kept in page 0 and SDXTXT is reassembled that the program will be a couple dozen bytes shorter and somewhat faster due to the use of page zero addressing rather than absolute addressing when these locations are accessed.

(7)

As given in the program listing, SDXTXT is about 1.2K bytes in length. This may be reduced to just under 1K (for storage in a single 2708 PROM) if the lower case characters are deleted from the font table. The routine is completely ROMable since it does not modify itself but it is not reentrant due to the fixed temporary storage locations. If SDXTXT is placed in ROM, it is suggested that the 4 bytes that must be in the base page be assigned just below the KIM monitor area. It may even be possible use the KIM monitor area itself since the routine is already debugged and therefore need not be single-stepped. Actually, many other programs could make use of these two address pointers as well. The remaining temporary storage may be put anywhere. Although page zero is a desirable location, the 96 invisible bytes at the end of the VM is also a good choice for this and any other programs associated with the display.

It is unlikely that the user will want SDXTXT to reside in the locations it was assembled for, which is the last 1.2K of a 16K expansion starting at 2000. While a full 6502 compatible assembler is best for configuring the program, hand relocation is not difficult. All underlined addresses must be changed if the program itself is relocated. If the temporary storage locations are also moved (quite likely), addresses referencing them will also have to be changed. While not specifically designated in the listing, they are easily spotted simply by noting references to CSRX, CSRY, DCNT1, etc. in the operand field of the instruction.

USING SDXTXT

Using SDXTXT is exceptionally simple. The user merely loads the ASCII character code to be displayed or control code to be interpreted into register A and does a JSR SDXTXT. The subroutine will then display the character at the present cursor location or do the indicated operation and then return with all registers intact. The condition codes will however be altered. SDXTXT expects the decimal mode flag to be OFF.

It cannot be emphasized enough that VMORG must be set to the page number of the first VM location before SDXTXT is used. For example, if the VM is jumpered for addresses 2000-3FFF, then VMORG should be 2016. Failure to set VMORG will change SDXTXT into MEMCLR!

It is also important that CSRX and CSRY have valid contents before any printable characters are sent to SDXTXT. The best way to accomplish this is to give SDXTXT an ASCII FF character (OC) as the very first operation. This action not only initializes the cursor to the top left side, it also clears the screen.

CSRX and CSRY hold the character and line number respectively of the present cursor location. Numbering starts at zero thus the top line is line 0 and the leftmost character is character 0. SDXTXT automatically moves the cursor as appropriate. The user may also move the cursor anywhere at any time by directly changing the values of CSRX and CSRY. Before this is done however, a call to CSRCLR must be executed to clear the existing cursor from the screen. The user then can change the cursor location. Following this, a call to CSRSET will display the cursor at its new position. CSRX must always be between 0 and 52₁₀ and CSRY must be between 0 and 21₁₀ inclusive. Violation of this range restriction is not checked and can cause random storing anywhere in memory.

(8)

In the present implementation, if more characters are received than will fit on a line the cursor simply remains at the rightmost character position on the line rather than forcing an automatic carriage return line feed sequence. This capability is easily added but can lead to problems in interfacing with BASIC unless the terminal width is set to 52 rather than 53. A line feed that runs off the bottom of the screen causes an upward scroll of the text instead with the top line being lost.

Two other useful subroutines are available as part of SDXTXT. FMOVE is an extremely fast memory move subroutine that can move any number of bytes from anywhere in memory to anywhere in memory at an average speed of 16 microseconds per byte. The address of the first source byte should be stored in ADP1 and the first destination address should be stored in ADP2. A double precision move count should be stored in DCNT1. Although A is destroyed, the index registers are preserved. FCLR is similar except that it can quickly clear any amount of memory. Set up the first address to be cleared in ADP2 and a double precision count in DCNT1 and call FCLR. X and Y are preserved but A is destroyed.

LIMITATIONS

Unfortunately, even though a lot of effort was put into making SDXTXT efficient, it takes a finite amount of time to draw a character and move the cursor. For normal applications, such as displaying text typed in or conversing with BASIC, this time will never be noticed. Using the KIM and the VM to simulate a teletype terminal however will most likely uncover limitations in the maximum baud rate that can be handled.

Approximately 2.68 milliseconds are required to draw a character and move the cursor. All control characters except FF and LF when it causes a scroll take even less time. FF takes nearly 100 milliseconds and an LF that scrolls requires about 120 MS. Ignoring these and only considering characters it is easily determined that the absolute maximum baud rate that can be handled is a little more than 3600 baud. This rate can be closely approached if a standard UART is used for the serial communication. If the timed loop (software UART) serial routines in the KIM monitor are used then only the stop bit duration is available for character generation. This would limit the rate to 300 baud with one stop bit or 600 baud with two stop bits.

Even with a UART, simple one-track programming would only allow 110 baud if LF and FF characters are to be received. Many terminal systems do allow one or more nulls to be sent after such control characters which would directly affect the maximum rate possible without dropping characters. Three nulls would allow operation at 300 baud and 6 would be good for 600 baud. If instead the UART is connected as an interrupting device (such as on the MTU K-1012 PROM/IO board) and a short first-in-first-out queue is programmed, baud rates approaching the theoretical maximum could be handled without the need for extra nulls. In any case the maximum communication speed is highly application dependent.

As mentioned earlier, a text editing application of the VM with SDXTXT would require a separate text buffer to hold the ASCII representations of the characters displayed. The most straightforward method of handling this would be to write a text buffer subroutine that parallels the operation of SDXTXT except with ASCII codes in an ASCII text buffer. Every character handled would then be given to both routines which would do the same thing with their respective character representations. When text is to be read back or stored on a mass storage device, the ASCII text buffer could then be read to retrieve the ASCII codes.

More sophisticated functions such as line and paragraph movement could be performed in one of two ways. Using the movement of one text line to another location as an example, one could do the operation only in the ASCII text buffer and then clear and regenerate the VM image by dumping the ASCII text buffer through SDXTXT. Although a second or two would be required to rewrite the screen, this is adequate for many applications and in fact is exactly how storage tube terminals (such as the Tektronix series) work.

The other alternative is to write a move routine that moves the VM image directly and add it to SDXTXT to parallel the same operation in the ASCII text buffer. For the one line move example, a routine is needed that would move all text below a given line down one line and open up a single line hole. A second routine that moves a line of characters from elsewhere on the screen into the hole would also be necessary. Finally a "close up" routine to fill the hole left by the line that was moved is needed. All of these routines would be little more than calls to other routines already in SDXTXT. Actually the vertical scrolling that occurs after an LF is a similar operation and can be used as an example. Clearly this is a much faster technique than rewriting the screen and can generally be performed in less than 100 milliseconds. Clever programming in which individual scan lines are moved instead of whole character lines can reduce the time required even further as well as reduce the need for "working storage" to hold the overflow line during the move.

IV. THE GRAPHICS SUPPORT SUBROUTINE PACKAGE

This package combines in one program all of the low level graphic and character drawing functions needed for most applications. Point plotting, line drawing, and character and text display are all provided. For the most part, structured programming discipline and ease of understanding of the code were emphasized more than absolute minimum code size or peak performance. Nevertheless a lot of function has been packed into the 3.2K bytes required by the complete package. Since the programming is modular, unused routines may simply be omitted to reduce the size for specific applications. For example, deleting the "windowed" text display routine will save about 1K. Removing all character display functions will cut the size to less than 1K. Using SDXTXT (simplified display text) instead of DTEXT will give a total package size of less than 2K or two 2708 type PROM's.

Some RAM storage is required by the routines in this package. Four bytes of temporary storage must be located on the base page for use as address pointers. An additional 13 bytes of temporary storage may be located anywhere else. All temporary storage may be used by other programs between calls to the graphic support routines. Finally, 17 bytes of permanent storage for parameters are required. These may not be disturbed between calls unless the user wants to specifically change them. Considerable savings in program size and execution time can be realized by assigning all RAM storage to page zero and reassembling the program.

As assembled, this package occupies locations 5500 - 5F75. Base page temporary storage is from 00EA - 00ED and general temporary storage is from 0111 - 011D. Permanent storage is from 0100 - 0110. The program code itself may be hand relocated anywhere in memory by changing all addresses designated by underlining in the listing. Moving the temporary storage by hand is more difficult but can be accomplished by noting all references to locations to be moved and changing accordingly. Hopefully, assignment of temporary storage to the end of the stack area will be appropriate for the majority of users.

SIGNIFICANCE OF THE PARAMETERS

Information to most of the graphics routines is passed via parameters in memory rather than in the registers. VMORG is the most important parameter. It should be set to the first page number of the Visible Memory before ANY of the graphics routines are called. For example, if the VM is jumpered for addresses 6000 - 7FFF then VMORG should be set to 60₁₆. Once set it will never be changed by any of these routines. Failure to set VMORG will usually cause total program wipeout.

Most graphic routines use one or two sets of coordinates. X1CORD and Y1CORD define one set of coordinates and X2CORD and Y2CORD define another set. All coordinate values are double precision and must always be positive. The double precision representation is with the least significant byte first (lower address) just like memory addresses in the 6502. Furthermore all coordinate values must be in the proper range. This means that 0XX319 and 0XY<199 (decimal numbers). Although Y never exceeds one byte in size, consistency and future compatibility with even higher resolution displays requires that Y be double precision also. Since both X and Y are positive, all coordinates are in the first quadrant.

(11)

Out of range coordinates can cause random storing anywhere in KIM memory. A verification routine is included that can be used in the checkout of an application program to prevent erroneous coordinate values and subsequent program destruction. A call to CKCRD1 will verify and correct if necessary X1CORD and Y1CORD. A call to CKCRD2 will check and correct X2CORD and Y2CORD. Correction, if necessary, is accomplished by subtracting the maximum allowable value of a coordinate until an in range result is obtained. The check routines do not alter any of the registers thus allowing calls to them to be inserted anywhere without problems.

If the text display routine is used, the text margins (TMAR, BMAR, LMAR, and RMAR) must be defined. Text may be written up to and including the margins but will not be written outside of the margins. By suitable manipulation of the margins, multiple, independent blocks of text may be displayed and manipulated on the screen simultaneously. Note that no checking for validity of the margins is performed. TMAR must be greater than BMAR and RMAR must be greater than LMAR. Further, the difference between the margins must be large enough to fit at least 1 line of 2 characters between them.

USE OF THE GRAPHIC POINT PLOT ROUTINES

All of the point oriented routines work with the point defined by X1CORD, Y1CORD. All of the routines preserve the X and Y index registers and do not change either pair of coordinates. The term "pixel" is used frequently. Pixel is a contracted form of "picture element" which is simply a dot on the display or a bit in the Visible Memory. The routines available are as follows:

STPIX - Sets the pixel at X1CORD, Y1CORD to a one (white dot)
CLPIX - Clears the pixel at X1CORD, Y1CORD to zero (black dot)
FLPIX - Changes the state of the pixel at X1CORD, Y1CORD from black to white or white to black
WRPIX - Stores bit 0 of the accumulator into the pixel at X1CORD, Y1CORD
RDPIX - Copies the state of the pixel at X1CORD, Y1CORD into all bits of the accumulator

Proper use of these routines should be self explanatory. For examples, see the Swirl demonstration program listing or some of the higher level routines (such as DRAW) in this package.

An internal subroutine frequently used by other routines in this package is PIXADR. Its purpose is to convert an X,Y coordinate into a VM memory address and a bit number. When called, X1CORD, Y1CORD is converted into an address. The address is stored in ADP1 and the bit number is stored in BTPT. Note that for the purpose of this routine that bit 0 is leftmost in a byte. Either of the indirect addressing modes on the 6502 may then be used to access the designated VM byte and the normal logical AND and OR instructions may be used to select the indicated bit. Mask tables MSKT1 and MSKT2 can be conveniently used as bit selection masks when indexed by the contents of BTPT.

(12)

USE OF THE LINE DRAWING ROUTINE

The line drawing routine is very similar to the point plotting routines. Basically a line is drawn from the point defined by X1CORD, Y1CORD to the point defined by X2CORD, Y2CORD. The line may be any length and at any angle and the routine will determine the best possible series of pixels to turn on between the endpoints. An iterative algorithm that requires no multiplications or divisions is utilized. The index registers are preserved but X1CORD is set equal to X2CORD and Y1CORD is set equal to Y2CORD before the routine returns. If the two sets of coordinates are already equal, the line becomes a single point.

ERASE is exactly like DRAW except that a black line is drawn between the endpoints. ERASE may be used to selectively erase a line that was previously drawn without having to clear the entire screen and regenerate the image. Note however that if a line that crosses other lines is erased a small gap will be left in the lines that it crossed.

USE OF THE CHARACTER DRAWING ROUTINES

DCHAR can be used to draw an ASCII character anywhere on the screen. X1CORD, Y1CORD determines where the character is drawn by specifying the location of the upper left corner of the character. The ASCII code of the character should be in the accumulator when DCHAR is called. The full 96 character set is supported and standard lower case shapes with descenders are used for lower case characters. ASCII control codes are completely ignored. The normal character baseline is 7 pixels below Y1CORD but lower case characters with descenders go as far down as 9 pixels. In any case, a 5 wide by 9 high rectangle is cleared and then a character is drawn into the space. The index registers and coordinates are preserved.

DTEXT is a more sophisticated text display routine than SDXTXT. Major differences are a cursor that works in terms of X and Y graphic coordinates, user defined margins for the text, and the ability to display superscripts and subscripts. A virtual "page" is defined by the margins. The ASCII FF control character for example only clears the display area defined by the margins. Vertical scrolling triggered by LF only scrolls between the margins. Control codes are defined for cursor movement by whole lines and characters in 4 directions or the user may directly position the cursor using the same technique as described for SDXTXT. SI and SO control characters effect a 3 pixel baseline shift up and down respectively for super and subscripts.

DTEXT is called just like SDXTXT. X1CORD and Y1CORD define the cursor location. These may be conveniently initialized to the upper left corner of the virtual page by giving an ASCII FF character to DTEXT before outputting any text. The cursor is then automatically moved when characters are displayed. DTXTIN is a convenience routine that sets the margins for full screen operation, clears the screen and sets the cursor to the upper left corner. With a full screen, DTEXT can display 18 lines of 53 characters. More details on the use of DTEXT are found in the program listings.

COPYRIGHT NOTICE

The cassette, user's manual, and all program listings in this package are copyrighted. The user or customer may make backup copies only to protect against loss or erasure. The copyright notices must remain intact on all such backup copies.

The programs may be used only on the computer systems owned directly by the customer himself and may not be reproduced and shipped with systems sold or rented by the customer.

Volume discounts are available for this software product. In cases of large anticipated volume, licenses and royalties may be negotiated for the reproduction of the package.

Micro Technology Unlimited,
Box 4596
29 Mead Street
Manchester, NH 03108

Dave Cox, Sales manager 603-432-7386
Hal Chamberlin, Engineer 603-669 0170

SWIRL KIM VM SWIRL DEMO
DOCUMENTATION, EQUATES, STORAGE

```

3      ; .PAGE 'DOCUMENTATION, EQUATES, STORAGE'
4      ; SWIRL DRAWING DEMONSTRATION FOR THE MICRO TECHNOLOGY UNLIMITED
5      ; VISIBLE MEMORY 320 BY 200 PIXEL DISPLAY
6      ; ENTER AT SWIRL WITH LINES, FREQ, AND DAMP SET TO APPROPRIATE
7      ; VALUES TO GENERATE AN SWIRLING DISPLAY. INTERRUPT WITH RESET
8      ; KEY WHEN PATTERN IS COMPLETED TO DESIRED EXTENT.
9
10     ; ENTER AT RSWIRL FOR AN ENDLESS SERIES OF PATTERNS USING
11    ; RANDOMLY SELECTED PARAMETERS.
12
13     ; GENERAL EQUATES
14
15 1C22 KIMMON = X'1C22 ; RESET ENTRY INTO KIM MONITOR
16 0140 NX = 320 ; NUMBER OF BITS IN A ROW
17 00C8 NY = 200 ; NUMBER OF ROWS (CHANGE FOR HALF SCREEN)
18
19 FA00 NPIX = NX*NY ; NUMBER OF PIXELS
20
21 0000 .= 0 ; START PROGRAM AT ZERO
22
23     ; STORAGE FOR SWIRL GENERATOR PROGRAM
24
25 0000 01 LINES: .BYTE 1 ; CONNECTING LINES IF NON-ZERO
26 0001 127E FREQ: .WORD X'7E12 ; FREQUENCY
27 0003 007E DAMP: .WORD X'7E00 ; 1-(DAMPING FACTOR)
28 0005 0078 COSINT: .WORD X'7800 ; INITIAL COSINE VALUE
29
30
31 0007 COS: .=+ 2 ; COSINE VALUE
32 0009 SIN: .=+ 2 ; SINE VALUE
33
34
35     ; GENERAL STORAGE
36
37 000B 20 VMORG: .BYTE X'20 ; PAGE NUMBER OF FIRST VISIBLE MEMORY
38
39 000C 3412 RANDNO: .WORD X'1234 ; INITIAL RANDOM NUMBER, MUST NOT BE ZERO
40 000E ADP1: .=+ 2 ; ADDRESS POINTER 1
41 0010 ADP2: .=+ 2 ; ADDRESS POINTER 2
42 0012 BTPT: .=+ 1 ; BIT NUMBER
43 0013 XICORD: .=+ 2 ; COORDINATE PAIR 1
44 0015 YICORD: .=+ 2
45 0017 X2CORD: .=+ 2 ; COORDINATE PAIR 2
46 0019 Y2CORD: .=+ 2
47
48     ; STORAGE FOR ARBITRARY LINE DRAW ROUTINE
49
50 001B DELTAX: .=+ 2 ; DELTA X
51 001D DELTAY: .=+ 2 ; DELTA Y
52 001F ACC: .=+ 2 ; ACCUMULATOR
53 0021 XDIR: .=+ 1 ; X MOVEMENT DIRECTION, ZERO=+
54 0022 YDIR: .=+ 1 ; Y MOVEMENT DIRECTION, ZERO=+
55 0023 XCHFLG: .=+ 1 ; EXCHANGE X AND Y FLAG, EXCHANGE IF NOT 0
56 0024 COLOR: .=+ 1 ; COLOR OF LINE DRAWN -1=WHITE

```

SWIRL KIM VM SWIRL DEMO
DOCUMENTATION, EQUATES, STORAGE

57 0025	TEMP:	.+= 2	; TEMPORARY STORAGE
58		;	STORAGE FOR THE ARITHMETIC SUBROUTINES
60			
61 0027	PROD:	.+= 4	; PRODUCT FOR ARITHMETIC ROUTINES
62 0028	MPCD:	.+= 2	; MULTIPLICAND FOR ARITHMETIC
63 0027	MPLR:	* PROD	; MULTIPLIER FOR ARITHMETIC ROUTINES
64 0020	MPSAVE:	.+= 2	; TEMPORARY STORAGE FOR MULTIPLY
65			

SWIRL KIM VM SWIRL DEMO
MAIN SWIRL GENERATION ROUTINE

```

66      ; .PAGE 'MAIN SWIRL GENERATION ROUTINE'
67
68 002F 208D00 SWIRL: JSR SWINIT    ; INITIALIZE COS AND SIN
69 0032 20A500 SWIRL1: JSR SCALE    ; SCALE SIN AND COS FOR DISPLAY
70 0035 A500 LDA LINES    ; TEST IF LINES BETWEEN POINTS DESIRED
71 0037 D003 BNE SWIRL2'  ; SKIP IF SO
72 0039 205D01 JSR C2TOC1    ; IF NOT, SET LINE LENGTH TO ZERO
73 003C 202202 SWIRL2: JSR DRAW     ; DRAW THE LINE OR POINT
74 003F 200001 JSR POINT    ; COMPUTE THE NEXT POINT
75 0042 4C3200 JMP SWIRL1    ; GO DO NEXT POINT
76
77      ; SWIRL ROUTINE WITH RANDOM PARAMETERS
78
79 0045 208D00 RSWIRL: JSR SWINIT    ; INITIALIZE COS AND SIN
80 0048 209503 RSWR1: JSR RAND     ; INITIALIZE FREQ RANDOMLY WITH UNIFORM
81 004B 8501 STA FREQ    ; DISTRIBUTION
82 004D 209503 JSR RAND
83 0050 8502 STA FREQ+1
84 0052 208103 JSR RNDEXP    ; INITIALIZE DAMP RANDOMLY WITH A NEGATIVE
85 0055 4A LSR4    ; EXPONENTIAL DISTRIBUTION
86 0056 497F EOR #X'7F    ; IN THE UPPER BYTE AND UNIFORM
87 0058 8504 STA DAMP+1    ; DISTRIBUTION IN THE LOWER BYTE
88 005A 209503 JSR RAND
89 005D 8503 STA DAMP
90 005F 209503 JSR RAND    ; RANDOMLY DETERMINE PRESENCE OF
91 0062 2901 AND #1        ; CONNECTING LINES
92 0064 8500 STA LINES
93 0066 20CB03 JSR RANGCK    ; VERIFY ACCEPTABLE RANGES OF PARAMETERS
94 0069 B0D0 BCS RSWR1    ; TRY AGAIN IF NOT ACCEPTABLE
95 006B 20A500 RSWR2: JSR SCALE    ; SCALE THE CURRENT POINT FOR PLOTTING
96 006E A500 LDA LINES    ; TEST IF CONNECTING LINES SPECIFIED
97 0070 D003 BNE RSWR3    ; SKIP AHEAD IF SO
98 0072 205D01 JSR C2TOC1    ; IF NOT, SET ZERO LINE LENGTH
99 0075 202202 RSWR3: JSR DRAW     ; DRAW A LINE FROM THE LAST POINT PLOTTED
100 0078 200001 JSR POINT    ; COMPUTE THE NEXT POINT
101 007B A50A RSWR4: LDA SIN+1    ; TEST IF PATTERN HAS DECAYED TO NEARLY
102 007D F004 BEQ RSWR5    ; ZERO
103 007F C9FF CMP #X'FF
104 0081 D0E8 BNE RSWR2    ;
105 0083 A508 RSWR5: LDA COS+1    ; GO START A NEW PATTERN IF SO
106 0085 F0BE BEQ RSWIRL    ; GO COMPUTE NEXT POINT IF NOT
107 0087 C9FF CMP #X'FF
108 0089 F0BA BEQ RSWIRL    ;
109 008B D0DE BNE RSWR2    ;
110
111      ; SWINIT - INITIALIZE COS FROM COSINT, ZERO SIN, CLEAR SCREEN
112
113 008D A505 SWINIT: LDA COSINT    ; INITIALIZE COS
114 008F 8507 STA COS
115 0091 A506 LDA COSINT+1
116 0093 8508 STA COS+1
117 0095 A900 LDA #0        ; ZERO SIN
118 0097 8509 STA SIN
119 0099 850A STA SIN+1

```

SWIRL KIM VM SWIRL DEMO
MAIN SWIRL GENERATION ROUTINE

```

120 009B 200002 JSR CLEAR    ; CLEAR THE VM SCREEN
121 009E 20A500 JSR SCALE    ; SCALE THE INITIAL POINT AND PUT INTO
122 00A1 205D01 JSR C2TOC1    ; IN BOTH SETS OF COORDINATES
123 00A4 60 RTS      ; RETURN
124
125      ; SCALE - TAKE VALUE OF SIN, SCALE ACCORDING TO NX, AND PUT INTO
126      ; XCORD. THEN TAKE VALUE OF COS, SCALE ACCORDING TO NY, AND
127      ; PUT INTO YCORD.
128      ; SIN AND COS ARE ASSUMED TO BE DOUBLE LENGTH BINARY FRACTIONS
129      ; BETWEEN -1 AND +1.
130
131 00A5 A507 SCALE: LDA COS    ; XCORD=NX/2*SIN+NX/2
132 00A7 8528 STA MPCD    ; TRANSFER SIN TO MULTIPLICAND
133 00A9 A508 LDA COS+1    ; (BINARY FRACTION)
134 00AB 852C STA MPCD+1
135 00AD A9A0 LDA #NX/2&X'FF    ; TRANSFER NX/2 TO MULTIPLIER
136 00AF 8527 STA MPLR    ; (INTEGER)
137 00B1 A900 LDA #NX/2/256
138 00B3 8528 STA MPLR+1
139 00B5 202B03 JSR SGNMPY    ; PERFORM A SIGNED MULTIPLICATION
140 00B8 208B03 JSR SLQL
141 00BB A529 LDA PROD+2    ; SIGNED INTEGER RESULT IN PROD+2 (LOW)
142 00BD 18 CLC      ; AND PROD+3 (HIGH)
143 00BE 69A0 ADC #NX/2&X'FF    ; ADD NX/2 TO PRODUCT AND PUT INTO XCORD
144 00C0 8517 STA XCORD
145 00C2 A52A LDA PROD+3
146 00C4 6900 ADC #NX/2/256
147 00C6 8518 STA XCORD+1
148
149 00C8 A509 LDA SIN    ; YCORD=NY/2*COS+NY/2
150 00CA 8528 STA MPCD    ; TRANSFER COS TO MULTIPLICAND
151 00CC A50A LDA SIN+1    ; (BINARY FRACTION)
152 00CD 852C STA MPCD+1
153 00DD A964 LDA #NY/2&X'FF    ; TRANSFER NY/2 TO MULTIPLIER
154 00DE 8527 STA MPLR    ; (INTEGER)
155 00D4 A900 LDA #NY/2/256
156 00D6 8528 STA MPLR+1
157 00D8 202B03 JSR SGNMPY    ; PERFORM A SIGNED MULTIPLICATION
158 00DB 208B03 JSR SLQL
159 00DE A529 LDA PROD+2    ; SIGNED INTEGER RESULT IN PROD+2 (LOW)
160 00E0 18 CLC      ; AND PROD+3 (HIGH)
161 00E1 6964 ADC #NY/2&X'FF    ; ADD NY/2 TO PRODUCT AND PUT INTO YCORD
162 00E3 8519 STA YCORD
163 00E5 A52A LDA PROD+3
164 00E7 6900 ADC #NY/2/256
165 00E9 851A STA YCORD+1
166 00EB 60 RTS      ; RETURN
167

```

SWIRL KIM VM SWIRL DEMO
POINT - COMPUTE NEXT POINT

```

168      ; .PAGE 'POINT - COMPUTE NEXT POINT'
169      ; POINT - COMPUTE NEXT VALUE OF COS,SIN FROM CURRENT VALUE OF
170      ; COS,SIN ACCORDING TO FREQ AND DAMP. DIFFERENCE EQUATION FOR
171      ; AN ELLIPSE IS USED.
172 00EC   .= X'100
173
174 0100 A509  POINT: LDA SIN      ; FIRST COMPUTE DAMP*SIN AND PUT INTO SIN
175 0102 8528  STA MPCD
176 0104 A50A  LDA SIN+1
177 0106 852C  STA MPCD+1
178 0108 A503  LDA DAMP
179 010A 8527  STA MPLR
180 010C A504  LDA DAMP+1
181 010E 8528  STA MPLR+1
182 0110 202B03 JSR SGNMPY
183 0113 208B03 JSR SLQL      ; SHIFT PRODUCT LEFT ONE FOR FRACTIONAL
184 0116 A529  LDA PROD+2
185 0118 8509  STA SIN      ; RESULT
186 011A A52A  LDA PROD+3
187 011C 850A  STA SIN+1
188
189 011E A507  LDA COS      ; NEXT COMPUTE COS*FREQ
190 0120 8527  STA MPLR
191 0122 A508  LDA COS+1
192 0124 8528  STA MPLR+1
193 0126 A501  LDA FREQ
194 0128 852B  STA MPCD
195 012A A502  LDA FREQ+1
196 012C 852C  STA MPCD+1
197 012E 202B03 JSR SGNMPY
198 0131 208B03 JSR SLQL
199 0134 A509  LDA SIN      ; ADD RESULT TO SIN AND PUT SUM BACK INTO
200 0136 18    CLC
201 0137 6529  ADC PROD+2
202 0139 8509  STA SIN
203 013B A50A  LDA SIN+1
204 013D 652A  ADC PROD+3
205 013F 850A  STA SIN+1
206
207 0141 A509  LDA SIN      ; NEXT COMPUTE FREQ*SIN
208 0143 8527  STA MPLR
209 0145 A50A  LDA SIN+1
210 0147 8528  STA MPLR+1
211 0149 202B03 JSR SGNMPY
212 014C 208B03 JSR SLQL
213
214 014F A507  LDA COS      ; SUBTRACT RESULT FROM COS AND PUT RESULT
215 0151 38    SEC
216 0152 E529  SBC PROD+2
217 0154 8507  STA COS
218 0156 A508  LDA COS+1
219 0158 E52A  SBC PROD+3
220 015A 8508  STA COS+1
221 015C 60    RTS      ; RETURN

```

SWIRL KIM VM SWIRL DEMO
POINT - COMPUTE NEXT POINT

```

222
223      ; SUBROUTINE TO MOVE THE CONTENTS OF COORDINATE PAIR 2 TO
224      ; COORDINATE PAIR 1.
225
226 015D A517  C2TOC1: LDA X2CORD      ; DO THE MOVING
227 015F 8513  STA X1CORD
228 0161 A518  LDA X2CORD+1
229 0163 8514  STA X1CORD+1
230 0165 A519  LDA Y2CORD
231 0167 8515  STA Y1CORD
232 0169 A51A  LDA Y2CORD+1
233 016B 8516  STA Y1CORD+1
234 016D 60    RTS      ; RETURN
235

```

SWIRL KIM VM SWIRL DEMO
ABBREVIATED GRAPHICS ROUTINES

```

.PAGE 'ABBREVIATED GRAPHICS ROUTINES'
236 . PIXADR - FIND THE BYTE ADDRESS AND BIT NUMBER OF PIXEL AT
237 . XICORD,Y1CORD
238 . PUTS BYTE ADDRESS IN ADP1 AND BIT NUMBER (BIT 0 IS LEFTMOST)
239 . IN BTPT.
240 . DOES NOT CHECK MAGNITUDE OF COORDINATES FOR MAXIMUM SPEED
241 . PRESERVES X AND Y REGISTERS, DESTROYS A
242 . BYTE ADDRESS = VMORG*256+(199-Y1CORD)*40+INT(XCORD/8)
243 . BIT ADDRESS = REM(XCORD/8)
244 . OPTIMIZED FOR SPEED THEREFORE CALLS TO A DOUBLE SHIFT ROUTINE
245 . ARE NOT DONE
246 .
247 016E A513 PIXADR: LDA XICORD ; COMPUTE BIT ADDRESS FIRST
248 0170 850E STA ADP1 ; ALSO TRANSFER XICORD TO ADP1
249 0172 2907 AND #X'07 ; WHICH IS SIMPLY THE LOW 3 BITS OF X
250 0174 8512 STA BTPT
251 0176 A514 LDA XICORD+1 ; FINISH TRANSFERRING XICORD TO ADP1
252 0178 850F STA ADP1+1
253 017A 460F LSR ADP1+1 ; DOUBLE SHIFT ADP1 RIGHT 3 TO GET
254 017C 660E ROR ADP1 ; INT(XCORD/8)
255 017E 460F LSR ADP1+1
256 0180 660E ROR ADP1
257 0182 460F LSR ADP1+1
258 0184 660E ROR ADP1
259 0186 A9C7 LDA #199 ; TRANSFER (199-Y1CORD) TO ADP2
260 0188 38 SEC ; AND TEMPORARY STORAGE
261 0189 E515 SBC Y1CORD
262 018B 8510 STA ADP2
263 018D 8525 STA TEMP
264 018F A900 LDA #0
265 0191 E516 SBC Y1CORD+1
266 0193 8511 STA ADP2+1
267 0195 8526 STA TEMP+1
268 0197 0610 ASL ADP2 ; COMPUTE 40*(199-Y1CORD)
269 0199 2611 ROL ADP2+1 ; 2*(199-Y1CORD)
270 019B 0610 ASL ADP2
271 019D 2611 ROL ADP2+1 ; 4*(199-Y1CORD)
272 019F A510 LDA ADP2 ; ADD IN TEMPORARY SAVE OF (199-Y1CORD)
273 01A1 18 CLC ; TO MAKE 5*(199-Y1CORD)
274 01A2 6525 ADC TEMP
275 01A4 8510 STA ADP2
276 01A6 A511 LDA ADP2+1
277 01A8 6526 ADC TEMP+1
278 01AA 8511 STA ADP2+1 ; 5*(199-Y1CORD)
279 01AC 0610 ASL ADP2 ; 10*(199-Y1CORD)
280 01AE 2611 ROL ADP2+1
281 01B0 0610 ASL ADP2 ; 20*(199-Y1CORD)
282 01B2 2611 ROL ADP2+1
283 01B4 0610 ASL ADP2 ; 40*(199-Y1CORD)
284 01B6 2611 ROL ADP2+1
285 01B8 A510 LDA ADP2 ; ADD IN INT(XICORD/8) COMPUTED EARLIER
286 01BA 18 CLC
287 01BB 650E ADC ADP1
288 01BD 850E STA ADP1
289 01BF A511 LDA ADP2+1

```

SWIRL KIM VM SWIRL DEMO
ABBREVIATED GRAPHICS ROUTINES

```

290 01C1 650F ADC ADP1+1
291 01C3 650B ADC VMORG ; ADD IN VMORG*256
292 01C5 850F STA ADP1+1 ; FINAL RESULT
293 01C7 60 RTS ; RETURN
294 .
295 . STPIX - SETS THE PIXEL AT XICORD,Y1CORD TO A ONE (WHITE DOT)
296 . DOES NOT ALTER XICORD OR Y1CORD
297 . PRESERVES X AND Y
298 . ASSUMES IN RANGE COORDINATES
299 .
300 01C8 206E01 STPIX: JSR PIXADR ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
301 . ; INTO ADP1
302 01CB 98 TYA ; SAVE Y
303 01CC 48 PHA
304 01CD A412 LDY BTPT ; GET BIT NUMBER IN Y
305 01CF B91A02 LDA MSKTB1,Y ; GET A BYTE WITH THAT BIT =1, OTHERS =0
306 01D2 A000 LDY #0 ; ZERO Y
307 01D4 110E ORA (ADP1),Y ; COMBINE THE BIT WITH THE ADDRESSED VM
308 01D6 910E STA (ADP1),Y ; BYTE
309 01D8 68 PLA ; RESTORE Y
310 01D9 A8 TAY
311 01DA 60 RTS ; AND RETURN
312 .
313 01DB .= X'200
314 .
315 . ; CLEAR DISPLAY MEMORY ROUTINE
316 .
317 0200 A000 CLEAR: LDY #0 ; INITIALIZE ADDRESS POINTER
318 0202 840E STY ADP1 ; AND ZERO INDEX Y
319 0204 A50B LDA VMORG
320 0206 850F STA ADP1+1
321 0208 18 CLC
322 0209 6920 ADC #X'20
323 020B AA TAX
324 020C 98 CLEAR1: TYA ; CLEAR A BYTE
325 020D 910E STA (ADP1),Y ; INCREMENT ADDRESS POINTER
326 020F E60E INC ADP1
327 0211 DDF9 BNE CLEAR1
328 0213 E60F INC ADP1+1
329 0215 E40F CPX ADP1+1 ; TEST IF DONE
330 0217 DDF3 BNE CLEAR1
331 0219 60 RTS ; RETURN
332 .
333 . ; MASK TABLES FOR INDIVIDUAL PIXEL SUBROUTINES
334 . MSKTB1 IS A TABLE OF 1 BITS CORRESPONDING TO BIT NUMBERS
335 .
336 021A 80402010 MSKTB1: .BYTE X'80,X'40,X'20,X'10
337 021E 08040201 .BYTE X'08,X'04,X'02,X'01
338 .

```

SWIRL KIM VM SWIRL DEMO
LINE DRAWING ROUTINES

```

.PAGE 'LINE DRAWING ROUTINES'
339   ; DRAW - DRAW THE BEST STRAIGHT LINE FROM XICORD,YICORD TO
340   ; X2CORD,Y2CORD.
341   ; X2CORD,Y2CORD COPIED TO XICORD,YICORD AFTER DRAWING
342   ; PRESERVES X AND Y
343   ; USES AN ALGORITHM THAT REQUIRES NO MULTIPLICATION OR DIVISON
344
345 0222 8A   DRAW: TXA      ; SAVE X AND Y
346 0223 48   PHA
347 0224 98   TYA
348 0225 48   PHA
349
350 350      ; COMPUTE SIGN AND MAGNITUDE OF DELTA X = X2-X1
351 351      ; PUT MAGNITUDE IN DELTAX AND SIGN IN XDIR
352
353 0226 A900 LDA #0      ; FIRST ZERO XDIR
354 0228 B521 STA XDIR
355 022A A517 LDA X2CORD ; NEXT COMPUTE TWOS COMPLEMENT DIFFERENCE
356 022C 38   SEC
357 022D E513 SBC X1CORD
358 022F B518 STA DELTAX
359 0231 A518 LDA X2CORD+1
360 0233 E514 SBC X1CORD+1
361 0235 B51C STA DELTAX+1
362 0237 100F BPL DRAW2  ; SKIP AHEAD IF DIFFERENCE IS POSITIVE
363 0239 C621 DEC XDIR   ; SET XDIR TO -1
364 023B 38   SEC
365 023C A900 LDA #0      ; NEGATE DELTAX
366 023E E518 SBC DELTAX
367 0240 B518 STA DELTAX
368 0242 A900 LDA #0
369 0244 E51C SBC DELTAX+1
370 0246 B51C STA DELTAX+1
371
372 372      ; COMPUTE SIGN AND MAGNITUDE OF DELTA Y = Y2-Y1
373 373      ; PUT MAGNITUDE IN DELTAY AND SIGN IN YDIR
374
375 0248 A900 DRAW2: LDA #0      ; FIRST ZERO YDIR
376 024A B522 STA YDIR
377 024C A519 LDA Y2CORD ; NEXT COMPUTE TWOS COMPLEMENT DIFFERENCE
378 024E 38   SEC
379 024F E515 SBC Y1CORD
380 0251 B510 STA DELTAY
381 0253 A51A LDA Y2CORD+1
382 0255 E516 SBC Y1CORD+1
383 0257 B51E STA DELTAY+1
384 0259 100F BPL DRAW3  ; SKIP AHEAD IF DIFFERENCE IS POSITIVE
385 025B C622 DEC YDIR   ; SET YDIR TO -1
386 025D 38   SEC
387 025E A900 LDA #0
388 0260 E51D SBC DELTAY
389 0262 B510 STA DELTAY
390 0264 A900 LDA #0
391 0266 E51E SBC DELTAY+1
392 0268 B51E STA DELTAY+1

```

SWIRL KIM VM SWIRL DEMO
LINE DRAWING ROUTINES

```

393
394 394      ; DETERMINE IF DELTAY IS LARGER THAN DELTAX
395 395      ; IF SO, EXCHANGE DELTAY AND DELTAX AND SET XCHFLG NONZERO
396 396      ; ALSO INITIALIZE ACC TO DELTAX
397 397      ; PUT A DOT AT THE INITIAL ENDPOINT
398
399 026A A900 DRAW3: LDA #0      ; FIRST ZERO XCHFLG
400 026C 8523 STA XCHFLG
401 026E A51D LDA DELTAY   ; COMPARE DELTAY WITH DELTAX
402 0270 38   SEC
403 0271 E51B SBC DELTAX
404 0273 A51E LDA DELTAY+1
405 0275 E51C SBC DELTAX+1
406 0277 9012 BCC DRAW4  ; SKIP EXCHANGE IF DELTAX IS GREATER THAN
407
408 0279 A61D LDX DELTAY
409 027B A51B LDA DELTAX
410 027D 851D STA DELTAY
411 027F 861B STX DELTAX
412 0281 A61E LDX DELTAY+1
413 0283 A51C LDA DELTAX+1
414 0285 851E STA DELTAY+1
415 0287 861C STX DELTAX+1
416 0289 C623 DEC XCHFLG ; SET XCHFLG TO -1
417 028B A51B DRAW4: LDA DELTAX ; INITIALIZE ACC TO DELTAX
418 028D 851F STA ACC
419 028F A51C LDA DELTAX+1
420 0291 8520 STA ACC+1
421 0293 20C801 JSR STPIX ; PUT A DOT AT THE INITIAL ENDPOINT;
422
423
424 424      ; HEAD OF MAIN DRAWING LOOP
425 425      ; TEST IF DONE
426
427 0296 A523 DRAW45: LDA XCHFLG ; TEST IF X AND Y EXCHANGED
428 0298 D00E BNE DRAW5 ; JUMP AHEAD IF SO
429 029A A513 LDA X1CORD ; TEST FOR XICORD=X2CORD
430 029C C517 CMP X2CORD
431 029E D019 BNE DRAW7 ; GO FOR ANOTHER ITERATION IF NOT
432 02A0 A514 LDA X1CORD+1
433 02A2 C518 CMP X2CORD+1
434 02A4 D013 BNE DRAW7 ; GO FOR ANOTHER ITERATION IF NOT
435 02A6 F00C BEQ DRAW6 ; GO RETURN IF SO
436 02A8 A515 DRAW5: LDA Y1CORD ; TEST FOR YICORD=Y2CORD
437 02AA C519 CMP Y2CORD
438 02AC D00B BNE DRAW7 ; GO FOR ANOTHER ITERATION IF NOT
439 02AE A516 LDA Y1CORD+1
440 02B0 C51A CMP Y2CORD+1
441 02B2 D005 BNE DRAW7 ; GO FOR ANOTHER ITERATION IF NOT
442 02B4 68  PLA
443 02B5 A8  TAY
444 02B6 68  PLA
445 02B7 AA  TAX
446 02B8 60  RTS ; AND RETURN
447

```

SWIRL KIM VM SWIRL DEMO
LINE DRAWING ROUTINES

```

448      ; DO A CLACULATION TO DETERMINE IF ONE OR BOTH AXES ARE TO BE
449      ; BUMPED (INCREMENTED OR DECREMENTED ACCORDING TO XDIR AND YDIR)
450      ; AND DO THE BUMPING
451
452 02B9 A523  DRAW7: LDA XCHFLG ; TEST IF X AND Y EXCHANGED
453 02B8 D006    BNE DRAW8  ; JUMP IF SO
454 02BD 200303  JSR BMPX   ; BUMP X IF NOT
455 02C0 4CC602  JMP DRAW9
456 02C3 201703  DRAW8: JSR BMPY  ; BUMP Y IF SO
457 02C6 20E702  DRAW9: JSR SBDY  ; SUBTRACT DY FROM ACC TWICE
458 02C9 20E702  JSR SBDY
459 02CC 1013    BPL DRAW12 ; SKIP AHEAD IF ACC IS NOT NEGATIVE
460 02CE A523    LDA XCHFLG ; TEST IF X AND Y EXCHANGED
461 02D0 D006    BNE DRAW10 ; JUMP IF SO
462 02D2 201703  JSR BMPY  ; BUMP Y IF NOT
463 02D5 4CD802  JMP DRAW11
464 02D8 200303  DRAW10: JSR BMPX ; BUMP X IF SO
465 02DB 20F502  DRAW11: JSR ADDX ; ADD DX TO ACC TWICE
466 02DE 20F502  JSR ADDX
467
468 02E1 20C801  DRAW12: JSR STPIX  ; OUTPUT THE NEW POINT
469 02E4 4C9602  JMP DRAW45 ; GO TEST IF DONE
470
471      ; SUBROUTINES FOR DRAW
472
473 02E7 A51F    SBDY: LDA ACC   ; SUBTRACT DELTAY FROM ACC AND PUT RESULT
474 02E9 38      SEC
475 02EA E51D    SBC DELTAY
476 02EC 851F    STA ACC
477 02EE A520    LDA ACC+1
478 02F0 E51E    SBC DELTAY+1
479 02F2 8520    STA ACC+1
480 02F4 60      RTS
481
482
483 02F5 A51F    ADDX: LDA ACC   ; ADD DELTAX TO ACC AND PUT RESULT IN ACC
484 02F7 18      CLC
485 02F8 651B    ADC DELTAX
486 02FA 851F    STA ACC
487 02FC A520    LDA ACC+1
488 02FE 651C    ADC DELTAX+1
489 0300 8520    STA ACC+1
490 0302 60      RTS
491
492
493 0303 A521    BMPX: LDA XDIR   ; BUMP XICORD BY +1 OR -1 ACCORDING TO
494 0305 D007    BNE BMPX2 ; XDIR
495 0307 E613    INC XICORD ; DOUBLE INCREMENT XICORD IF XDIR=0
496 0309 D002    BNE BMPX1
497 030B E614    INC XICORD+1
498 030D 60      BMPX1: RTS
499 030E A513    BMPX2: LDA XICORD ; DOUBLE DECREMENT XICORD IF XDIR<>0
500 0310 D002    BNE BMPX3
501 0312 C614    DEC XICORD+1
502 0314 C613    DEC XICORD

```

SWIRL KIM VM SWIRL DEMO
LINE DRAWING ROUTINES

```

503 0316 60      RTS
504
505
506 0317 A522    BMPY: LDA YDIR   ; BUMP YICORD BY +1 OR -1 ACCORDING TO
507 0319 D007    BNE BMPY2 ; YDIR
508 031B E615    INC YICORD ; DOUBLE INCREMENT YICORD IF YDIR=0
509 031D D002    BNE BMPY1
510 031F E616    INC YICORD+1
511 0321 60      BMPY1: RTS
512 0322 A515    BMPY2: LDA YICORD ; DOUBLE DECREMENT YICORD IF YDIR<>0
513 0324 D002    BNE BMPY3
514 0326 C616    DEC YICORD+1
515 0328 C615    BMPY3: DEC YICORD
516 032A 60      RTS
517

```

SWIRL KIM VM SWIRL DEMO
MULTIPLY, SHIFT, AND RANDOM NUMBER ROUTINES

```

.PAGE 'MULTIPLY, SHIFT, AND RANDOM NUMBER ROUTINES'
518      ; SIGNED MULTIPLY SUBROUTINE
519      ; ENTER WITH SIGNED MULTIPLIER IN PROD AND PROD+1
520      ; ENTER WITH SIGNED MULTPLICAND IN MPCD AND MPCD+1
521      ; RETURN WITH 16 BIT SIGNED PRODUCT IN PROD (LOW) THROUGH
522      ; PROD+3 (HIGH)
523      ; A DESTROYED, X AND Y PRESERVED
524
525 032B A527  SGNMPY: LDA PROD    ; GET MULTIPLIER
526 032D 852D  STA MPSAVE   ; AND SAVE IT
527 032F A528  LDA PROD+1
528 0331 852E  STA MPSAVE+1
529 0333 205903 JSR UNSMPY  ; DO AN UNSIGNED MULTIPLY
530 0336 A52C  LDA MPCD+1  ; TEST SIGN OF MULTPLICAND
531 0338 1000  BPL SGNMP1  ; JUMP IF POSITIVE
532 033A A529  LDA PROD+2  ; SUBTRACT MULTIPLIER FROM HIGH PRODUCT IF
533 033C 38    SEC        ; NEGATIVE
534 033D E52D  SBC MPSAVE
535 033F 8529  STA PROD+2
536 0341 A52A  LDA PROD+3
537 0343 E52E  SBC MPSAVE+1
538 0345 852A  STA PROD+3
539 0347 A52E  SGNMP1: LDA MPSAVE+1  ; TEST SIGN OF MULTIPLIER
540 0349 1000  BPL SGNMP2  ; GO RETURN IF POSITIVE
541 034B A529  LDA PROD+2  ; SUBTRACT MULTPLICAND FROM HIGH PRODUCT
542 034D 38    SEC        ; IF NEGATIVE
543 034E E52B  SBC MPCD
544 0350 8529  STA PROD+2
545 0352 A52A  LDA PROD+3
546 0354 E52C  SBC MPCD+1
547 0356 852A  STA PROD+3
548 0358 60    SGNMP2: RTS       ; RETURN
549
550      ; 16 X 16 UNSIGNED MULTIPLY SUBROUTINE
551      ; ENTER WITH UNSIGNED MULTIPLIER IN PROD AND PROD+1
552      ; ENTER WITH UNSIGNED MULTPLICAND IN MPCD AND MPCD+1
553      ; RETURN WITH 16 BIT UNSIGNED PRODUCT IN PROD (LOW) THROUGH
554      ; PROD+3 (HIGH)
555      ; A DESTROYED, X AND Y PRESERVED
556
557 0359 8A    UNSMPY: TXA      ; SAVE X INDEX
558 035A 48    PHA
559 035B A900  LDA #0      ; CLEAR UPPER PRODUCT
560 035D 852A  STA PROD+3
561 035F 8529  STA PROD+2
562 0361 A211  LDX #17      ; SET 17 MULTIPLY CYCLE COUNT
563 0363 18    CLC        ; INITIALLY CLEAR CARRY
564 0364 208203 UNSM1: JSR SRLQ
565
566 0367 CA    DEX
567 0368 F012  BEQ UNSM2
568 036A 90F8  BCC UNSM1
569
570 036C A529  LDA PROD+2  ; ADD MULTPLICAND TO UPPER PRODUCT
571 036E 18    CLC

```

SWIRL KIM VM SWIRL DEMO
MULTIPLY, SHIFT, AND RANDOM NUMBER ROUTINES

```

572 036F 652B  ADC MPCD
573 0371 8529  STA PROD+2
574 0373 A52A  LDA PROD+3
575 0375 652C  ADC MPCD+1
576 0377 852A  STA PROD+3
577 0379 4C6403 JMP UNSM1      ; GO FOR NEXT CYCLE
578 037C 68    UNSM2: PLA      ; RESTORE X
579 037D AA    RTS       ; RETURN
581
582      ; QUAD SHIFT RIGHT SUBROUTINE
583      ; ENTER AT SRQA FOR ALGEBRAIC SHIFT RIGHT
584      ; ENTER AT SRQL FOR LOGICAL SHIFT
585      ; ENTER WITH QUAD PRECISION VALUE TO SHIFT IN PROD THROUGH PROD+3
586      ; DESTROYS A, PRESERVES X AND Y, RETURNS BIT SHIFTED OUT IN CARRY
587
588 037F A52A  SRQA: LDA PROD+3  ; GET SIGN BIT OF PROD IN CARRY
589 0381 0A    ASLA
590 0382 662A  SRQL: ROR PROD+3  ; LOGICAL SHIFT RIGHT ENTRY
591 0384 6629  ROR PROD+2
592 0386 662B  ROR PROD+1
593 0388 6627  ROR PROD
594 038A 60    RTS       ; RETURN
595
596
597      ; QUAD SHIFT LEFT SUBROUTINE
598      ; ENTER AT SLQL TO SHIFT IN A ZERO BIT
599      ; ENTER AT RLQL TO SHIFT IN THE CARRY
600      ; ENTER WITH QUAD PRECISION VALUE TO SHIFT IN PROD THROUGH PROD+3
601      ; DESTROYS A, PRESERVES X AND Y, RETURNS BIT SHIFTED OUT IN CARRY
602
603 038B 18    SLQL: CLC      ; SHIFT IN ZERO BIT ENTRY; CLEAR CARRY
604 038C 2627  RLQL: ROL      ; SHIFT IN CARRY ENTRY
605 038E 2628  ROL PROD+1
606 0390 2629  ROL PROD+2
607 0392 262A  ROL PROD+3
608 0394 60    RTS       ; RETURN
609
610      ; RANDOM NUMBER GENERATOR SUBROUTINE
611      ; ENTER WITH SEED IN RANDNO
612      ; EXIT WITH NEW RANDOM NUMBER IN RANDNO AND A
613      ; USES 16 BIT FEEDBACK SHIFT REGISTER METHOD
614      ; DESTROYS REGISTER A AND Y
615
616 0395 A008  RAND: LDY #8      ; SET COUNTER FOR 8 RANDOM BITS
617 0397 A50C  RAND1: LDA RANDNO  ; EXCLUSIVE-OR BITS 3, 12, 14, AND 15
618 0399 4A    LSRA      ; OF SEED
619 039A 450C  EOR RANDNO
620 039C 4A    LSRA
621 039D 4A    LSRA
622 039E 450C  EOR RANDNO
623 03A0 4A    LSRA
624 03A1 450D  EOR RANDNO+1  ; RESULT IS IN BIT 3 OF A
625 03A3 4A    LSRA      ; SHIFT INTO CARRY
626 03A4 4A    LSRA

```

SWIRL KIM VM SWIRL DEMO
MULTIPLY, SHIFT, AND RANDOM NUMBER ROUTINES

```

627 03A5 4A      LSRA
628 03A6 4A      LSRA
629 03A7 260D    ROL  RANDNO+1 ; SHIFT RANDNO LEFT ONE BRINGING IN CARRY
630 03A9 260C    ROL  RANDNO
631 03AB 88      DEY
632 03AC DOE9    BNE  RAND1 ; TEST IF 8 NEW RANDOM BITS COMPUTED
633 03AE A50C    LDA  RANDNO ; LOOP FOR MORE IF NOT
634 03B0 60      RTS
635
636      ; EXPONENTIALLY DISTRIBUTED RANDOM NUMBER SUBROUTINE
637      ; RULES OF USE SAME AS RAND, 8 BIT RESULT RETURNED IN A
638      ; AN EXPONENTIAL DISTRIBUTION MEANS THAT THE PROBABILITY OF A
639      ; RESULT BETWEEN 10 AND 20 IS THE SAME AS THE PROBABILITY OF A
640      ; RESULT BETWEEN 100 AND 200.
641      ; NOTE THAT THE PROBABILITY OF A ZERO RESULT IS ZERO.
642
643 03B1 209503  RNDEXP: JSR  RAND ; GET TWO NEW RANDOM BYTES
644 03B4 209503  JSR  RAND
645 03B7 A50C    LDA  RANDNO ; CONVERT ONE OF THE BYTES TO A RANDOM
646 03B9 2907    AND  #7   ; VALUE BETWEEN 0 AND 7 AND PUT IN Y AS A
647 03BB A8      TAY
648 03BC C8      INY
649 03BD A50D    LDA  RANDNO+1 ; GET THE OTHER RANDOM NUMBER AND SHIFT IT
650 03BF 88      DEY
651 03C0 F004    BEQ  RNDXP2 ; RIGHT ACCORDING TO Y
652 03C2 4A      LSRA
653 03C3 4CBF03  JMP  RNDXP1
654 03C6 0900    ORA  #0   ; TEST FOR A ZERO RESULT
655 03C8 FOE7    BEQ  RNDEXP ; PROHIBIT ZERO RESULTS
656 03CA 60      RTS
657
658      ; RANGCK - CHECK FOR ACCEPTABLE RANGE OF FREQ AND DAMP PARAMETERS
659      ; RETURN WITH CARRY OFF IF OK
660
661 03CB A502    RANGCK: LDA  FREQ+1 ; MINIMUM ABSOLUTE VALUE FOR FREQ IS X'0100
662 03CD F01C    BEQ  RANGNK ; GO TO FAILURE RETURN IF HIGH BYTE IS 0
663 03CF C9FF    CMP  #X'FF
664 03D1 F018    BEQ  RANGNK ; GO TO FAILURE RETURN IF HIGH BYTE IS FF
665 03D3 A504    RANG2:  LDA  DAMP+1 ; CHECK THAT DAMP IS NOT GREATER THAN
666 03D5 C97F    CMP  #X'7F ; X'7EFF
667 03D7 F012    BEQ  RANGNK ; GO TO FAILURE RETURN IF SO
668 03D9 A502    RANG3:  LDA  FREQ+1 ; IF FREQ AND DAMP ARE INDIVIDUALLY OK,
669 03D8 1002    BPL  RANG4 ; VERIFY THAT DAMP IS ACCEPTABLY HIGH IF
670 03D0 45FF    EOR  X'FF ; ABSOLUTE VALUE OF FREQ IS SMALL
671 03DF C908    RANG4:  CMP  #8
672 03E1 1006    BPL  RANGOK ; GO TO SUCCESS RETURN IF FREQ IS HIGH
673 03E3 A504    LDA  DAMP+1 ; IF FREQ IS LOW, REQUIRE DAMP TO BE HIGH
674 03E5 C97E    CMP  #X'7E
675 03E7 3002    JMI  RANGNK ; GO TO FAILURE RETURN IF DAMP NOT HIGH
676
677 03E9 18      RANGOK: CLC
678 03EA 60      RTS ; CLEAR CARRY TO INDICATE SUCCESS
679 03EB 38      RANGNK: SEC
680 03EC 60      RTS ; SET CARRY TO INDICATE FAILURE
681
682
683 0000      .END

```

VMLIF VISIBLE MEMORY LIFE
DOCUMENTATION, EQUATES, STORAGE

```

3   ; .PAGE 'DOCUMENTATION, EQUATES, STORAGE'
4   ; MTU VISIBLE MEMORY DEMONSTRATION PROGRAM
5   ; JOSEPH CONWAY'S GAME OF LIFE ON A 320 BY 200 MATRIX
6   ; ENTRY POINT "DEMO" GENERATES AN INITIAL PATTERN OF CELLS AND
7   ; THEN EXECUTES THE LIFE ALGORITHM ON IT.
8   ;
9   ; FOR USER ENTERED PATTERNS, THE SCREEN SHOULD FIRST BE CLEARED
10  ; BY EXECUTING "INIT". THE KIM KEYBOARD MONITOR OR "KYPT" MAY
11  ; THEN BE USED TO ENTER THE INITIAL CELL PATTERN. AFTER PATTERN
12  ; ENTRY, A JUMP TO "LIFE" WILL START COMPUTING THE SUCCEEDING
13  ; GENERATIONS.
14  ;
15  ; LIFE MAY BE INTERRUPTED AT THE END OF A GENERATION BY PRESSING
16  ; ANY KEY (EXCEPT RESET OR ST) ON THE KIM KEYPAD AND HOLDING
17  ; UNTIL THE END OF THE GENERATION. THIS WILL TRANSFER CONTROL
18  ; TO "KYPT" FOR USER MODIFICATION OF THE DISPLAYED PATTERN.
19  ;
20  ; KYPT IS USED FOR CONVENIENT ENTRY AND MODIFICATION OF CELL
21  ; PATTERNS. WHEN ENTERED, A BLINKING GRAPHIC CURSOR IS
22  ; DISPLAYED IN THE MIDDLE OF THE SCREEN. THE USER MAY MOVE THE
23  ; CURSOR IN ANY DIRECTION AND EITHER SET OR CLEAR CELLS AT THE
24  ; CURRENT CURSOR POSITION. THE CURSOR IS MOSTLY ON IF IT COVERS
25  ; A LIVE CELL AND MOSTLY OFF OTHERWISE.
26  ; THE KIM KEYBOARD IS USED FOR CONTROL OF THE PROGRAM. THE
27  ; FOLLOWING KEYS ARE ACTIVE:
28  ;     1 CURSOR DOWN
29  ;     6 CURSOR RIGHT
30  ;     9 CURSOR UP
31  ;     4 CURSOR LEFT
32  ;     + SET A CELL
33  ;     F CLEAR A CELL
34  ;
35  ; GO GO TO LIFE ROUTINE USING THE CURRENT PATTERN
36  ; PARTICULARLY INTERESTING INITIAL PATTERNS MAY BE SAVED ON KIM
37  ; CASSETTE AND RELOADED LATER FOR DEMONSTRATIONS, ETC.
38  ;
39  ; GENERAL EQUATES
40 1C22 KIMMON = X'1C22 ; ENTRY TO KIM MONITOR
41 1F6A GETKEY = X'1F6A ; ADDRESS OF MONITOR KEYBOARD READ ROUTINE
42 0140 NX = 320 ; NUMBER OF BITS IN A ROW
43 00C8 NY = 200 ; NUMBER OF ROWS (CHANGE FOR HALF SCREEN
44 ; OPERATION)
45 FA00 NPIX = NX*NY ; NUMBER OF PIXELS
46 0032 DBCDLA = 50 ; KIM KEYBOARD DEBOUNCE DELAY TIME
47
48 0000 .= 0 ; START DEMO PROGRAM AT LOCATION ZERO
49
50 ; PARAMETER STORAGE
51
52 0000 20 VMORG: .BYTE X'20 ; FIRST PAGE IN DISPLAY MEMORY
53
54 ; MISCELLANEOUS STORAGE
55
56 0001 NCYSV: .=+ 1 ; TEMPORARY STORAGE FOR NEIGHBOR COUNT

```

VMLIF VISIBLE MEMORY LIFE
DOCUMENTATION, EQUATES, STORAGE

57			ROUTINE
58 0002	NCNT:	.=.+ 1	; COUNT OF LIVE NEIGHBORS
59 0003	LNCNT:	.=.+ 1	; CELL LINE COUNTER
60 0004	NGEN:	.=.+ 1	; BYTE TO ACCUMULATE NEW CELLS
61 0005	ADP1:	.=.+ 2	; ADDRESS POINTER 1
62 0007	ADP2:	.=.+ 2	; ADDRESS POINTER 2
63 0009	BTPT:	.=.+ 1	; BIT NUMBER
64 000A	X1CORD:	.=.+ 2	; COORDINATE PAIR 1
65 000C	Y1CORD:	.=.+ 2	
66 000E	X2CORD:	.=.+ 2	; COORDINATE PAIR 2
67 0010	Y2CORD:	.=.+ 2	
68 0012	TEMP:	.=.+ 2	; TEMPORARY STORAGE
69 0014	FLASHC:	.=.+ 2	; TIME DELAY COUNTER FOR CURSOR FLASHING
70 0001	LSTKEY	= NCYSV	; CODE OF LAST KEY PRESSED ON KIM KEYBOARD
71 0002	DBCNT	= NCNT	; KIM KEYBOARD DEBOUNCE COUNTER
72 0003	REALST	= LNCNT	; STATE OF CELL UNDER THE CURSOR
73			
74			TABLE OF MASKS FOR NEIGHBOR COUNTING
75			
76 0016 01		.BYTE X'01	
77 0017 80402010	MSK:	.BYTE X'80,X'40,X'20,X'10	
78 0018 08040201		.BYTE X'08,X'04,X'02,X'01	
79 001F 80		.BYTE X'80	
80			
81			STORAGE TO BUFFER 3 FULL SCAN LINES OF CELLS
82			
83 0020 00		.BYTE 0	
84 0021	TR:	.=.+ 40	; ROW ABOVE CENTRAL ROW
85 0049	CR:	.=.+ 40	; CENTRAL ROW
86 0071	BR:	.=.+ 40	; ROW BELOW CENTRAL ROW
87 0099 00		.BYTE 0	
88			

VMLIF VISIBLE MEMORY LIFE
INITIAL PATTERN GENERATION ROUTINES

```

89      ; .PAGE 'INITIAL PATTERN GENERATION ROUTINES'
90      ; CLEAR DISPLAY MEMORY AND INITIALIZE ROUTINE
91      ; USED TO PREPARE SCREEN FOR USER ENTERED PATTERN
92 009A D8  INIT: CLD      ; INITIALIZE MACHINE AND DISPLAY
93 009B 202C02 JSR      CLEAR    ; CLEAR THE SCREEN
94 009E 4C221C JMP      KIMMON   ; RETURN TO THE MONITOR
95
96      ; MAIN DEMO ROUTINE, DRAW INITIAL PATTERN
97      ; DRAWS A FIGURE DEFINED BY "LIST" AND THEN JUMPS TO LIFE
98
99 00A1 D8  DEMO: CLD      ; CLEAR DECIMAL MODE
100 00A2 202C02 JSR      CLEAR    ; CLEAR THE SCREEN
101 00A5 A200 LDX      #0       ; INITIALIZE INDEX FOR COORDINATE LIST
102 00A7 BD3603 DEMO1: LDA      LIST+1,X ; GET HIGH BYTE OF X COORDINATE
103 00A8 101A BPL      DEMO2    ; JUMP IF A DRAW COMMAND
104 00AC C9FF CMP      #X'FF   ; IF MOVE, TEST FOR END OF LIST FLAG
105 00AE F050 BEQ      LIFE     ; GO TO LIFE IF SO
106 00B0 297F AND      #X'7F   ; DELETE SIGN BIT
107 00B2 8508 STA      XICORD+1 ; FOR MOVE JUST COPY COORDINATES FROM LIST
108 00B4 BD3503 LDA      LIST,X   ; INTO XICORD,YICORD
109 00B7 850A STA      XICORD
110 00B9 BD3703 LDA      LIST+2,X
111 00BC 850C STA      YICORD
112 00BE BD3803 LDA      LIST+3,X
113 00C1 850D STA      YICORD+1
114 00C3 4CDA00 JMP      DEMO3
115 00C6 850F DEMO2: STA      X2CORD+1 ; FOR DRAW, COPY COORDINATES FROM LIST
116 00C8 BD3503 LDA      LIST,X   ; INTO X2CORD,Y2CORD
117 00CB 850E STA      X2CORD
118 00CD BD3703 LDA      LIST+2,X
119 00D0 8510 STA      Y2CORD
120 00D2 BD3803 LDA      LIST+3,X
121 00D5 8511 STA      Y2CORD+1
122 00D7 20F502 JSR      SDRAW   ; DRAW LINE FROM XICORD,YICORD TO X2CORD,
123 00DA E8  DEMO3: INX      ; Y2CORD
124 00DB E8  INX      ; BUMP INDEX TO NEXT SET OF COORDINATES
125 00DC E8  INX
126 00DD E8  INX
127 00DE D0C7 BNE      DEMO1   ; LOOP UNTIL END OF LIST REACHED
128 00E0 FO1E BEQ      LIFE    ; GO TO LIFE ROUTINE WHEN DONE
129
130      ; CSRINS - INSERT GRAPHIC CURSOR AT XICORD,YICORD
131      ; SAVES STATE OF THE CELL ALREADY THERE IN REALST
132
133 00E2 20CC02 CSRINS: JSR      RDPIX   ; READ CURRENT STATE OF CELL UNDER CURSOR
134 00E5 8503 STA      REALST   ; SAVE THE STATE
135 00E7 60  RTS      ; RETURN
136
137      ; CSRDEL - DELETE THE GRAPHIC CURSOR AT XICORD,YICORD
138      ; AND RESTORE THE CELL THAT WAS ORIGINALLY THERE
139
140 00E8 A503 CSRDEL: LDA      REALST   ; GET SAVED CELL STATE
141 00EA 20C402 JSR      WRPIX   ; PUT IT BACK INTO DISPLAY MEMORY
142 00ED 60  RTS      ; RETURN

```

VMLIF VISIBLE MEMORY LIFE
MAIN LIFE ROUTINE

```

144 00EE .PAGE 'MAIN LIFE ROUTINE'
145      ; X'100
146 0100 A900 LIFE: LDA      #0       ; PRIME THE THREE LINE BUFFERS
147 0102 8505 STA      ADP1    ; INITIALIZE VM POINTER TO TOP OF SCREEN
148 0104 A500 LDA      VMORG
149 0106 8506 STA      ADP1+1
150 0108 201D02 JSR      PRIME   ; DO THE PRIMING
151
152      ; MAIN LIFE LOOP
153
154 010B A9C6 LDA      #198   ; SET COUNT OF ROWS TO PROCESS
155 010D 8503 STA      LWCNT
156 010F A505 LIFE1: LDA      ADP1   ; INCREMENT THE ADDRESS POINTER TO THE
157 0111 18 CLC
158 0112 6928 ADC      #40
159 0114 8505 STA      ADP1
160 0116 9002 BCC      LIFE2
161 0118 E606 INC      ADP1+1
162 011A 203101 LIFE2: JSR      LFBUF  ; EXECUTE LIFE ALGORITHM ON CENTRAL ROW
163
164
165 011D C603 DEC      LWCNT
166 011F F006 BEQ      LIFE3
167 0121 200002 JSR      ROLL   ; JUMP OUT IF 198 LINES BEEN PROCESSED
168 0124 4C0F01 JMP      LIFE1  ; ROLL THE BUFFERS UP ONE POSITION
169
170
171      ; END OF GENERATION, TEST KIM KEYBOARD
172 0127 206A1F LIFE3: JSR      GETKEY
173 012A C915 CMP      #21
174 012C B0D2 BCS      LIFE   ; GO FOR NEXT GENERATION IF NO KEY PRESSED
175 012E 4CC703 JMP      KYPT   ; GO TO KEYBOARD PATTERN ENTRY IF A
176
177

```

VMLIF VISIBLE MEMORY LIFE
LIFE NEXT GENERATION ROUTINE FOR BUFFER CONTENTS

```

.PAGE 'LIFE NEXT GENERATION ROUTINE FOR BUFFER CONTENTS'
178    ; LIFE NEXT GENERATION ROUTINE
179    ; THE CELLS IN THE MIDDLE LINE BUFFER ARE SCANNED AND THEIR
180    ; NEIGHBORS COUNTED TO DETERMINE IF THEY LIVE, DIE, OR GIVE
181    ; BIRTH. THE UPDATED CENTRAL LINE IS STORED BACK INTO DISPLAY
182    ; TO IMPROVE SPEED, WHEN PROCESSING THE CENTRAL 6 BITS IN A BYTE
183    ; THE ENTIRE BYTE AND ITS NEIGHBORS ARE CHECKED FOR ZERO.
184    ; IF ALL ARE ZERO, THE 6 BITS ARE SKIPPED.
185
186
187 0131 A000 LFBUF: LDY #0      ; INITIALIZE BYTE ADDRESS
188 0133 A207 LFBUF1: LDX #7      ; PREPARE FOR THE NEXT BYTE
189 0135 A900 STA NGEN          ; ZERO NEXT GENERATION BYTE
190 0137 8504
191 0139 E006 LFBUF2: CPX #6      ; TEST IF TO PROCESS BIT 6
192 013B D00D BNE LFBUF3          ; JUMP IF NOT
193 013D B92100 LDA TR,Y          ; TEST IF CENTRAL BYTE AND ITS NEIGHBORS
194 0140 194900 ORA CR,Y          ; ARE ALL ZEROES MEANING THAT NO CHANGE IS
195 0143 197100 ORA BR,Y          ; POSSIBLE IN THE CENTRAL 6 BITS OF THE
196 0146 D002 BNE LFBUF3          ; CURRENT BYTE
197 0148 A200 LDX #0      ; IF ZEROES, SKIP 6 CENTRAL BITS
198 014A 207501 LFBUF3: JSR NCNTC ; COUNT NEIGHBORS
199 014D A502 LDA NCNT
200 014F F01B BEQ LFBUF6          ; JUMP IF EXACTLY 3 LIVE NEIGHBORS
201 0151 3004 BMI LFBUF4          ; JUMP IF MORE THAN 3 LIVE NEIGHBORS
202 0153 C901 CMP #1
203 0155 F00D BEQ LFBUF5          ; JUMP IF EXACTLY 2 LIVE NEIGHBORS
204 0157 CA LFBUF4: DEX          ; DECREMENT BIT NUMBER
205 0158 10DF BPL LFBUF2          ; GO PROCESS NEXT BIT IF NOT DONE WITH BYT
206 015A A504 LDA NGEN          ; STORE NEXT GENERATION BYTE INTO DISPLAY
207 015C 9105 STA (ADP1),Y          ; MEMORY
208 015E C8 INY
209 015F C028 CPY #40          ; TEST IF DONE
210 0161 D000 BNE LFBUF1          ; LOOP IF NOT
211 0163 60 RTS
212
213 0164 B94900 LFBUF5: LDA CR,Y          ; WHEN EXACTLY 2 NEIGHBORS, TEST CURRENT
214 0167 3517 AND MSK,X          ; CELL
215 0169 4C6E01 JMP LFBUF7          ; NEW CELL IF CURRENT CELL IS ALIVE
216
217 016C B517 LFBUF6: LDA MSK,X          ; CREATE A CELL IN THE NEXT GENERATION
218 016E 0504 LFBUF7: ORA NGEN
219 0170 8504 STA NGEN
220 0172 4C5701 JMP LFBUF4
221

```

VMLIF VISIBLE MEMORY LIFE
NEIGHBOR COUNT ROUTINE

```

.PAGE 'NEIGHBOR COUNT ROUTINE'
222
223
224
225
226
227
228
229 0175 8401 NCNTC: STY NCYSV ; SAVE Y
230 0177 A903 LDA #3      ; INITIALIZE THE NEIGHBOR COUNT
231 0178 8502 STA NCNT
232 0178 B92100 N1: LDA TR,Y          ; CHECK CELLS DIRECTLY ABOVE AND BELOW
233 017E 3517 AND MSK,X          ; CENTRAL CELL FIRST
234 0180 F002 BEQ N2
235 0182 C602 DEC NCNT
236 0184 B97100 N2: LDA BR,Y          ; TEST COLUMN OF 3 LEFT CELLS NEXT
237 0187 3517 AND MSK,X
238 0189 F002 BEQ N3
239 0188 C602 DEC NCNT
240 0180 E000 N3: CPX #0      ; SKIP AHEAD IF IN THE SAME BYTE
241 018F D001 BNE N3A
242 0191 88 DEY ; OTHERWISE MOVE 1 BYTE LEFT
243 0192 B92100 N3A: LDA TR,Y          ; OTHERWISE MOVE 1 BYTE LEFT
244 0195 3516 AND MSK-1,X
245 0197 F002 BEQ N4
246 0199 C602 DEC NCNT
247 0198 B94900 N4: LDA CR,Y          ; QUICK EXIT IF MORE THAN 3 NEIGHBORS
248 019E 3516 AND MSK-1,X
249 01A0 F004 BEQ N5
250 01A2 C602 DEC NCNT
251 01A4 302F BMI NCXIT
252 01A6 B97100 N5: LDA BR,Y          ; QUICK EXIT IF MORE THAN 3 NEIGHBORS
253 01A9 3516 AND MSK-1,X
254 01AB F004 BEQ N6
255 01AD C602 DEC NCNT
256 01AF 3024 BMI NCXIT
257 01B1 A401 N6: LDY NCYSV ; RESTORE Y
258 01B3 E007 CPX #7      ; TEST COLUMN OF 3 RIGHT CELLS LAST
259 01B5 D001 BNE N6A
260 01B7 C8 INY ; SKIP AHEAD IF IN THE SAME BYTE
261 01B8 B92100 N6A: LDA TR,Y          ; OTHERWISE MOVE 1 BYTE RIGHT
262 01B8 3518 AND MSK+1,X
263 01BD F004 BEQ N7
264 01BF C602 DEC NCNT
265 01C1 3012 BMI NCXIT
266 01C3 B94900 N7: LDA CR,Y          ; QUICK EXIT IF MORE THAN 3 NEIGHBORS
267 01C6 3518 AND MSK+1,X
268 01C8 F002 BEQ N8
269 01CA C602 DEC NCNT
270 01CC B97100 N8: LDA BR,Y          ; RESTORE Y
271 01CF 3518 AND MSK+1,X
272 01D1 F002 BEQ NCXIT
273 01D3 C602 DEC NCNT
274 01D5 A401 NCXIT: LDY NCYSV ; AND RETURN
275 01D7 60 RTS

```

VMLIF VISIBLE MEMORY LIFE
CELL LINE MOVE ROUTINES

```

.PAGE 'CELL LINE MOVE ROUTINES'
277    ; ROLL THE THREE LINE BUFFERS UP ONE POSITION
278    ; AND BRING IN A NEW LINE FROM DISPLAY MEMORY STARTING AT
279    ; (ADP1)+80 PRESERVES INDEX REGISTERS
280
281 01D8   .= X'200
282 0200 98  ROLL: TYA      ; SAVE INDEX Y
283 0201 48  PHA
284 0202 A050 LDY #80      ; INITIALIZE INDEX
285 0204 B9F9FF ROLL1: LDA CR-80,Y ; ROLL A BYTE
286 0207 9901FF STA TR-80,Y
287 020A B92100 LDA BR-80,Y
288 020D 99F9FF STA CR-80,Y
289 0210 B105 LDA (ADP1),Y
290 0212 992100 STA BR-80,Y
291 0215 C8  INY      ; INCREMENT INDEX
292 0216 C078 CPY #120     ; TEST IF 40 BYTES ROLLED
293 0218 D0EA BNE ROLL1   ; LOOP IF NOT
294 021A 68  PLA      ; RESTORE Y
295 021B A8  TAY
296 021C 60  RTS      ; RETURN
297
298    ; PRIME THE LINE BUFFERS WITH THE FIRST THREE LINES OF DISPLAY
299    ; MEMORY
300    ; MOVES 120 BYTES STARTING AT (ADP1) INTO LINE BUFFERS STARTING
301    ; AT TR
302
303 021D 98  PRIME: TYA      ; SAVE INDEX Y
304 021E 48  PHA
305 021F A077 LDY #119     ; INITIALIZE INDEX
306 0221 B105 PRIME1: LDA (ADP1),Y ; MOVE A BYTE
307 0223 992100 STA TR,Y
308 0226 88  DEY      ; DECREMENT INDEX
309 0227 10F8 BPL PRIME1   ; LOOP IF NOT DONE
310 0229 68  PLA      ; RESTORE Y
311 022A A8  TAY
312 022B 60  RTS      ; RETURN
313
314    ; CLEAR DISPLAY MEMORY ROUTINE
315
316 022C A000 CLEAR: LDY #0       ; INITIALIZE ADDRESS POINTER
317 022E 8405 STY ADP1      ; AND ZERO INDEX Y
318 0230 A500 LDA VMORG
319 0232 8506 STA ADP1+1
320 0234 18  CLC
321 0235 6920 ADC #X'20
322 0237 AA  TAX
323 0238 98  CLEAR1: TYA      ; CLEAR A BYTE
324 0239 9105 STA (ADP1),Y
325 023B E605 INC ADP1      ; INCREMENT ADDRESS POINTER
326 023D D0F9 BNE CLEAR1
327 023F E606 INC ADP1+1
328 0241 E406 CPX ADP1+1
329 0243 D0F3 BNE CLEAR1
330 0245 60  RTS      ; RETURN

```

VMLIF VISIBLE MEMORY LIFE
GRAPHICS ROUTINES FOR GENERATING THE INITIAL PATTERN

12

```

.PAGE 'GRAPHICS ROUTINES FOR GENERATING THE INITIAL PATTERN'
332    ; PIXADR - FIND THE BYTE ADDRESS AND BIT NUMBER OF PIXEL AT
333    ; XICORD,YICORD
334    ; PUTS BYTE ADDRESS IN ADP1 AND BIT NUMBER (BIT 0 IS LEFTMOST)
335    ; IN BTPT.
336    ; DOES NOT CHECK MAGNITUDE OF COORDINATES FOR MAXIMUM SPEED
337    ; PRESERVES X AND Y REGISTERS, DESTROYS A
338    ; BYTE ADDRESS = VMORG*256+(199-YICORD)*40+INT(XCORD/8)
339    ; BIT ADDRESS = REM(XCORD/8)
340    ; OPTIMIZED FOR SPEED THEREFORE CALLS TO A DOUBLE SHIFT ROUTINE
341    ; ARE NOT DONE
342
343 0246 A50A PIXADR: LDA XICORD ; COMPUTE BIT ADDRESS FIRST
344 0248 8505 STA ADP1   ; ALSO TRANSFER XICORD TO ADP1
345 024A 2907 AND #X'07 ; WHICH IS SIMPLY THE LOW 3 BITS OF X
346 024C 8509 STA BTPT
347 024E A508 LOA XICORD+1 ; FINISH TRANSFERRING XICORD TO ADP1
348 0250 8506 STA ADP1+1
349 0252 4606 LSR ADP1+1 ; DOUBLE SHIFT ADP1 RIGHT 3 TO GET
350 0254 6605 ROR ADP1   ; INT(XCORD/8)
351 0256 4606 LSR ADP1+1
352 0258 6605 ROR ADP1
353 025A 4606 LSR ADP1+1
354 025C 6605 ROR ADP1
355 025E A9C7 LDA #199   ; TRANSFER (199-YICORD) TO ADP2
356 0260 38 SEC
357 0261 E50C SBC YICORD ; AND TEMPORARY STORAGE
358 0263 8507 STA ADP2
359 0265 8512 STA TEMP
360 0267 A900 LDA #0
361 0269 E500 SBC YICORD+1
362 026B 8508 STA ADP2+1
363 026D 8513 STA TEMP+1
364 026F 0607 ASL ADP2 ; COMPUTE 40*(199-YICORD)
365 0271 2608 ROL ADP2+1 ; 2*(199-YICORD)
366 0273 0607 ASL ADP2
367 0275 2608 ROL ADP2+1 ; 4*(199-YICORD)
368 0277 A507 LDA ADP2 ; ADD IN TEMPORARY SAVE OF (199-YICORD)
369 0279 18 CLC
370 027A 6512 ADC TEMP
371 027C 8507 STA ADP2
372 027E A508 LDA ADP2+1
373 0280 6513 ADC TEMP+1
374 0282 8508 STA ADP2+1 ; 5*(199-YICORD)
375 0284 0607 ASL ADP2 ; 10*(199-YICORD)
376 0286 2608 ROL ADP2+1
377 0288 0607 ASL ADP2 ; 20*(199-YICORD)
378 028A 2608 ROL ADP2+1
379 028C 0607 ASL ADP2 ; 40*(199-YICORD)
380 028E 2608 ROL ADP2+1
381 0290 A507 LDA ADP2 ; ADD IN INT(XICORD/8) COMPUTED EARLIER
382 0292 18 CLC
383 0293 6505 ADC ADP1
384 0295 8505 STA ADP1
385 0297 A508 LDA ADP2+1

```

VMLIF VISIBLE MEMORY LIFE
GRAPHICS ROUTINES FOR GENERATING THE INITIAL PATTERN

```

386 0299 6506    ADC  ADP1+1
387 0298 6500    ADC  VMORG      ; ADD IN VMORG*256
388 0290 8506    STA  ADP1+1      ; FINAL RESULT
389 029F 60       RTS             ; RETURN
390
391 ; STPIX - SETS THE PIXEL AT XICORD,YICORD TO A ONE (WHITE DOT)
392 ; DOES NOT ALTER XICORD OR YICORD
393 ; PRESERVES X AND Y
394 ; ASSUMES IN RANGE COORDINATES
395
396 02A0 204602    STPIX: JSR  PIXADR    ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
397 ; INTO ADP1
398 02A3 98        TYA             ; SAVE Y
399 02A4 48        PHA             ; PRESERVE X
400 02A5 A409      LDY  BTPT        ; GET BIT NUMBER IN Y
401 02A7 B9E502    LDA  MSKTB1,Y   ; GET A BYTE WITH THAT BIT =1, OTHERS =0
402 02AA A000      LDY  #0          ; ZERO Y
403 02AC 1105      ORA  (ADP1),Y   ; COMBINE THE BIT WITH THE ADDRESSED VM
404 ; BYTE
405 02AE 4CBF02    JMP  CLPIX1     ; GO STORE RESULT, RESTORE Y, AND RETURN
406
407 ; CLPIX - CLEARS THE PIXEL AT XICORD,YICORD TO A ZERO (BLACK DOT)
408 ; DOES NOT ALTER XICORD OR YICORD
409 ; PRESERVES X AND Y
410 ; ASSUMES IN RANGE COORDINATES
411
412 02B1 204602    CLPIX: JSR  PIXADR    ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
413 ; INTO ADP1
414 02B4 98        TYA             ; SAVE Y
415 02B5 48        PHA             ; PRESERVE X
416 02B6 A409      LDY  BTPT        ; GET BIT NUMBER IN Y
417 02B8 B9ED02    LDA  MSKTB2,Y   ; GET A BYTE WITH THAT BIT =0, OTHERS =1
418 02BB A000      LDY  #0          ; ZERO Y
419 02BD 3105      AND  (ADP1),Y   ; REMOVE THE BIT FROM THE ADDRESSED VM
420 02BF 9105      CLPIX1: STA  (ADP1),Y   ; BYTE
421 02C1 68        PLA             ; RESTORE Y
422 02C2 A8        TAY             ; PRESERVE Y
423 02C3 60        RTS             ; AND RETURN
424
425 ; WRPIX - SETS THE PIXEL AT XICORD,YICORD ACCORDING TO THE STATE
426 ; OF BIT 0 (RIGHTMOST) OF A
427 ; DOES NOT ALTER XICORD OR YICORD
428 ; PRESERVES X AND Y
429 ; ASSUMES IN RANGE COORDINATES
430
431 02C4 2CCB02    WRPIX: BIT  WRPIXM    ; TEST LOW BIT OF A
432 02C7 F0E8      BEQ  CLPIX      ; JUMP IF A ZERO TO BE WRITTEN
433 02C9 D005      BNE  STPIX      ; OTHERWISE WRITE A ONE
434
435 02CB 01        WRPIXM: .BYTE 1      ; BIT TEST MASK FOR BIT 0
436
437 ; RDPIX - READS THE PIXEL AT XICORD,YICORD AND SETS A TO ALL
438 ; ZEROES IF IT IS A ZERO OR TO ALL ONES IF IT IS A ONE
439 ; LOW BYTE OF ADP1 IS EQUAL TO A ON RETURN
440 ; DOES NOT ALTER XICORD OR YICORD

```

VMLIF VISIBLE MEMORY LIFE
GRAPHICS ROUTINES FOR GENERATING THE INITIAL PATTERN

```

441 ; PRESERVES X AND Y
442 ; ASSUMES IN RANGE COORDINATES
443
444 02CC 204602    RDPIX: JSR  PIXADR    ; GET BYTE AND BIT ADDRESS OF PIXEL
445 02CF 98        TYA             ; SAVE Y
446 02D0 48        PHA             ; PRESERVE X
447 02D1 A000      LDY  #0          ; GET ADDRESSED BYTE FROM VM
448 02D3 B105      LDA  (ADP1),Y   ; GET BIT NUMBER IN Y
449 02D5 A409      LDY  BTPT        ; CLEAR ALL BUT ADDRESSED BIT
450 02D7 39E502    AND  MSKTB1,Y   ; SKIP AHEAD IF IT WAS A ZERO
451 02DA F002      BEQ  RDPIX1     ; SET TO ALL ONES IF IT WAS A ONE
452 02DC A9FF      LDA  #X'FF      ; SAVE A TEMPORARILY IN ADP1 WHILE
453 02DE 8505      RDPIX1: STA  ADP1      ; RESTORING Y
454 02E0 68        PLA             ; RESTORE Y
455 02E1 A8        TAY             ; PRESERVE Y
456 02E2 A505      LDA  ADP1      ; RETURN
457 02E4 60        RTS             ; PRESERVE Y
458
459 ; MASK TABLES FOR INDIVIDUAL PIXEL SUBROUTINES
460 ; MSKTB1 IS A TABLE OF 1 BITS CORRESPONDING TO BIT NUMBERS
461 ; MSKTB2 IS A TABLE OF 0 BITS CORRESPONDING TO BIT NUMBERS
462
463 02E5 80402010  MSKTB1: .BYTE X'80,X'40,X'20,X'10
464 02E9 08040201  .BYTE X'08,X'04,X'02,X'01
465 02ED 7FBDFEEF  MSKTB2: .BYTE X'7F,X'BF,X'DF,X'EF
466 02F1 F7FBDFEEF .BYTE X'F7,X'FB,X'FD,X'FE
467
468 ; SDRAW - SIMPLIFIED DRAW ROUTINE
469 ; DRAWS A LINE FROM XICORD,YICORD TO X2CORD,Y2CORD
470 ; WHEN DONE COPIES X2CORD AND Y2CORD INTO XICORD AND YICORD
471 ; RESTRICTED TO HORIZONTAL, VERTICAL, AND 45 DEGREE DIAGONAL
472 ; LINES (SLOPE=1)
473 ; PRESERVES BOTH INDEX REGISTERS
474
475 02F5 8A        SDRAW: TXA           ; SAVE INDEX REGS
476 02F6 48        PHA             ; PRESERVE X
477 02F7 98        TAY             ; PRESERVE Y
478 02F8 48        PHA             ; PRESERVE X
479 02F9 20A002    JSR  STPIX      ; PUT A DOT AT INITIAL ENDPOINT
480 02FC A000      SDRAW1: LDY  #0      ; CLEAR "SOMETHING DONE" FLAG
481 02FE A200      LDX  #0      ; UPDATE X COORDINATE
482 0300 201303    JSR  UPDC      ; UPDATE Y COORDINATE
483 0303 A202      LDX  #YICORD-XICORD;UPDATE Y COORDINATE
484 0305 201303    JSR  UPDC      ; UPDATE Y COORDINATE
485 0308 20A002    JSR  STPIX      ; PUT A DOT AT INTERMEDIATE POINT
486 0308 88        DEY             ; TEST IF EITHER COORDINATE CHANGED
487 030C 10EE      BPL  SDRAW1     ; ITERATE AGAIN IF SO
488 030E 68        PLA             ; RESTORE INDEX REGISTERS
489 030F A8        TAY             ; PRESERVE Y
490 0310 68        PLA             ; PRESERVE X
491 0311 AA        TAX             ; PRESERVE Y
492 0312 60        RTS             ; RETURN
493
494 ; INTERNAL SUBROUTINE FOR UPDATING COORDINATES
495

```

VMLIF VISIBLE MEMORY LIFE
GRAPHICS ROUTINES FOR GENERATING THE INITIAL PATTERN

```

496 0313 B50F    UPDC:   LDA    X2CORD+1,X ; COMPARE ENDPOINT WITH CURRENT POSITION
497 0315 D50B    CMP    X1CORD+1,X
498 0317 9012    BCC    UPDC3 ; JUMP IF CURRENT POSITION IS LARGER
499 0319 D008    BNE    UPDC1 ; JUMP IF ENDPOINT IS LARGER
500 031B B50E    LDA    X2CORD,X
501 031D D50A    CMP    X1CORD,X
502 031F 900A    BCC    UPDC3 ; JUMP IF CURRENT POSITION IS LARGER
503 0321 F011    BEQ    UPDC5 ; GO RETURN IF EQUAL
504 0323 F60A    INC    X1CORD,X ; ENDPOINT IS LARGER, INCREMENT CURRENT
505 0325 D002    BNE    UPDC2 ; POSITION
506 0327 F60B    INC    X1CORD+1,X
507 0329 C8    UPDC2:  INY    ; SET "DONE SOMETHING" FLAG
508 032A 60    RTS    ; RETURN
509 032B B50A    LDA    X1CORD,X ; CURRENT POSITION IS LARGER, DECREMENT
510 032D D002    BNE    UPDC4 ; CURRENT POSITION
511 032F D60B    DEC    X1CORD+1,X
512 0331 D60A    UPDC4:  DEC    X1CORD,X
513 0333 C8    INY    ; SET "DONE SOMETHING" FLAG
514 0334 60    UPDC5:  RTS    ; RETURN
515

```

VMLIF VISIBLE MEMORY LIFE
COORDINATE LIST FOR DRAWING INITIAL FIGURE

.PAGE 'COORDINATE LIST FOR DRAWING INITIAL FIGURE'
COORDINATE LIST DEFINING THE INITIAL PATTERN FOR LIFE
EACH VERTEX IN THE FIGURE IS REPRESENTED BY 4 BYTES
THE FIRST TWO BYTES ARE THE X COORDINATE OF THE NEXT ENDPOINT
AND THE NEXT TWO BYTES ARE THE Y COORDINATE.
IF THE HIGH BYTE OF X HAS THE SIGN BIT ON, A MOVE FROM THE
CURRENT POSITION TO THE NEW POSITION IS DONE (THE SIGN BIT IS
IS DELETED BEFORE MOVING)
IF THE HIGH BYTE OF X HAS THE SIGN BIT OFF, A DRAW FROM THE
CURRENT POSITION TO THE NEW POSITION IS DONE.
IF THE HIGH BYTE OF X = X'FF, IT IS THE END OF THE LIST.

516	:	.WORD 56+X'8000,60 ; 1 MOVE
517	:	.WORD 56,140 ; 2 DRAW
518	:	.WORD 72,140 ; 3 DRAW
519	:	.WORD 72,76 ; 4
520	:	.WORD 104,76 ; 5
521	:	.WORD 104,60 ; 6
522	:	.WORD 56,60 ; 7
523	:	.WORD 120+X'8000,60 ; 8 MOVE
524	:	.WORD 120,140 ; 9
525	:	.WORD 136,140 ; 10
526	:	.WORD 136,60 ; 11
527 0335 38803C00	LIST:	.WORD 120,60 ; 12
528 0339 38008C00		.WORD 152+X'8000,60 ; 13 MOVE
529 033D 48008C00		.WORD 152,140 ; 14
530 0341 48004C00		.WORD 200,140 ; 15
531 0345 68004C00		.WORD 200,124 ; 16
532 0349 68003C00		.WORD 168,124 ; 17
533 034D 38003C00		.WORD 168,108 ; 18
534 0351 78803C00		.WORD 192,108 ; 19
535 0355 78008C00		.WORD 192,92 ; 20
536 0359 88008C00		.WORD 168,92 ; 21
537 035D 88003C00		.WORD 168,60 ; 22
538 0361 78003C00		.WORD 152,60 ; 23
539 0365 98803C00		.WORD 216+X'8000,60 ; 24 MOVE
540 0369 98008C00		.WORD 216,140 ; 25
541 036D C8008C00		.WORD 264,140 ; 26
542 0371 C8007C00		.WORD 264,124 ; 27
543 0375 A8007C00		.WORD 232,124 ; 28
544 0379 A8006C00		.WORD 232,108 ; 29
545 037D C0006C00		.WORD 256,108 ; 30
546 0381 C0005C00		.WORD 256,92 ; 31
547 0385 A8005C00		.WORD 232,92 ; 32
548 0389 A8003C00		.WORD 232,76 ; 33
549 038D 98003C00		.WORD 264,76 ; 34
550 0391 D8803C00		.WORD 264,60 ; 35
551 0395 D8008C00		.WORD 216,60 ; 36
552 0399 08018C00		.WORD X'FFFF ; END OF LIST
553 039D 08017C00		
554 03A1 E8007C00		
555 03A5 E8006C00		
556 03A9 00016C00		
557 03AD 00015C00		
558 03B1 E8005C00		
559 03B5 E8004C00		
560 03B9 08014C00		
561 03BD 08013C00		
562 03C1 D8003C00		
563 03C5 FFFF		

VMLIF VISIBLE MEMORY LIFE
KEYBOARD PATTERN ENTRY ROUTINES

```

565 .PAGE 'KEYBOARD PATTERN ENTRY ROUTINES'
566 ; KEYBOARD PATTERN ENTRY ROUTINES
567 ; USES THE KIM KEYBOARD AND A CURSOR TO SIMPLIFY THE ENTRY
568 ; OF INITIAL LIFE PATTERNS
569 03C7 A900 KYPT: LDA #0 ; SET INITIAL CURSOR POSITION IN CENTER
570 03C9 8508 STA XICORD+1 ; OF SCREEN
571 03CB 850D STA YICORD+1
572 03CD A9A0 LDA #160
573 03CF 850A STA XICORD
574 03D1 A964 LDA #100
575 03D3 850C STA YICORD
576 03D5 20E200 JSR CSRINS ; INSERT A CURSOR ON THE SCREEN
577 03D8 A932 KYPT0: LDA #DBC0LA ; RESET THE DEBOUNCE COUNT
578 03DA 8502 STA DBCNT
579 03DC E614 KYPT1: INC FLASHC ; DOUBLE INCREMENT CURSOR FLASH COUNT
580 03DE D002 BNE KYPT2
581 03E0 E615 INC FLASHC+1
582
583 ; GENERATE A 25% DUTY CURSOR IF CELL IS DEAD AND 75% IF ALIVE
584
585 03E2 A515 KYPT2: LDA FLASHC+1 ; GET HIGH BYTE OF FLASH COUNTER
586 03E4 4A LSRA ; COMPUTE LOGICAL "AND" OF BITS 0 AND 1
587 03E5 2515 AND FLASHC+1 ; IN ACC BIT 0
588 03E7 4503 EOR REALST ; EXCLUSIVE-OR WITH REAL STATE OF CELL
589 03E9 20C402 JSR WRPIX ; DISPLAY THE CURSOR
590
591 ; READ KIM KEYBOARD AND DETECT ANY CHANGE IN KEYS PRESSED
592
593 03EC 206A1F JSR GETKEY ; GET CURRENTLY PRESSED KEY
594 03EF C501 CMP LSTKEY ; TEST IF SAME AS BEFORE
595 03F1 F0E5 BEQ KYPT0 ; IGNORE IF SO
596 03F3 C602 DEC DBCNT ; IF DIFFERENT, DECREMENT AND TEST
597 03F5 10E5 BPL KYPT1 ; DEBOUNCE COUNT AND IGNORE KEY IF NOT RUN
598 ; OUT
599 03F7 8501 STA LSTKEY ; AFTER DEBOUNCE, UPDATE KEY LAST PRESSED
600 03F9 4C8017 JMP KYPT6 ; AND GO PROCESS THE KEYSTROKE
601
602 03FC .= X'1780 ; CONTINUE PROGRAM IN 6530 RAM
603
604 1780 C901 KYPT6: CMP #1 ; TEST "1" KEY
605 1782 F01B BEQ CSRD ; JUMP IF CURSOR DOWN
606 1784 C909 CMP #9 ; TEST "9" KEY
607 1786 F01F BEQ CSRU ; JUMP IF CURSOR UP
608 1788 C904 CMP #4 ; TEST "4" KEY
609 178A F023 BEQ CSRL ; JUMP IF CURSOR LEFT
610 178C C906 CMP #6 ; TEST "6" KEY
611 178E F02D BEQ CSRR ; JUMP IF CURSOR RIGHT
612 1790 C913 CMP #19 ; TEST "GO" KEY
613 1792 F043 BEQ GO ; JUMP IF GO KEY
614 1794 C912 CMP #18 ; TEST "+" KEY
615 1796 F034 BEQ SETCEL ; JUMP IF SET CELL KEY
616 1798 C90F CMP #15 ; TEST "F" KEY
617 179A F034 BEQ CLRCEL ; JUMP IF CLEAR CELL KEY
618 179C 4CD803 JMP KYPT0 ; IGNORE ANY OTHER KEYS

```

VMLIF VISIBLE MEMORY LIFE
KEYBOARD PATTERN ENTRY ROUTINES

```

619
620 179F 20E800 CSRD: JSR CSRDEL ; DELETE EXISTING CURSOR
621 17A2 C60C DEC YICORD ; DECREMENT Y COORDINATE FOR CURSOR DOWN
622 17A4 4CC617 JMP CSRMOV
623
624 17A7 20E800 CSRU: JSR CSRDEL ; DELETE EXISTING CURSOR
625 17AA E60C INC YICORD ; INCREMENT Y COORDINATE FOR CURSOR UP
626 17AC 4CC617 JMP CSRMOV
627
628 17AF 20E800 CSRL: JSR CSRDEL ; DELETE EXISTING CURSOR
629 17B2 A50A LDA XICORD ; DECREMENT X COORDINATE FOR CURSOR LEFT
630 17B4 D002 BNE CSRL1
631 17B6 C60B DEC XICORD+1
632 17B8 C60A CSRL1: DEC YICORD
633 17BA 4CC617 JMP CSRMOV
634
635 17BD 20E800 CSRR: JSR CSRDEL ; DELETE EXISTING CURSOR
636 17C0 E60A INC XICORD ; INCREMENT X COORDINATE FOR CURSOR RIGHT
637 17C2 D002 BNE CSRMOV
638 17C4 E60B INC XICORD+1
639
640 17C6 20E200 CSRMOV: JSR CSRINS ; INSERT CURSOR AT NEW LOCATION
641 17C9 4CD803 JMP KYPT0 ; GO BACK TO KEYBOARD INPUT LOOP
642
643 17CC A9FF SETCEL: LDA #X'FF ; SET REAL CELL STATE TO LIVE
644 17CE D002 BNE CLRCL1
645
646 17D0 A900 CLRCEL: LDA #0 ; SET REAL CELL STATE TO DEAD
647 17D2 8503 CLRCL1: STA REALST ; GO BACK TO KEYBOARD INPUT LOOP
648 17D4 4CD803 JMP KYPT0 ; TEST THE CURSOR
649
650 17D7 20E800 GO: JSR CSRDEL ; DELETE CURSOR AND RESTORE THE CELL UNDER
651
652 17DA 4C0001 JMP LIFE ; THE CURSOR
653
654
655 0000 .END

```

NO ERROR LINES

SDTXT SIMPLIFIED DISPLAY TE
SIMPLIFIED VISABLE MEMORY TEXT DISPLAY SUBROUTINE

```

3 . PAGE 'SIMPLIFIED VISABLE MEMORY TEXT DISPLAY SUBROUTINE'
4 . THIS SUBROUTINE TURNS THE VISABLE MEMORY INTO A DATA DISPLAY
5 . TERMINAL (GLASS TELETYPE).
6 . CHARACTER SET IS 96 FULL ASCII UPPER AND LOWER CASE.
7 . CHARACTER MATRIX IS 5 BY 7 SET INTO A 6 BY 9 RECTANGLE.
8 . LOWER CASE IS REPRESENTED AS SMALL (5 BY 5) CAPITALS.
9 . SCREEN CAPACITY IS 22 LINES OF 53 CHARACTERS FOR FULL SCREEN
10 . OR 11 LINES FOR HALF SCREEN.
11 . CURSOR IS A NON-BLINKING UNDERLINE.
12 . CONTROL CODES RECOGNIZED:
13 . CR X'0D SETS CURSOR TO LEFT SCREEN EDGE
14 . LF X'0A MOVES CURSOR DOWN ONE LINE, SCROLLS
15 . DISPLAY UP ONE LINE IF ALREADY ON BOTTOM
16 . LINE
17 . BS X'08 MOVES CURSOR ONE CHARACTER LEFT, DOES
18 . NOTHING IF ALREADY AT LEFT SCREEN EDGE
19 . FF X'0C CLEARS SCREEN AND PUTS CURSOR AT TOP LEFT
20 . OF SCREEN, SHOULD BE CALLED FOR
21 . INITIALIZATION
22 . ALL OTHER CONTROL CODES IGNORED.
23 . ENTER WITH CHARACTER TO BE DISPLAYED IN A.
24 . X AND Y PRESERVED.
25 . 3 BYTES OF RAM STORAGE REQUIRED FOR KEEPING TRACK OF THE
26 . CURSOR
27 . 4 BYTES OF TEMPORARY STORAGE IN BASE PAGE REQUIRED FOR ADDRESS
28 . POINTERS. (CAN BE DESTROYED BETWEEN CALLS TO SDXT)
29 . 4 BYTES OF TEMPORARY STORAGE ANYWHERE (CAN BE DESTROYED
30 . BETWEEN CALLS TO SDXT)
31 . * **** VMORG #MUST# BE SET TO THE PAGE NUMBER OF THE VISIBLE *
32 . * MEMORY BEFORE CALLING SDXT ****
33 .
34 . ; GENERAL EQUATES
35 .
36 1F40 NLOC = 8000 ; NUMBER OF VISIBLE LOCATIONS
37 0009 CHHI = 9 ; CHARACTER WINDOW HEIGHT
38 0006 CHWID = 6 ; CHARACTER WINDOW WIDTH
39 0035 NCHR = 320/CHWID ; NUMBER OF CHARACTERS PER LINE
40 0016 NLIN = NLOC/40/CHHI ; NUMBER OF TEXT LINES
41 1D88 NSCRL = NLIN-1*CHHI*40 ; NUMBER OF LOCATIONS TO SCROLL
42 0188 NCLR = NLOC-NSCRL ; NUMBER OF LOCATIONS TO CLEAR AFTER SCROL
43 .
44 . ; BASE PAGE TEMPORARY STORAGE
45 .
46 0000 ADP1: .= X'EA
47 00EA ADP1: .= .+ 2 ; ADDRESS POINTER 1
48 00EC ADP2: .= .+ 2 ; ADDRESS POINTER 2
49 .
50 . ; GENERAL TEMPORARY STORAGE
51 .
52 00EE BTPT: .= .+ 1 ; BIT NUMBER TEMPORARY STORAGE
53 5B00 DCNT1: .= .+ 2 ; DOUBLE PRECISION COUNTER
54 5B01 MRGT1: .= .+ 1 ; TEMPORARY STORAGE FOR MERGE

```

SDTXT SIMPLIFIED DISPLAY TE
SIMPLIFIED VISABLE MEMORY TEXT DISPLAY SUBROUTINE

```

57 .
58 . ; PERMANENT RAM STORAGE
59 .
60 5B04 CSRX: .= .+ 1 ; CURRENT CHARACTER NUMBER (0=LEFT CHAR)
61 5B05 CSRY: .= .+ 1 ; CURRENT LINE NUMBER (0=TOP LINE)
62 5B06 VMORG: .= .+ 1 ; FIRST PAGE NUMBER OF VISABLE MEMORY
63 .
64 5B07 48 SDXT: PHA ; SAVE REGISTERS
65 5B08 8A TXA
66 5B09 48 PHA
67 5B0A 98 TYA
68 5B0B 48 PHA
69 5B0C A900 LDA #0 ; CLEAR UPPER ADP2
70 5B0E B5ED STA ADP2+1
71 5B10 BA TSX
72 5B11 BD0301 LDA X'103,X
73 5B14 297F AND #X'7F ; INSURE 7 BIT ASCII INPUT
74 5B16 38 SEC
75 5B17 E920 SBC #X'20 ; TEST IF A CONTROL CHARACTER
76 5B19 3047 BMI SDTX10 ; JUMP IF SO
77 .
78 . ; CALCULATE TABLE ADDRESS FOR CHAR SHAPE AND PUT IT INTO ADP1
79 .
80 5B1B 85EC SDXT1: STA ADP2 ; SAVE CHARACTER CODE IN ADP2
81 5B1D 20225C JSR SADP2L ; COMPUTE 8*CHARACTER CODE IN ADP2
82 5B20 20225C JSR SADP2L
83 5B23 20225C JSR SADP2L
84 5B26 49FF EOR #X'FF ; NEGATE CHARACTER CODE
85 5B28 38 SEC ; SUBTRACT CHARACTER CODE FROM ADP2 AND
86 5B29 65EC ADC ADP2 ; PUT RESULT IN ADP1 FOR A FINAL RESULT OF
87 5B2B 85EA STA ADP1 ; 7*CHARACTER CODE
88 5B2D A5ED LDA ADP2+1
89 5B2F 69FF ADC #X'FF
90 5B31 85EB STA ADP1+1
91 5B33 A5EA LDA ADP1 ; ADD IN ORIGIN OF CHARACTER TABLE
92 5B35 18 CLC
93 5B36 6921 ADC #CHTB&X'FF
94 5B38 85EA STA ADP1
95 5B3A A5EB LDA ADP1+1
96 5B3C 6950 ADC #CHTB/256
97 5B3E 85EB STA ADP1+1 ; ADP1 NOW HAS ADDRESS OF TOP ROW OF
98 . ; CHARACTER SHAPE
99 . ; COMPUTE BYTE AND BIT ADDRESS OF FIRST SCAN LINE OF
100 . ; CHARACTER AT CURSOR POSITION
101 .
102 5B40 20355C JSR CSRTAD ; COMPUTE BYTE AND BIT ADDRESSES OF FIRST
103 . ; SCAN LINE OF CHARACTER AT CURSOR POS.
104 .
105 . ; SCAN OUT THE 7 CHARACTER ROWS
106 .
107 5B43 A000 LDY #0 ; INITIALIZE Y INDEX=FONT TABLE POINTER
108 5B45 B1EA STA (ADP1),Y ; GET A DOT ROW FROM THE FONT TABLE
109 5B47 20805C JSR MERGE ; MERGE IT WITH GRAPHIC MEMORY AT (ADP2)
110 5B4A 20275C JSR DN1SCN ; ADD 40 TO ADP2 TO MOVE DOWN ONE SCAN
111 . ; LINE IN GRAPHIC MEMORY

```

**SDXTX SIMPLIFIED DISPLAY TE
SIMPLIFIED VISABLE MEMORY TEXT DISPLAY SUBROUTINE**

```

112 5B4D C8      INY      ; BUMP UP POINTER INTO FONT TABLE
113 5B4E C007    CPY #7   ; TEST IF DONE
114 5B50 D0F3    BNE SDTX2 ; GO DO NEXT SCAN LINE IF NOT
115 5B52 AD045B   LDA CSRX ; DO A CURSOR RIGHT
116 5B55 C934    CMP #NCHR-1 ; TEST IF LAST CHARACTER ON THE LINE
117 5B57 1006    BPL SDTX3 ; SKIP CURSOR RIGHT IF SO
118 5B59 201A5C   JSR CSRCLR ; CLEAR OLD CURSOR
119 5B5C EE045B   INC CSRX ; MOVE CURSOR ONE POSITION RIGHT
120 5B5F 4CF85B   SDTX3: JMP SDTXRT ; GO INSERT CURSOR, RESTORE REGISTERS,
121           ; AND RETURN
122
123           ; INTERPRET CONTROL CODES
124
125 5B62 C9ED    SDTX10: CMP #X'0D-X'20 ; TEST IF CR
126 5B64 F00F    BEQ SDTXCR ; JUMP IF SO
127 5B66 C9EA    CMP #X'0A-X'20 ; TEST IF LF
128 5B68 F047    BEQ SDTXLF ; JUMP IF SO
129 5B6A C9E8    CMP #X'08-X'20 ; TEST IF BS
130 5B6E F012    BEQ SDTXCL ; JUMP IF SO
131 5B6E C9EC    CMP #X'0C-X'20 ; TEST IF FF
132 5B70 F01E    BEQ SDTXFF ; JUMP IF SO
133 5B72 4CF85B   JMP SDTXRT ; GO RETURN IF UNRECOGNIZABLE CONTROL
134
135 5B75 201A5C   SDTXCR: JSR CSRCLR ; CARRIAGE RETURN, FIRST CLEAR CURSOR
136 5B78 A900    LDA #0   ; ZERO CURSOR HORIZONTAL POSITION
137 5B7A 80045B   STA CSRX
138 5B7D 4CF85B   JMP SDTXRT ; GO SET CURSOR AND RETURN
139
140 5B80 201A5C   SDTXCL: JSR CSRCLR ; CURSOR LEFT, FIRST CLEAR CURSOR
141 5B83 AD045B   LDA CSRX ; GET CURSOR HORIZONTAL POSITION
142 5B86 C900    CMP #0   ; TEST IF AGAINST LEFT EDGE
143 5B88 F003    BEQ SDTX20 ; SKIP UPDATE IF SO
144 5B8A CE045B   DEC CSRX ; OTHERWISE DECREMENT CURSOR X POSITION
145 5B8D 4CF85B   SDTX20: JMP SDTXRT ; GO SET CURSOR AND RETURN
146
147 5B94 AD065B   SDTXFF: LDA VMORG ; FORM FEED, CLEAR SCREEN TO ZEROES
148 5B93 85ED    STA ADP2+1 ; TRANSFER VISABLE MEMORY ORIGIN ADDRESS
149 5B95 A900    LDA #0   ; TO ADP2
150 5B97 85EC    STA ADP2
151 5B99 A940    LDA #NLOC&X'FF ; SET COUNT OF LOCATIONS TO CLEAR IN DCNT1
152 5B99 80015B   STA DCNT1
153 5B9E A91F    LDA #NLOC/256
154 5BA0 80025B   STA DCNT1+1
155 5BA3 20015D   JSR FCLR ; CLEAR THE SCREEN
156 5BA6 A900    LDA #0
157 5BA8 80045B   STA CSRX ; PUT CURSOR IN UPPER LEFT CORNER
158 5BAB 80055B   STA CSRY
159 5BAE 4CF85B   JMP SDTXRT ; GO SET CURSOR AND RETURN
160
161 5BB1 201A5C   SDTXLF: JSR CSRCLR ; LINE FEED, FIRST CLEAR CURSOR
162 5BB4 AD055B   LDA CSRY ; GET CURRENT LINE POSITION
163 5BB7 C915    CMP #NLIN-1 ; TEST IF AT BOTTOM OF SCREEN
164 5BB9 1005    BPL SDTX40 ; GO SCROLL IF SO
165 5BBC EE055B   INC CSRY ; INCREMENT LINE NUMBER IF NOT AT BOTTOM
166 5BBE D038    BNE SDTXRT ; GO INSERT CURSOR AND RETURN

```

**SDXTX SIMPLIFIED DISPLAY TE
SIMPLIFIED VISABLE MEMORY TEXT DISPLAY SUBROUTINE**

```

167 5BC0 A900    SDTX40: LDA #0   ; SET UP ADDRESS POINTERS FOR MOVE
168 5BC2 85ED    STA ADP2 ; ADP1 = SOURCE FOR MOVE = FIRST BYTE OF
169 5BC4 AD065B   LDA VMORG ; SECOND LINE OF TEXT
170 5BC7 85ED    STA ADP2+1 ; ADP2 = DESTINATION FOR MOVE = FIRST BYTE
171 5BC9 18     CLC   ; IN VISIBLE MEMORY
172 5BCA 6901    ADC #CHHI*40/256
173 5BCC 85EB    STA ADP1+1
174 5BCE A968    LDA #CHHI*40&X'FF
175 5BD0 85EA    STA ADP1
176 5BD2 A988    LDA #NSCRL&X'FF ; SET NUMBER OF LOCATIONS TO MOVE
177 5BD4 80015B   STA DCNT1 ; LOW PART
178 5BD7 A91D    LDA #NSCRL/256 ; HIGH PART
179 5BD9 80025B   STA DCNT1+1
180 5BDC 20D35C   JSR FMOVE ; EXECUTE MOVE USING AN OPTIMIZED, HIGH
181           ; SPEED MEMORY MOVE ROUTINE
182
183
184 5BDF A988    LDA #NLIN-1*CHHI*40&X'FF ; SET ADDRESS POINTER
185 5BE1 85EC    STA ADP2 ; LOW BYTE
186 5BE3 A91D    LDA #NLIN-1*CHHI*40/256
187 5BE5 18     CLC   ; CLEAR LAST LINE OF TEXT
188 5BE6 6D065B   ADC VMORG
189 5BE9 85ED    STA ADP2+1 ; HIGH BYTE
190 5BE8 A988    LDA #NCLR&X'FF ; SET LOW BYTE OF CLEAR COUNT
191 5BED 80015B   STA DCNT1
192 5BF0 A901    LDA #NCLR/256 ; SET HIGH BYTE OF CLEAR COUNT
193 5BF2 80025B   STA DCNT1+1
194 5BF5 20015D   JSR FCLR ; CLEAR THE DESIGNATED AREA
195
196
197
198 5BF8 20125C   SDTXRT: JSR CSRSET ; RETURN SEQUENCE, INSERT CURSOR
199 5BF8 68     PLA   ; RESTORE REGISTERS FROM THE STACK
200 5BFC A8     TAY
201 5BFD 68     PLA
202 5BFF AA     TAX
203 5BFF 68     PLA
204 5C00 60     RTS
205           ; RETURN

```

SDXTXT SIMPLIFIED DISPLAY TE
SUBROUTINES FOR SDXTXT

```

206      ; COMPUTE ADDRESS OF BYTE CONTAINING LAST SCAN LINE OF
207      ; CHARACTER AT CURSOR POSITION
208      ; ADDRESS = CSRTAD+(CHHI-1)*40 SINCE CHHI IS A CONSTANT 9,
209      ; (CHHI-1)*40=320
210      ; BTPT HOLDS BIT ADDRESS, 0=LEFTMOST
211
212 5C01 20355C  CSRBAD: JSR CSRTAD    ; COMPUTE ADDRESS OF TOP OF CHARACTER CELL
213          ; FIRST
214 5C04 A5EC  LDA ADP2      ; ADD 320 TO RESULT = 8 SCAN LINES
215 5C06 18    CLC
216 5C07 6940  ADC #320&X'FF
217 5C09 85EC  STA ADP2
218 5C0B A5ED  LDA ADP2+1
219 5C0D 6901  ADC #320/256
220 5C0F 85ED  STA ADP2+1
221 5C11 60    RTS
222
223      ; SET CURSOR AT CURRENT POSITION
224
225 5C12 20015C CSRSET: JSR CSRBAD    ; GET BYTE AND BIT ADDRESS OF CURSOR
226 5C15 A9F8  LDA #X'FB    ; DATA = UNDERLINE CURSOR
227 5C17 4C805C CSRST1: JMP MERGE     ; MERGE CURSOR WITH GRAPHIC MEMORY
228          ; AND RETURN
229
230      ; CLEAR CURSOR AT CURRENT POSITION
231
232 5C1A 20015C CSRCLR: JSR CSRBAD    ; GET BYTE AND BIT ADDRESS OF CURSOR
233 5C1D A900  LDA #0       ; DATA = BLANK DOT ROW
234 5C1F 4C805C JMP MERGE     ; REMOVE DOT ROW FROM GRAPHIC MEMORY
235          ; AND RETURN
236
237      ; SHIFT ADP2 LEFT ONE BIT POSITION
238
239 5C22 06EC  SADP2L: ASL ADP2
240 5C24 26ED  ROL ADP2+1
241 5C26 60    RTS
242
243      ; MOVE DOWN ONE SCAN LINE DOUBLE ADDS 40 TO ADP2
244
245 5C27 A5EC  DNISCM: LDA ADP2      ; ADD 40 TO LOW BYTE
246 5C29 18    CLC
247 5C2A 6928  ADC #40
248 5C2C 85EC  STA ADP2
249 5C2E A900  LDA #0       ; EXTEND CARRY INTO UPPER BYTE
250 5C30 65ED  ADC ADP2+1
251 5C32 85ED  STA ADP2+1
252 5C34 60    RTS
253
254      ; COMPUTE BYTE ADDRESS CONTAINING FIRST SCAN LINE OF
255      ; CHARACTER AT CURSOR POSITION AND PUT IN ADP2
256      ; BIT ADDRESS (BIT 0 IS LEFTMOST) AT BTPT
257      ; BYTE ADDRESS =VMORG*256+CHHI*40*CSRY+INT(CSRX*6/8)
258      ; SINCE CHHI IS A CONSTANT 9, THEN CHHI*40=360
259      ; BIT ADDRESS=REM(CSRX*5/8)

```

SDXTXT SIMPLIFIED DISPLAY TE
SUBROUTINES FOR SDXTXT

```

260
261 5C35 A900  CSRTAD: LDA #0      ; ZERO UPPER ADP2
262 5C37 85ED  STA ADP2+1
263 5C39 AD055B  LDA CSRY     ; FIRST COMPUTE 360*CSRY
264 5C3C 0A    ASLA      ; COMPUTE 9*CSRY DIRECTLY IN A
265 5C3D 0A    ASLA
266 5C3E 0A    ASLA
267 5C3F 6D055B  ADC CSRY     ; STORE 9*CSRY IN LOWER ADP2
268 5C42 85EC  STA ADP2
269 5C44 20225C  JSR SADP2L   ; 18*CSRY IN ADP2
270 5C47 20225C  JSR SADP2L   ; 36*CSRY IN ADP2
271 5C4A 65EC  ADC ADP2      ; ADD IN 9*CSRY TO MAKE 45*CSRY
272 5C4C 85EC  STA ADP2
273 5C4E A900  LDA #0
274 5C50 65ED  ADC ADP2+1
275 5C52 85ED  STA ADP2+1
276 5C54 20225C  JSR SADP2L   ; 45*CSRY IN ADP2
277 5C57 20225C  JSR SADP2L   ; 90*CSRY IN ADP2
278 5C5A 20225C  JSR SADP2L   ; 180*CSRY IN ADP2
279 5C5D AD045B  LDA CSRX     ; 360*CSRY IN ADP2
280 5C60 0A    ASLA      ; NEXT COMPUTE 6*CSRX WHICH IS A 9 BIT
281 5C61 6D045B  ADC CSRX
282 5C64 0A    ASLA
283 5C65 8D005B  STA BTPT     ; SAVE RESULT TEMPORARILY
284 5C68 6A    RORA      ; DIVIDE BY 8 AND TRUNCATE FOR INT
285 5C69 4A    LSRA      ; FUNCTION
286 5C6A 4A    LSRA      ; NOW HAVE INT(CSRX*6/8)
287 5C6B 18    CLC
288 5C6C 65EC  ADC ADP2
289 5C6E 85EC  STA ADP2
290 5C70 A5ED  LDA ADP2+1
291 5C72 6D065B  ADC VMORG   ; ADD IN VMORG*256
292 5C75 85ED  STA ADP2+1
293 5C77 AD005B  LDA BTPT     ; FINISHED WITH ADP2
294 5C7A 2907  AND #7      ; COMPUTE REM(CSRX*6/8) WHICH IS LOW 3
295 5C7C 80005B  STA BTPT     ; BITS OF CSRX*6
296 5C7F 60    RTS       ; KEEP IN BTPT
297
298
299
300
301
302
303 5C80 8D035B  MERGE: STA MRGT1   ; SAVE INPUT DATA
304 5C83 98    TYA        ; SAVE Y
305 5C84 48    PHA
306 5C85 AC005B  LDY BTPT     ; OPEN UP A 5 BIT WINDOW IN GRAPHIC MEMORY
307 5C88 B9C35C  LDA MERGT,Y  ; LEFT BITS
308 5C88 A000  LDY #0      ; ZERO Y
309 5C88 31EC  AND (ADP2),Y
310 5C8F 91EC  STA (ADP2),Y
311 5C91 AC005B  LDY BTPT
312 5C94 B9CB5C  LDA MERGT+8,Y ; RIGHT BITS
313 5C97 A001  LDY #1
314 5C99 31EC  AND (ADP2),Y

```

SDXTX SIMPLIFIED DISPLAY TE
SUBROUTINES FOR SDXTX

```

315 5C9B 91EC STA (ADP2),Y
316 5C9D AD035B LDA MRGT1 ; SHIFT DATA RIGHT TO LINE UP LEFTMOST
317 5CA0 A005B LDY BTPT ; DATA BIT WITH LEFTMOST GRAPHIC FIELD
318 5CA3 F004 BEQ MERGE2 ; SHIFT BTPT TIMES
319 5CA5 4A
320 5CA6 88
321 5CA7 D0FC
322 5CA9 11EC
323 5CAB 91EC
324 5CAD A908
325 5CAF 3B
326 5CBO ED005B
327 5CB3 A8
328 5CB4 AD035B
329 5CB7 0A
330 5CB8 88
331 5CB9 D0FC
332 5CBB C8
333 5CBC 11EC
334 5CBE 91EC
335 5CC0 68
336 5CC1 A8
337 5CC2 60
338
339 5CC3 0783C1E0 MERGT: .BYTE X'07,X'83,X'C1,X'E0 ; TABLE OF MASKS FOR OPENING UP
340 5CC7 F0F8FCFE .BYTE X'F0,X'F8,X'FC,X'FE ; A 5 BIT WINDOW ANYWHERE
341 5CCB FFFFFFFF .BYTE X'FF,X'FF,X'FF,X'FF ; IN GRAPHIC MEMORY
342 5CCF 7F3F1F0F .BYTE X'7F,X'3F,X'1F,X'0F
343
344 ; FAST MEMORY MOVE ROUTINE
345 ; ENTER WITH SOURCE ADDRESS IN ADPT1 AND DESTINATION ADDRESS IN
346 ; ADPT2 AND MOVE COUNT (DOUBLE PRECISION) IN DCNT1.
347 ; MOVE PROCEEDS FROM LOW TO HIGH ADDRESSES AT APPROXIMATELY 16US
348 ; PER BYTE.
349 ; EXIT WITH ADDRESS POINTERS AND COUNT IN UNKNOWN STATE.
350 ; PRESERVES X AND Y REGISTERS.
351
352 5CD3 8A FMOVE: TXA ; SAVE X AND Y ON THE STACK
353 5CD4 48 PHA
354 5CD5 98 TYA
355 5CD6 48 PHA
356 5CD7 CE025B FMOVE1: DEC DCNT1+1 ; TEST IF LESS THAN 256 LEFT TO MOVE
357 5CDA 3015 BMI FMOVE3 ; JUMP TO FINAL MOVE IF SO
358 5CDC A000 LDY #0 ; MOVE A BLOCK OF 256 BYTES QUICKLY
359 5CDE B1EA FMOVE2: LDA (ADP1),Y ; TWO BYTES AT A TIME
360 5CEO 91EC STA (ADP2),Y
361 5CE2 C8 INY
362 5CE3 B1EA LDA (ADP1),Y
363 5CE5 91EC STA (ADP2),Y
364 5CE7 C8 INY
365 5CE8 D0F4 BNE FMOVE2 ; CONTINUE UNTIL DONE
366 5CEA E6EB INC ADP1+1 ; BUMP ADDRESS POINTERS TO NEXT PAGE
367 5CEC E6ED INC ADP2+1
368 5CEE 4CD75C JMP FMOVE1 ; GO MOVE NEXT PAGE
369 5CF1 AE015B FMOVE3: LDX DCNT1 ; GET REMAINING BYTE COUNT INTO X

```

SDXTX SIMPLIFIED DISPLAY TE
SUBROUTINES FOR SDXTX

```

370 5CF4 B1EA FMOVE4: LDA (ADP1),Y ; MOVE A BYTE
371 5CF6 91EC STA (ADP2),Y
372 5CF8 C8 INY
373 5CF9 CA DEX
374 5CFA D0F8 BNE FMOVE4 ; CONTINUE UNTIL DONE
375 5CFC 68 PLA ; RESTORE INDEX REGISTERS
376 5CFD A8 TAY
377 5CFE 68 PLA
378 5CF9 AA TAX
379 5D00 60 RTS ; AND RETURN
380
381 ; FAST MEMORY CLEAR ROUTINE
382 ; ENTER WITH ADDRESS OF BLOCK TO CLEAR IN ADP2 AND CLEAR COUNT
383 ; IN DCNT1.
384 ; EXIT WITH ADDRESS POINTERS AND COUNT IN UNKNOWN STATE
385 ; PRESERVES X AND Y REGISTERS
386
387 5D01 98 FCLR: TYA ; SAVE Y
388 5D02 48 PHA
389 5D03 A000 FCLR1: LDY #0
390 5D05 CE025B DEC DCNT1+1 ; TEST IF LESS THAN 256 LEFT TO MOVE
391 5D08 300B BMI FCLR3 ; JUMP TO FINAL CLEAR IF SO
392 5D0A 98 TYA ; CLEAR A BLOCK OF 256 QUICKLY
393 5D0B 91EC FCLR2: STA (ADP2),Y ; CLEAR A BYTE
394 5D00 C8 INY
395 5D0E D0F8 BNE FCLR2
396 5D10 E6ED INC ADP2+1 ; BUMP ADDRESS POINTER TO NEXT PAGE
397 5D12 4C035D JMP FCLR1 ; GO CLEAR NEXT PAGE
398 5D15 98 FCLR3: TYA ; CLEAR REMAINING PARTIAL PAGE
399 5D16 91EC FCLR4: STA (ADP2),Y
400 5D18 C8 INY
401 5D19 CE015B DEC DCNT1
402 5D1C D0F8 BNE FCLR4
403 5D1E 68 PLA ; RESTORE Y
404 5D1F A8 TAY
405 5D20 60 RTS ; RETURN
406

```

SDTXT SIMPLIFIED DISPLAY TE
CHARACTER FONT TABLE

```

407      : .PAGE    'CHARACTER FONT TABLE'
408      : CHARACTER FONT TABLE
409      : ENTRIES IN ORDER STARTING AT ASCII BLANK
410      : 96 ENTRIES
411      : EACH ENTRY CONTAINS 7 BYTES
412      : 7 BYTES ARE CHARACTER MATRIX, TOP ROW FIRST, LEFTMOST DOT
413      : IS LEFTMOST IN BYTE
414      : LOWER CASE FONT IS SMALL UPPER CASE, 5 BY 5 MATRIX
415 5D21 000000 CHTB: .BYTE   X'00,X'00,X'00 ; BLANK
416 5D24 00000000 .BYTE   X'00,X'00,X'00,X'00
417 5D28 202020 .BYTE   X'20,X'20,X'20 ; !
418 5D2B 20200020 .BYTE   X'20,X'20,X'00,X'20
419 5D2F 505050 .BYTE   X'50,X'50,X'50 ; "
420 5D32 00000000 .BYTE   X'00,X'00,X'00,X'00
421 5D36 5050F8 .BYTE   X'50,X'50,X'F8 ; #
422 5D39 50F85050 .BYTE   X'50,X'F8,X'50,X'50
423 5D3D 2078A0 .BYTE   X'20,X'78,X'A0 ; $
424 5D40 7028F020 .BYTE   X'70,X'28,X'F0,X'20
425 5D44 C8C810 .BYTE   X'C8,X'C8,X'10 ; %
426 5D47 20409898 .BYTE   X'20,X'40,X'98,X'98
427 5D48 40A0A0 .BYTE   X'40,X'A0,X'A0 ; &
amp;
428 5D4E 40A89068 .BYTE   X'40,X'A8,X'90,X'68
429 5D52 303030 .BYTE   X'30,X'30,X'30 ; '
430 5D55 00000000 .BYTE   X'00,X'00,X'00,X'00
431 5D59 20404040 .BYTE   X'20,X'40,X'40 ; (
432 5D5C 40404020 .BYTE   X'40,X'40,X'40,X'20
433 5D60 201010 .BYTE   X'20,X'10,X'10 ; )
434 5D63 10101020 .BYTE   X'10,X'10,X'10,X'20
435 5D67 20A870 .BYTE   X'20,X'A8,X'70 ; *
436 5D6A 2070A820 .BYTE   X'20,X'70,X'A8,X'20
437 5D6E 002020 .BYTE   X'00,X'20,X'20 ; +
438 5D71 F8202000 .BYTE   X'F8,X'20,X'20,X'00
439 5D75 000000 .BYTE   X'00,X'00,X'00 ; ,
440 5D78 30301020 .BYTE   X'30,X'30,X'10,X'20
441 5D7C 000000 .BYTE   X'00,X'00,X'00 ; -
442 5D7F F8000000 .BYTE   X'F8,X'00,X'00,X'00
443 5D83 000000 .BYTE   X'00,X'00,X'00 ; .
444 5D86 00003030 .BYTE   X'00,X'00,X'30,X'30
445 5D8A 080810 .BYTE   X'08,X'08,X'10 ; /
446 5D8D 20408080 .BYTE   X'20,X'40,X'80,X'80
447 5D91 609090 .BYTE   X'60,X'90,X'90 ; 0
448 5D94 90909060 .BYTE   X'90,X'90,X'90,X'60
449 5D98 206020 .BYTE   X'20,X'60,X'20 ; 1
450 5D9B 20202070 .BYTE   X'20,X'20,X'20,X'70
451 5D9F 708810 .BYTE   X'70,X'88,X'10 ; 2
452 5DA2 204080F8 .BYTE   X'20,X'40,X'80,X'F8
453 5DA6 708808 .BYTE   X'70,X'88,X'08 ; 3
454 5DA9 30088870 .BYTE   X'30,X'08,X'88,X'70
455 5DAD 103050 .BYTE   X'10,X'30,X'50 ; 4
456 5DB0 90F81010 .BYTE   X'90,X'F8,X'10,X'10
457 5DB4 F880F0 .BYTE   X'F8,X'80,X'F0 ; 5
458 5DB7 080808F0 .BYTE   X'08,X'08,X'08,X'F0
459 5DBB 708808 .BYTE   X'70,X'80,X'80 ; 6
460 5DBE F0888870 .BYTE   X'F0,X'88,X'88,X'70

```

SDTXT SIMPLIFIED DISPLAY TE
CHARACTER FONT TABLE

```

461 5DC2 F80810 .BYTE   X'F8,X'08,X'10 ; 7
462 5DC5 20408080 .BYTE   X'20,X'40,X'80,X'80
463 5DC9 708888 .BYTE   X'70,X'88,X'88 ; 8
464 5DCC 70888870 .BYTE   X'70,X'88,X'88,X'70
465 5DD0 708888 .BYTE   X'70,X'88,X'88 ; 9
466 5DD3 78080870 .BYTE   X'78,X'08,X'08,X'70
467 5DD7 303000 .BYTE   X'30,X'30,X'00 ; :
468 5DDA 00003030 .BYTE   X'00,X'00,X'30,X'30
469 5DDC 303000 .BYTE   X'30,X'30,X'00 ; :
470 5DE1 30301020 .BYTE   X'30,X'30,X'10,X'20
471 5DE5 102040 .BYTE   X'10,X'20,X'40 ; LESS THAN
472 5DE8 80402010 .BYTE   X'80,X'40,X'20,X'10
473 5DEF 0000F8 .BYTE   X'00,X'00,X'F8 ; =
474 5DEF 00F80000 .BYTE   X'00,X'F8,X'00,X'00
475 5DF3 402010 .BYTE   X'40,X'20,X'10
476 5DF6 08102040 .BYTE   X'08,X'10,X'20,X'40
477 5DFA 708808 .BYTE   X'70,X'88,X'08 ; ?
478 5DFD 10200020 .BYTE   X'10,X'20,X'00,X'20
479 5E01 708808 .BYTE   X'70,X'88,X'08
480 5E04 68A8A8D0 .BYTE   X'68,X'A8,X'A8,X'D0
481 5E08 205088 .BYTE   X'20,X'50,X'88 ; A
482 5E0B 88F88888 .BYTE   X'88,X'F8,X'88,X'88
483 5EOF F04848 .BYTE   X'F0,X'48,X'48 ; B
484 5E12 704848F0 .BYTE   X'70,X'48,X'48,X'F0
485 5E16 708880 .BYTE   X'70,X'88,X'80 ; C
486 5E19 80808870 .BYTE   X'80,X'80,X'88,X'70
487 5E1D F04848 .BYTE   X'F0,X'48,X'48 ; D
488 5E20 484848F0 .BYTE   X'48,X'48,X'48,X'F0
489 5E24 F88080 .BYTE   X'F8,X'80,X'80 ; E
490 5E27 F80808F8 .BYTE   X'F0,X'80,X'80,X'F8
491 5E2B F88080 .BYTE   X'F8,X'80,X'80 ; F
492 5E2E F808080 .BYTE   X'FO,X'80,X'80,X'80
493 5E32 708880 .BYTE   X'70,X'88,X'80 ; G
494 5E35 B8888870 .BYTE   X'B8,X'88,X'88,X'70
495 5E39 888888 .BYTE   X'88,X'88,X'88 ; H
496 5E3C F8888888 .BYTE   X'F8,X'88,X'88,X'88
497 5E40 702020 .BYTE   X'70,X'20,X'20 ; I
498 5E43 20202070 .BYTE   X'20,X'20,X'20,X'70
499 5E47 381010 .BYTE   X'38,X'10,X'10 ; J
500 5E4A 10109060 .BYTE   X'10,X'10,X'90,X'60
501 5E4E 8890AO .BYTE   X'88,X'90,X'A0 ; K
502 5E51 COA09088 .BYTE   X'CO,X'A0,X'90,X'88
503 5E55 808080 .BYTE   X'80,X'80,X'80 ; L
504 5E58 808080F8 .BYTE   X'80,X'80,X'80,X'F8
505 5E5C 88D8A8 .BYTE   X'88,X'D8,X'A8 ; M
506 5E5F A8888888 .BYTE   X'A8,X'88,X'88,X'88
507 5E63 8888C8 .BYTE   X'88,X'88,X'C8 ; N
508 5E66 A8988888 .BYTE   X'A8,X'98,X'88,X'88
509 5E6A 708888 .BYTE   X'70,X'88,X'88 ; O
510 5E6D 88888870 .BYTE   X'88,X'88,X'88,X'70
511 5E71 F08888 .BYTE   X'F0,X'88,X'88 ; P
512 5E74 F0808080 .BYTE   X'F0,X'80,X'80,X'80
513 5E78 708888 .BYTE   X'70,X'88,X'88 ; Q
514 5E7B 88A89068 .BYTE   X'88,X'A8,X'90,X'68
515 5E7F F08888 .BYTE   X'F0,X'88,X'88 ; R

```

SDTXT SIMPLIFIED DISPLAY TE
CHARACTER FONT TABLE

```

516 5E82 F0A09088 .BYTE X'F0,X'A0,X'90,X'88
517 5E86 788080 .BYTE X'78,X'80,X'80
518 5E89 700808F0 .BYTE X'70,X'08,X'08,X'F0
519 5E80 F82020 .BYTE X'F8,X'20,X'20
520 5E90 20202020 .BYTE X'20,X'20,X'20,X'20
521 5E94 888888 .BYTE X'88,X'88,X'88
522 5E97 88888870 .BYTE X'88,X'88,X'88,X'70
523 5E98 888888 .BYTE X'88,X'88,X'88
524 5E9E 50502020 .BYTE X'50,X'50,X'20,X'20
525 5EA2 888888 .BYTE X'88,X'88,X'88
526 5EA5 A8A8D888 .BYTE X'A8,X'A8,X'D8,X'88
527 5EA6 888850 .BYTE X'88,X'88,X'50
528 5EAC 20508888 .BYTE X'20,X'50,X'88,X'88
529 5EB0 888850 .BYTE X'88,X'88,X'50
530 5EB3 20202020 .BYTE X'20,X'20,X'20,X'20
531 5EB7 F80810 .BYTE X'F8,X'08,X'10
532 5EB8 204080F8 .BYTE X'20,X'40,X'80,X'F8
533 5EBE 704040 .BYTE X'70,X'40,X'40,X'40
534 5EC1 40404070 .BYTE X'40,X'40,X'40,X'70
535 5EC5 808040 .BYTE X'80,X'80,X'40
536 5EC8 20100808 .BYTE X'20,X'10,X'08,X'08
537 5EC C06030 .BYTE X'70,X'10,X'10
538 5ECF 10101070 .BYTE X'10,X'10,X'10,X'70
539 5ED3 205088 .BYTE X'20,X'50,X'88
540 5ED6 00000000 .BYTE X'00,X'00,X'00,X'00
541 5ED9 000000 .BYTE X'00,X'00,X'00
542 5EDD 000000F8 .BYTE X'00,X'00,X'00,X'F8
543 5EE1 C06030 .BYTE X'CO,X'60,X'30
544 5EE4 00000000 .BYTE X'00,X'00,X'00,X'00
545 5EE8 000020 .BYTE X'00,X'00,X'20
546 5EEB 5088F888 .BYTE X'50,X'88,X'F8,X'88
547 5EEF 0000F0 .BYTE X'00,X'00,X'F0
548 5EF2 487048F0 .BYTE X'48,X'70,X'48,X'F0
549 5EF9 000078 .BYTE X'00,X'00,X'78
550 5EF9 80808078 .BYTE X'80,X'80,X'80,X'78
551 5EFD 0000F0 .BYTE X'00,X'00,X'F0
552 5F00 484848F0 .BYTE X'48,X'48,X'48,X'F0
553 5F04 0000F8 .BYTE X'00,X'00,X'F8
554 5F07 80E080F8 .BYTE X'80,X'E0,X'80,X'F8
555 5F08 0000F8 .BYTE X'00,X'00,X'F8
556 5F0E 80E08080 .BYTE X'80,X'E0,X'80,X'80
557 5F12 000078 .BYTE X'00,X'00,X'78
558 5F15 80988878 .BYTE X'80,X'98,X'88,X'78
559 5F19 000088 .BYTE X'00,X'00,X'88
560 5F1C 88F88888 .BYTE X'88,X'F8,X'88,X'88
561 5F20 000070 .BYTE X'00,X'00,X'70
562 5F23 20202070 .BYTE X'20,X'20,X'20,X'70
563 5F27 000038 .BYTE X'00,X'00,X'38
564 5F2A 10105020 .BYTE X'10,X'10,X'50,X'20
565 5F2E 000090 .BYTE X'00,X'00,X'90
566 5F31 A0C0A090 .BYTE X'A0,X'CO,X'A0,X'90
567 5F35 000080 .BYTE X'00,X'00,X'80
568 5F38 808080F8 .BYTE X'80,X'80,X'80,X'F8
569 5F3C 000088 .BYTE X'00,X'00,X'88
570 5F3F D8A88888 .BYTE X'D8,X'A8,X'88,X'88

```

SDTXT SIMPLIFIED DISPLAY TE
CHARACTER FONT TABLE

```

571 5F43 000088 .BYTE X'00,X'00,X'88 ; N (LC)
572 5F46 C8A89888 .BYTE X'C8,X'A8,X'98,X'88 ; O (LC)
573 5F4A 000070 .BYTE X'00,X'00,X'70 ; P (LC)
574 5F4D 88888870 .BYTE X'88,X'88,X'88,X'70 ; Q (LC)
575 5F51 0000F0 .BYTE X'00,X'00,X'F0 ; R (LC)
576 5F54 88F08080 .BYTE X'88,X'F0,X'80,X'80 ; S (LC)
577 5F58 000070 .BYTE X'00,X'00,X'70 ; T (LC)
578 5F5B 88A89068 .BYTE X'88,X'A8,X'90,X'68 ; U (LC)
579 5F5F 0000F0 .BYTE X'00,X'00,X'F0 ; V (LC)
580 5F62 88F0A090 .BYTE X'88,X'F0,X'A0,X'90 ; W (LC)
581 5F66 000078 .BYTE X'00,X'00,X'78 ; X (LC)
582 5F69 807008F0 .BYTE X'80,X'70,X'08,X'F0 ; Y (LC)
583 5F6D 0000F8 .BYTE X'00,X'00,X'F8 ; Z (LC)
584 5F70 20202020 .BYTE X'20,X'20,X'20,X'20 ; LEFT BRACE
585 5F74 000088 .BYTE X'00,X'00,X'88 ; VERTICAL BAR
586 5F77 88888870 .BYTE X'88,X'88,X'88,X'70 ; RIGHT BRACE
587 5F7B 000088 .BYTE X'00,X'00,X'88 ; TILDAS
588 5F7E 88885020 .BYTE X'88,X'88,X'50,X'20 ; RUBOUT
589 5F82 000088 .BYTE X'00,X'00,X'88 ; RUBOUT
590 5F85 88A89888 .BYTE X'88,X'A8,X'D8,X'88 ; RUBOUT
591 5F89 000088 .BYTE X'00,X'00,X'88 ; RUBOUT
592 5F8C 50205088 .BYTE X'50,X'20,X'50,X'88 ; RUBOUT
593 5F90 000088 .BYTE X'00,X'00,X'88 ; RUBOUT
594 5F93 50202020 .BYTE X'50,X'20,X'20,X'20 ; RUBOUT
595 5F97 0000F8 .BYTE X'00,X'00,X'F8 ; RUBOUT
596 5F9A 102040F8 .BYTE X'10,X'20,X'40,X'F8 ; RUBOUT
597 5F9E 102020 .BYTE X'10,X'20,X'20 ; RUBOUT
598 5FA1 60202010 .BYTE X'60,X'20,X'20,X'10 ; RUBOUT
599 5FA5 202020 .BYTE X'20,X'20,X'20 ; RUBOUT
600 5FA8 20202020 .BYTE X'20,X'20,X'20,X'20 ; RUBOUT
601 5FAC 402020 .BYTE X'40,X'20,X'20 ; RUBOUT
602 5FAD 30202040 .BYTE X'30,X'20,X'20,X'40 ; RUBOUT
603 5FB3 10A840 .BYTE X'10,X'A8,X'40 ; RUBOUT
604 5FB6 00000000 .BYTE X'00,X'00,X'00,X'00 ; RUBOUT
605 5FBA A850A8 .BYTE X'A8,X'50,X'A8 ; RUBOUT
606 5FBD 50A850A8 .BYTE X'50,X'A8,X'50,X'A8 ; RUBOUT
607
608 0000 .END

```

NO ERROR LINES

MSUP K-1008 VM GRAPHIC SUP
DOCUMENTATION, EQUATES, STORAGE

```

3 .PAGE 'DOCUMENTATION, EQUATES, STORAGE'
4
5 THIS PACKAGE PROVIDES FUNDAMENTAL GRAPHICS ORIENTED
6 SUBROUTINES NEEDED FOR EFFECTIVE USE OF THE VISIBLE MEMORY AS
7 A GRAPHIC DISPLAY DEVICE. MAJOR SUBROUTINES INCLUDED ARE AS
8 FOLLOWS:
9     CLEAR - CLEARS THE ENTIRE VISIBLE MEMORY AS DEFINED BY
10    NPIX/8
11    PIXADR- RETURNS BYTE AND BIT ADDRESS OF PIXEL AT XICORD,
12        YICORD
13    CKCRD1- PERFORM A RANGE CHECK ON XICORD,YICORD
14    CKCRD2- PERFORM A RANGE CHECK ON X2CORD,Y2CORD
15    STPIX - SET PIXEL AT XICORD,YICORD TO A ONE (WHITE DOT)
16    CLPIX - CLEAR PIXEL AT XICORD,YICORD TO ZERO (BLACK DOT)
17    FLPIX - FLIP THE PIXEL AT XICORD,YICORD
18    WRPIX - UPDATE PIXEL AT XICORD,YICORD ACCORDING TO THE
19        STATE OF THE ACCUMULATOR
20    RDPIX - COPY THE STATE OF THE PIXEL AT XICORD,YICORD INTO
21        THE ACCUMULATOR
22    DRAW - DRAW THE BEST STRAIGHT LINE FROM XICORD,YICORD
23        TO X2CORD,Y2CORD. X2CORD,Y2CORD COPIED TO
24        XICORD,YICORD AFTER DRAWING
25    ERASE - SAME AS DRAW EXCEPT A BLACK LINE IS DRAWN
26    DCHAR - DISPLAYS A CHARACTER WHOSE UPPER LEFT CORNER IS
27        XICORD,YICORD. CHARACTER MATRIX IS 5 WIDE BY 9
28        HIGH INCLUDING LOWER CASE DESCENDERS BUT NOT
29        INCLUDING CHARACTER AND LINE SPACING.
30    DTEXT - ACCEPTS ASCII CHARACTERS AND FORMATS THEM INTO
31        TEXT. A STANDARD (BUT EASILY MODIFIED) CHARACTER
32        FIELD 6 WIDE BY 11 HIGH ALLOWS UP TO 18 LINES OF 53
33        CHARACTERS. SUBSCRIPT AND SUPERSCRIPT VIA CONTROL
34        CHARACTERS IS IMPLEMENTED.
35    DTXTIN- INITIALIZE PARAMETERS FOR USE OF DTEXT ON FULL
36        SCREEN.
37
38 ALL SUBROUTINES DEPEND ON ONE OR TWO PAIRS OF COORDINATES.
39 EACH COORDINATE IS A STANDARD PRECISION, UNSIGNED NUMBER WITH
40 THE LOW BYTE FIRST (I.E. LIKE MEMORY ADDRESSES IN THE 6502)
41 THE ORIGIN OF THE COORDINATE SYSTEM IS AT THE LOWER LEFT
42 CORNER OF THE SCREEN THEREFORE THE ENTIRE SCREEN IS IN THE
43 FIRST QUADRANT. ALLOWABLE RANGE OF THE X COORDINATE IS 0 TO
44 319 (DECIMAL) AND THE RANGE OF THE Y COORDINATE IS 0 TO 199.
45 FOR MAXIMUM SPEED ALL SUBROUTINES ASSUME THAT THE COORDINATE
46 VALUES ARE IN RANGE. IF THEY ARE NOT, WILD STORING INTO ANY
47 PART OF KIM RAM IS POSSIBLE. FOR DEBUGGING, CALLS TO CKCRD1
48 AND CKCRD2 SHOULD BE PERFORMED PRIOR TO GRAPHIC ROUTINE CALLS
49 IN ORDER TO DETECT AND CORRECT ERRONEOUS COORDINATE VALUES.
50
51 ; GENERAL EQUATES
52 0140 NX = 320 ; NUMBER OF BITS IN A ROW
53 00C8 NY = 200 ; NUMBER OF ROWS (CHANGE FOR HALF SCREEN
54 ; OPERATION)
55 FA00 NPIX = NX*NY ; NUMBER OF PIXELS
56 0008 HHIW = 11 ; HEIGHT OF CHARACTER WINDOW

```

VMSUP K-1008 VM GRAPHIC SUP
DOCUMENTATION, EQUATES, STORAGE

```

57 0006 CHWIDW = 6 ; WIDTH OF CHARACTER WINDOW
58 0009 CHHIM = 9 ; HEIGHT OF CHARACTER MATRIX
59 0005 CHWIDM = 5 ; WIDTH OF CHARACTER MATRIX
60
61 ; BASE PAGE TEMPORARY STORAGE (MAY BE DESTROYED BETWEEN CALLS)
62
63 0000 .= X'EA
64
65 00EA ADP1: .=+ 2 ; ADDRESS POINTER 1
66 00EC ADP2: .=+ 2 ; ADDRESS POINTER 2
67
68 ; PERMANENT RAM STORAGE (MUST BE PRESERVED BETWEEN CALLS)
69 ;***** THESE PARAMETERS MUST BE SET BEFORE USING GRAPHIC *****
70 ;***** ROUTINES THAT REFERENCE THEM *****
71
72 00EE .= X'100 ; PUT IN STACK AREA FOR CONVENIENCE
73
74 0100 VMORG: .=+ 1 ; PAGE NUMBER OF FIRST VISIBLE MEMORY
75
76 0101 X1CORD: .=+ 2 ; COORDINATE PAIR 1 AND CURSOR LOCATION
77 0103 Y1CORD: .=+ 2
78 0105 X2CORD: .=+ 2 ; COORDINATE PAIR 2
79 0107 Y2CORD: .=+ 2
80 0109 TMAR: .=+ 2 ; TOP MARGIN FOR DTEXT
81 0108 BMAR: .=+ 2 ; BOTTOM MARGIN FOR DTEXT
82 010D LMAR: .=+ 2 ; LEFT MARGIN FOR DTEXT
83 010F RMAR: .=+ 2 ; RIGHT MARGIN FOR DTEXT
84
85 ; GENERAL TEMPORARY STORAGE (CAN BE DESTROYED BETWEEN CALLS)
86
87 0111 BTPT: .=+ 1 ; BIT NUMBER
88 0112 DELTAX: .=+ 2 ; DELTA X FOR LINE DRAW
89 0114 DELTAY: .=+ 2 ; DELTA Y FOR LINE DRAW
90 0116 ACC: .=+ 2 ; ACCUMULATOR FOR LINE DRAW
91 0118 XDIR: .=+ 1 ; X MOVEMENT DIRECTION, ZERO=+
92 0119 YDIR: .=+ 1 ; Y MOVEMENT DIRECTION, ZERO=+
93 011A XCHFLG: .=+ 1 ; EXCHANGE X AND Y FLAG, EXCHANGE IF NOT 0
94 011B COLOR: .=+ 1 ; COLOR OF LINE DRAWN -1=WHITE
95 011C TEMP: .=+ 2 ; TEMPORARY STORAGE
96 0112 TBLT: = DELTAX ; TOP LEFT BYTE ADDRESS FOR TEXT WINDOW
97 0118 TBLIT: = XDIR ; TOP LEFT BIT ADDRESS FOR TEXT WINDOW
98 0114 TRBT: = DELTAY ; TOP RIGHT BYTE ADDRESS FOR TEXT WINDOW
99 0119 TRBIT: = YDIR ; TOP RIGHT BIT ADDRESS FOR TEXT WINDOW
100 0116 BRBYT: = ACC ; BOTTOM RIGHT BYTE ADDRESS FOR TXT WINDOW
101

```

VMSUP K-1008 VM GRAPHIC SUP
CLEAR ENTIRE SCREEN ROUTINE

```

.PAGE 'CLEAR ENTIRE SCREEN ROUTINE'
CLEAR ENTIRE SCREEN ROUTINE
USES BOTH INDICES AND ADP1
102      ; X'5500      ; PUT AT END OF 16K EXPANSION
103      ; USES BOTH INDICES AND ADP1
104
105 011E  .= X'5500      ; INITIALIZE ADDRESS POINTER
106
107 5500 A000  CLEAR: LDY #0          ; AND ZERO INDEX Y
108 5502 84EA  STY ADP1
109 5504 A00001 LDA VMORG
110 5507 85EB  STA ADP1+1
111 5509 18    CLC
112 550A 691F  ADC #NPIX/8/256
113 550C AA    TAX
114 550D 98    STA (ADP1),Y
115 550E 91EA  INC ADP1
116 5510 E6EA  BNE CLEAR2
117 5512 D002  INC ADP1+1
118 5514 E6EB  LDA ADP1
119 5516 A5EA  TEST IF DONE
120 5518 C940  CMP #NPIX/8&X'FF
121 551A D0F1  BNE CLEAR1
122 551C E4EB  CPX ADP1+1
123 551E D0ED  BNE CLEAR1
124 5520 60    RTS
125

```

VMSUP K-1008 VM GRAPHIC SUP
PIXADR - BYTE AND BIT ADDRESS OF A PIXEL

```

.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
126      ; X1CORD,Y1CORD
127      ; PUTS BYTE ADDRESS IN ADP1 AND BIT NUMBER (BIT 0 IS LEFTMOST)
128      ; IN BTPT.
129      ; DOES NOT CHECK MAGNITUDE OF COORDINATES FOR MAXIMUM SPEED
130      ; PRESERVES X AND Y REGISTERS, DESTROYS A
131      ; BYTE ADDRESS = VMORG*256+(199-Y1CORD)*40+INT(XCORD/8)
132      ; BIT ADDRESS = REM(XCORD/8)
133      ; OPTIMIZED FOR SPEED THEREFORE CALLS TO A DOUBLE SHIFT ROUTINE
134      ; ARE NOT DONE
135      ; COMPUTE BIT ADDRESS FIRST
136 5521 A00101  PIXADR: LDA X1CORD
137 5524 85EA  STA ADP1
138 5526 2907  AND #X'07
139 5528 801101 STA BTPT
140 552B 801101 LDA X1CORD+1
141 552B A0201  STA ADP1+1
142 552E 85EB  LSR ADP1+1
143 5530 46EB  ROR ADP1
144 5532 66EA  LSR ADP1+1
145 5534 46EB  ROR ADP1
146 5536 66EA  ROR ADP1
147 5538 46EB  LSR ADP1+1
148 553A 66EA  ROR ADP1
149 553C A9C7  LDA #199
150 553E 38    SEC
151 553F ED0301 SBC Y1CORD
152 5542 85EC STA ADP2
153 5544 801C01 STA TEMP
154 5547 A900 LDA #0
155 5549 ED0401 SBC Y1CORD+1
156 554C 85ED STA ADP2+1
157 554E 801D01 STA TEMP+1
158 5551 06EC ASL ADP2
159 5553 26ED ROL ADP2+1
160 5555 06EC ASL ADP2
161 5557 26ED ROL ADP2+1
162 5559 A5EC LDA ADP2
163 555B 18    CLC
164 555C 601C01 ADC TEMP
165 555F 85EC STA ADP2
166 5561 A5ED LDA ADP2+1
167 5563 601D01 ADC TEMP+1
168 5566 85ED STA ADP2+1
169 5568 06EC ASL ADP2
170 556A 26ED ROL ADP2+1
171 556C 06EC ASL ADP2
172 556E 26ED ROL ADP2+1
173 5570 06EC ASL ADP2
174 5572 26ED ROL ADP2+1
175 5574 A5EC LDA ADP2
176 5576 18    CLC
177 5577 65EA ADC ADP1
178 5579 85EA STA ADP1
179 557B A5ED LDA ADP2+1

```

VMSUP K-1008 VM GRAPHIC SUP
PIXADR - BYTE AND BIT ADDRESS OF A PIXEL

```

180 557D 65EB ADC ADP1+1
181 557F 6D0001 ADC VMORG ; ADD IN VMORG*256
182 5582 85EB STA ADP1+1 ; FINAL RESULT
183 5584 60 RTS ; RETURN
184

```

VMSUP K-1008 VM GRAPHIC SUP
INDIVIDUAL PIXEL SUBROUTINES

```

.PAGE 'INDIVIDUAL PIXEL SUBROUTINES'
STPIX - SETS THE PIXEL AT XICORD,YICORD TO A ONE (WHITE DOT)
DOES NOT ALTER XICORD OR YICORD
PRESERVES X AND Y
ASSUMES IN RANGE COORDINATES

185 ; STPIX: JSR PIXADR ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
186 ; INTO ADP1
187 ; SAVE Y
188 ; LDY BTPT ; GET BIT NUMBER IN Y
189 ; LDA MSKTB1,Y ; GET A BYTE WITH THAT BIT =1, OTHERS =0
190 ; LDY #0 ; ZERO Y
191 ; ORA (ADP1),Y ; COMBINE THE BIT WITH THE ADDRESSED VM
192 ; STA (ADP1),Y ; BYTE
193 ; PLA ; RESTORE Y
194 ; RTS ; AND RETURN
195 ; CLPIX - CLEARS THE PIXEL AT XICORD,YICORD TO A ZERO (BLACK DOT)
196 ; DOES NOT ALTER XICORD OR YICORD
197 ; PRESERVES X AND Y
198 ; ASSUMES IN RANGE COORDINATES
199 ; LDY BTPT ; GET BIT NUMBER IN Y
200 ; LDA MSKTB2,Y ; GET A BYTE WITH THAT BIT =0, OTHERS =1
201 ; LDY #0 ; ZERO Y
202 ; AND (ADP1),Y ; REMOVE THE BIT FROM THE ADDRESSED VM
203 ; STA (ADP1),Y ; BYTE
204 ; PLA ; RESTORE Y
205 ; RTS ; AND RETURN
206 ; FLPIX - FLIPS THE PIXEL AT XICORD,YICORD
207 ; DOES NOT ALTER XICORD OR YICORD
208 ; PRESERVES X AND Y
209 ; ASSUMES IN RANGE COORDINATES
210 ; LDY BTPT ; GET BIT NUMBER IN Y
211 ; LDA MSKTB1,Y ; GET A BYTE WITH THAT BIT =1, OTHERS =0
212 ; LDY #0 ; ZERO Y
213 ; EOR (ADP1),Y ; FLIP THAT BIT IN THE ADDRESSED VM BYTE
214 ; STA (ADP1),Y ; RESTORE Y
215 ; RTS ; AND RETURN
216 ; PLA ; RESTORE Y
217 ; TAY ; AND RETURN
218 ; RTS ; AND RETURN
219 ; CLPIX: JSR PIXADR ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
220 ; INTO ADP1
221 ; SAVE Y
222 ; LDY BTPT ; GET BIT NUMBER IN Y
223 ; LDA MSKTB2,Y ; GET A BYTE WITH THAT BIT =0, OTHERS =1
224 ; LDY #0 ; ZERO Y
225 ; AND (ADP1),Y ; REMOVE THE BIT FROM THE ADDRESSED VM
226 ; STA (ADP1),Y ; BYTE
227 ; PLA ; RESTORE Y
228 ; RTS ; AND RETURN
229 ; CLPIX: JSR PIXADR ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
230 ; INTO ADP1
231 ; SAVE Y
232 ; LDY BTPT ; GET BIT NUMBER IN Y
233 ; LDA MSKTB1,Y ; GET A BYTE WITH THAT BIT =1, OTHERS =0
234 ; LDY #0 ; ZERO Y
235 ; EOR (ADP1),Y ; FLIP THAT BIT IN THE ADDRESSED VM BYTE
236 ; STA (ADP1),Y ; RESTORE Y
237 ; RTS ; AND RETURN
238

```

VMSUP K-1008 VM GRAPHIC SUP
INDIVIDUAL PIXEL SUBROUTINES

```

239 : WRPIX - SETS THE PIXEL AT XICORD,YICORD ACCORDING TO THE ST
240 . OF BIT 0 (RIGHTMOST) OF A
241 . DOES NOT ALTER XICORD OR YICORD
242 . PRESERVES X AND Y AND A
243 . ASSUMES IN RANGE CORRIDINATES
244

245 55C1 2CD155 WRPIX: BIT WRPIXM- ; TEST LOW BIT OF A
246 55C4 48 PHA
247 55C5 F005 BEQ WRPIXI ; JUMP IF A ZERO TO BE WRITTEN
248 55C7 208555 JSR STPIX ; OTHERWISE WRITE A ONE
249 55CA 68 PLA ; RESTORE A AND RETURN
250 55CB 60 RTS
251 55CC 209955 WRPIXI: JSR CLPIX ; CLEAR THE PIXEL
252 55CF 68 PLA ; RESTORE A AND RETURN
253 55D0 60 RTS
254
255 55D1 01 WRPIXM: .BYTE 1 ; BIT TEST MASK FOR BIT 0
256
257 : RDPIX - READS THE PIXEL AT XICORD,YICORD AND SETS A TO ALL
258 . ZEROES IF IT IS A ZERO OR TO ALL ONES IF IT IS A ONE
259 . LOW BYTE OF ADP1 IS EQUAL TO A ON RETURN
260 . DOES NOT ALTER XICORD OR YICORD
261 . PRESERVES X AND Y
262 . ASSUMES IN RANGE CORRIDINATES
263
264 55D2 202155 RDPIX: JSR PIXADR ; GET BYTE AND BIT ADDRESS OF PIXEL
265 55D5 98 TYA ; SAVE Y
266 55D6 48 PHA
267 55D7 A000 LDY #0 ; GET ADDRESSED BYTE FROM VM
268 55D9 B1EA LDA (ADP1),Y
269 55DB AC1101 LDY BTPT ; GET BIT NUMBER IN Y
270 55DE 39EC55 AND MSKTB1,Y ; CLEAR ALL BUT ADDRESSED BIT
271 55E1 F002 BEQ RDPIX1 ; SKIP AHEAD IF IT WAS A ZERO
272 55E3 A9FF LDA #X'FF ; SET TO ALL ONES IF IT WAS A ONE
273 55E5 85EA STA ADP1 ; SAVE A TEMPORARILY IN ADP1 WHILE
274 55E7 68 PLA ; RESTORING Y
275 55E9 A8 TAY
276 55E9 A5EA LDA ADP1
277 55EB 60 RTS ; RETURN
278
279 : MASK TABLES FOR INDIVIDUAL PIXEL SUBROUTINES
280 : MSKTB1 IS A TABLE OF 1 BITS CORRESPONDING TO BIT NUMBERS
281 : MSKTB2 IS A TABLE OF 0 BITS CORRESPONDING TO BIT NUMBERS
282
283 55EC 80402010 MSKTB1: .BYTE X'80,X'40,X'20,X'10
284 55F0 08040201 .BYTE X'08,X'04,X'02,X'01
285 55F4 7FBDFEF MSKTB2: .BYTE X'7F,X'BF,X'DF,X'EF
286 55F8 F7FBDFFE .BYTE X'F7,X'FB,X'FD,X'FE
287

```

VMSUP K-1008 VM GRAPHIC SUP
COORDINATE CHECK ROUTINES

```

    .PAGE 'COORDINATE CHECK ROUTINES'
    CKCRD1 - CHECK X1CORD,Y1CORD 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.
    FOR MAXIMUM FLEXIBILITY IN USE, ALL REGISTERS ARE PRESERVED

288
289
290
291
292
293
294
295
296 55FC 48      CKCRD1: PHA          ; SAVE ALL REGISTERS
297 55FD 8A      TXA
298 55FE 48      PHA
299 55FF 98      TYA
300 5600 48      PHA
301 5601 A200    LDX #X1CORD-X1CORD ; CHECK X1CORD
302 5603 A000    LDY #XLIMIT-LIMTAB
303 5605 202B56  JSR CK
304 5608 A202    LDX #Y1CORD-X1CORD ; CHECK Y1CORD
305 560A A002    LDY #YLIMIT-LIMTAB
306 560C 202B56  JSR CK
307 560F 68      CKCRDR: PLA         ; RESTORE REGISTERS
308 5610 8A      TAY
309 5611 68      PLA
310 5612 AA      TAX
311 5613 68      PLA
312 5614 60      RTS          ; AND RETURN
313
314 ;           CKCRD2 - SAME AS CKCRD1 EXCEPT CHECKS X2CORD,Y2CORD
315
316 5615 48      CKCRD2: PHA         ; SAVE ALL REGISTERS
317 5616 8A      TXA
318 5617 48      PHA
319 5618 98      TYA
320 5619 48      PHA
321 561A A204    LDX #X2CORD-X1CORD ; CHECK X2CORD
322 561C A000    LDY #XLIMIT-LIMTAB
323 561E 202B56  JSR CK
324 5621 A206    LDX #Y2CORD-X1CORD ; CHECK Y2CORD
325 5623 A002    LDY #YLIMIT-LIMTAB
326 5625 202B56  JSR CK
327 5628 4C0F56  JMP CKCRDR   ; GO RESTORE REGISTERS AND RETURN
328
329 562B BD0201  CK:  LDA X1CORD+1,X ; CHECK UPPER BYTE
330 562E D95556  CMP LIMITB+1,Y ; AGAINST UPPER BYTE OF LIMIT
331 5631 9020    BCC CK4        ; OK IF LESS THAN UPPER BYTE OF LIMIT
332 5633 F016    BEQ CK3        ; GO CHECK LOWER BYTE IF EQUAL TO
333 ;           UPPER BYTE OF LIMIT
334 5635 BD0101  CK2: LDA X1CORD,X ; SUBTRACT THE LIMIT
335 5638 38      SEC
336 5639 F95456  SBC LIMITTAB,Y ; LOWER BYTE FIRST
337 563C 9D0101  STA X1CORD,X
338 563F BD0201  LDA X1CORD+1,X
339 5642 F95556  SBC LIMITTAB+1,Y
340 5645 9D0201  STA X1CORD+1,X
341 5648 4C2B56  JMP CK          ; AND THEN GO CHECK RANGE AGAIN

```

VMSUP K-1008 VM GRAPHIC SUP
COORDINATE CHECK ROUTINES

```

342 564B BD0101 CK3: LDA X1CORD,X ; CHECK LOWER BYTE OF X
343 564E D95456 CMP LIMTAB,Y
344 5651 B0E2 BCS CK2 ; GO ADJUST IF TOO LARGE
345 5653 60 CK4: RTS ; RETURN
346
347 .LIMTAB: ; TABLE OF LIMITS
348 5654 4001 XLIMIT: .WORD NX
349 5656 C800 YLIMIT: .WORD NY
350

```

VMSUP K-1008 VM GRAPHIC SUP
LINE DRAWING ROUTINES

```

351 ; .PAGE 'LINE DRAWING ROUTINES'
352 ; DRAW - DRAW THE BEST STRAIGHT LINE FROM XICORD,Y1CORD TO
353 ; X2CORD,Y2CORD.
354 ; X2CORD,Y2CORD COPIED TO XICORD,Y1CORD AFTER DRAWING
355 ; PRESERVES X AND Y
356 ; USES AN ALGORITHM THAT REQUIRES NO MULTIPLICATION OR DIVISION
357 5658 A900 ERASE: LDA #X'00 ; SET LINE COLOR TO BLACK
358 565A F002 BEQ DRAW1 ; GO DRAW THE LINE
359
360 565C A9FF DRAW: LDA #X'FF ; SET LINE COLOR TO WHITE
361 565E 801B01 DRAW1: STA COLOR
362 5661 8A TXA ; SAVE X AND Y
363 5662 48 PHA
364 5663 98 TYA
365 5664 48 PHA
366
367 ; COMPUTE SIGN AND MAGNITUDE OF DELTA X = X2-X1
368 ; PUT MAGNITUDE IN DELTAX AND SIGN IN XDIR
369
370 5665 A900 LDA #0 ; FIRST ZERO XDIR
371 5667 801B01 STA XDIR
372 566A AD0501 LDA X2CORD ; NEXT COMPUTE TWOS COMPLEMENT DIFFERENCE
373 566D 38 SEC
374 566E ED0101 SBC X1CORD
375 5671 801201 STA DELTAX
376 5674 AD0601 LDA X2CORD+1
377 5677 ED0201 SBC X1CORD+1
378 567A 801301 STA DELTAX+1
379 567D 1014 BPL DRAW2 ; SKIP AHEAD IF DIFFERENCE IS POSITIVE
380 567F CE1801 DEC XDIR ; SET XDIR TO -1
381 5682 38 SEC ; NEGATE DELTAX
382 5683 A900 LDA #0
383 5685 ED1201 SBC DELTAX
384 5688 801201 STA DELTAX
385 568B A900 LDA #0
386 568D ED1301 SBC DELTAX+1
387 5690 801301 STA DELTAX+1
388
389 ; COMPUTE SIGN AND MAGNITUDE OF DELTA Y = Y2-Y1
390 ; PUT MAGNITUDE IN DELTAY AND SIGN IN YDIR
391
392 5693 A900 DRAW2: LDA #0 ; FIRST ZERO YDIR
393 5695 801901 STA YDIR
394 5698 AD0701 LDA Y2CORD ; NEXT COMPUTE TWOS COMPLEMENT DIFFERENCE
395 569B 38 SEC
396 569C ED0301 SBC Y1CORD
397 569F 801401 STA DELTAY
398 56A2 AD0801 LDA Y2CORD+1
399 56A5 ED0401 SBC Y1CORD+1
400 56A8 801501 STA DELTAY+1
401 56AB 1014 BPL DRAW3 ; SKIP AHEAD IF DIFFERENCE IS POSITIVE
402 56AD CE1901 DEC YDIR ; SET YDIR TO -1
403 56B0 38 SEC ; NEGATE DELTAX
404 56B1 A900 LDA #0

```

VMSUP K-1008 VM GRAPHIC SUP
LINE DRAWING ROUTINES

```

405 56B3 ED1401    SBC  DELTAY
406 56B6 8D1401    STA  DELTAY
407 56B9 A900      LDA  #0
408 56BB ED1501    SBC  DELTAY+1
409 56BE 8D1501    STA  DELTAY+1
410
411 ; DETERMINE IF DELTAY IS LARGER THAN DELTAX
412 ; IF SO, EXCHANGE DELTAY AND DELTAX AND SET XCHFLG NONZERO
413 ; ALSO INITIALIZE ACC TO DELTAX
414 ; PUT A DOT AT THE INITIAL DENPOINT
415
416 56C1 A900      DRAW3: LDA  #0      ; FIRST ZERO XCHFLG
417 56C3 8D1A01    STA  XCHFLG
418 56C6 AD1401    LDA  DELTAY      ; COMPARE DELTAY WITH DELTAX
419 56C9 38        SEC
420 56CA ED1201    SBC  DELTAX
421 56CD AD1501    LDA  DELTAX+1
422 56D0 ED1301    SBC  DELTAX+1
423 56D3 9018      BCC  DRAW4      ; SKIP EXCHANGE IF DELTAX IS GREATER THAN
424 ; DELTAY
425 56D5 AE1401    LDX  DELTAY      ; EXCHANGE DELTAX AND DELTAY
426 56D8 AD1201    LDA  DELTAX
427 56D8 8D1401    STA  DELTAY
428 56DE 8E1201    STX  DELTAX
429 56E1 AE1501    LDX  DELTAY+1
430 56E4 AD1301    LDA  DELTAX+1
431 56E7 8D1501    STA  DELTAY+1
432 56EA 8E1301    STX  DELTAX+1
433 56ED CE1A01    DEC  XCHFLG      ; SET XCHFLG TO -1
434 56F0 AD1201    DRAW4: LDA  DELTAX      ; INITIALIZE ACC TO DELTAX
435 56F3 8D1601    STA  ACC
436 56F6 AD1301    LDA  DELTAX+1
437 56F9 8D1701    STA  ACC+1
438 56FC AD1B01    LDA  COLOR       ; PUT A DOT AT THE INITIAL ENDPOINT
439 56FF 20C155    JSR  WRPIX      ; XICORD,Y1CORD
440
441 ; HEAD OF MAIN DRAWING LOOP
442 ; TEST IF DONE
443
444 5702 AD1A01    DRAW45: LDA  XCHFLG      ; TEST IF X AND Y EXCHANGED
445 5705 D012      BNE  DRAW5      ; JUMP AHEAD IF SO
446 5707 AD0101    LDA  X1CORD      ; TEST FOR X1CORD=X2CORD
447 570A CD0501    CMP  X2CORD
448 570D D01F      BNE  DRAW7      ; GO FOR ANOTHER ITERATION IF NOT
449 570F AD0201    LDA  X1CORD+1
450 5712 CD0601    CMP  X2CORD+1
451 5715 D017      BNE  DRAW7      ; GO FOR ANOTHER ITERATION IF NOT
452 5717 F010      BEQ  DRAW6      ; GO RETURN IF SO
453 5719 AD0301    DRAW5: LDA  Y1CORD      ; TEST FOR Y1CORD=Y2CORD
454 571C CD0701    CMP  Y2CORD
455 571F D00D      BNE  DRAW7      ; GO FOR ANOTHER ITERATION IF NOT
456 5721 AD0401    LDA  Y1CORD+1
457 5724 CD0801    CMP  Y2CORD+1
458 5727 D005      BNE  DRAW7      ; GO FOR ANOTHER ITERATION IF NOT
459 5729 68        DRAW6: PLA

```

VMSUP K-1008 VM GRAPHIC SUP
LINE DRAWING ROUTINES

```

460 572A A8        461 572B 68      TAY
462 572C AA        TAX
463 572D 60        RTS      ; AND RETURN
464
465 ; DO A CLACULATION TO DETERMINE IF ONE OR BOTH AXES ARE TO BE
466 ; BUMPED (INCREMENTED OR DECREMENTED ACCORDING TO XDIR AND YDIR)
467 ; AND DO THE BUMPING
468
469 572E AD1A01    DRAW7: LDA  XCHFLG      ; TEST IF X AND Y EXCHANGED
470 5731 D006      BNE  DRAW8      ; JUMP IF SO
471 5733 20B957    JSR  BMPX
472 5736 4C3C57    JMP  DRAW9
473 5739 20A357    DRAW8: JSR  SBDY      ; BUMP Y IF SO
474 573C 206157    JSR  SBDY      ; SUBTRACT DY FROM ACC TWICE
475 573F 206157    DRAW9: JSR  BPL       ; SKIP AHEAD IF ACC IS NOT NEGATIVE
476 5742 1014      LDA  XCHFLG
477 5744 AD1A01    TEST  IF X AND Y EXCHANGED
478 5747 D006      BNE  DRAW10     ; JUMP IF SO
479 5749 20A357    JSR  BMPY      ; BUMP Y IF NOT
480 574C 4C5257    DRAW10: JSR  BMPX     ; BUMP X IF SO
481 574F 20B957    JSR  ADDX      ; ADD DX TO ACC TWICE
482 5752 207557    DRAW11: JSR  ADDX
483 5755 207557    JSR  ADDX
484
485 5758 AD1B01    DRAW12: LDA  COLOR      ; OUTPUT THE NEW POINT
486 575B 20C155    JSR  WRPIX
487 575E 4C0257    JMP  DRAW45     ; GO TEST IF DONE
488
489 ; SUBROUTINES FOR DRAW
490
491 5761 AD1601    SBDY: LDA  ACC      ; SUBTRACT DELTAY FROM ACC AND PUT RESULT
492 5764 38        SEC
493 5765 ED1401    SBC  DELTAY
494 5768 8D1601    STA  ACC
495 576B AD1701    LDA  ACC+1
496 576E ED1501    SBC  DELTAY+1
497 5771 8D1701    STA  ACC+1
498 5774 60        RTS
499
500
501 5775 AD1601    ADDX: LDA  ACC      ; ADD DELTAX TO ACC AND PUT RESULT IN ACC
502 5778 18        CLC
503 5779 6D1201    ADC  DELTAX
504 577C 8D1601    STA  ACC
505 577F AD1701    LDA  ACC+1
506 5782 6D1301    ADC  DELTAX+1
507 5785 8D1701    STA  ACC+1
508 5788 60        RTS
509
510
511 5789 AD1B01    BMPX: LDA  XDIR      ; BUMP XICORD BY +1 OR -1 ACCORDING TO
512 578C D009      BNE  BMPX2     ; XDIR
513 578E EO0101    INC  X1CORD     ; DOUBLE INCREMENT XICORD IF XDIR=0
514 5791 D003      BNE  BMPX1

```

VMSUP K-1008 VM GRAPHIC SUP
LINE DRAWING ROUTINES

```

515 5793 EE0201    INC   XICORD+1
516 5796 60        RTS
517 5797 AD0101    BMPX1: LDA   XICORD      ; DOUBLE DECREMENT XICORD IF XDIR<>0
518 579A D003        BNE   BMPX3
519 579C CEO201     DEC   XICORD+1
520 579F CEO101     DEC   XICORD
521 57A2 60        RTS
522
523
524 57A3 AD1901    BMPY: LDA   YDIR      ; BUMP YICORD BY +1 OR -1 ACCORDING TO
525 57A6 D009        BNE   BMPY2      ; YDIR
526 57A8 EEO301     INC   YICORD      ; DOUBLE INCREMENT YICORD IF YDIR=0
527 57AB D003        BNE   BMPY1
528 57AD EEO401     INC   YICORD+1
529 57B0 60        BMPY1: RTS
530 57B1 AD0301    BMPY2: LDA   YICORD      ; DOUBLE DECREMENT YICORD IF YDIR<>0
531 57B4 D003        BNE   BMPY3
532 57B6 CEO401     DEC   YICORD+1
533 57B9 CEO301    BMPY3: DEC   YICORD
534 57BC 60        RTS
535

```

VMSUP K-1008 VM GRAPHIC SUP
DCHAR - DRAW A CHARACTER

```

536          ;.PAGE 'DCHAR - DRAW A CHARACTER'
537          ;DCHAR - DRAW A CHARACTER WHOSE UPPER LEFT CORNER IS AT
538          ;XICORD,YICORD
539          ;XICORD AND YICORD ARE NOT ALTERED
540          ;THIS ROUTINE DISPLAYS A 5 BY 9 DOT MATRIX CHARACTER AT THE
541          ;SPECIFIED LOCATION. THE 5 BY 9 BLOCK IS CLEARED AND THEN THE
542          ;CHARACTER IS WRITTEN INTO IT.
543          ;THE 5 BY 9 MATRIX INCLUDES 2 LINE DESCENDERS ON LOWER CASE
544          ;CHARACTERS.
545          ;BOTH INDEX REGISTERS AND THE ACCUMULATOR ARE PRESERVED.
546          ;THE CHARACTER CODE TO BE DISPLAYED SHOULD BE IN A.
547          ;ASCII CONTROL CODES ARE IGNORED AND NO DRAWING IS DONE
548          ;THIS ROUTINE ASSUMES IN RANGE COORDINATES INCLUDING WIDTH AND
549          ;HEIGHT OF CHARACTER.
550 57BD 48        DCHAR: PHA      ; SAVE REGISTERS
551 57BE 8A        TXA
552 57BF 48        PHA
553 57C0 98        TYA
554 57C1 48        PHA
555 57C2 BA        TSX      ; GET INPUT CHARACTER BACK
556 57C3 BD0301    LDA   X'103,X
557 57C6 297F    AND   #X'7F      ; INSURE 7 BIT ASCII INPUT
558 57C8 38        SEC
559 57C9 E920    SBC   #X'20      ; TEST IF A CONTROL CHARACTER
560 57CB 3062    BMI   DCHAR5      ; DO A QUICK RETURN IF SO
561
562          ; CALCULATE FONT TABLE ADDRESS FOR CHAR
563
564 57CD 48        PHA      ; SAVE VERIFIED, ZERO ORIGIN CHAR CODE
565 57CE 202155    JSR   PIXADR      ; GET BYTE AND BIT ADDRESS OF FIRST SCAN
566
567 57D1 68        PLA      ; LINE OF CHARACTER INTO ADP1 AND BTPT
568 57D2 85EC    STA   ADP2      ; RESTORE ZERO ORIGIN CHARACTER CODE
569 57D4 A900    LDA   #0
570 57D6 85ED    STA   ADP2+1
571 57D8 20DC5A    JSR   SADP2L      ; COMPUTE 8*CHARACTER CODE IN ADP2
572 57D9 20DC5A    JSR   SADP2L
573 57D0 20DC5A    JSR   SADP2L
574 57E1 A5EC    LDA   ADP2      ; ADD IN ORIGIN OF CHARACTER TABLE
575 57E3 18        CLC
576 57E4 6976    ADC   #CHTB&X'FF
577 57E6 85EC    STA   ADP2
578 57E8 A5ED    LDA   ADP2+1
579 57EA 695C    ADC   #CHTB/256
580 57EC 85ED    STA   ADP2+1      ; ADP2 NOW HAS ADDRESS OF TOP ROW OF
581          ; CHARACTER SHAPE
582
583
584 57EE A000    LDY   #0      ; INITIALIZE Y INDEX = FONT TABLE POINTER
585 57F0 A200    LDX   #0      ; INITIALIZE X = SCAN LINE COUNTER
586
587          ; CLEAR THE FIRST TWO SCAN LINES OF DESCENDING CHARACTERS
588          ; FOR LOWER CASE "J", PUT IN THE DOT AS A SPECIAL CASE
589

```

VMSUP K-1008 VM GRAPHIC SUP
DCHAR - DRAW A CHARACTER

```

590 57F2 B1EC LDA (ADP2),Y ; GET THE FIRST ROW FROM THE TABLE
591 57F4 F01C BEQ DCHAR3 ; SKIP AHEAD IF NOT A DESCENDING CHARACTER
592 57F6 A5EC LDA ADP2 ; IF DESCENDING, TEST IF LOWER CASE J
593 57F8 C9C6 CMP #X'6A-X'20*8+CHTB&X'FF
594 57FA D004 BNE DCHAR1 ; CLEAR FIRST SCAN LINE IF NOT
595 57FC A920 LDA #X'20 ; LOAD THE DOT FOR THE J IF A J
596 57FE D002 BNE DCHAR2 ; DO THE FIRST SCAN LINE
597 5800 A900 DCHAR1: LDA #0 ; COUNT SCAN LINES DONE
598 5802 208558 JSR MERGE5
599 5805 20E15A JSR DN1SCN ; GO DOWN 1 SCAN LINE
600 5808 E8 INX ; COUNT SCAN LINES DONE
601 5809 A900 LDA #0 ; CLEAR THE SECOND SCAN LINE
602 580B 208558 JSR MERGE5
603 580E 20E15A JSR DN1SCN ; GO DOWN ANOTHER SCAN LINE
604 5811 E8 INX ; COUNT SCAN LINES DONE
605
606 ; SCAN OUT THE BODY OF THE CHARACTER
607
608 5812 C8 DCHAR3: INY ; GO TO NEXT SCAN LINE OF THE FONT
609 5813 B1EC LDA (ADP2),Y ; GET THE SCAN LINE
610 5815 208558 JSR MERGE5 ; MERGE IT WITH GRAPHIC MEMORY AT (ADP1)
611 5818 20E15A JSR DN1SCN ; GO DOWN 1 SCAN LINE
612 581B E8 INX ; COUNT SCAN LINES OUTPUTTED
613 581C C007 CPY #7 ; TEST IF WHOLE CHARACTER SCANNED OUT
614 581E D0F2 BNE DCHAR3 ; GO SCAN OUT ANOTHER ROW IF NOT
615 5820 E009 CPX #9 ; TEST IF THE WHOLE CHARACTER CELL SCANNED
616 5822 F008 BEQ DCHAR5 ; JUMP OUT IF SO
617 5824 A900 LDA #0 ; CLEAR TRAILING SCAN LINES ON
618 5826 208558 JSR MERGE5 ; NON-DESCENDING CHARACTERS
619 5829 20E15A JSR DN1SCN ; TO NEXT LINE
620 582C E8 INX ; COUNT LINES
621 582D D0F1 BNE DCHAR4 ; LOOP UNTIL DONE
622
623 ; RESTORE REGISTERS AND RETURN
624
625 582F 68 DCHAR5: PLA
626 5830 A8 TAY
627 5831 68 PLA
628 5832 AA TAX
629 5833 68 PLA
630 5834 60 RTS
631

```

VMSUP K-1008 VM GRAPHIC SUP
GRAPHIC MERGE ROUTINES

```

632 ; .PAGE 'GRAPHIC MERGE ROUTINES'
633 ; MERGEL - MERGE LEFT ROUTINE
634 ; MERGES ACCUMULATOR CONTENTS WITH A BYTE OF GRAPHIC MEMORY
635 ; ADDRESSED BY ADP1 AND BTPT.
636 ; BITS TO THE LEFT OF (BTPT) ARE PRESERVED IN GRAPHIC MEMORY.
637 ; BIT (BTPT) AND BITS TO THE RIGHT ARE SET EQUAL TO
638 ; CORRESPONDING BIT POSITIONS IN THE ACCUMULATOR.
639 ; NO REGISTERS ARE BOthered.

640 5835 48 MERGEL: PHA ; SAVE REGISTERS
641 5836 8A TXA
642 5837 48 PHA
643 5838 98 TYA
644 5839 48 PHA
645 583A BA TSX ; GET INPUT BACK
646 583B BD0301 LDA X'103,X
647 583E AC1101 LDY BTPT ; GET BIT NUMBER INTO Y
648 5841 39D058 AND MERGTR-1,Y ; CLEAR BITS TO BE PRESERVED IN MEMORY
649 5844 9D0301 STA X'103,X ; FROM A
650 5847 A000 LDY #0 ; CLEAR BITS FROM MEMORY TO BE CHANGED
651 5849 AE1101 LDX BTPT
652 584C B1EA LDA (ADP1),Y ; GET MEMORY BYTE
653 584E 3DCB58 AND MERGTL,X ; CLEAR THE BITS
654 5851 BA TSX ; DO THE MERGING
655 5852 1D0301 ORA X'103,X
656 5855 91EA STA (ADP1),Y ; RESTORE REGISTERS
657 5857 68 PLA
658 5858 A8 TAY
659 5859 68 PLA
660 585A AA TAX
661 585B 68 PLA
662 585C 60 RTS ; RETURN

663
664 ; MERGR - MERGE RIGHT ROUTINE
665 ; MERGES ACCUMULATOR CONTENTS WITH A BYTE OF GRAPHIC MEMORY
666 ; ADDRESSED BY ADP1 AND BTPT.
667 ; BITS TO THE RIGHT OF (BTPT) ARE PRESERVED IN GRAPHIC MEMORY.
668 ; BIT (BTPT) AND BITS TO THE LEFT ARE SET EQUAL TO CORRESPONDING
669 ; BIT POSITIONS IN THE ACCUMULATOR.
670 ; NO REGISTERS ARE BOthered.
671
672 585D 48 MERGR: PHA ; SAVE REGISTERS
673 585E 8A TXA
674 585F 48 PHA
675 5860 98 TYA
676 5861 48 PHA
677 5862 BA TSX ; GET INPUT BACK
678 5863 BD0301 LDA X'103,X
679 5866 AC1101 LDY BTPT ; GET BIT NUMBER INTO Y
680 5869 39C958 AND MERGTL+1,Y ; CLEAR BITS TO BE PRESERVED IN MEMORY
681 586C 9D0301 STA X'103,X ; FROM A
682 586F A000 LDY #0 ; CLEAR BITS FROM MEMORY TO BE CHANGED
683 5871 AE1101 LDX BTPT
684 5874 B1EA LDA (ADP1),Y ; GET MEMORY BYTE
685 5876 3DD158 AND MERGTR,X ; CLEAR THE BITS

```

VMSUP K-1008 VM GRAPHIC SUP
GRAPHIC MERGE ROUTINES

```

686 5879 BA      TSX      ; DO THE MERGING
687 587A 100301   ORA X'103,X
688 587D 91EA     STA (ADP1),Y
689 587F 68       PLA      ; RESTORE REGISTERS
690 5880 A8       TAY
691 5881 68       PLA
692 5882 AA       TAX
693 5883 68       PLA
694 5884 60       RTS      ; RETURN
695
696      ; MERGE A ROW OF 5 DOTS WITH GRAPHIC MEMORY STARTING AT BYTE
697      ; ADDRESS AND BIT NUMBER IN ADP1 AND BTPT
698      ; 5 DOTS TO MERGE LEFT JUSTIFIED IN A
699      ; PRESERVES X AND Y
700
701 5885 8D1D01   MERGE5: STA TEMP+1    ; SAVE INPUT DATA
702 5888 98       TYA      ; SAVE Y
703 5889 48       PHA
704 588A AC1101   LDY BTPT      ; OPEN UP A 5 BIT WINDOW IN GRAPHIC MEMORY
705 588D B90958   LDA MERGT5,Y ; LEFT BITS
706 5890 A000     LDY #0        ; ZERO Y
707 5892 31EA     AND (ADP1),Y
708 5894 91EA     STA (ADP1),Y
709 5896 AC1101   LDY BTPT
710 5899 B9E158   LDA MERGT5+8,Y ; RIGHT BITS
711 589C A001     LDY #1
712 589E 31EA     AND (ADP1),Y
713 58A0 91EA     STA (ADP1),Y
714 58A2 AD1D01   LDA TEMP+1    ; SHIFT DATA RIGHT TO LINE UP LEFTMOST
715 58A5 AC1101   LDY BTPT      ; DATA BIT WITH LEFTMOST GRAPHIC FIELD
716 58A8 F004     BEQ MERGE2    ; SHIFT BTPT TIMES
717 58A8 4A       MERGE1: LSR A
718 58AB 88       DEY
719 58AC DOFC     BNE MERGE1
720 58AE 11EA     MERGE2: ORA (ADP1),Y ; OVERLAY WITH GRAPHIC MEMORY
721 58B0 91EA     STA (ADP1),Y
722 58B2 A908     LDA #8        ; SHIFT DATA LEFT TO LINE UP RIGHTMOST
723 58B4 38       SEC
724 58B5 ED1101   SBC BTPT      ; DATA BIT WITH RIGHTMOST GRAPHIC FIELD
725 58B8 A8       TAY
726 58B9 AD1D01   LDA TEMP+1
727 58B C0        MERGE3: ASLA
728 58BD 88       DEY
729 58BE DOFC     BNE MERGE3
730 58C0 C8       INY
731 58C1 11EA     ORA (ADP1),Y ; OVERLAY WITH GRAPHIC MEMORY
732 58C3 91EA     STA (ADP1),Y
733 58C5 68       PLA      ; RESTORE Y
734 58C6 A8       TAY
735 58C7 60       RTS      ; RETURN
736
737 58C8 0080C0E0  MERGTL: .BYTE X'00,X'80,X'C0,X'E0 ; MASKS FOR MERGE LEFT
738 58CC F0FBFCFE .BYTE X'FO,X'FB,X'FC,X'FE ; CLEAR ALL BITS TO THE RIGHT OF
739 58D0 FF        .BYTE X'FF          ; AND INCLUDING BIT N (0=MSB)
740

```

VMSUP K-1008 VM GRAPHIC SUP
GRAPHIC MERGE ROUTINES

```

741 58D1 7F3F1FOF  MERGTR: .BYTE X'7F,X'3F,X'1F,X'0F ; MASKS FOR MERGE RIGHT
742 58D5 07030100   .BYTE X'07,X'03,X'01,X'00 ; CLEAR ALL BITS TO THE LEFT OF
743                                     ; AND INCLUDING BIT N (0=MSB)
744
745 58D9 0783C1E0  MERGT5: .BYTE X'07,X'83,X'C1,X'E0 ; TABLE OF MASKS FOR OPENING UP
746 58D0 F0F8FCFE   .BYTE X'FO,X'FB,X'FC,X'FE ; A 5 BIT WINDOW ANYWHERE
747 58E1 FFFFFFFF   .BYTE X'FF,X'FF,X'FF,X'FF ; IN GRAPHIC MEMORY
748 58E5 7F3F1FOF   .BYTE X'7F,X'3F,X'1F,X'0F
749

```

VMSUP K-1008 VM GRAPHIC SUP
DTEXT - SOPHISTICATED TEXT DISPLAY ROUTINE

```

.PAGE 'DTEXT - SOPHISTICATED TEXT DISPLAY ROUTINE'
DTEXT - SOPHISTICATED TEXT DISPLAY ROUTINE
CURSOR IS ADDRESSED IN TERMS OF X AND Y COORDINATES.
CURRENT CURSOR POSITION IS IN XCORD AND YCORD WHICH IS THE
COORDINATES OF THE UPPER LEFT CORNER OF THE CHARACTER POINTED
TO BY THE CURSOR.
CURSOR POSITIONING MAY BE ACCOMPLISHED BY DIRECTLY
MODIFYING XCORD,YCORD OR BY ASCII CONTROL CODES OR BY
CALLING THE CURSOR MOVEMENT SUBROUTINES DIRECTLY.
LIKEWISE BASELINE SHIFT FOR SUB AND SUPERSCRIPT MAY BE DONE
DIRECTLY OR WITH CONTROL CHARACTERS.
ADDITIONAL CONTROL CHARACTER FUNCTIONS ARE EASILY ADDED BY
ADDING ENTRIES TO A DISPATCH TABLE AND CORRESPONDING SERVICE
ROUTINES
CURSOR IS A NON-BLINKING UNDERLINE

CONTROL CODES RECOGNIZED:
CR X'0D SETS CURSOR TO LEFT SCREEN EDGE
LF X'0A MOVES CURSOR DOWN ONE LINE, SCROLLS DISPLAY BOUNDED
BY THE MARGINS UP ONE LINE IF ALREADY ON BOTTOM LINE
BS X'08 MOVES CURSOR ONE CHARACTER LEFT
FF X'0C CLEARS SCREEN BETWEEN THE MARGINS AND PUTS CURSOR AT
TOP AND LEFT MARGIN
SI X'0F MOVES BASELINE UP 3 SCAN LINES FOR SUPERSCRIPTS
SO X'0E MOVES BASELINE DOWN 3 SCAN LINES FOR SUBSCRIPTS
DC1 X'11 MOVES CURSOR LEFT ONE CHARACTER WIDTH
DC2 X'12 MOVES CURSOR RIGHT ONE CHARACTER WIDTH
DC3 X'13 MOVES CURSOR UP ONE CHARACTER HEIGHT
DC4 X'14 MOVES CURSOR DOWN ONE CHARACTER HEIGHT
NO WRAPAROUND OR SCROLLING IS DONE WHEN DC1-DC4 IS
USED TO MOVE THE CURSOR.

WHEN CALLS TO DTEXT ARE INTERMINGLED WITH CALLS TO THE GRAPHIC
ROUTINES, CSRINS AND CSRDEL SHOULD BE CALLED TO INSERT AND
DELETE THE CURSOR RESPECTIVELY. LIKEWISE THESE ROUTINES
SHOULD BE USED WHEN THE USER PROGRAM DIRECTLY MODIFIES THE
CURSOR POSITION BY CHANGING XCORD AND YCORD. IF THIS IS
NOT DONE, THE CURSOR SYMBOL MAY NOT SHOW UNTIL THE FIRST
CHARACTER HAS BEEN DRAWN OR MAY REMAIN AT THE LAST CHARACTER
DRAWN.

DTEXT USES A VIRTUAL PAGE DEFINED BY TOP, BOTTOM, LEFT, AND
RIGHT MARGINS. CURSOR MOVEMENT, SCROLLING, CLEARING, AND TEXT
DISPLAY IS RESTRICTED TO THE AREA DEFINED BY TMAR, BMAR, LMAR,
AND RMAR RESPECTIVELY. VALID MARGIN SETTINGS ARE ASSUMED
WHICH MEANS THAT THE MARGINS DEFINE SPACE AT LEAST TWO
CHARACTERS WIDE BY ONE LINE HIGH AND THAT ALL OF THEM ARE
VALID COORDINATES. A CONVENIENCE ROUTINE, DTXTIN, MAY BE
CALLED TO INITIALIZE THE MARGINS FOR USE OF THE FULL SCREEN IN
PURE TEXT DISPLAY APPLICATIONS.

AUTOMATIC SCROLLING IS PERFORMED BY THE LINE FEED CONTROL
CHARACTER PROCESSOR. FOR SCROLLING TO FUNCTION PROPERLY, AT
LEAST TWO LINES OF CHARACTERS MUST FIT BETWEEN THE TOP AND
BOTTOM MARGINS AND SUPERSCRIPTS AND SUBSCRIPTS SHOULD BE

```

VMSUP K-1008 VM GRAPHIC SUP
DTEXT - SOPHISTICATED TEXT DISPLAY ROUTINE

```

804      ; AVOIDED UNLESS CHHIW IS REDEFINED TO PROVIDE ENOUGH WINDOW
805      ; AREA TO HOLD THE SHIFTED CHARACTERS WITHOUT OVERLAP WITH
806      ; ADJECANT LINES.

808      ; DTXTIN MAY BE CALLED TO INITIALIZE DTEXT FOR USE AS A FULL
809      ; SCREEN TEXT DISPLAY ROUTINE. SETS MARGINS FOR FULL SCREEN
810      ; OPERATION, CLEARS THE SCREEN, AND SETS THE CURSOR AT THE UPPER
811      ; LEFT CORNER OF THE SCREEN. THE USER MUST STILL SET VMORG
812      ; HOWEVER!
813      ; DTXTIN - CONVENIENT INITIALIZE ROUTINE FOR FULL SCREEN USE OF
814      ; DTEXT.
815      ; DTXTIN: LDA #0      ; SET LEFT AND BOTTOM MARGINS TO ZERO
816      ; STA LMAR
817 58E9 A900 DTXTIN: LDA #0      ; SET TOP MARGIN TO TOP OF SCREEN
818 58EB 800001 STA LMAR+1
819 58EE 800E01 STA BMAR
820 58F1 800B01 STA BMAR+1
821 58F4 800C01 STA NY-1/256
822 58F7 A9C7 LDA #NY-18X'FF ; SET RIGHT MARGIN TO RIGHT EDGE OF SCREEN
823 58F9 800901 STA RMAR
824 58FC A900 STA RMAR+1
825 58FE 800A01 LDA #NX-1/256 ; SET RIGHT MARGIN TO RIGHT EDGE OF SCREEN
826 5901 A93F STA RMAR
827 5903 800F01 LDA #NX-1/256 ; SET RIGHT MARGIN TO RIGHT EDGE OF SCREEN
828 5906 A901 STA RMAR+1
829 5908 801001 LDA #X'0C ; CLEAR SCREEN AND PUT CURSOR AT UPPER
830 590B A90C ; LEFT CORNER BY SENDING AN ASCII FF
831           ; CONTROL CHARACTER TO DTEXT. THEN FALL
832           ; INTO DTEXT.
833           ; DTXT - DISPLAY ASCII TEXT ROUTINE
834           ; ENTER WITH ASCII CHARACTER CODE TO DISPLAY OR INTERPRET IN A.
835           ; PRESERVES ALL REGISTERS.
836           ; DTEXT: PHA      ; SAVE THE REGISTERS
837           ; TXA
838           ; PHA
839 590D 48 DTEXT: PHA      ; SAVE THE REGISTERS
840 590E 8A TXA
841 590F 48 PHA
842 5910 98 TYA
843 5911 48 PHA
844 5912 BA TSX      ; GET INPUT BACK
845 5913 BD0301 LDA X'103,X ; FROM THE STACK
846 5916 297F AND #X'7F ; INSURE 7 BIT ASCII INPUT
847 5918 C920 CMP #X'20 ; TEST IF A CONTROL CHARACTER
848 591A 300C BMI DTXTI ; JUMP AHEAD IF SO
849 591C 20BD57 JSR DCHAR ; FOR A REGULAR TEXT CHARACTER, DISPLAY IT
850 591F 20F05B JSR CSRR ; DO A CURSOR RIGHT
851 5922 68 DTEXTR: PLA ; RESTORE THE REGISTERS
852 5923 A8 TAY
853 5924 68 PLA
854 5925 AA TAX
855 5926 68 PLA
856 5927 60 RTS ; AND RETURN
857           ; DTEXT1: LDX #0 ; SET UP A LOOP TO SEARCH THE CONTROL
858 5928 A200

```

VMSUP K-1008 VM GRAPHIC SUP
DTEXT - SOPHISTICATED TEXT DISPLAY ROUTINE

```

859 592A DD585C DTEXT2: CMP CCTAB,X ; CHARACTER TABLE FOR A MATCH
860 592D F009 BEQ DTEXT3 ; JUMP IF A MATCH
861 592F E8 INX ; BUMP X TO POINT TO NEXT TABLE ENTRY
862 5930 E8 INX
863 5931 E8 INX
864 5932 E01E CPX #CCTAB-E-CCTAB; TEST IF ENTIRE TABLE SEARCHED
865 5934 D0F4 BNE DTEXT2 ; LOOP IF NOT
866 5936 FOEA BEQ DTEXTR ; GO RETURN IF ENTIRE TABLE SEARCHED
867
868 5938 BD5A5C DTEXT3: LDA CCTAB+2,X ; JUMP TO THE ADDRESS IN THE NEXT TWO
869 593B 48 PHA ; TABLE BYTES
870 593C BD595C LDA CCTAB+1,X
871 593F 48 PHA
872 5940 60 RTS

```

VMSUP K-1008 VM GRAPHIC SUP
SERVICE ROUTINES FOR CONTROL CHARACTERS

```

874 ; .PAGE 'SERVICE ROUTINES FOR CONTROL CHARACTERS'
875 ; SERVICE ROUTINES FOR CONTROL CHARACTERS. DO THE INDICATED
876 ; FUNCTION AND JUMP TO DTEXTR TO RESTORE REGISTERS AND RETURN.
877 ; CRR - CURSOR RIGHT
878
879 5941 20F05B CRR: JSR CSRR ; MOVE CURSOR RIGHT
880 5944 4C2259 JMP DTEXTR ; GO RETURN
881
882 ; CRL - CURSOR LEFT AND BACKSPACE
883
884 5947 200A5C CRL: JSR CSRL ; MOVE CURSOR LEFT
885 594A 4C2259 JMP DTEXTR ; GO RETURN
886
887 ; CRU - CURSOR UP
888
889 594D 20245C CRU: JSR CSRU ; MOVE CURSOR UP
890 5950 4C2259 JMP DTEXTR ; GO RETURN
891
892 ; CRD - CURSOR DOWN
893
894 5953 203E5C CRD: JSR CSRD ; MOVE CURSOR DOWN
895 5956 4C2259 JMP DTEXTR ; GO RETURN
896
897 ; BASUP - SHIFT BASELINE UP 3 SCAN LINES
898 ; NOTE - NO RANGE CHECK ON THE Y COORDINATE IS MADE
899 ; BASELINE SHIFTING SHOULD ONLY BE DONE AT A BLANK CHARACTER
900 ; POSITION
901
902 5959 20C95B BASUP: JSR CSRDEL ; DELETE CURRENT CURSOR
903 595C AD030I LDA Y1CORD ; INCREMENT Y COORDINATE BY 3
904 595F 18 CLC
905 5960 6903 ADC #3
906 5962 8D0301 STA Y1CORD
907 5965 9003 BCC BASUP1
908 5967 EE0401 INC Y1CORD+1
909 596A 20C55B BASUP1: JSR CSRINS ; DISPLAY CURSOR AT NEW LOCATION
910 596D 4C2259 JMP DTEXTR ; GO RETURN
911
912 ; BASDN - SHIFT BASELINE DOWN 3 SCAN LINES
913 ; NOTE - NO RANGE CHECK ON THE Y COORDINATE IS MADE
914 ; BASELINE SHIFTING SHOULD ONLY BE DONE AT A BLANK CHARACTER
915 ; POSITION
916
917 5970 20C95B BASDN: JSR CSRDEL ; DELETE CURRENT CURSOR
918 5973 AD030I LDA Y1CORD ; DECREMENT Y COORDINATE BY 3
919 5976 38 SEC
920 5977 E903 SBC #3
921 5979 8D0301 STA Y1CORD
922 597C B003 BCS BASDN1
923 597E CE0401 DEC Y1CORD+1
924 5981 20C55B BASDN1: JSR CSRINS ; DISPLAY CURSOR AT NEW LOCATION
925 5984 4C2259 JMP DTEXTR ; GO RETURN
926
927 ; CARRET - CARRIAGE RETURN

```

VMSUP K-1008 VM GRAPHIC SUP
SERVICE ROUTINES FOR CONTROL CHARACTERS

```

928
929 5987 20C95B  CARRET: JSR CSRDEL ; DELETE CURRENT CURSOR
930 598A A0D001  LDA LMAR ; SET XICORD TO THE LEFT MARGIN
931 598D 8D0101  STA XICORD
932 5990 A0D0E01  LDA LMAR+1
933 5993 8D0201  STA XICORD+1
934 5996 20C55B  JSR CSRINS ; DISPLAY CURSOR AT NEW LOCATION
935 5999 4C2259  JMP DTEXTR ; GO RETURN
936
937 ; LNFFED - LINE FEED ROUTINE, SCROLLS IF NOT SUFFICIENT SPACE
938 ; AT THE BOTTOM FOR A NEW LINE
939
940 599C 20695B  LNFFED: JSR DNST ; TEST IF CURSOR IS TOO FAR DOWN TO ALLOW
941 599F 9006  BCC LNFFED1 ; MOVEMENT
942 59A1 203E5C  JSR CSRD ; IF OK, DO A SIMPLE CURSOR DOWN
943 59A4 4C2259  JMP DTEXTR ; AND GO RETURN
944 59A7 20C95B  JSR CSRDEL ; DELETE CURRENT CURSOR
945 59AA 20E05A  JSR RECTP ; SAVE CURSOR COORDINATES AND PROCESS
946 ; CORNER DATA
947 59AD AD1201  LNFFED0: LDA TLBYT ; ADD CHHIW SCAN LINES TO ADDRESS OF TOP
948 5980 18  CLC ; LEFT CORNER TO ESTABLISH ADDRESS OF
949 5981 6988  ADC #CHHIW^NX/8&X^FF ; FIRST SCAN LINE TO SCROLL
950 5983 85EC  STA ADP2 ; AND PUT INTO ADP2
951 5985 AD1301  LDA TLBYT+1
952 5988 6901  ADC #CHHIW^NX/8/256
953 598A 85ED  STA ADP2+1
954
955 ; MOVE LEFT PARTIAL BYTE
956
957 59BC AD1201  LNFFED2: LDA TLBYT ; MOVE CURRENT TOP LEFT BYTE ADDRESS INTO
958 59BF 85EA  STA ADP1 ; ADP1
959 59C1 AD1301  LDA TLBYT+1
960 59C4 85EB  STA ADP1+1
961 59C6 AD1801  LDA TLBIT ; MOVE LEFT BIT ADDRESS TO BTPT
962 59C9 8D1101  STA BTPT
963 59CC A000  LDY #0
964 59CE B1EC  LDA (ADP2),Y ; MOVE A PARTIAL BYTE FROM (ADP2)
965 59D0 203558  JSR MERGEL ; TO (ADP1) ACCORDING TO BTPT
966
967 ; MOVE FULL BYTES IN THE MIDDLE
968
969 59D3 E6EA  LNFFED3: INC ADP1 ; INCREMENT ADP1
970 59D5 D002  BNE LNFFED4
971 59D7 E6EB  INC ADP1+1
972 59D9 E6EC  LNFFED4: INC ADP2 ; INCREMENT ADP2
973 59D8 D002  BNE LNFFED5
974 59D0 E6ED  INC ADP2+1
975 59D9 A5EA  LNFFED5: LDA ADP1 ; TEST IF EQUAL TO CURRENT TOP RIGHT BYTE
976 59E1 CD1401  CMP TRBYT ; ADDRESS
977 59E4 D007  BNE LNFFED6 ; SKIP AHEAD IF NOT
978 59E6 A5EB  LDA ADP1+1
979 59E8 CD1501  CMP TRBYT+1
980 59EB F007  BEQ LNFFED7 ; GO TO RIGHT PARTIAL BYTE PROCESSING IF =
981 59ED B1EC  LNFFED6: LDA (ADP2),Y ; MOVE A BYTE
982 59EF 91EA  STA (ADP1),Y

```

VMSUP K-1008 VM GRAPHIC SUP
SERVICE ROUTINES FOR CONTROL CHARACTERS

```

983 59F1 4CD359
984 ; LNFFED3 ; GO PROCESS NEXT BYTE
985 ; MOVE RIGHT PARTIAL BYTE
986
987 59F4 AD1901  LNFFED7: LDA TRBIT ; MOVE RIGHT BIT ADDRESS TO BTPT
988 59F7 8D1101  STA BTPT
989 59FA B1EC  LDA (ADP2),Y ; MOVE A PARTIAL BYTE FROM (ADP2) TO
990 59FC 205058  JSR MERGER ; (ADP1) ACCORDING TO BTPT
991 59FF A5EC  LDA ADP2 ; TEST IF ADP2 = BRBYT
992 5A01 CD1601  CMP BRBYT
993 5A04 D009  BNE LNFFED8 ; JUMP AHEAD IF NOT
994 5A06 A5ED  LDA ADP2+1
995 5A08 CD1701  CMP BRBYT+1
996 5A0B D002  BNE LNFFED8 ; JUMP AHEAD IF NOT
997 5A0D F01F  BEQ LNFFEDB ; FINISHED WITH MOVE PART OF SCROLL, GO
998 ; CLEAR AREA LEFT AT BOTTOM OF RECTANGLE
1000
1001 ; PREPARE TO START NEXT LINE
1002 5A0F AD1201  LNFFED8: LDA TLBYT ; ADD NX/8 TO TOP LEFT BYTE ADDRESS
1003 5A12 18  CLC ; #NX/8
1004 5A13 6928  ADC TLBYT
1005 5A15 8D1201  STA LNFED9
1006 5A18 9003  BCC INC TLBYT+1
1007 5A1A EE1301  INC TLBYT+1
1008 5A1D AD1401  LNFED9: LDA TRBYT ; ADD NX/8 TO TOP RIGHT BYTE ADDRESS
1009 5A20 18  CLC ; #NX/8
1010 5A21 6928  ADC TRBYT
1011 5A23 8D1401  STA LNFED0
1012 5A26 9085  BCC INC TRBYT+1
1013 5A28 EE1501  INC LNFED0 ; GO MOVE NEXT SCAN LINE
1014 5A2B 4CAD59  JMP LNFED0
1015
1016 ; CLEAR REGION AT BOTTOM OF RECTANGLE FOR NEW LINE OF TEXT
1017 ; AND REINSERT CURSOR
1018
1019 5A2E 20735A  LNFEDB: JSR LNCLR ; DO THE CLEARING
1020 5A31 AD0501  LDA X2CORD ; RESTORE CURSOR COORDINATES
1021 5A34 8D0101  STA XICORD
1022 5A37 AD0601  LDA X2CORD+1
1023 5A3A 8D0201  STA XICORD+1
1024 5A3D AD0701  LDA Y2CORD
1025 5A40 8D0301  STA YICORD
1026 5A43 AD0801  LDA Y2CORD+1
1027 5A46 8D0401  STA YICORD+1
1028 5A49 20C55B  JSR CSRINS ; INSERT CURSOR AT THE SAME POSITION
1029 5A4C 4C2259  JMP DTEXTR ; GO RETURN
1030
1031 ; FMFED - FORM FEED ROUTINE, CLEARS THE SCREEN BETWEEN THE
1032 ; MARGINS AND PLACES CURSOR AT UPPER LEFT CORNER OF
1033 ; RECTANGLE DEFINED BY THE MARGINS.
1034 ; NOTE: ROUTINE MODIFIES BOTH ADDRESS POINTERS AND BOTH SETS OF
1035 ; COORDINATES.
1036
1037 5A4F 20E05A  FMFED: JSR RECTP ; PROCESS MARGIN DATA INTO CORNER

```

VMSUP K-1008 VM GRAPHIC SUP
SERVICE ROUTINES FOR CONTROL CHARACTERS

```

1038          ; BYTE AND BIT ADDRESSES
1039 5A52 20735A  JSR LNCLR  ; CLEAR THE AREA DEFINED BY THE CORNERS
1040 5A55 AD0001  LDA LMAR   ; POSITION CURSOR AT TOP AND LEFT MARGINS
1041 5A58 800101  STA X1CORD
1042 5A5B ADE001  LDA LMAR+1
1043 5A5E 8D0201  STA X1CORD+1
1044 5A61 AD0901  LDA TMAR
1045 5A64 8D0301  STA Y1CORD
1046 5A67 AD0A01  LDA TMAR+1
1047 5A6A 8D0401  STA Y1CORD+1
1048 5A6D 20C55B  JSR CSRINS ; INSERT CURSOR
1049 5A70 4C2259  JMP DTEXTR ; FINISHED WITH FORM FEED
1050

```

VMSUP K-1008 VM GRAPHIC SUP
MISCELLANEOUS INTERNAL SUBROUTINES

```

1051          ; .PAGE 'MISCELLANEOUS INTERNAL SUBROUTINES'
1052          ; LNCLR - SUBROUTINE TO CLEAR AREA INSIDE OF THE MARGINS
1053          ; DEFINED BY TLBYT,TLBIT; TRBYT,TRBIT; BRBYT
1054          ; USED BY FORM FEED AND SCROLL TO CLEAR BETWEEN THE MARGINS
1055          ; CLEAR LEFT PARTIAL BYTE
1056          ; USES INDEX Y
1057 5 73 AD1201  LNCLR: LDA TLBYT    ; MOVE CURRENT TOP LEFT BYTE ADDRESS INTO
1058 5A76 85EA    STA ADP1      ; ADP1
1059 5A78 AD1301  LDA TLBYT+1
1060 5A7B 85EB    STA ADP1+1
1061 5A7D AD1801  LDA TLBIT     ; MOVE LEFT BIT ADDRESS TO BTPT
1062 5A80 8D1101  STA BTPT
1063 5A83 A900    LDA #0       ; CLEAR LEFT PARTIAL BYTE
1064 5A85 203558  JSR MERGL
1065
1066          ; CLEAR FULL BYTES IN THE MIDDLE
1067          ; LNCLR1: INC ADP1      ; INCREMENT ADP1
1068 5A88 E6EA    BNE LNCLR2
1069 5A8A D002    INC ADP1+1
1070 5ABC E6EB    LNCLR2: LDA ADP1      ; TEST IF EQUAL TO CURRENT TOP RIGHT BYTE
1071 5A8E A5EA    CMP TRBYT
1072 5A90 CD1401  BNE LNCLR3
1073 5A93 D007    LDA ADP1+1
1074 5A95 A5EB    CMP TRBYT+1
1075 5A97 CD1501  BEQ LNCLR4
1076 5A9A F007    LNCLR3: LDA #0      ; GO TO RIGHT PARTIAL BYTE PROCESSING IF =
1077 5A9C A900    TAY
1078 5A9E A8      STA (ADP1),Y
1079 5A9F 91EA    BEQ LNCLR1
1080 5AA1 FOE5    ; LOOP UNTIL ALL FULL BYTES ON THIS LINE
1081          ; HAVE BEEN CLEARED
1082
1083          ; CLEAR RIGHT PARTIAL BYTE
1084          ; LNCLR4: LDA TRBIT    ; MOVE RIGHT BIT ADDRESS TO BTPT
1085 5AA3 AD1901  STA BTPT
1086 5AA6 8D1101  LDA #0      ; CLEAR RIGHT PARTIAL BYTE
1087 5AA9 A900    JSR MERGL
1088 5AAB 205058  LDA ADP1      ; TEST IF ADP1 = BRBYT
1089 5AAE A5EA    CMP BRBYT
1090 5AB0 CD1601  BNE LNCLR5
1091 5AB3 D008    LDA ADP1+1
1092 5AB5 A5EB    CMP BRBYT+1
1093 5AB7 CD1701  BNE LNCLR5
1094 5ABA D001    RTS
1095 5ABC 60      ; JUMP AHEAD IF NOT
1096          ; FINISHED WITH CLEAR IF SO
1097          ; PREPARE TO START NEXT LINE
1098
1099 5ABD AD1201  LNCLR5: LDA TLBYT    ; ADD NX/8 TO TOP LEFT BYTE ADDRESS
1100 5AC0 18      CLC
1101 5AC1 6928    ADC #NX/8
1102 5AC3 8D1201  STA TLBYT
1103 5AC6 9003    BCC LNCLR6
1104 5AC8 EE1301  INC TLBYT+1

```

VMSUP K-1008 VM GRAPHIC SUP
MISCELLANEOUS INTERNAL SUBROUTINES

```

1105 5ACB AD1401 LNCLR6: LDA TRBYT ; ADD NX/8 TO TOP RIGHT BYTE ADDRESS
1106 5ACE 18 CLC
1107 5ACF 6928 ADC #NX/8
1108 5AD1 8D1401 STA TRBYT
1109 5AD4 9090 BCC LNCLR ; GO PROCESS NEXT LINE
1110 5AD6 EE1501 INC TRBYT+1
1111 5AD9 4C735A JMP LNCLR
1112
1113 ; SADP2L - SHIFT ADP2 LEFT 1 BIT POSITION
1114 ; NO REGISTERS BOTHERED
1115
1116 5ADC 06EC SADP2L: ASL ADP2 ; SHIFT LOW PART
1117 5ADE 26ED ROL ADP2+1 ; SHIFT HIGH PART
1118 5AE0 60 RTS ; RETURN
1119
1120 ; DN1SCN - SUBROUTINE TO ADD NX/8 TO ADP1 TO EFFECT A DOWN
1121 ; SHIFT OF ONE SCAN LINE
1122 ; INDEX REGISTERS PRESERVED
1123
1124 5AE1 A5EA DN1SCN: LDA ADP1 ; ADD NX/8 TO LOW ADP1
1125 5AE3 18 CLC
1126 5AE4 6928 ADC #NX/8
1127 5AE6 85EA STA ADP1
1128 5AE8 9002 BCC DN1SC1
1129 5AEA E6EB INC ADP1+1 ; INCREMENT HIGH PART IF CARRY FROM LOW
1130 5AEC 60 DN1SC1: RTS ; RETURN
1131
1132 ; SUBROUTINE TO ESTABLISH USEFUL DATA ABOUT THE RECTANGLE
1133 ; DEFINED BY THE TEXT MARGINS IN TERMS OF BYTE AND BIT ADDR.
1134 ; TLBYT AND TLBIT DEFINE THE UPPER LEFT CORNER, TRBYT AND TRBIT
1135 ; DEFINE UPPER RIGHT CORNER, BRBYT DEFINES BOTTOM RIGHT CORNER
1136
1137 5AED AD0101 RECTP: LDA X1CORD ; SAVE CURRENT CURSOR POSITION IN
1138 5AF0 8D0501 STA X2CORD ; X2CORD AND Y2CORD
1139 5AF3 AD0201 LDA X1CORD+1
1140 5AF6 8D0601 STA X2CORD+1
1141 5AF9 AD0301 LDA Y1CORD
1142 5AFC 8D0701 STA Y2CORD
1143 5AFF AD0401 LDA Y1CORD+1
1144 5B02 8D0801 STA Y2CORD+1
1145 5B05 AD0001 LDA LMAR ; ESTABLISH BYTE AND BIT ADDRESSES OF
1146 5B08 8D0101 STA X1CORD ; TOP LEFT CORNER
1147 5B08 AD0E01 LDA LMAR+1
1148 5B0E 8D0201 STA X1CORD+1
1149 5B11 AD0901 LDA TMAR
1150 5B14 8D0301 STA Y1CORD
1151 5B17 AD0A01 LDA TMAR+1
1152 5B1A 8D0401 STA Y1CORD+1
1153 5B1D 202155 JSR PIXADR
1154 5B20 A5EA LDA ADP1
1155 5B22 8D1201 STA TLBYT
1156 5B25 A5EB LDA ADP1+1
1157 5B27 8D1301 STA TLBYT+1
1158 5B2A AD1101 LDA BTPT
1159 5B2D 8D1801 STA TLBIT

```

VMSUP K-1008 VM GRAPHIC SUP
MISCELLANEOUS INTERNAL SUBROUTINES

```

1160 5B30 AD0F01 LDA RMAR ; ESTABLISH BYTE AND BIT ADDRESSES OF TOP
1161 5B33 8D0101 STA X1CORD ; RIGHT CORNER
1162 5B36 AD1001 LDA RMAR+1
1163 5B39 8D0201 STA X1CORD+1
1164 5B3C 202155 JSR PIXADR
1165 5B3F A5EA LDA ADP1
1166 5B41 8D1401 STA TRBYT
1167 5B44 A5EB LDA ADP1+1
1168 5B46 8D1501 STA TRBYT+1
1169 5B49 AD1101 LDA BTPT
1170 5B4C 8D1901 STA TRBIT
1171 5B4F AD0B01 LDA BMAR ; ESTABLISH BYTE ADDRESS OF BOTTOM RIGHT
1172 5B52 8D0301 STA Y1CORD ; CORNER; BIT ADDRESS IS SAME AS BIT
1173 5B55 AD0C01 LDA BMAR+1 ; ADDRESS OF TOP RIGHT CORNER
1174 5B58 8D0401 STA Y1CORD+1
1175 5B5B 202155 JSR PIXADR
1176 5B5E A5EA LDA ADP1
1177 5B60 8D1601 STA BRBYT
1178 5B63 A5EB LDA ADP1+1
1179 5B65 8D1701 STA BRBYT+1
1180 5B68 60 RTS ; RETURN
1181

```

VMSUP K-1008 VM GRAPHIC SUP
CURSOR-BORDER LIMIT TEST ROUTINES

```

.PAGE 'CURSOR-BORDER LIMIT TEST ROUTINES'
1182      ; CURSOR-BORDER LIMIT TEST ROUTINES
1183      ; TESTS IF ENOUGH SPACE TO ALLOW CURSOR MOVEMENT IN ANY OF 4
1184      ; DIRECTIONS. RETURNS WITH POSITIVE OR ZERO RESULT IF ENOUGH
1185      ; SPACE AND A NEGATIVE RESULT IF NOT ENOUGH SPACE.
1186      ; SUBROUTINES USE A AND X
1187
1188 5B69 AD0301  DNTST: LDA Y1CORD    ; COMPUTE Y1CORD-BMAR-(2*CHHIW-2)
1189 5B6C 38          SEC
1190 5B6D ED0B01  SBC BMAR      ; SIGN OF RESULT
1191 5B70 AA          TAX         ; - NOT OK
1192 5B71 AD0401  LDA Y1CORD+1  ; Z OK
1193 5B74 EDOC01  SBC BMAR+1   ; + OK
1194 5B77 48          PHA
1195 5B78 8A          TXA
1196 5B79 38          SEC
1197 5B7A E914  SBC #2*CHHIW-2
1198 5B7C 68          PLA
1199 5B7D E900  SBC #0
1200 5B7F 60          RTS
1201
1202 5B80 AD0901  UPTST: LDA TMAR    ; COMPUTE TMAR-Y1CORD-CHHIW
1203 5B83 38          SEC
1204 5B84 ED0301  SBC Y1CORD    ; SIGN OF RESULT
1205 5B87 AA          TAX         ; - NOT OK
1206 5B88 AD0A01  LDA TMAR+1   ; Z OK
1207 5B8B ED0401  SBC Y1CORD+1  ; + OK
1208 5B8E 48          PHA
1209 5B8F 8A          TXA
1210 5B90 38          SEC
1211 5B91 E90B  SBC #CHHIW
1212 5B93 68          PLA
1213 5B94 E900  SBC #0
1214 5B96 60          RTS
1215
1216 5B97 AD0101  LFTST: LDA X1CORD  ; COMPUTE X1CORD-LMAR-CHWIDW
1217 5B9A 38          SEC
1218 5B9B ED0001  SBC LMAR      ; SIGN OF RESULT
1219 5B9E AA          TAX         ; - NOT OK
1220 5B9F AD0201  LDA X1CORD+1  ; Z OK
1221 5BAC ED0E01  SBC LMAR+1   ; + OK
1222 5BAC 48          PHA
1223 5BAC 8A          TXA
1224 5BAC 38          SEC
1225 5BAC E906  SBC #CHWIDW
1226 5BAC 68          PLA
1227 5BAC E900  SBC #0
1228 5BAC 60          RTS
1229
1230 5BAE AD0F01  RTTST: LDA RMAR    ; COMPUTE RMAR-X1CORD-(2*CHWIDW-2)
1231 5BB1 38          SEC
1232 5BB2 ED0101  SBC X1CORD  ; SIGN OF RESULT
1233 5BB5 AA          TAX         ; - NOT OK
1234 5BB6 AD1001  LDA RMAR+1   ; Z OK
1235 5BB9 ED0201  SBC X1CORD+1  ; + OK

```

VMSUP K-1008 VM GRAPHIC SUP
CURSOR-BORDER LIMIT TEST ROUTINES

1236 5BBC 48	PHA	
1237 5BBD 8A	TXA	
1238 5BBE 38	SEC	
1239 5BBF E90A	SBC	#2*CHWIDW-2
1240 5BC1 68	PLA	
1241 5BC2 E900	SBC	#0
1242 5BC4 60	RTS	
1243		

VMSUP K-1008 VM GRAPHIC SUP
CURSOR MANIPULATION ROUTINES

```

1244      ; .PAGE 'CURSOR MANIPULATION ROUTINES'
1245      ; CSRINS - INSERT A CURSOR AT THE CURRENT CURSOR POSITION
1246      ; WHICH IS DEFINED BY XICORD,YICORD
1247      ; CSRDEL - REMOVE THE CURSOR WHICH IS ASSUMED TO BE AT THE
1248      ; CURRENT CURSOR POSITION
1249      ; CURSOR IS DISPLAYED AS AN UNDERLINE CHHIM+1 SCAN LINES BELOW
1250      ; ACTUAL CHARACTER COORDINATES WHICH SPECIFY THE LOCATION OF THE
1251      ; UPPER LEFT CORNER OF THE CHARACTER
1252      ; INDEX REGISTERS PRESERVED
1253 5BC5 A9FB  CSRINS: LDA #X'F8    ; SET A FOR INSERTING THE CURSOR
1254 5BC7 D002  BNE CSR
1255 5BC9 A900  CSRDEL: LDA #0     ; SET A FOR DELETING THE CURSOR
1256
1257 5BCB 48    CSR: PHA      ; SAVE A
1258 5BC C AD0301 LDA YICORD   ; TEMPORARILY SUBTRACT CHHIM FROM YICORD
1259 5BCF 38    SEC
1260 5BD0 E909  SBC #CHHIM
1261 5BD2 800301 STA YICORD
1262 5BD5 B003  BCS CSR1
1263 5BD7 CE0401 DEC YICORD+1
1264 5BD8 202155 CSR1: JSR PIXADR ; COMPUTE ADDRESS OF CURSOR MARK
1265 5BD0 68    PLA      ; RESTORE SAVED A
1266 5BD E 208558 JSR MERGE5  ; MERGE CURSOR DATA WITH DISPLAY MEMORY
1267 5BE1 AD0301 LDA YICORD   ; RESTORE YICORD BY ADDING CHHIM BACK
1268 5BE4 18    CLC
1269 5BE5 6909  ADC #CHHIM
1270 5BE7 8D0301 STA YICORD
1271 5BEA 9003  BCC CSR2
1272 5BEC EE0401 INC YICORD+1
1273 5BEF 60    CSR2: RTS     ; RETURN
1274
1275      ; CSRR - MOVE CURSOR RIGHT ROUTINE
1276      ; DO NOTHING IF AGAINST RIGHT MARGIN
1277      ; USES X AND A
1278
1279 5BF0 20AE5B CSRR: JSR RTTST  ; TEST IF CURSOR CAN GO RIGHT
1280 5BF3 3014    BMI CSRR2   ; GO RETURN IF NOT ENOUGH ROOM
1281 5BF5 20C95B  JSR CSRDEL  ; DELETE THE PRESENT CURSOR
1282 5BF8 AD0101  LDA XICORD  ; ADD CHARACTER WINDOW WIDTH TO X
1283 5BF8 18    CLC
1284 5BFC 6906  ADC #CHWIDW
1285 5BFE 800101 STA XICORD
1286 5C01 9003  BCC CSR1
1287 5C03 EE0201  INC XICORD+1
1288 5C06 20C55B CSRR1: JSR CSRINS ; DISPLAY CURSOR AT THE NEW LOCATION
1289 5C09 60    CSRR2: RTS     ; RETURN
1290
1291      ; CSRL - MOVE CURSOR LEFT
1292      ; DO NOTHING IF AGAINST LEFT MARGIN
1293      ; USES A AND X
1294
1295 5COA 20975B CSRL: JSR LFTST  ; TEST IF CURSOR IS TOO FAR LEFT
1296 5COD 3014    BMI CSR2    ; JUMP IF IT IS TOO FAR LEFT
1297 5COF 20C95B  JSR CSRDEL  ; DELETE THE PRESENT CURSOR

```

VMSUP K-1008 VM GRAPHIC SUP
CURSOR MANIPULATION ROUTINES

```

1298 5C12 AD0101  LDA XICORD  ; SUBTRACT CHARACTER WINDOW WIDTH FROM
1299 5C15 38    SEC      ; X COORDINATE
1300 5C16 E906  SBC #CHWIDW
1301 5C18 8D0101 STA XICORD
1302 5C1B B003  BCS CSR1
1303 5C1D CE0201 DEC XICORD+1
1304 5C20 20C55B CSRL1: JSR CSRINS ; DISPLAY CURSOR AT THE NEW LOCATION
1305 5C23 60    CSRL2: RTS     ; RETURN
1306
1307      ; CSRU - CURSOR UP
1308      ; DO NOTHING IF AGAINST TOP MARGIN
1309      ; USES A AND X
1310
1311 5C24 20805B CSRU: JSR UPTST ; TEST IF CURSOR IS TOO FAR UP
1312 5C27 3014    BMI CSR2    ; JUMP IF IT IS TOO HIGH
1313 5C29 20C95B JSR CSRDEL  ; DELETE THE PRESENT CURSOR
1314 5C2C AD0301 LDA YICORD  ; ADD CHARACTER WINDOW HEIGHT TO Y
1315 5C2F 18    CLC
1316 5C30 690B  ADC #CHHIW
1317 5C32 8D0301 STA YICORD
1318 5C35 9003  BCC CSR1
1319 5C37 EE0401 INC YICORD+1
1320 5C3A 20C55B CSRU1: JSR CSRINS ; DISPLAY CURSOR AT THE NEW LOCATION
1321 5C3D 60    CSRU2: RTS     ; RETURN
1322
1323      ; CSRD - CURSOR DOWN
1324      ; DO NOTHING IF AGAINST BOTTOM MARGIN
1325      ; USES X AND A
1326
1327 5C3E 20695B CSRD: JSR DNTST ; TEST IF CURSOR IS TOO FAR DOWN
1328 5C41 3014    BMI CSR2    ; JUMP IF NOT ENOUGH SPACE
1329 5C43 20C95B JSR CSRDEL  ; DELETE THE PRESENT CURSOR
1330 5C46 AD0301 LDA YICORD  ; SUBTRACT CHARACTER WINDOW HEIGHT FROM
1331 5C49 38    SEC      ; Y COORDINATE
1332 5C4A E90B  SBC #CHHIW
1333 5C4C 8D0301 STA YICORD
1334 5C4F B003  BCS CSR1
1335 5C51 CE0401 DEC YICORD+1
1336 5C54 20C55B CSRD1: JSR CSRINS ; DISPLAY CURSOR AT THE NEW LOCATION
1337 5C57 60    CSRD2: RTS     ; RETURN
1338

```

VMSUP K-1008 VM GRAPHIC SUP
CONTROL CHARACTER DISPATCH TABLE

```

1339      ; .PAGE 'CONTROL CHARACTER DISPATCH TABLE'
1340      ; CONTROL CHARACTER DISPATCH TABLE FOR DTEXT
1341      ; FIRST BYTE IS ASCII CONTROL CHARACTER CODE
1342      ; SECOND AND THIRD BYTES ARE ADDRESS OF SERVICE ROUTINE
1343 5C58 0D  CCTAB: .BYTE X'0D      ; CR
1344 5C59 8659  .WORD CARRET-1    ; CARRIAGE RETURN
1345 5C5B 0A    .BYTE X'0A      ; LF
1346 5C5C 9859  .WORD LNFED-1    ; LINE FEED
1347 5C5E 08    .BYTE X'08      ; BS
1348 5C5F 4659  .WORD CRL-1     ; BACKSPACE
1349 5C61 0C    .BYTE X'0C      ; FF
1350 5C62 4E5A  .WORD FMFED-1   ; FORMFEED (CLEAR SCREEN)
1351 5C64 0F    .BYTE X'0F      ; SI
1352 5C65 5859  .WORD BASUP-1   ; BASELINE SHIFT UP
1353 5C67 0E    .BYTE X'0E      ; SO
1354 5C68 6F59  .WORD BASDN-1   ; BASELINE SHIFT DOWN
1355 5C6A 11    .BYTE X'11      ; DC1
1356 5C6B 4659  .WORD CRL-1     ; CURSOR LEFT
1357 5C6D 12    .BYTE X'12      ; DC2
1358 5C6E 4059  .WORD CRR-1     ; CURSOR RIGHT
1359 5C70 13    .BYTE X'13      ; DC3
1360 5C71 4C59  .WORD CRU-1     ; CURSOR UP
1361 5C73 14    .BYTE X'14      ; DC4
1362 5C74 5259  .WORD CRD-1     ; CURSOR DOWN
1363          CCTABE: .BYTE 00      ; END OF LIST
1364

```

VMSUP K-1008 VM GRAPHIC SUP
CHARACTER FONT TABLE

```

1365      ; .PAGE 'CHARACTER FONT TABLE'
1366      ; CHARACTER FONT TABLE 5 WIDE BY 7 HIGH PLUS 2 DESCENDING
1367      ; ENTRIES IN ORDER STARTING AT ASCII BLANK
1368      ; 96 ENTRIES
1369      ; EACH ENTRY CONTAINS 8 BYTES
1370      ; SIGN BIT OF FIRST BYTE IS A DESCENDER FLAG, CHARACTER DESCENDS
1371      ; 2 ROWS IF IT IS A ONE
1372      ; NEXT 7 BYTES ARE CHARACTER MATRIX, TOP ROW FIRST, LEFTMOST DOT
1373      ; IS LEFTMOST IN BYTE
1374 5C76 00000000  CHTB: .BYTE X'00,X'00,X'00,X'00      ; BLANK
1375 5C7A 00000000  .BYTE X'00,X'00,X'00,X'00
1376 5C7E 00202020  .BYTE X'00,X'20,X'20,X'20      ; !
1377 5C82 20200200  .BYTE X'20,X'20,X'00,X'20
1378 5C86 00505050  .BYTE X'00,X'50,X'50,X'50      ; "
1379 5C8A 00000000  .BYTE X'00,X'00,X'00,X'00
1380 5C8E 005050F8  .BYTE X'00,X'50,X'50,X'F8      ; #
1381 5C92 50F85050  .BYTE X'50,X'F8,X'50,X'50
1382 5C96 002078A0  .BYTE X'00,X'20,X'78,X'A0      ; $
1383 5C9A 7028F020  .BYTE X'70,X'28,X'F0,X'20
1384 5C9E 00C8C810  .BYTE X'00,X'C8,X'C8,X'10      ; %
1385 5CA2 20409898  .BYTE X'20,X'40,X'98,X'98
1386 5C46 0040A0A0  .BYTE X'00,X'40,X'A0,X'A0      ; ^
1387 5CAA 40A89068  .BYTE X'40,X'AB,X'90,X'68
1388 5CAE 00303030  .BYTE X'00,X'30,X'30,X'30      ; '
1389 5CB2 00000000  .BYTE X'00,X'00,X'00,X'00
1390 5CB6 00204040  .BYTE X'00,X'20,X'40,X'40      ; (
1391 5CBA 40404020  .BYTE X'40,X'40,X'40,X'20
1392 5CBE 00201010  .BYTE X'00,X'20,X'10,X'10      ; )
1393 5CC2 10101020  .BYTE X'10,X'10,X'10,X'20
1394 5CC6 0020A870  .BYTE X'00,X'20,X'AB,X'70      ; *
1395 5CCA 2070A820  .BYTE X'20,X'70,X'AB,X'20
1396 5CCE 00002020  .BYTE X'00,X'00,X'20,X'20      ; +
1397 5CD2 F8202000  .BYTE X'F8,X'20,X'20,X'00
1398 5CD6 80000000  .BYTE X'80,X'00,X'00,X'00      ; :
1399 5CDA 30301020  .BYTE X'30,X'30,X'10,X'20
1400 5CDE 00000000  .BYTE X'00,X'00,X'00,X'00      ; -
1401 5CE2 F8000000  .BYTE X'F8,X'00,X'00,X'00
1402 5CE6 00000000  .BYTE X'00,X'00,X'00,X'00      ; .
1403 5CEA 00003030  .BYTE X'00,X'00,X'30,X'30
1404 5CEE 00080810  .BYTE X'00,X'08,X'08,X'10      ; /
1405 5CF2 20408080  .BYTE X'20,X'40,X'80,X'80
1406 5CF6 00609090  .BYTE X'00,X'60,X'90,X'90      ; 0
1407 5CFA 90909060  .BYTE X'90,X'90,X'90,X'60
1408 5CFE 00206020  .BYTE X'00,X'20,X'60,X'20      ; 1
1409 5D02 20202070  .BYTE X'20,X'20,X'20,X'70
1410 5D04 00708810  .BYTE X'00,X'70,X'88,X'10      ; 2
1411 5D0A 204080F8  .BYTE X'20,X'40,X'80,X'F8
1412 5D0E 00708808  .BYTE X'00,X'70,X'88,X'08      ; 3
1413 5D12 30088870  .BYTE X'30,X'08,X'88,X'70
1414 5D16 00103050  .BYTE X'00,X'10,X'30,X'50      ; 4
1415 5D1A 90F81010  .BYTE X'90,X'F8,X'10,X'10
1416 5D1E 00F880F0  .BYTE X'00,X'F8,X'80,X'F0      ; 5
1417 5D22 080808F0  .BYTE X'08,X'08,X'08,X'F0
1418 5D26 00708080  .BYTE X'00,X'70,X'80,X'80      ; 6

```

VMSUP K-1008 VM GRAPHIC SUP
CHARACTER FONT TABLE

```

1419 5D2A F0888870 .BYTE X'F0,X'88,X'88,X'70
1420 5D2E 00F80810 .BYTE X'00,X'F8,X'08,X'10 ; 7
1421 5D32 20408080 .BYTE X'20,X'40,X'80,X'80
1422 5D36 00708888 .BYTE X'00,X'70,X'88,X'88 ; 8
1423 5D3A 70888870 .BYTE X'70,X'88,X'88,X'70
1424 5D3E 00708888 .BYTE X'00,X'70,X'88,X'88 ; 9
1425 5D42 78080870 .BYTE X'78,X'08,X'08,X'70
1426 5D46 00303000 .BYTE X'00,X'30,X'30,X'00 ; :
1427 5D4A 00003030 .BYTE X'00,X'00,X'30,X'30
1428 5D4E 80303000 .BYTE X'80,X'30,X'30,X'00 ; :
1429 5D52 30301020 .BYTE X'30,X'30,X'10,X'20
1430 5D56 00102040 .BYTE X'00,X'10,X'20,X'40 ; LESS THAN
1431 5D5A 80402010 .BYTE X'80,X'40,X'20,X'10
1432 5D5E 000000F8 .BYTE X'00,X'00,X'00,X'F8 ; =
1433 5D62 00F80000 .BYTE X'00,X'F8,X'00,X'00
1434 5D66 00402010 .BYTE X'00,X'40,X'20,X'10 ; GREATER THAN
1435 5D6A 80102040 .BYTE X'08,X'10,X'20,X'40
1436 5D6E 00708808 .BYTE X'00,X'70,X'88,X'08 ; ?
1437 5D72 10200020 .BYTE X'10,X'20,X'00,X'20
1438 5D76 00708808 .BYTE X'00,X'70,X'88,X'08 ; @
1439 5D7A 68A8A800 .BYTE X'68,X'A8,X'A8,X'D0
1440 5D7E 00205088 .BYTE X'00,X'20,X'50,X'88 ; A
1441 5D82 88F88888 .BYTE X'88,X'F8,X'88,X'88
1442 5D86 00F04848 .BYTE X'00,X'F0,X'48,X'48 ; B
1443 5D8A 704848F0 .BYTE X'70,X'48,X'48,X'F0
1444 5D8E 00708880 .BYTE X'00,X'70,X'88,X'80 ; C
1445 5D92 80808870 .BYTE X'80,X'80,X'88,X'70
1446 5D96 00F04848 .BYTE X'00,X'F0,X'48,X'48 ; D
1447 5D9A 484848F0 .BYTE X'48,X'48,X'48,X'F0
1448 5D9E 00F88880 .BYTE X'00,X'F8,X'80,X'80 ; E
1449 5D9A F0888870 .BYTE X'F0,X'80,X'80,X'F8
1450 5D9E 00F88880 .BYTE X'00,X'F8,X'80,X'80 ; F
1451 5DA0 F0888880 .BYTE X'F0,X'80,X'80,X'80
1452 5D9E 00708880 .BYTE X'00,X'70,X'88,X'80 ; G
1453 5D9B B8888870 .BYTE X'B8,X'88,X'88,X'70
1454 5D96 00888888 .BYTE X'00,X'88,X'88,X'88 ; H
1455 5DB4 F8888888 .BYTE X'F8,X'88,X'88,X'88
1456 5D9E 00702020 .BYTE X'00,X'70,X'20,X'20 ; I
1457 5DC2 20202070 .BYTE X'20,X'20,X'20,X'70
1458 5DC6 00381010 .BYTE X'00,X'38,X'10,X'10 ; J
1459 5DCA 10109060 .BYTE X'10,X'10,X'90,X'60
1460 5DCE 008890A0 .BYTE X'00,X'88,X'90,X'A0 ; K
1461 5DD2 COA09088 .BYTE X'CO,X'A0,X'90,X'88
1462 5DDE 00808080 .BYTE X'00,X'80,X'80,X'80 ; L
1463 5DDA 808080F8 .BYTE X'80,X'80,X'80,X'F8
1464 5DDE 0088D8A8 .BYTE X'00,X'88,X'D8,X'A8 ; M
1465 5DE2 A8888888 .BYTE X'A8,X'88,X'88,X'88
1466 5DE6 008888C8 .BYTE X'00,X'88,X'88,X'C8 ; N
1467 5DEA A8988888 .BYTE X'A8,X'98,X'88,X'88
1468 5DEE 00708888 .BYTE X'00,X'70,X'88,X'88 ; O
1469 5DF2 88888870 .BYTE X'88,X'88,X'88,X'70
1470 5DF6 00F08888 .BYTE X'00,X'F0,X'88,X'88 ; P
1471 5DFA F0808080 .BYTE X'F0,X'80,X'80,X'80
1472 5DFF 00708888 .BYTE X'00,X'70,X'88,X'88 ; Q
1473 5E02 88A89068 .BYTE X'88,X'A8,X'90,X'68

```

VMSUP K-1008 VM GRAPHIC SUP
CHARACTER FONT TABLE

```

1474 5E06 00F08888 .BYTE X'00,X'F0,X'88,X'88 - ; R
1475 5E0A FOA09088 .BYTE X'F0,X'A0,X'90,X'88
1476 5E0E 00788080 .BYTE X'00,X'78,X'80,X'80 ; S
1477 5E12 700808F0 .BYTE X'70,X'08,X'08,X'F0
1478 5E16 00F82020 .BYTE X'00,X'F8,X'20,X'20 ; T
1479 5E1A 20202020 .BYTE X'20,X'20,X'20,X'20
1480 5E1E 00888888 .BYTE X'00,X'88,X'88,X'88 ; U
1481 5E22 88888870 .BYTE X'88,X'88,X'88,X'70
1482 5E26 00888888 .BYTE X'00,X'88,X'88,X'88 ; V
1483 5E2A 50502020 .BYTE X'50,X'50,X'20,X'20
1484 5E2E 00888888 .BYTE X'00,X'88,X'88,X'88 ; W
1485 5E32 A8A80888 .BYTE X'A8,X'A8,X'D8,X'88
1486 5E36 00888850 .BYTE X'00,X'88,X'88,X'50 ; X
1487 5E3A 20508888 .BYTE X'20,X'50,X'88,X'88
1488 5E3E 00888850 .BYTE X'00,X'88,X'88,X'50 ; Y
1489 5E42 20202020 .BYTE X'20,X'20,X'20,X'20
1490 5E46 00F80810 .BYTE X'00,X'F8,X'08,X'10 ; Z
1491 5E4A 204080F8 .BYTE X'20,X'40,X'80,X'F8
1492 5E4E 00704040 .BYTE X'00,X'70,X'40,X'40 ; LEFT BRACKET
1493 5E52 40404070 .BYTE X'40,X'40,X'40,X'70
1494 5E56 00808040 .BYTE X'00,X'80,X'80,X'40 ; BACKSLASH
1495 5E5A 20100808 .BYTE X'20,X'10,X'08,X'08
1496 5E5E 00701010 .BYTE X'00,X'70,X'10,X'10 ; RIGHT BRACKET
1497 5E62 10101070 .BYTE X'10,X'10,X'10,X'70
1498 5E66 00205088 .BYTE X'00,X'20,X'50,X'88 ; CARROT
1499 5E6A 00000000 .BYTE X'00,X'00,X'00,X'00 ; UNDERLINE
1500 5E6E 00000000 .BYTE X'00,X'00,X'00,X'00 ; UNDERLINE
1501 5E72 000000F8 .BYTE X'00,X'00,X'00,X'F8
1502 . . .
1503 5E76 00C06030 .BYTE X'00,X'CO,X'60,X'30 ; GRAVE ACCENT
1504 5E7A 00000000 .BYTE X'00,X'00,X'00,X'00
1505 5E7E 00006010 .BYTE X'00,X'00,X'60,X'10 ; A (LC)
1506 5E82 70909068 .BYTE X'70,X'90,X'90,X'68
1507 5E86 008080F0 .BYTE X'00,X'80,X'80,X'F0 ; B (LC)
1508 5E8A 888888F0 .BYTE X'88,X'88,X'88,X'F0
1509 5E8C 00000078 .BYTE X'00,X'00,X'00,X'78 ; C (LC)
1510 5E92 80808078 .BYTE X'80,X'80,X'80,X'78
1511 5E96 00080878 .BYTE X'00,X'08,X'08,X'78 ; D (LC)
1512 5E9A 88888878 .BYTE X'88,X'88,X'88,X'78
1513 5E9C 00000070 .BYTE X'00,X'00,X'00,X'70 ; E (LC)
1514 5EA2 88F08078 .BYTE X'88,X'F0,X'80,X'78
1515 5EA6 00304040 .BYTE X'00,X'30,X'40,X'40 ; F (LC)
1516 5EAA E0404040 .BYTE X'E0,X'40,X'40,X'40
1517 5EAC 80708888 .BYTE X'80,X'70,X'88,X'88 ; G (LC)
1518 5EB2 98680870 .BYTE X'98,X'68,X'08,X'70
1519 5EB6 00808080 .BYTE X'00,X'80,X'80,X'80 ; H (LC)
1520 5EBA C8888888 .BYTE X'C8,X'88,X'88,X'88
1521 5EBE 00200060 .BYTE X'00,X'20,X'00,X'60 ; I (LC)
1522 5EC2 20202070 .BYTE X'20,X'20,X'20,X'70
1523 5EC6 80701010 .BYTE X'80,X'70,X'10,X'10 ; J (LC)
1524 5ECA 10109060 .BYTE X'10,X'10,X'90,X'60
1525 5ECE 00808090 .BYTE X'00,X'80,X'80,X'90 ; K (LC)
1526 5ED2 A0C0A090 .BYTE X'A0,X'CO,X'A0,X'90
1527 5ED6 00602020 .BYTE X'00,X'60,X'20,X'20 ; L (LC)
1528 5EDA 20202020 .BYTE X'20,X'20,X'20,X'20

```

VMSUP K-1008 VM GRAPHIC SUP
CHARACTER FONT TABLE

```

1529 5EDE 00000000 .BYTE X'00,X'00,X'00,X'D0 ; M (LC)
1530 5EE2 A8A8A8A8 .BYTE X'A8,X'A8,X'A8,X'A8
1531 5EE6 00000080 .BYTE X'00,X'00,X'00,X'B0 ; N (LC)
1532 5EEA C8888888 .BYTE X'C8,X'88,X'88,X'88
1533 5EEE 00000070 .BYTE X'00,X'00,X'00,X'70 ; O (LC)
1534 5EF2 88888870 .BYTE X'88,X'88,X'88,X'70
1535 5EF6 80F08888 .BYTE X'80,X'F0,X'88,X'88 ; P (LC)
1536 5EFA 88F08080 .BYTE X'88,X'F0,X'80,X'80
1537 5EFE 80788888 .BYTE X'80,X'78,X'88,X'88 ; Q (LC)
1538 5F02 88780808 .BYTE X'88,X'78,X'08,X'08
1539 5F06 00000080 .BYTE X'00,X'00,X'00,X'80 ; R (LC)
1540 5F0A C8808080 .BYTE X'C8,X'80,X'80,X'80
1541 5F0E 00000078 .BYTE X'00,X'00,X'00,X'78 ; S (LC)
1542 5F12 807008F0 .BYTE X'80,X'70,X'08,X'F0
1543 5F16 004040E0 .BYTE X'00,X'40,X'40,X'E0 ; T (LC)
1544 5F1A 40405020 .BYTE X'40,X'40,X'50,X'20
1545 5F1E 00000090 .BYTE X'00,X'00,X'00,X'90 ; U (LC)
1546 5F22 90909068 .BYTE X'90,X'90,X'90,X'68
1547 5F26 00000088 .BYTE X'00,X'00,X'00,X'88 ; V (LC)
1548 5F2A 88505020 .BYTE X'88,X'50,X'50,X'20
1549 5F2E 000000A8 .BYTE X'00,X'00,X'00,X'A8 ; W (LC)
1550 5F32 A8A8A850 .BYTE X'A8,X'A8,X'A8,X'50
1551 5F36 00000088 .BYTE X'00,X'00,X'00,X'88 ; X (LC)
1552 5F3A 50205088 .BYTE X'50,X'20,X'50,X'88
1553 5F3E 80888888 .BYTE X'80,X'88,X'88,X'88 ; Y (LC)
1554 5F42 50204080 .BYTE X'50,X'20,X'40,X'80
1555 5F46 000000F8 .BYTE X'00,X'00,X'00,X'F8 ; Z (LC)
1556 5F4A 102040F8 .BYTE X'10,X'20,X'40,X'F8
1557 5F4E 00102020 .BYTE X'00,X'10,X'20,X'20 ; LEFT BRACE
1558 5F52 60202010 .BYTE X'60,X'20,X'20,X'10
1559 5F56 00202020 .BYTE X'00,X'20,X'20,X'20 ; VERTICAL BAR
1560 5F5A 20202020 .BYTE X'20,X'20,X'20,X'20
1561 5F5E 00402020 .BYTE X'00,X'40,X'20,X'20 ; RIGHT BRACE
1562 5F62 30202040 .BYTE X'30,X'20,X'20,X'40
1563 5F66 0010A840 .BYTE X'00,X'10,X'A8,X'40 ; TILDA
1564 5F6A 00000000 .BYTE X'00,X'00,X'00,X'00 ; RUBOUT
1565 5F6E 00A850A8 .BYTE X'00,X'A8,X'50,X'A8
1566 5F72 50A850A8 .BYTE X'50,X'A8,X'50,X'A8
1567
1568 0000 .END
NO ERROR LINES

```