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

The graphics software package for the K-1008 Visable 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. SDTXT 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 your 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.

RUNNING THE DEMONSTRATION PROGRAMS

I. SWIRL

Swirl is a demonstration program that generates a variety of interesting spirl 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 elipse when evaluated digitally 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 ($7FFF_{16}$) gives 6 points per cycle, +.5 (4000_{16}) gives about 12, and so forth. <u>Negative</u> 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 <u>exactly</u> 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.

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 defalut 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 preceeding 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 occurance 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 to set to 7000 to prevent the possibility of overflow. The sequence of patterns will not repeat for days!

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 <u>live</u> 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 accross 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.

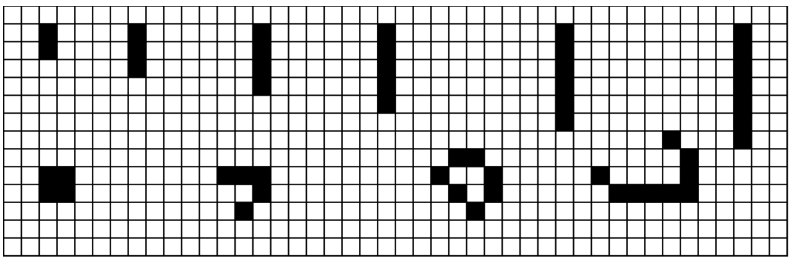
The Life demonstration program consists of four entry points. INIT (009A) when entered will merely clear the screen and return to the KIM monitor. This is generally necessary before entering a pattern by hand. KYPT (03C7) allows entry of an initial pattern of cells using a graphic cursor and the KIM 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 supprise!). 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).

Try the initial patterns shown below and note their fate.

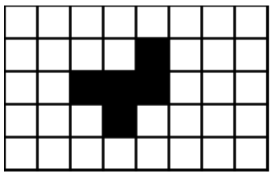


The patterns that evolve from those on the previous page are fundamental and well known to every Life fan. They are so common in the result of many initial patterns that they have been given discriptive names. See if you can match the following names with the corresponding final patterns: Block, Honeyfarm,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

III. USING SDTXT FOR TEXT DISPLAY ON THE VISIBLE MEMORY

SDTXT 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, SDTXT 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 SDTXT 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 SDTXT 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 SDTXT 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 CHWID 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.

SDTXT requires some RAM for parameter and temporary storage. There are three types of storage required. Base page temporary storage <u>must</u> be in page zero since the indirect addressing modes require this. Four bytes are required but they need not be preserved between calls to SDTXT 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 SDTXT unless the user desires to change these parameters. Note that if all RAM storage is kept in page 0 and SDTXT 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.

As given in the program listing, SDTXT 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 SDTXT 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 SDTXT 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 <u>underlined</u> 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 SDTXT

Using SDTXT 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 SDTXT. 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. SDTXT expects the decimal mode flag to be OFF.

It cannot be emphasized enough that VMORG <u>must</u> be set to the page number of the first VM location before SDTXT is used. For example, if the VM is jumpered for addresses 2000-3FFF, then VMORG should be 20_{16}. Failure to set VMORG will change SDTXT into MEMCLR!

It is also important that CSRX and CSRY have valid contents before any printable characters are sent to SDTXT. The best way to accomplish this is to give SDTXT an ASCII FF character (0C) 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. SDTXT 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 52_{10} and CSRY must be between 0 and 2149 inclusive. Violation of this range restriction is not checked and can cause random storing anywhere in memory.

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 SDTXT. FMOVE is an extremely fast memory move subroutine that can move any number of bytes from anywhere 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 SDTXT 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 theorectical 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 SDTXT 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 SDTXT 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 retireve 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 SDTXT. 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 SDTXT 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 SDTXT. 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 SDTXT (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 60_{16}- Once set it wiil 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 $0 \leq X \leq 319$ and $0 \leq Y \leq 199$ (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.

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 amywhere 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.

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 divisons 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 SDTXT. 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 SDTXT. SI and SO control characters effect a 3 pixel baseline shift up and down respectively for super and subscripts.

DTEXT is called just like SDTXT. 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 opper 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.

Micro Technology Unlimited,
Box 4596
29 Mead Street
Manchester, NH 03108
Dave Cox, Sales manager 603-432-7386
Hal Chamberlin, Engineer 603-669 0170

```
                              .PAGE  'DOCUMENTATION, EQUATES, STORAGE'
                     ;        SWIRL DRAWING DEMONSTRATION FOR THE MICRO TECHNOLOGY UNLIMITED
                     ;        VISIBLE MEMORY 320 BY 200 PIXEL DISPLAY

                     ;        ENTER AT SWIRL WITH LINES, FREQ, AND DAMP SET TO APPROPRIATE
                     ;        VALUES TO GENERATE AN SWIRLING DISPLAY. INTERRUPT WITH RESET
                     ;        KEY WHEN PATTERN IS COMPLETED TO DESIRED EXTENT.

                     ;        ENTER AT RSWIRL FOR AN ENDLESS SERIES OF PATTERNS USING
                     ;        RANDOMLY SELECTED PARAMETERS.

                     ;        GENERAL EQUATES

   1C22              KIMMON   =      X'1C22       ; RESET ENTRY INTO KIM MONITOR
   0140              NX       =      320          ; NUMBER OF BITS IN A ROW
   00C8              NY       =      200          ; NUMBER OF ROWS (CHANGE FOR HALF SCREEN
                                                  ; OPERATION)
   FA00              NPIX     =      NX*NY        ; NUMBER OF PIXELS

   0000                       .=     0            ; START PROGRAM AT ZERO

                     ;        STORAGE FOR SWIRL GENERATOR PROGRAM

   0000 01           LINES:   .BYTE  1            ; CONNECTING LINES IF NON-ZERO
   0001 127E         FREQ:    .WORD  X'7E12       ; FREQUENCY
   0003 007E         DAMP:    .WORD  X'7E00       ; 1-(DAMPING FACTOR)
   0005 0078         COSINT:  .WORD  X'7800       ; INITIAL COSINE VALUE
                                                  ; GOOD VALUE FOR GENERAL USE BUT SHOULD BE
                                                  ; REDUCED TO X'70 TO PREVENT OVERFLOW WITH
                                                  ; RANDOMLY SELECTED PARAMETERS
   0007              COS:     .=.+   2            ; COSINE VALUE
   0009              SIN:     .=.+   2            ; SINE VALUE

                     ;        GENERAL STORAGE

   000B 20           VMORG:   .BYTE  X'20         ; PAGE NUMBER OF FIRST VISIBLE MEMORY
                                                  ; LOCATION
   000C 3412         RANDNO:  .WORD  X'1234       ; INITIAL RANDON NUMBER, MUST NOT BE ZERO
   000E              ADP1:    .=.+   2            ; ADDRESS POINTER 1
   0010              ADP2:    .=.+   2            ; ADDRESS POINTER 2
   0012              BTPT:    .=.+   1            ; BIT NUMBER
   0013              X1CORD:  .=.+   2            ; COORDINATE PAIR 1
   0015              Y1CORD:  .=.+   2
   0017              X2CORD:  .=.+   2            ; COORDINATE PAIR 2
   0019              Y2CORD:  .=.+   2

                     ;        STORAGE FOR ARBITRARY LINE DRAW ROUTINE

   001B              DELTAX:  .=.+   2            ; DELTA X
   001D              DELTAY:  .=.+   2            ; DELTA Y
   001F              ACC:     .=.+   2            ; ACCUMULATOR
   0021              XDIR:    .=.+   1            ; X MOVEMENT DIRECTION, ZERO=+
   0022              YDIR:    .=.+   1            ; Y MOVEMENT DIRECTION, ZERO=+
   0023              XCHFLG:  .=.+   1            ; EXCHANGE X AND Y FLAG, EXCHANGE IF NOT 0
   0024              COLOR:   .=.+   1            ; COLOR OF LINE DRAWN -1=WHITE
```

```
0025           TEMP:    .=.+   2            ; TEMPORARY STORAGE

               ;        STORAGE FOR THE ARITHMETIC SUBROUTINES

0027           PROD:    .=.+   4            ; PRODUCT FOR ARITHMETIC ROUTINES
002B           MPCD:    .=.+   2            ; MUPTIPLICAND FOR ARITHMETIC
002D           MPLR     =      PROD         ; MULTIPLIER FOR ARITHMETIC ROUTINES
002D           MPSAVE:  .=.+   2            ; TEMPORARY STORAGE FOR MULTIPLY

```

```
                        .PAGE  'MAIN SWIRL GENERATION ROUTINE'
                ;       SWIRL ROUTINE FOR STRAIGHT LINES CONNECTING THE POINTS

002F 208D00    SWIRL:   JSR    SWINIT       ; INITIALIZE COS AND SIN
0032 20A500    SWIRL1:  JSR    SCALE        ; SCALE SIN AND COS FOR DISPLAY
0035 A500               LDA    LINES        ; TEST IF LINES BETWEEN POINTS DESIRED
0037 D003               BNE    SWIRL2       ; SKIP IF SO
0039 205D01             JSR    C2TOC1       ; IF NOT, SET LINE LENGTH TO ZERO
003C 202202    SWIRL2:  JSR    DRAW         ; DRAW THE LINE OR POINT
003F 200001             JSR    POINT        ; COMPUTE THE NEXT POINT
0042 4C3200             JMP    SWIRL1

                ;       SWIRL ROUTINE WITH RANDOM PARAMETERS

0045 208D00    RSWIRL:  JSR    SWINIT       ; INITIALIZE COS AND SIN
0048 209503    RSWR1:   JSR    RAND         ; INITIALIZE FREQ RANDOMLY WITH UNIFORM
004B 8501               STA    FREQ         ; DISTRIBUTION
004D 209503             JSR    RAND
0050 8502               STA    FREQ+1
0052 20B103             JSR    RNDEXP       ; INITIALIZE DAMP RANDOMLY WITH A NEGATIVE
0055 4A                 LSRA                ; EXPONENTIAL DISTRIBUTION
0056 497F               EOR    #X'7F        ; IN THE UPPER BYTE AND UNIFORM
0058 8504               STA    DAMP+1       ; DISTRIBUTION IN THE LOWER BYTE
005A 209503             JSR    RAND
005D 8503               STA    DAMP
005F 209503             JSR    RAND         ; RANDOMLY DETERMINE PRESENCE OF
0062 2901               AND    #1           ; CONNECTING LINES
0064 8500               STA    LINES
0066 20CB03             JSR    RANGCK       ; VERIFY ACCEPTABLE RANGES OF PARAMETERS
0069 B0DD               BCS    RSWR1        ; TRY AGAIN IF NOT ACCEPTABLE
006B 20A500    RSWR2:   JSR    SCALE        ; SCALE THE CURRENT POINT FOR PLOTTING
006E A500               LDA    LINES        ; TEST IF CONNECTING LINES SPECIFIED
0070 D003               BNE    RSWR3        ; SKIP AHEAD IF SO
0072 205D01             JSR    C2TOC1       ; IF NOT, SET ZERO LINE LENGTH
0075 202202    RSWR3:   JSR    DRAW         ; ORAW A LINE FROM THE LAST POINT PLOTTED
0078 200001             JSR    POINT        ; COMPUTE THE NEXT POINT
007B A50A      RSWR4:   LDA    SIN+1        ; TEST IF PATTERN HAS DECAYED TO NEARLY
007D F004               BEQ    RSWR5        ; ZERO
007F C9FF               CMP    #X'FF
0081 D0E8               BNE    RSWR2
0083 A508      RSWR5:   LDA    COS+1
0085 F0BE               BEQ    RSWIRL       ; GO START A NEW PATTERN IF SO
0087 C9FF               CMP    #X'FF
0089 F0BA               BEQ    RSWIRL
008B D0DE               BNE    RSWR2        ; GO COMPUTE NEXT POINT IF NOT

                ;       SWINIT - INITIALIZE COS FROM COSINT, ZERO SIN, CLEAR SCREEN

008D A505      SWINIT:  LDA    COSINT       ; INITIALIZE COS
008F 8507               STA    COS
0091 A506               LDA    COSINT+1
0093 8508               STA    COS+1
0095 A900               LDA    #0           ; ZERO SIN
0097 8509               STA    SIN
0099 850A               STA    SIN+1
```

```
009B 200002             JSR    CLEAR          ; CLEAR THE VM SCREEN
009E 20A500             JSR    SCALE          ; SCALE THE INITIAL POINT AND PUT INTO
00A1 205D01             JSR    C2TOC1         ; IN BOTH SETS OF COORDINATES
00A4 60                 RTS                   ; RETURN

               ;        SCALE - TAKE VALUE OF SIN, SCALE ACCORDING TO NX, AND PUT INTO
               ;        X2CORD. THEN TAKE VALUE OF COS, SCALE ACCORDING TO NY, AND
               ;        PUT INTO Y2CORD.
               ;        SIN AND COS ARE ASSUMED TO BE DOUBLE LENGTH BINARY FRACTIONS
               ;        BETWEEN -1 AND +1.

00A5 A507      SCALE:   LDA    COS            ; X2CORD=NX/2*SIN4NX/2
00A7 852B               STA    MPCD           ; TRANSFER SIN TO MULTIPLICAND
00A9 A508               LDA    COS+1          ; (BINARY FRACTION)
00AB 852C               STA    MPCD+1
00AD A9A0               LDA    #NX/2&X'FF     ; TRANSFER NX/2 TO MULTIPLIER
00AF 8527               STA    MPLR           ; (INTEGER)
00B1 A900               LDA    #NX/2/256
00B3 8528               STA    MPLR+1
00B5 202B03             JSR    SGNMPY         ; PERFORM A SIGNED MULTIPLICATION
00B8 208B03             JSR    SLQL
00BB A529               LDA    PROD+2         ; SIGNED INTEGER RESULT IN PROD+2 (LOW)
00BD 18                 CLC                   ; AND PROD+3 (HIGH)
00BE 69A0               ADC    #NX/2&X'FF     ; ADD NX/2 TO PRODUCT AND PUT INTO X2CORD
00C0 8517               STA    X2CORD
00C2 A52A               LDA    PROD+3
00C4 6900               ADC    #NX/2/256
00C6 8518               STA    X2CORD+1

00C8 A509               LDA    SIN            ; Y2CORD=NY/2*COS+NX/2
00CA 852B               STA    MPCD           ; TRANSFER COS TO MULTIPLICAND
00CC A50A               LDA    SIN+1          ; (BINARY FRACTION)
00CE 852C               STA    MPCD+1
00D0 A964               LDA    #NY/2&X'FF     ; TRANSFER NY/2 TO MULTIPLIER
00D2 8527               STA    MPLR           ; (INTEGER)
00D4 A900               LDA    #NY/2/256
00D6 8528               STA    MPLR+1
00D8 202B03             JSR    SGNMPY         ; PERFORM A SIGNED MULTIPLICATION
00DB 208B03             JSR    SLQL
00DE A529               LDA    PROD+2         ; SIGNED INTEGER RESULT IN PROD+2 (LOW)
00E0 18                 CLC                   ; AND PROD+3 (HIGH)
00E1 6964               ADC    #NY/2&X'FF     ; ADD NY/2 TO PRODUCT AND PUT INTO Y2CORD
00E3 8519               STA    Y2CORD
00E5 A52A               LDA    PROD+3
00E7 6900               ADC    #NY/2/256
00E9 851A               STA    Y2CORD+1
00EB 60                 RTS                   ; RETURN
```

```
                            .PAGE  'POINT - COMPUTE NEXT POINT'
                   ;        POINT - COMPUTE NEXT VALUE OF COS,SIN FROM CURRENT VALUE OF
                   ;        COS,SIN ACCORDING TO FREQ AND DAMP. DIFFERENCE EQUATION FOR
                   ;        AN ELIPSE IS USED

00EC                        .=     X'100

0100 A509          POINT:   LDA    SIN          ; FIRST COMPUTE DAMP*SIN AND PUT INTO SIN
0102 852B                   STA    MPCD
0104 A50A                   LDA    SIN+1
0106 852C                   STA    MPCD+1
0108 A503                   LDA    DAMP
010A 8527                   STA    MPLR
010C A504                   LDA    DAMP+1
010E 8528                   STA    MPLR+1
0110 202B03                 JSR    SGNMPY
0113 208B03                 JSR    SLQL         ; SHIFT PRODUCT LEFT ONE FOR FRACTIONAL
0116 A529                   LDA    PROD+2       ; RESULT
0118 8509                   STA    SIN          ; AND PUT BACK INTO SIN
011A A52A                   LDA    PROD+3
011C 850A                   STA    SIN+1

011E A507                   LDA    COS          ; NEXT COMPUTE COS*FREQ
0120 8527                   STA    MPLR
0122 A508                   LDA    COS+1
0124 8528                   STA    MPLR+1
0126 A501                   LDA    FREQ
0128 852B                   STA    MPCD
012A A502                   LDA    FREQ+1
012C 852C                   STA    MPCD+1
012E 202B03                 JSR    SGNMPY
0131 208B03                 JSR    SLQL
0134 A509                   LDA    SIN          ; ADD RESULT TO SIN AND PUT SUM BACK INTO
0136 18                     CLC                 ; SIN
0137 6529                   ADC    PROD+2
0139 8509                   STA    SIN
013B A50A                   LDA    SIN+1
013D 652A                   ADC    PROD+3
013F 850A                   STA    SIN+1

0141 A509                   LDA    SIN          ; NEXT COMPUTE FREQ*SIN
0143 8527                   STA    MPLR
0145 A50A                   LDA    SIN+1
0147 8528                   STA    MPLR+1       ; FREQ ALREADY IN MPCD
0149 202B03                 JSR    SGNMPY
014C 208B03                 JSR    SLQL

014F A507                   LDA    COS          ; SUBSTRACT RESULT FROM COS AND PUT RESULT
0151 38                     SEC                 ; IN COS
0152 E529                   SBC    PROD+2
0154 8507                   STA    COS
0156 A508                   LDA    COS+1
0158 E52A                   SBC    PROD+3
015A 8508                   STA    COS+1
015C 60                     RTS                 ; RETURN
```

```

                    ;        SUBROUTINE TO MOVE THE CONTENTS OF COORDINATE PAIR 2 TO
                    ;        COORDINATE PAIR 1.

    015D A517       C2TOC1:  LDA    X2CORD       ; DO THE MOVING
    015F 8513                STA    X1CORD
    0161 A518                LDA    X2CORD+1
    0163 8514                STA    X1CORD+1
    0165 A519                LDA    Y2CORD
    0167 8515                STA    Y1CORD
    0169 A51A                LDA    Y2CORD+1
    016B 8516                STA    Y1CORD+1
    016D 60                  RTS                 ; RETURN

```

```
                        .PAGE  'ABBREVIATED GRAPHICS ROUTINES'
               ;        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

016E A513      PIXADR:  LDA    X1CORD       ; COMPUTE BIT ADDRESS FIRST
0170 850E               STA    ADP1         ; ALSO TRANSFER X1CORD TO ADP1
0172 2907               AND    #X'07        ; WHICH IS SIMPLY THE LOW 3 BITS OF X
0174 8512               STA    BTPT
0176 A514               LDA    X1CORD+1     ; FINISH TRANSFERRING X1CORD TO ADP1
0178 850F               STA    ADP1+1
017A 460F               LSR    ADP1+1       ; DOUBLE SHIFT ADP1 RIGHT 3 TO GET
017C 660E               ROR    ADP1         ; INT(XCORD/8)
017E 460F               LSR    ADP1+1
0180 660E               ROR    ADP1
0182 460F               LSR    ADP1+1
0184 660E               ROR    ADP1
0186 A9C7               LDA    #199         ; TRANSFER (199-Y1CORD) TO ADP2
0188 38                 SEC                 ; AND TEMPORARY STORAGE
0189 E515               SBC    Y1CORD
018B 8510               STA    ADP2
018D 8525               STA    TEMP
018F A900               LDA    #0
0191 E516               SBC    Y1CORD+1
0193 8511               STA    ADP2+1
0195 8526               STA    TEMP+1
0197 0610               ASL    ADP2         ; COMPUTE 40*(199-Y1CORD)
0199 2611               ROL    ADP2+1       ;  2*(199-Y1CORD)
019B 0610               ASL    ADP2
019D 2611               ROL    ADP2+1       ;  4*(199+Y1CORD)
019F A510               LDA    ADP2         ;  ADD IN TEMPORARY SAVE OF (199-Y1CORD)
01A1 18                 CLC                 ;  TO MAKE 5*(199-Y1CORD)
01A2 6525               ADC    TEMP
01A4 8510               STA    ADP2
01A6 A511               LDA    ADP2+1
01A8 6526               ADC    TEMP+1
01AA 8511               STA    ADP2+1       ;  5*(199-Y1CORD)
01AC 0610               ASL    ADP2         ;  10*(199-Y1CORD)
01AE 2611               ROL    ADP2+1
01B0 0610               ASL    ADP2         ;  20*(199-Y1CORD)
01B2 2611               ROL    ADP2+1
01B4 0610               ASL    ADP2         ;  40*(199-Y1CORD)
01B6 2611               ROL    ADP2+1
01B8 A510               LDA    ADP2         ; ADD IN INT(X1CORD/8) COMPUTED EARLIER
01BA 18                 CLC
01BB 650E               ADC    ADP1
01BD 850E               STA    ADP1
01BF A511               LDA    ADP2+1
```

```
01C1 650F                ADC    ADP1+1
01C3 650B                ADC    VMORG        ; ADD IN VMORG*256
01C5 850F                STA    ADP1+1       ; FINAL RESULT
01C7 60                  RTS                 ; RETURN

               ;         STPIX - SETS THE PIXEL AT X1CORD,Y1CORD TO A ONE (WHITE DOT)
               ;         DOES NOT ALTER X1CORD OR Y1CORD
               ;         PRESERVES X AND Y
               ;         ASSUMES IN RANGE CORRDINATES

01C8 206E01    STPIX:    JSR    PIXADR       ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
                                             ; INTO ADP1
01CB 98                  TYA                 ; SAVE Y
01CC 48                  PHA
01CD A412                LDY    BTPT         ; GET BIT NUMBER IN Y
01CF B91A02              LDA    MSKTB1,Y     ; GET A BYTE WITH THAT BIT =1, OTHERS =0
01D2 A000                LDY    #0           ; ZERO Y
01D4 110E                ORA    (ADP1),Y     ; COMBINE THE BIT WITH THE ADDRESSED VM
01D6 910E                STA    (ADP1),Y     ; BYTE
01D8 68                  PLA                 ; RESTORE Y
01D9 A8                  TAY
01DA 60                  RTS                 ; AND RETURN

01DB                     .=     X'200

               ;         CLEAR DISPLAY MEMORY ROUTINE

0200 A000      CLEAR:    LDY    #0           ; INITIALIZE ADDRESS POINTER
0202 840E                STY    ADP1         ; AND ZERO INDEX Y
0204 A50B                LDA    VMORG
0206 850F                STA    ADP1+1
0208 18                  CLC
0209 6920                ADC    #X'20
020B AA                  TAX
020C 98        CLEAR1:   TYA                 ; CLEAR A BYTE
020D 910E                STA    (ADP1),Y
020F E60E                INC    ADP1         ; INCREMENT ADDRESS POINTER
0211 D0F9                BNE    CLEAR1
0213 E60F                INC    ADP1+1
0215 E40F                CPX    ADP1+1       ; TEST IF DONE
0217 D0F3                BNE    CLEAR1
0219 60                  RTS                 ; RETURN

               ;         MASK TABLES FOR INDIVIDUAL PIXEL SUBROUTINES
               ;         MSKTB1 IS A TABLE OF 1 BITS CORRESPONDING TO BIT NUMBERS

021A 80402010  MSKTB1:   .BYTE  X'80,X'40,X' 20,X'10
021E 08040201            .BYTE  X'08,X'04,X' 02,X'01

```

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

0222 8A        DRAW:    TXA                 ; SAVE X AND Y
0223 48                 PHA
0224 98                 TYA
0225 48                 PHA

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

0226 A900               LDA    #0           ; FIRST ZERO XDIR
0228 8521               STA    XDIR
022A A517               LDA    X2CORD       ; NEXT COMPUTE TWOS COMPLEMENT DIFFERENCE
022C 38                 SEC
022D E513               SBC    X1CORD
022F 851B               STA    DELTAX
0231 A518               LDA    X2CORD+1
0233 E514               SBC    X1CORD+1
0235 851C               STA    DELTAX+1
0237 100F               BPL    DRAW2        ; SKIP AHEAD IF DIFFERENCE IS POSITIVE
0239 C621               DEC    XDIR         ; SET XDIR TO -1
023B 38                 SEC                 ; NEGATE DELTAX
023C A900               LDA    #0
023E E51B               SBC    DELTAX
0240 851B               STA    DELTAX
0242 A900               LDA    #0
0244 E51C               SBC    DELTAX+1
0246 851C               STA    DELTAX+1

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

0248 A900      DRAW2:   LDA    #0           ; FIRST ZERO YDIR
024A 8522               STA    YDIR
024C A519               LDA    Y2CORD       ; NEXT COMPUTE TWOS COMPLEMENT DIFFERENCE
024E 38                 SEC
024F E515               SBC    Y1CORD
0251 851D               STA    DELTAY
0253 A51A               LDA    Y2CORD+1
0255 E516               SBC    Y1CORD+1
0257 851E               STA    DELTAY+1
0259 100F               BPL    DRAW3        ; SKIP AHEAD IF DIFFERENCE IS POSITIVE
025B C622               DEC    YDIR         ; SET YDIR TO -1
025D 38                 SEC                 ; NEGATE DELTAX
025E A900               LDA    #0
0260 E51D               SBC    DELTAY
0262 851D               STA    DELTAY
0264 A900               LDA    #0
0266 E51E               SBC    DELTAY+1
0268 851E               STA    DELTAY+1
```

```

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

026A A900       DRAW3:   LDA    #0            ; FIRST ZERO XCHFLG
026C 8523                STA    XCHFLG
026E A51D                LDA    DELTAY        ; COMPARE DELTAY WITH DELTAX
0270 38                  SEC
0271 E51B                SBC    DELTAX
0273 A51E                LDA    DELTAY+1
0275 E51C                SBC    DELTAX+1
0277 9012                BCC    DRAW4         ; SKIP EXCHANGE IF DELTAX IS GREATER THAN
                                              ; DELTAY
0279 A61D                LDX    DELTAY        ; EXCHANGE DELTAX AND DELTAY
027B A51B                LDA    DELTAX
027D 851D                STA    DELTAY
027F 861B                STX    DELTAX
0281 A61E                LDX    DELTAY+1
0283 A51C                LDA    DELTAX+1
0285 851E                STA    DELTAY+1
0287 861C                STX    DELTAX+1
0289 C623                DEC    XCHFLG        ; SET XCHFLG TO -1
028B A51B       DRAW4:   LDA    DELTAX        ; INITIALIZE ACC TO DELTAX
028D 851F                STA    ACC
028F A51C                LDA    DELTAX+1
0291 8520                STA    ACC+1
0293 20C801              JSR    STPIX         ; PUT A DOT AT THE INITIAL ENDPOINT;
                                              ; X1CORD, Y1CORD

                ;        HEAD OF MAIN DRAWING LOOP
                ;        TEST IF DONE

0296 A523       DRAW45:  LDA    XCHFLG        ; TEST IF X AND Y EXCHANGED
0298 D00E                BNE    DRAW5         ; JUMP AHEAD IF SO
029A A513                LDA    X1CORD        ; TEST FOR X1CORD=X2CORD
029C C517                CMP    X2CORD
029E D019                BNE    DRAW7         ; GO FOR ANOTHER ITERATION IF NOT
02A0 A514                LDA    X1CORD+1
02A2 C518                CMP    X2CORD+1
02A4 D013                BNE    DRAW7         ; GO FOR ANOTHER ITERATION IF NOT
02A6 F00C                BEQ    DRAW6         ; GO RETURN IF SO
02A8 A515       DRAW5:   LDA    Y1CORD        ; TEST FOR Y1CORD=Y2CORD
02AA C519                CMP    Y2CORD
02AC D00B                BNE    DRAW7         ; GO FOR ANOTHER ITERATION IF NOT
02AE A516                LDA    Y1CORD+1
02B0 C51A                CMP    Y2CORD+1
02B2 D005                BNE    DRAW7         ; GO FOR ANOTHER ITERATION IF NOT
02B4 68         DRAW6:   PLA                  ; RESTORE INDEX REGISTERS
02B5 A8                  TAY
02B6 68                  PLA
02B7 AA                  TAX
02B8 60                  RTS                  ; AND RETURN

```

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

02B9 A523      DRAW7:   LDA    XCHFLG       ; TEST IF X AND Y EXCHANGED
02BB D006               BNE    DRAW8        ; JUMP IF SO
02BD 200303             JSR    BMPX         ; BUMP X IF NOT
02C0 4CC602             JMP    DRAW9
02C3 201703    DRAW8:   JSR    BMPY         ; BUMP Y IF SO
02C6 20E702    DRAW9:   JSR    SBDY         ; SUBSTRACT DY FROM ACC TWICE
02C9 20E702             JSR    SBDY
02CC 1013               BPL    DRAW12       ; SKIP AHEAD IF ACC IS NOT NEGATIVE
02CE A523               LDA    XCHFLG       ; TEST IF X AND Y EXCHANGED
02D0 D006               BNE    DRAW10       ; JUMP IF SO
02D2 201703             JSR    BMPY         ; BUMP Y IF NOT
02D5 4CDB02             JMP    DRAW11
02D8 200303    DRAW10:  JSR    BMPX         ; BUMP X IF SO
02DB 20F502    DRAW11:  JSR    ADDX         ; ADD DX TO ACC TWICE
02DE 20F502             JSR    ADDX

02E1 20C801    DRAW12:  JSR    STPIX        ; OUTPUT THE NEW POINT
02E4 4C9602             JMP    DRAW45       ; GO TEST IF DONE

               ;        SUBROUTINES FOR DRAW

02E7 A51F      SBDY:    LDA    ACC          ; SUBSTRACT DELTAY FROM ACC AND PUT RESULT
02E9 38                 SEC                 ; IN ACC
02EA E51D               SBC    DELTAY
02EC 851F               STA    ACC
02EE A520               LDA    ACC+1
02F0 E51E               SBC    DELTAY+1
02F2 8520               STA    ACC+1
02F4 60                 RTS


02F5 A51F      ADDX:    LDA    ACC          ; ADD DELTAX TO ACC AND PUT RESULT IN ACC
02F7 18                 CLC
02F8 651B               ADC    DELTAX
02FA 851F               STA    ACC
02FC A520               LDA    ACC+1
02FE 651C               ADC    DELTAX+1
0300 8520               STA    ACC+1
0302 60                 RTS


0303 A521      BMPX:    LDA    XDIR         ; BUMP X1CORD BY +1 OR -1 ACCORDING TO
0305 D007               BNE    BMPX2        ; XDIR
0307 E613               INC    X1CORD       ; DOUBLE INCREMENT X1CORD IF XDIR=0
0309 D002               BNE    BMPX1
030B E614               INC    X1CORD+1
030D 60        BMPX1:   RTS
030E A513      BMPX2:   LDA    X1CORD       ; DOUBLE DECREMENT X1CORD IF XDIR<>0
0310 D002               BNE    BMPX3
0312 C614               DEC    X1CORD+1
0314 C613      BMPX3:   DEC    X1CORD
```

```
0316 60                  RTS


0317 A522      BMPY:     LDA    YDIR          ; BUMP Y1CORD BY +1 OR -1 ACCORDING TO
0319 D007                BNE    BMPY2         ; YDIR
031B E615                INC    Y1CORD        ; DOUBLE INCREMENT Y1CORD IF YDIR=0
031D D002                BNE    BMPY1
031F E616                INC    Y1CORD+1
0321 60        BMPY1:    RTS
0322 A515      BMPY2:    LDA    Y1CORD        ; DOUBLE DECREMENT Y1CORD IF YDIR<>0
0324 D002                BNE    BMPY3
0326 C616                DEC    Y1CORD+1
0328 C615      BMPY3:    DEC    Y1CORD
032A 60                  RTS

```

```
                        .PAGE  'MULTIPLY, SHIFT, AND RANDOM NUMBER ROUTINES'
               ;        SIGNED MULTIPLY SUBROUTINE
               ;        ENTER WITH SIGNED MULTIPLIER IN PROD AND PROD+1
               ;        ENTER WITH SIGNED MULTIPLICAND IN MPCD AND MPCD+1
               ;        RETURN WITH 16 BIT SIGNED PRODUCT IN PROD (LOW) THROUGH
               ;        PROD+3 (HIGH)
               ;        A DESTROYED, X AND Y PRESERVED

032B A527      SGNMPY:  LDA    PROD          ; GET MULTIPLIER
032D 852D               STA    MPSAVE        ; AND SAVE IT
032F A528               LDA    PROD+1
0331 852E               STA    MPSAVE+1
0333 205903             JSR    UNSMPY        ; DO AN UNSIGNED MULTIPLY
0336 A52C               LDA    MPCD+1        ; TEST SIGN OF MULTIPLICAND
0338 100D               BPL    SGNMP1        ; JUMP IF POSITIVE
033A A529               LDA    PROD+2        ; SUBTRACT MULTIPLIER FROM HIGH PRODUCT IF
033C 38                 SEC                  ; NEGATIVE
033D E52D               SBC    MPSAVE
033F 8529               STA    PROD+2
0341 A52A               LDA    PROD+3
0343 E52E               SBC    MPSAVE+1
0345 852A               STA    PROD+3
0347 A52E      SGNMP1:  LDA    MPSAVE+1      ; TEST SIGN OF MULTIPLIER
0349 100D               BPL    SGNMP2        ; GO RETURN IF POSITIVE
034B A529               LDA    PROD+2        ; SUBTRACT MULTIPLICAND FROM HIGH PRODUCT
034D 38                 SEC                  ; IF NEGATIVE
034E E52B               SBC    MPCD
0350 8529               STA    PROD+2
0352 A52A               LDA    PROD+3
0354 E52C               SBC    MPCD+1
0356 852A               STA    PROD+3
0358 60        SGNMP2:  RTS                  ; RETURN

               ;        16 X 16 UNSIGNED MULTIPLY SUBROUTINE
               ;        ENTER WITH UNSIGNED MULTIPLIER IN PROD AND PROD+1
               ;        ENTER WITH UNSIGNED MULTIPLICAND IN MPCD AND MPCD+1
               ;        RETURN WITH 16 BIT UNSIGNED PRODUCT IN PROD (LOW) THROUGH
               ;        PROD+3 (HIGH)
               ;        A DESTROYED, X AND Y PRESERVED

0359 8A        UNSMPY:  TXA                  ; SAVE X INDEX
035A 48                 PHA
035B A900               LDA    #0            ; CLEAR UPPER PRODUCT
035D 852A               STA    PROD+3
035F 8529               STA    PROD+2
0361 A211               LDX    #17           ; SET 17 MULTIPLY CYCLE COUNT
0363 18                 CLC                  ; INITIALLY CLEAR CARRY
0364 208203    UNSM1:   JSR    SRQL          ; SHIFT MULTIPLIER AND PRODUCT RIGHT 1
                                             ; PUTTING A MULTIPLIER BIT IN CARRY
0367 CA                 DEX                  ; DECREMENT AND CHECK CYCLE COUNT
0368 F012               BEQ    UNSM2         ; JUMP OUT IF DONE
036A 90F8               BCC    UNSM1         ; SKIP MULTIPLICAND ADD IF MULTIPLIER BIT
                                             ; IS ZERO
036C A529               LDA    PROD+2        ; ADD MULTIPLICAND TO UPPER PRODUCT
036E 18                 CLC
```

```
036F 652B               ADC    MPCD
0371 8529               STA    PROD+2
0373 A52A               LDA    PROD+3
0375 652C               ADC    MPCD+1
0377 852A               STA    PROD+3
0379 4C6403             JMP    UNSM1        ; GO FOR NEXT CYCLE
037C 68        UNSM2:   PLA                 ; RESTORE X
037D AA                 TAX
037E 60                 RTS                 ; RETURN

               ;        QUAD SHIFT RIGHT SUBROUTINE
               ;        ENTER AT SRQA FOR ALGEBRAIC SHIFT RIGHT
               ;        ENTER AT SRQL FOR LOGICAL SHIFT
               ;        ENTER WITH QUAD PRECISION VALUE TO SHIFT IN PROD THROUGH PROD+3
               ;        DESTROYS A, PRESERVES X AND Y, RETURNS BIT SHIFTED OUT IN CARRY

037F A52A      SRQA:    LDA    PROD+3       ; GET SIGN BIT OF PROD IN CARRY
0381 0A                 ASLA
0382 662A      SRQL:    ROR    PROD+3       ; LOGICAL SHIFT RIGHT ENTRY
0384 6629               ROR    PROD+2
0386 6628               ROR    PROD+1
0388 6627               ROR    PROD
038A 60                 RTS                 ; RETURN


               ;        QUAD SHIFT LEFT SUBROUTINE
               ;        ENTER AT SLQL TO SHIFT IN A ZERO BIT
               ;        ENTER AT RLQL TO SHIFT IN THE CARRY
               ;        ENTER WITH QUAD PRECISION VALUE TO SHIFT IN PROD THROUGH PROD+3
               ;        DESTROYS A, PRESERVES X AND Y, RETURNS BIT SHIFTED OUT IN CARRY

038B 18        SLQL:    CLC                 ; SHIFT IN ZERO BIT ENTRY; CLEAR CARRY
038C 2627      RLQL:    ROL    PROD         ; SHIFT IN CARRY ENTRY
038E 2628               ROL    PROD+1
0390 2629               ROL    PROD+2
0392 262A               ROL    PROD+3
0394 60                 RTS                 ; RETURN

               ;        RANDOM NUMBER GENERATOR SUBROUTINE
               ;        ENTER WITH SEED IN RANDNO
               ;        EXIT WITH NEW RANDOM NUMBER IN RANDNO AND A
               ;        USES 16 BIT FEEDBACK SHIFT REGISTER METHOD
               ;        DESTROYS REGISTER A AND Y

0395 A008      RAND:    LDY    #8           ; SET COUNTER FOR 8 RANDOM BITS
0397 A50C      RAND1:   LDA    RANDNO       ; EXCLUSIVE-OR BITS 3, 12, 14, AND 15
0399 4A                 LSRA                ; OF SEED
039A 450C               EOR    RANDNO
039C 4A                 LSRA
039D 4A                 LSRA
039E 450C               EOR    RANDNO
03A0 4A                 LSRA
03A1 450D               EOR    RANDNO+1     ; RESULT IS IN BIT 3 OF A
03A3 4A                 LSRA                ; SHIFT INTO CARRY
03A4 4A                 LSRA
```

```
03A5 4A                 LSRA
03A6 4A                 LSRA
03A7 260D               ROL    RANDNO+1     ; SHIFT RANDNO LEFT ONE BRINGING IN CARRY
03A9 260C               ROL    RANDNO
03AB 88                 DEY                 ; TEST IF 8 NEW RANDOM BITS COMPUTED
03AC D0E9               BNE    RAND1        ; LOOP FOR MORE IF NOT
03AE A50C               LDA    RANDNO
03B0 60                 RTS                 ; RETURN

               ;        EXPONENTIALLY DISTRIBUTED RANDOM NUMBER SUBROUTINE
               ;        RULES OF USE SAME AS RAND, 8 BIT RESULT RETURNED IN A
               ;        AN EXPONENTIAL DISTRIBUTION MEANS THAT THE PROBABILITY OF A
               ;        RESULT BETWEEN 10 AND 20 IS THE SAME AS THE PROBABILITY OF A
               ;        RESULT BETWEEN 100 AND 200.
               ;        NOTE THAT THE PROBABILITY OF A ZERO RESULT IS ZERO.

03B1 209503    RNDEXP:  JSR    RAND         ; GET TWO NEW RANDOM BYTES
03B4 209503             JSR    RAND
03B7 A50C               LDA    RANDNO       ; CONVERT ONE OF THE BYTES TO A RANDOM
03B9 2907               AND    #7           ; VALUE BETWEEN 0 AND 7 AND PUT IN Y AS A
03BB A8                 TAY                 ; SHIFT COUNT
03BC C8                 INY
03BD A50D               LDA    RANDNO+1     ; GET THE OTHER RANDOM NUMBER AND SHIFT IT
03BF 88        RNDXP1:  DEY                 ; RIGHT ACCORDING TO Y
03C0 F004               BEQ    RNDXP2
03C2 4A                 LSRA
03C3 4CBF03             JMP    RNDXP1
03C6 0900      RNDXP2:  ORA    #0           ; TEST FOR A ZERO RESULT
03C8 F0E7               BEQ    RNDEXP       ; PROHIBIT ZERO RESULTS
03CA 60                 RTS                 ; RETURN

               ;        RANGCK - CHECK FOR ACCEPTABLE RANGE OF FREQ AND DAMP PARAMETERS
               ;        RETURN WITH CARRY OFF IF OK

03CB A502      RANGCK:  LDA    FREQ+1       ; MINIMUM ABSOLUTE VALUE FOR FREQ IS X'0100
03CD F01C               BEQ    RANGNK       ; GO TO FAILURE RETURN IF HIGH BYTE IS 0
03CF C9FF               CMP    #X'FF
03D1 F018               BEQ    RANGNK       ; GO TO FAILURE RETURN IF HIGH BYTE IS FF
03D3 A504      RANG2:   LDA    DAMP+1       ; CHECK THAT DAMP IS NOT GREATER THAN
03D5 C97F               CMP    #X'7F        ; X'7EFF
03D7 F012               BEQ    RANGNK       ; GO TO FAILURE RETURN IF SO
03D9 A502      RANG3:   LDA    FREQ+1       ; IF FREQ AND DAMP ARE INDIVIDUALLY OK,
03DB 1002               BPL    RANG4        ; VERIFY THAT DAMP IS ACCEPTABLY HIGH IF
03DD 45FF               EOR    X'FF         ; ABSOLUTE VALUE OF FREQ IS SMALL
03DF C908      RANG4:   CMP    #8
03E1 1006               BPL    RANGOK       ; GO TO SUCCESS RETURN IF FREQ IS HIGH
03E3 A504               LDA    DAMP+1       ; IF FREQ IS LOW, REQUIRE DAMP TO BE HIGH
03E5 C97E               CMP    #X'7E
03E7 3002               BMI    RANGNK       ; GO TO FAILURE RETURN IF DAMP NOT HIGH
                                            ; ENOUGH WHEN FREQ IS LESS THAN X'10
03E9 18        RANGOK:  CLC                 ; CLEAR CARRY TO INDICATE SUCCESS
03EA 60                 RTS                 ; RETURN
03EB 38        RANGNK:  SEC                 ; SET CARRY TO INDICATE FAILURE
03EC 60                 RTS                 ; RETURN


0000                    .END
```

```
                        .PAGE 'DOCUMENTATION, EQUATES, STORAGE'
               ;        MTU VISIBLE MEMORY DEMONSTRATION PROGRAM
               ;        JOSEPH CONWAY'S GAME OF LIFE ON A 320 BY 200 MATRIX

               ;        ENTRY POINT "DEMO" GENERATES AN INITIAL PATTERN OF CELLS AND
               ;        THEN EXECUTES THE LIFE ALGORITHM ON IT.

               ;        FOR USER ENTERED PATTERNS, THE SCREEN SHOULD FIRST BE CLEARED
               ;        BY EXECUTING "INIT". THE KIM KEYBOARD MONITOR OR "KYPT" MAY
               ;        THEN BE USED TO ENTER THE INITIAL CELL PATTERN. AFTER PATTERN
               ;        ENTRY, A JUMP TO "LIFE" WILL START COMPUTING THE SUCCEEDING
               ;        GENERATIONS.

               ;        LIFE MAY BE INTERRUPTED AT THE END OF A GENERATION BY PRESSING
               ;        ANY KEY (EXCEPT RESET OR ST) ON THE KIM KEYPAD AND HOLDING
               ;        UNTIL THE END OF THE GENERATION. THIS WILL TRANSFER CONTROL
               ;        TO "KYPT" FOR USER MODIFICATION OF THE DISPLAYED PATTERN.

               ;        KYPT IS USED FOR CONVENIENT ENTRY AND MODIFICATION OF CELL
               ;        PATTERNS. WHEN ENTERED, A BLINKING GRAPHIC CURSOR IS
               ;        DISPLAYED IN THE MIDDLE OF THE SCREEN. THE USER MAY MOVE THE
               ;        CURSOR IN ANY DIRECTION AND EITHER SET OR CLEAR CELLS AT THE
               ;        CURRENT CURSOR POSITION. THE CURSOR IS MOSTLY ON IF IT COVERS
               ;        A LIVE CELL AND MOSTLY OFF OTHERWISE.
               ;            THE KIM KEYBOARD IS USED FOR CONTROL OF THE PROGRAM. THE
               ;        FOLLOWING KEYS ARE ACTIVE:
               ;               1  CURSOR DOWN
               ;               6  CURSOR RIGHT
               ;               9  CURSOR UP
               ;               4  CURSOR LEFT
               ;               +  SET A CELL
               ;               F  CLEAR A CELL
               ;               GO GO TO LIFE ROUTINE USING THE CURRENT PATTERN
               ;        PARTICULARLY INTERESTING INITIAL PATTERNS MAY BE SAVED ON KIM
               ;        CASSETTE AND RELOADED LATER FOR DEMONSTRATIONS, ETC.

               ;        GENERAL EQUATES

1C22           KIMMON   =      X'1C22       ; ENTRY TO KIM MONITOR
1F6A           GETKEY   =      X'1F6A       ; ADDRESS OF MONITOR KEYBOARD READ ROUTINE
0140           NX       =      320          ; NUMBER OF BITS IN A ROW
00C8           NY       =      200          ; NUMBER OF ROWS  (CHANGE FOR HALF SCREEN
                                            ; OPERATION)
FA00           NPIX     =      NX*NY        ; NUMBER OF PIXELS
0032           DBCDLA   =      50           ; KIM KEYBOARD DEBOUNCE DELAY TIME

0000                    .=     0            ; START DEMO PROGRAM AT LOCATION ZERO

               ;        PARAMETER STORAGE

0000 20        VMORG:   .BYTE  X'20         ; FIRST PAGE IN DISPLAY MEMORY

               ;        MISCELLANEOUS STORAGE

0001           NCYSV:   .=.+   1            ; TEMPORARY STORAGE FOR NEIGHBOR COUNT
```

```
                                            ; ROUTINE
0002           NCNT:    .=.+   1            ; COUNT OF LIVE NEIGHBORS
0003           LNCNT:   .=.+   1            ; CELL LINE COUNTER
0004           NGEN:    .=.+   1            ; BYTE TO ACCUMULATE NEW CELLS
0005           ADP1:    .=.+   2            ; ADDRESS POINTER 1
0007           ADP2:    .=.+   2            ; ADDRESS POINTER 2
0009           BTPT:    .=.+   1            ; BIT NUMBER
000A           X1CORD:  .=.+   2            ; COORDINATE PAIR 1
000C           Y1CORD:  .=.+   2
000E           X2CORD:  .=.+   2            ; COORDINATE PAIR 2
0010           Y2CORD:  .=.+   2
0012           TEMP:    .=.+   2            ; TEMPORARY STORAGE
0014           FLASHC:  .=.+   2            ; TIME DELAY COUNTER FOR CURSOR FLASHING
0016           LSTKEY   =      NCYSV        ; CODE OF LAST KEY PRESSED ON KIM KEYBOARD
0016           DBCNT    =      NCNT         ; KIM KEYBOARD DEBOUNCE COUNTER
0016           REALST   =      LNCNT        ; STATE OF CELL UNDER THE CURSOR

               ;        TABLE OF MASKS FOR NEIGHBOR COUNTING

0016 01                 .BYTE  X'01
0017 80402010  MSK:     .BYTE  X'80,X'40,X'20,X'10
001B 08040201           .BYTE  X'08,X'04,X'02,X'01
001F 80                 .BYTE  X'80

               ;        STORAGE TO BUFFER 3 FULL SCAN LINES OF CELLS

0020 00                 .BYTE  0
0021           TR:      .=.+   40           ; ROW ABOVE CENTRAL ROW
0049           CR:      .=.+   40           ; CENTRAL ROW
0071           BR:      .=.+   40           ; ROW BELOW CENTRAL ROW
0099 00                 .BYTE  0

```

```
                          .PAGE  'INITIAL PATTERN GENERATION ROUTINES'
                ;         CLEAR DISPLAY MEMORY AND INITIALIZE ROUTINE
                ;         USED TO PREPARE SCREEN FOR USER ENTERED PATTERN

 009A D8        INIT:     CLD                   ; INITIALIZE MACHINE AND DISPLAY
 009B 202C02              JSR    CLEAR          ; CLEAR THE SCREEN
 009E 4C221C              JMP    KIMMON         ; RETURN TO THE MONITOR

                ;         MAIN DEMO ROUTINE, DRAW INITIAL PATTERN
                ;         DRAWS A FIGURE DEFINED BY "LIST" AND THEN JUMPS TO LIFE

 00A1 D8        DEMO:     CLD                   ; CLEAR DECIMAL MODE
 00A2 202C02              JSR    CLEAR          ; CLEAR THE SCREEN
 00A5 A200                LDX    #0             ; INITIALIZE INDEX FOR COORDINATE LIST
 00A7 BD3603    DEMO1:    LDA    LIST+1,X       ; GET HIGH BYTE OF X COORDINATE
 00AA 101A                BPL    DEMO2          ; JUMP IF A DRAW COMMAND
 00AC C9FF                CMP    #X'FF          ; IF MOVE, TEST FOR END OF LIST FLAG
 00AE F050                BEQ    LIFE           ; GO TO LIFE IF SO
 00B0 297F                AND    #X'7F          ; DELETE SIGN BIT
 00B2 850B                STA    X1CORD+1       ; FOR MOVE JUST COPY COORDINATES FROM LIST
 00B4 BD3503              LDA    LIST,X         ; INTO X1CORD,Y1CORD
 00B7 850A                STA    X1CORD
 00B9 BD3703              LDA    LIST+2,X
 00BC 850C                STA    Y1CORD
 00BE BD3803              LDA    LIST+3,X
 00C1 850D                STA    Y1CORD+1
 00C3 4CDA00              JMP    DEMO3
 00C6 850F      DEMO2:    STA    X2CORD+1       ; FOR DRAW, COPY COORDINATES FROM LIST
 00C8 BD3503              LDA    LIST,X         ; INTO X2CORD,Y2CORD
 00CB 850E                STA    X2CORD
 00CD BD3703              LDA    LIST+2,X
 00D0 8510                STA    Y2CORD
 00D2 BD3803              LDA    LIST+3,X
 00D5 8511                STA    Y2CORD+1
 00D7 20F502              JSR    SDRAW          ; DRAW LINE FROM X1CORD,Y1CORD TO X2CORD,
 00DA E8        DEMO3:    INX                   ; Y2CORD
 00DB E8                  INX                   ; BUMP INDEX TO NEXT SET OF COORDINATES
 00DC E8                  INX
 00DD E8                  INX
 00DE D0C7                BNE    DEMO1          ; LOOP UNTIL END OF LIST REACHED
 00E0 F01E                BEQ    LIFE           ; GO TO LIFE ROUTINE WHEN DONE

                ;         CSRINS - INSERT GRAPHIC CURSOR AT X1CORD,Y1CORD
                ;         SAVES STATE OF THE CELL ALREADY THERE IN REALST

 00E2 20CC02    CSRINS:   JSR    RDPIX          ; READ CURRENT STATE OF CELL UNDER CURSOR
 00E5 8503                STA    REALST         ; SAVE THE STATE
 00E7 60                  RTS                   ; RETURN

                ;         CSRDEL - DELETE THE GRAPHIC CURSOR AT X1CORD,Y1CORD
                ;         AND RESTORE THE CELL THAT WAS ORIGINALLY THERE

 00E8 A503      CSRDEL:   LDA    REALST         ; GET SAVED CELL STATE
 00EA 20C402              JSR    WRPIX          ; PUT IT BACK INTO DISPLAY MEMORY
 00ED 60                  RTS                   ; RETURN
```

```
                        .PAGE  'MAIN LIFE ROUTINE'
00EE                    .=     X'100

0100 A900      LIFE:    LDA    #0            ; PRIME THE THREE LINE BUFFERS
0102 8505               STA    ADP1          ; INITIALIZE VM POINTER TO TOP OF SCREEN
0104 A500               LDA    VMORG
0106 8506               STA    ADP1+1
0108 201D02             JSR    PRIME         ; DO THE PRIMING

               ;        MAIN LIFE LOOP

010B A9C6               LDA    #198          ; SET THE COUNT OF ROWS TO PROCESS
010D 8503               STA    LNCNT
010F A505      LIFE1:   LDA    ADP1          ; INCREMENT THE ADDRESS POINTER TO THE
0111 18                 CLC                  ; NEXT LINE
0112 6928               ADC    #40
0114 8505               STA    ADP1
0116 9002               BCC    LIFE2
0118 E606               INC    ADP1+1
011A 203101    LIFE2:   JSR    LFBUF         ; EXECUTE LIFE ALGORITHM ON CENTRAL ROW
                                             ; IN BUFFER AND UPDATE THE CURRENT ROW IN
                                             ; DISPLAY MEMORY
011D C603               DEC    LNCNT         ; DECREMENT THE LINE COUNT
011F F006               BEQ    LIFE3         ; JUMP OUT IF 198 LINES BEEN PROCESSED
0121 200002             JSR    ROLL          ; ROLL THE BUFFERS UP ONE POSITION
0124 4C0F01             JMP    LIFE1         ; GO PROCESS THE NEXT LINE

               ;        END OF GENERATION, TEST KIM KEYBOARD

0127 206A1F    LIFE3:   JSR    GETKEY
012A C915               CMP    #21
012C B0D2               BCS    LIFE          ; GO FOR NEXT GENERATION IF NO KET PRESSED
012E 4CC703             JMP    KYPT          ; GO TO KEYBOARD PATTERN ENTRY IF A
                                             ; KEY WAS PRESSED

                        .PAGE  'LIFE NEXT GENERATION ROUTINE FOR BUFFER CONTENTS'
               ;        LIFE NEXT GENERATION ROUTINE
               ;        THE CELLS IN THE MIDDLE LINE BUFFER ARE SCANNED AND THEIR
               ;        NEIGHBORS COUNTED TO DETERMINE IF THEY LIVE, DIE, OR GIVE
               ;        BIRTH. THE UPDATED CENTRAL LINE IS STORED BACK INTO DISPLAY
               ;        MEMORY STARTING AT (ADP1).
               ;        TO IMPROVE SPEED, WHEN PROCESSING THE CENTRAL 6 BITS IN A BYTE
               ;        THE ENTIRE BYTE AND ITS NEIGHBORS ARE CHECKED FOR ZERO.
               ;        IF ALL ARE ZERO, THE 6 BITS ARE SKIPPED.

0131 A000      LFBUF:   LDY    #0           ; INITIALIZE BYTE ADDRESS
0133 A207      LFBUF1:  LDX    #7           ; PREPARE FOR THE NEXT BYTE
0135 A900               LDA    #0           ; ZERO NEXT GEN BYTE
0137 8504               STA    NGEN
0139 E006      LFBUF2:  CPX    #6           ; TEST IF TO PROCESS BIT 6
013B D00D               BNE    LFBUF3       ; JUMP IF NOT
013D B92100             LDA    TR,Y         ; TEST IF CENTRAL BYTE AND ITS NEIGHBORS
0140 194900             ORA    CR,Y         ; ARE ALL ZEROES MEANING THAT NO CHANGE IS
0143 197100             ORA    BR,Y         ; POSSIBLE IN THE CENTRAL 6 BITS OF THE
0146 D002               BNE    LFBUF3       ; CURRENT BYTE
0148 A200               LDX    #0           ; IF ZEROES, SKIP 6 CENTRAL BITS
014A 207501    LFBUF3:  JSR    NCNTC        ; COUNT NEIGHBORS
014D A502               LDA    NCNT
014F F01B               BEQ    LFBUF6       ; JUMP IF EXACTLY 3 LIVE NEIGHBORS
0151 3004               BMI    LFBUF4       ; JUMP IF MORE THAN 3 LIVE NEIGHBORS
0153 C901               CMP    #1
0155 F00D               BEQ    LFBUF5       ; JUMP IF EXACTLY 2 LIVE NEIGHBORS
0157 CA        LFBUF4:  DEX                 ; DECREMENT BIT NUMBER
0158 10DF               BPL    LFBUF2       ; GO PROCESS NEXT BIT IF NOT DONE WITH BYTE
015A A504               LDA    NGEN         ; STORE NEXT GENERATION BYTE INTO DISPLAY
015C 9105               STA    (ADP1),Y     ; MEMORY
015E C8                 INY                 ; GO TO NEXT BYTE
015F C028               CPY    #40          ; TEST IF DONE
0161 D0D0               BNE    LFBUF1       ; LOOP IF NOT
0163 60                 RTS                 ; OTHERWISE RETURN

0164 B94900    LFBUF5:  LDA    CR,Y         ; WHEN EXACTLY 2 NEIGHBORS, TEST CURRENT
0167 3517               AND    MSK,X        ; CELL
0169 4C6E01             JMP    LFBUF7       ; NEW CELL IF CURRENT CELL IS ALIVE

016C B517      LFBUF6:  LDA    MSK,X        ; CREATE A CELL IN THE NEXT GENERATION
016E 0504      LFBUF7:  ORA    NGEN
0170 8504               STA    NGEN
0172 4C5701             JMP    LFBUF4

```

```
                                .PAGE 'NEIGHBOR COUNT ROUTINE'
                ;       NEIGHBOR COUNT ROUTINE FOR ALL EIGHT NEIGHBORS OF A CENTRAL
                ;       CELL. USES THREE SCAN LINE BUFFER IN BASE PAGE FOR MAXIMUM
                ;       SPEED. INDEX Y POINTS TO BYTE CONTAINING CENTRAL CELL
                ;       RELATIVE TO BEGINNING OF CENTRAL SCAN LINE. INDEX X HAS BIT
                ;       NUMBER OF CENTRAL CELL, O=LEFTMOST IN BYTE. EXITS WITH 3-N IN
                ;       NCNT WHERE N IS NUMBER OF LIVE NEIGHBORS. PRESERVES X AND Y.

0175 8401       NCNTC:  STY    NCYSV        ; SAVE Y
0177 A903               LDA    #3           ; INITIALIZE THE NEIGHBOR COUNT
0179 8502               STA    NCNT
017B B92100     N1:     LDA    TR,Y         ; CHECK CELLS DIRECTLY ABOVE AND BELOW
017E 3517               AND    MSK,X        ; CENTRAL CELL FIRST
0180 F002               BEQ    N2
0182 C602               DEC    NCNT
0184 B97100     N2:     LDA    BR,Y
0187 3517               AND    MSK,X
0189 F002               BEQ    N3
018B C602               DEC    NCNT
018D E000       N3:     CPX    #0           ; TEST COLUMN OF 3 LEFT CELLS NEXT
018F D001               BNE    N3A          ; SKIP AHEAD IF IN THE SAME BYTE
0191 88                 DEY                 ; OTHERWISE MOVE 1 BYTE LEFT
0192 B92100     N3A:    LDA    TR,Y
0195 3516               AND    MSK-1,X
0197 F002               BEQ    N4
0199 C602               DEC    NCNT
019B B94900     N4:     LDA    CR,Y
019E 3516               AND    MSK-1,X
01A0 F004               BEQ    N5
01A2 C602               DEC    NCNT
01A4 302F               BMI    NCXIT        ; QUICK EXIT IF MORE THAN 3 NEIGHBORS
01A6 B97100     N5:     LDA    BR,Y
01A9 3516               AND    MSK-1,X
01AB F004               BEQ    N6
01AD C602               DEC    NCNT
01AF 3024               BMI    NCXIT        ; QUICK EXIT IF MORE THAN 3 NEIGHBORS
01B1 A401       N6:     LDY    NCYSV        ; RESTORE Y
01B3 E007               CPX    #7           ; TEST COLUMN OF 3 RIGHT CELLS LAST
01B5 D001               BNE    N6A          ; SKIP AHEAD IF IN THE SAME BYTE
01B7 C8                 INY                 ; OTHERWISE MOVE 1 BYTE RIGHT
01B8 B92100     N6A:    LDA    TR,Y
01BB 3518               AND    MSK+1,X
01BD F004               BEQ    N7
01BF C602               DEC    NCNT
01C1 3012               BMI    NCXIT        ; QUICK EXIT IF MORE THAN 3 NEIGHBORS
01C3 B94900     N7:     LDA    CR,Y
01C6 3518               AND    MSK+1,X
01C8 F002               BEQ    N8
01CA C602               DEC    NCNT
01CC B97100     N8:     LDA    BR,Y
01CF 3518               AND    MSK+1,X
01D1 F002               BEQ    NCXIT
01D3 C602               DEC    NCNT
01D5 A401       NCXIT:  LDY    NCYSV        ; RESTORE Y
01D7 60                 RTS                 ; AND RETURN
```

```
                         .PAGE  'CELL LINE MOVE ROUTINES'
                ;        ROLL THE THREE LINE BUFFERS UP ONE POSITION
                ;        AND BRING IN A NEW LINE FROM DISPLAY MEMORY STARTING AT
                ;        (ADP1) +80 PRESERVES INDEX REGISTERS

01D8                     .=     X'200
0200 98         ROLL:    TYA                  ; SAVE INDEX Y
0201 48                  PHA
0202 A050                LDY    #80           ; INITIALIZE INDEX
0204 B9F9FF     ROLL1:   LDA    CR-80,Y       ; ROLL A BYTE
0207 99D1FF              STA    TR-80,Y
020A B92100              LDA    BR-80,Y
020D 99F9FF              STA    CR-80,Y
0210 B105                LDA    (ADP1),Y
0212 992100              STA    BR-80,Y
0215 C8                  INY                  ; INCREMENT INDEX
0216 C078                CPY    #120          ; TEST IF 40 BYTES ROLLED
0218 D0EA                BNE    ROLL1         ; LOOP IF NOT
021A 68                  PLA                  ; RESTORE Y
021B A8                  TAY
021C 60                  RTS                  ; RESTURN

                ;        PRIME THE LINE BUFFERS WITH THE FIRST THREE LINES OF DISPLAY
                ;        MEMORY
                ;        MOVES 120 BYTES STARTING AT (ADP1) INTO LINE BUFFERS STARTING
                ;        AT TR

021D 98         PRIME:   TYA                  ; SAVE INDEX Y
021E 48                  PHA
021F A077                LDY    #119          ; INITIALIZE INDEX
0221 B105       PRIME1:  LDA    (ADP1),Y      ; MOVE A BYTE
0223 992100              STA     TR,Y
0226 88                  DEY                  ; DECREMENT INDEX
0227 10F8                BPL    PRIME1        ; LOOP IF NOT DONE
0229 68                  PLA                  ; RESTORE Y
022A A8                  TAY
022B 60                  RTS                  ; RETURN

                ;        CLEAR DISPLAY MEMORY ROUTINE

022C A000       CLEAR:   LDY    #0            ; INITIALIZE ADDRESS POINTER
022E 8405                STY    ADP1          ; AND ZERO INDEX Y
0230 A500                LDA    VMORG
0232 8506                STA    ADP1+1
0234 18                  CLC
0235 6920                ADC    #X'20
0237 AA                  TAX
0238 98         CLEAR1:  TYA                  ; CLEAR A BYTE
0239 9105                STA    (ADP1),Y
023B E605                INC    ADP1          ; INCREMENT ADDRESS POINTER
023D D0F9                BNE    CLEAR1
023F E606                INC    ADP1+1
0241 E406                CPX    ADP1+1        ; TEST IF DONE
0243 D0F3                BNE    CLEAR1
0245 60                  RTS                  ; RETURN
```

```
                        .PAGE  'GRAPHICS ROUTINES FOR GENERATING THE INITIAL PATTERN'
               ;        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

0246 A50A      PIXADR:  LDA    X1CORD       ; COMPUTE BIT ADDRESS FIRST
0248 8505               STA    ADP1         ; ALSO TRANSFER X1CORD TO ADP1
024A 2907               AND    #X'07        ; WHICH IS SIMPLY THE LOW 3 BITS OF X
024C 8509               STA    BTPT
024E A50B               LDA    X1CORD+1     ; FINISH TRANSFERRING X1CORD TO ADP1
0250 8506               STA    ADP1+1
0252 4606               LSR    ADP1+1       ; DOUBLE SHIFT ADP1 RIGHT 3 TO GET
0254 6605               ROR    ADP1         ; INT(XCORD/8)
0256 4606               LSR    ADP1+1
0258 6605               ROR    ADP1
025A 4606               LSR    ADP1+1
025C 6605               ROR    ADP1
025E A9C7               LDA    #199         ; TRANSFER (199-Y1CORD) TO ADP2
0260 38                 SEC                 ; AND TEMPORARY STORAGE
0261 E50C               SBC    Y1CORD
0263 8507               STA    ADP2
0265 8512               STA    TEMP
0267 A900               LDA    #0
0269 E50D               SBC    Y1CORD+1
026B 8508               STA    ADP2+1
026D 8513               STA    TEMP+1
026F 0607               ASL    ADP2         ; COMPUTE 40*(199-Y1CORD)
0271 2608               ROL    ADP2+1       ;  2*(199-Y1CORD)
0273 0607               ASL    ADP2
0275 2608               ROL    ADP2+1       ;  4*(199+Y1CORD)
0277 A507               LDA    ADP2         ;  ADD IN TEMPORARY SAVE OF (199-Y1CORD)
0279 18                 CLC                 ;  TO MAKE 5*(199-Y1CORD)
027A 6512               ADC    TEMP
027C 8507               STA    ADP2
027E A508               LDA    ADP2+1
0280 6513               ADC    TEMP+1
0282 8508               STA    ADP2+1       ;  5*(199-Y1CORD)
0284 0607               ASL    ADP2         ;  10*(199-Y1CORD)
0286 2608               ROL    ADP2+1
0288 0607               ASL    ADP2         ;  20*(199-Y1CORD)
028A 2608               ROL    ADP2+1
028C 0607               ASL    ADP2         ;  40*(199-Y1CORD)
028E 2608               ROL    ADP2+1
0290 A507               LDA    ADP2         ; ADD IN INT(X1CORD/8) COMPUTED EARLIER
0292 18                 CLC
0293 6505               ADC    ADP1
0295 8505               STA    ADP1
0297 A508               LDA    ADP2+1
```

```
0299 6506                ADC    ADP1+1
029B 6500                ADC    VMORG         ; ADD IN VMORG*256
029D 8506                STA    ADP1+1        ; FINAL RESULT
029F 60                  RTS                  ; RETURN

                ;        STPIX - SETS THE PIXEL AT X1CORD,Y1CORD TO A ONE (WHITE DOT)
                ;        DOES NOT ALTER X1CORD OR Y1CORD
                ;        PRESERVES X AND Y
                ;        ASSUMES IN RANGE CORRDINATES

02A0 204602     STPIX:   JSR    PIXADR        ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
                                              ; INTO ADP1
02A3 98                  TYA                  ; SAVE Y
02A4 48                  PHA
02A5 A409                LDY    BTPT          ; GET BIT NUMBER IN Y
02A7 B9E502              LDA    MSKTB1,Y      ; GET A BYTE WITH THAT BIT =1, OTHERS =0
02AA A000                LDY    #0            ; ZERO Y
02AC 1105                ORA    (ADP1),Y      ; COMBINE THE BIT WITH THE ADDRESSED VM
                                              ; BYTE
02AE 4CBF02              JMP    CLPIX1        ; GO STORE RESULT, RESTORE Y, AND RETURN

                ;        CLPIX - CLEARS THE PIXEL AT X1CORD,Y1CORD TO A ZERO (BLACK DOT
                ;        DOES NOT ALTER X1CORD OR Y1CORD
                ;        PRESERVES X AND Y
                ;        ASSUMES IN RANGE COORDINATES

02B1 204602     CLPIX:   JSR    PIXADR        ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
                                              ; INTO ADP1
02B4 98                  TYA                  ; SAVE Y
02B5 48                  PHA
02B6 A409                LDY    BTPT          ; GET BIT NUMBER IN Y
02B8 B9ED02              LDA    MSKTB2,Y      ; GET A BYTE WITH THAT BIT =0, OTHERS =1
02BB A000                LDY    #0            ; ZERO Y
02BD 3105                AND    (ADP1),Y      ; REMOVE THE BIT FROM THE ADDRESSED VM
02BF 9105       CLPIX1:  STA    (ADP1),Y      ; BYTE
02C1 68                  PLA                  ; RESTORE Y
02C2 A8                  TAY
02C3 60                  RTS                  ; AND RETURN

                ;        WRPIX - SETS THE PIXEL AT X1CORD,Y1CORD ACCORDING TO THE STATE
                ;        OF BIT 0 (RIGHTMOST) OF A
                ;        DOES NOT ALTER X1CORD OR Y1CORD
                ;        PRESERVES X AND Y
                ;        ASSUMES IN RANGE CORRDINATES

02C4 2CCB02     WRPIX:   BIT    WRPIXM        ; TEST LOW BIT OF A
02C7 F0E8                BEQ    CLPIX         ; JUMP IF A ZERO TO BE WRITTEN
02C9 D0D5                BNE    STPIX         ; OTHERWISE WRITE A ONE

02CB 01         WRPIXM:  .BYTE  1             ; BIT TEST MASK FOR BIT 0

                ;        RDPIX - READS THE PIXEL AT X1CORD,Y1CORD AND SETS A TO ALL
                ;        ZEROES IF IT IS A ZERO OR TO ALL ONES IF IT IS A ONE
                ;        LOW BYTE OF ADP1 IS EQUAL TO A ON RETURN
                ;        DOES NOT ALTER X1CORD OR Y1CORD
```

```
                   ;        PRESERVES X AND Y
                   ;        ASSUMES IN RANGE CORRDINATES

02CC 204602    RDPIX:   JSR    PIXADR       ; GET BYTE AND BIT ADDRESS OF PIXEL
02CF 98                 TYA                 ; SAVE Y
02D0 48                 PHA
02D1 A000               LDY    #0           ; GET ADDRESSED BYTE FROM VM
02D3 B105               LDA    (ADP1),Y
02D5 A409               LDY    BTPT         ; GET BIT NUMBER IN Y
02D7 39E502             AND    MSKTB1,Y     ; CLEAR ALL BUT ADDRESSED BIT
02DA F002               BEQ    RDPIX1       ; SKIP AHEAD IF IT WAS A ZERO
02DC A9FF               LDA    #X'FF        ; SET TO ALL ONES IF IT WAS A ONE
02DE 8505      RDPIX1:  STA    ADP1         ; SAVE A TEMPORARILY IN ADP1 WHILE
02E0 68                 PLA                 ; RESTORING Y
02E1 A8                 TAY
02E2 A505               LDA    ADP1
02E4 60                 RTS                 ; RETURN

                   ;        MASK TABLES FOR INDIVIDUAL PIXEL SUBROUTINES
                   ;        MSKTB1 IS A TABLE OF 1 BITS CORRESPONDING TO BIT NUMBERS
                   ;        MSKTB2 IS A TABLE OF 0 BITS CORRESPONDING TO BIT NUMBERS

02E5 80402010  MSKTB1:  .BYTE  X'80,X'40,X'20,X'10
02E9 08040201           .BYTE  X'08,X'04,X'02,X'01
02ED 7FBFDFEF  MSKTB2:  .BYTE  X'7F,X'BF,X'DF,X'EF
02F1 F7FBFDFE           .BYTE  X'F7,X'FB,X'FD,X'FE

                   ;        SDRAW - SIMPLIFIED DRAW ROUTINE
                   ;        DRAWS A LINE FROM X1CORD,Y1CORD TO X2CORD,Y2CORD
                   ;        WHEN DONE COPIES X2CORD AND Y2CORD INTO X1CORD AND Y1CORD
                   ;        RESTRICTED TO HORIZONTAL, VERTICAL, AND 45 DEGREE DIAGONAL
                   ;        LINES (SLOPE=1)
                   ;        PRESERVES BOTH INDEX REGISTERS

02F5 8A        SDRAW:   TXA                 ; SAVE INDEX REGS
02F6 48                 PHA
02F7 98                 TYA
02F8 48                 PHA
02F9 20A002             JSR    STPIX        ; PUT A DOT AT INITIAL ENDPOINT
02FC A000      SDRAW1:  LDY    #0           ; CLEAR "SOMETHING DONE" FLAG
02FE A200               LDX    #0           ; UPDATE X COORDINATE
0300 201303             JSR    UPDC
0303 A202               LDX    #Y1CORD-X1CORD;UPDATE Y COORDINATE
0305 201303             JSR    UPDC
0308 20A002             JSR    STPIX        ; PUT A DOT AT INTERMEDIATE POINT
030B 88                 DEY                 ; TEST IF EITHER COORDINATE CHANGED
030C 10EE               BPL    SDRAW1       ; ITERATE AGAIN IF SO
030E 68                 PLA                 ; RESTORE INDEX REGISTERS
030F A8                 TAY
0310 68                 PLA
0311 AA                 TAX
0312 60                 RTS                 ; RETURN

                   ;        INTERNAL SUBROUTINE FOR UPDATING COORDINATES

```

```
0313 B50F      UPDC:    LDA    X2CORD+1,X   ; COMPARE ENDPOINT WITH CURRENT POSITION
0315 D50B               CMP    X1CORD+1,X
0317 9012               BCC    UPDC3        ; JUMP IF CURRENT POSITION IS LARGER
0319 D008               BNE    UPDC1        ; JUMP IF ENDPOINT IS LARGER
031B B50E               LDA    X2CORD,X
031D D50A               CMP    X1CORD,X
031F 900A               BCC    UPDC3        ; JUMP IF CURRENT POSITION IS LARGER
0321 F011               BEQ    UPDC5        ; GO RETURN IF EQUAL
0323 F60A      UPDC1:   INC    X1CORD,X     ; ENDPOINT IS LARGER, INCREMENT CURRENT
0325 D002               BNE    UPDC2        ; POSITION
0327 F60B               INC    X1CORD+1,X
0329 C8        UPDC2:   INY                 ; SET "DONE SOMETHING" FLAG
032A 60                 RTS                 ; RETURN
032B B50A      UPDC3:   LDA    X1CORD,X     ; CURRENT POSITION IS LARGER, DECREMENT
032D D002               BNE    UPDC4        ; CURRENT POSITION
032F D60B               DEC    X1CORD+1,X
0331 D60A      UPDC4:   DEC    X1CORD,X
0333 C8                 INY                 ; SET "DONE SOMETHING" FLAG
0334 60        UPDC5:   RTS                 ; RETURN
```

```
                          .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.

0335 38803C00  LIST:      .WORD  56+X'8000,60   ; 1     MOVE
0339 38008C00             .WORD  56,140         ; 2     DRAW
033D 48008C00             .WORD  72,140         ; 3     DRAW
0341 48004C00             .WORD  72,76          ; 4
0345 68004C00             .WORD  104,76         ; 5
0349 68003C00             .WORD  104,60         ; 6
034D 38003C00             .WORD  56,60          ; 7
0351 78803C00             .WORD  120+X'8000,60  ; 8     MOVE
0355 78008C00             .WORD  120,140        ; 9
0359 88008C00             .WORD  136,140        ; 10
035D 88003C00             .WORD  136,60         ; 11
0361 78003C00             .WORD  120,60         ; 12
0365 98803C00             .WORD  152+X'8000,60  ; 13    MOVE
0369 98008C00             .WORD  152,140        ; 14
036D C8008C00             .WORD  200,140        ; 15
0371 C8007C00             .WORD  200,124        ; 16
0375 A8007C00             .WORD  168,124        ; 17
0379 A8006C00             .WORD  168,108        ; 18
037D C0006C00             .WORD  192,108        ; 19
0381 C0005C00             .WORD  192,92         ; 20
0385 A8005C00             .WORD  168,92         ; 21
0389 A8003C00             .WORD  168,60         ; 22
038D 98003C00             .WORD  152,60         ; 23
0391 D8803C00             .WORD  216+X'8000,60  ; 24    MOVE
0395 D8008C00             .WORD  216,140        ; 25
0399 08018C00             .WORD  264,140        ; 26
039D 08017C00             .WORD  264,124        ; 27
03A1 E8007C00             .WORD  232,124        ; 28
03A5 E8006C00             .WORD  232,108        ; 29
03A9 00016C00             .WORD  256,108        ; 30
03AD 00015C00             .WORD  256,92         ; 31
03B1 E8005C00             .WORD  232,92         ; 32
03B5 E8004C00             .WORD  232,76         ; 33
03B9 08014C00             .WORD  264,76         ; 34
03BD 08013C00             .WORD  264,60         ; 35
03C1 D8003C00             .WORD  216,60         ; 36
03C5 FFFF                 .WORD  X'FFFF         ; END OF LIST

```

```
                        .PAGE  'KEYBOARD PATTERN ENTRY ROUTINES'
               ;        KEYBOARD PATTERN ENTRY ROUTINES
               ;        USES THE KIM KEYBOARD AND A CURSOR TO SIMPLIFY THE ENTRY
               ;        OF INITIAL LIFE PATTERNS

03C7 A900      KYPT:    LDA    #0           ; SET INITIAL CURSOR POSITION IN CENTER
03C9 850B               STA    X1CORD+1     ; OF SCREEN
03CB 850D               STA    Y1CORD+1
03CD A9A0               LDA    #160
03CF 850A               STA    X1CORD
03D1 A964               LDA    #100
03D3 850C               STA    Y1CORD
03D5 20E200             JSR    CSRINS       ; INSERT A CURSOR ON THE SCREEN
03D8 A932      KYPT0:   LDA    #DBCDLA      ; RESET THE DEBOUNCE COUNT
03DA 8502               STA    DBCNT
03DC E614      KYPT1:   INC    FLASHC       ; DOUBLE INCREMENT CURSOR FLASH COUNT
03DE D002               BNE    KYPT2
03E0 E615               INC    FLASHC+1

               ;        GENERATE A 25% DUTY CURSOR IF CELL IS DEAD AND 75% IF ALIVE

03E2 A515      KYPT2:   LDA    FLASHC+1     ; GET HIGH BYTE OF FLASH COUNTER
03E4 4A                 LSRA                ; COMPUTE LOGICAL "AND" OF BITS 0 AND 1
03E5 2515               AND    FLASHC+1     ; IN ACC BIT 0
03E7 4503               EOR    REALST       ; EXCLUSIVE-OR WITH REAL STATE OF CELL
03E9 20C402             JSR    WRPIX        ; DISPLAY THE CURSOR

               ;        READ KIM KEYBOARD AND DETECT ANY CHANGE IN KEYS PRESSED

03EC 206A1F             JSR    GETKEY       ; GET CURRENT PRESSED KEY
03EF C501               CMP    LSTKEY       ; TEST IF SAME AS BEFORE
03F1 F0E5               BEQ    KYPT0        ; IGNORE IF SO
03F3 C602               DEC    DBCNT        ; IF DIFFERENT, DECREMENT AND TEST
03F5 10E5               BPL    KYPT1        ; DEBOUNCE COUNT AND IGNORE KEY IF NOT RUN
                                            ; OUT
03F7 8501               STA    LSTKEY       ; AFTER DEBOUNCE, UPDATE KEY LAST PRESSED
03F9 4C8017             JMP    KYPT6        ; AND GO PROCESS THE KEYSTROKE

03FC                    .=     X'1780       ; CONTINUE PROGRAM IN 6530 RAM

1780 C901      KYPT6:   CMP    #1           ; TEST "1" KEY
1782 F01B               BEQ    CSRD         ; JUMP IF CURSOR DOWN
1784 C909               CMP    #9           ; TEST "9" KEY
1786 F01F               BEQ    CSRU         ; JUMP IF CURSOR UP
1788 C904               CMP    #4           ; TEST "4" KEY
178A F023               BEQ    CSRL         ; JUMP IF CURSOR LEFT
178C C906               CMP    #6           ; TEST "6" KEY
178E F02D               BEQ    CSRR         ; JUMP IF CURSOR RIGHT
1790 C913               CMP    #19          ; TEST "GO" KEY
1792 F043               BEQ    GO           ; JUMP IF GO KEY
1794 C912               CMP    #18          ; TEST "+" KEY
1796 F034               BEQ    SETCEL       ; JUMP IF SET CELL KEY
1798 C90F               CMP    #15          ; TEST "F" KEY
179A F034               BEQ    CLRCEL       ; JUMP IF CLEAR CELL KEY
179C 4CD803             JMP    KYPT0        ; IGNORE ANY OTHER KEYS
```

```

179F 20E800    CSRD:    JSR    CSRDEL       ; DELETE EXISTING CURSOR
17A2 C60C               DEC    Y1CORD       ; DECREMENT Y COORDINATE FOR CURSOR DOWN
17A4 4CC617             JMP    CSRMOV

17A7 20E800    CSRU:    JSR    CSRDEL       ; DELETE EXISTING CURSOR
17AA E60C               INC    Y1CORD       ; INCREMENT Y COORDINATE FOR CURSOR UP
17AC 4CC617             JMP    CSRMOV

17AF 20E800    CSRL:    JSR    CSRDEL       ; DELETE EXISTING CURSOR
17B2 A50A               LDA    X1CORD       ; DECREMENT X COORDINATE FOR CURSOR LEFT
17B4 D002               BNE    CSRL1
17B6 C60B               DEC    X1CORD+1
17B8 C60A      CSRL1:   DEC    X1CORD
17BA 4CC617             JMP    CSRMOV

17BD 20E800    CSRR:    JSR    CSRDEL       ; DELETE EXISTING CURSOR
17C0 E60A               INC    X1CORD       ; INCREMENT X COORDINATE FOR CURSOR RIGHT
17C2 D002               BNE    CSRMOV
17C4 E60B               INC    X1CORD+1

17C6 20E200    CSRMOV:  JSR    CSRINS       ; INSERT CURSOR AT NEW LOCATION
17C9 4CD803             JMP    KYPT0        ; GO BACK TO KEYBOARD INPUT LOOP

17CC A9FF      SETCEL:  LDA    #X'FF        ; SET REAL CELL STATE TO LIVE
17CE D002               BNE    CLRCL1

17D0 A900      CLRCEL:  LDA    #0           ; SET REAL CELL STATE TO DEAD
17D2 8503      CLRCL1:  STA    REALST
17D4 4CD803             JMP    KYPT0        ; GO BACK TO KEYBOARD INPUT LOOP

17D7 20E800    GO:      JSR    CSRDEL       ; DELETE CURSOR AND RESTORE THE CELL UNDER
                                            ; THE CURSOR
17DA 4C0001             JMP    LIFE         ; AND GO EXECUTE LIFE


0000                    .END
```
NO ERROR LINES

```
                        .PAGE  'SIMPLIFIED VISABLE MEMORY TEXT DISPLAY SUBROUTINE'
               ;        THIS SUBROUTINE TURNS THE VISABLE MEMORY INTO A DATA DISPLAY
               ;        TERMINAL (GLASS TELETYPE).
               ;        CHARACTER SET IS 96 FULL ASCII UPPER AND LOWER CASE.
               ;        CHARACTER MATRIX IS 5 BY 7 SET INTO A 6 BY 9 RECTANGLE.
               ;        LOWER CASE IS REPRESENTED AS SMALL (5 BY 5) CAPITALS.
               ;        SCREEN CAPACITY IS 22 LINES OF 53 CHARACTERS FOR FULL SCREEW
               ;        OR 11 LINES FOR HALF SCREEN.
               ;        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 UP ONE LINE IF ALREADY ON BOTTOM
               ;                            LINE
               ;        BS     X'08         MOVES CURSOR ONE CHARACTER LEFT, DOES
               ;                            NOTHING IF ALREADY AT LEFT SCREEN EDGE
               ;        FF     X'0C         CLEARS SCREEN AND PUTS CURSOR AT TOP LEFT
               ;                            OF SCREEN, SHOULD BE CALLED FOR
               ;                            INITIALIZATION
               ;        ALL OTHER CONTROL CODES IGNORED.
               ;        ENTER WITH CHARACTER TO BE DISPLAYED IN A.
               ;        X AND Y PRESERVED.
               ;        3 BYTES OF RAM STORAGE REQUIRED FOR KEEPING TRACK OF THE
               ;        CURSOR
               ;        4 BYTES OF TEMPORARY STORAGE IN BASE PAGE REQUIRED FOR ADDRESS
               ;        POINTERS. (CAN BE DESTROYED BETWEEN CALLS TO SDTXT
               ;        4 BYTES OF TEMPORARY STORAGE ANYWHERE (CAN BE DESTROYED
               ;        BETWEEN CALLS TO SDTXT)

               ;        * **** VMORG #MUST# BE SET TO THE PAGE NUMBER OF THE VISIBLE *
               ;        * MEMORY BEFORE CALLING SDTXT ****                           *

               ;        GENERAL EQUATES

1F40           NLOC     =      8000         ; NUMBER OF VISIBLE LOCATIONS
0009           CHHI     =      9            ; CHARACTER WINDOW HEIGHT
0006           CHWID    =      6            ; CHARACTER WINDOW WIDTH
0035           NCHR     =      320/CHWID    ; NUMBER OF CHARACTERS PER LINE
0016           NLIN     =      NLOC/40/CHHI ; NUMBER OF TEXT LINES
1D88           NSCRL    =      NLIN-1*CHHI*40 ; NUMBER OF LOCATIONS TO SCROLL
01B8           NCLR     =      NLOC-NSCRL   ; NUMBER OF LOCATIONS TO CLEAR AFTER SCROLL

               ;        BASE PAGE TEMPORARY STORAGE

0000                    .=     X'EA
00EA           ADP1     .=.+   2            ; ADDRESS POINTER 1
00EC           ADP2     .=.+   2            ; ADDRESS POINTER 2

               ;        GENERAL TEMPORARY STORAGE

00EE                    .=     X'5B00       ; PLACE AT END OF 16K EXPANSION

5B00           BTPT:    .=.+   1            ; BIT NUMBER TEMPORARY STORAGE
5B01           DCNT1:   .=.+   2            ; DOUBLE PRECISION COUNTER
5B03           MRGT1:   .=.+   1            ; TEMPORARY STORAGE FOR MERGE
```

```

               ;        PERMANENT RAM STORAGE

5B04           CSRX:    .=.+   1            ; CURRENT CHARACTER NUMBER (0=LEFT CHAR)
5B05           CSRY:    .=.+   1            ; CURRENT LINE NUMBER (0=TOP LINE)
5B06           VMORG:   .=.+   1            ; FIRST PAGE NUMBER OF VISIBLE MEMORY

5B07 48        SDTXT:   PHA                 ; SAVE REGISTERS
5B08 8A                 TXA
5B09 48                 PHA
5B0A 98                 TYA
5B0B 48                 PHA
5B0C A900               LDA    #0           ; CLEAR UPPER ADP2
5B0E 85ED               STA    ADP2+1
5B10 BA                 TSX                 ; GET INPUT BACK
5B11 BD0301             LDA    X'103,X
5B14 297F               AND    #X'7F        ; INSURE 7 BIT ASCII INPUT
5B16 38                 SEC
5B17 E920               SBC    #X'20        ; TEST IF A CONTROL CHARACTER
5B19 3047               BMI    SDTX10       ; JUMP IF SO

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

5B1B 85EC      SDTXT1:  STA    ADP2         ; SAVE CHARACTER CODE IN ADP2
5B1D 20225C             JSR    SADP2L       ; COMPUTE 8*CHARACTER CODE IN ADP2
5B20 20225C             JSR    SADP2L
5B23 20225C             JSR    SADP2L
5B26 49FF               EOR    #X'FF        ; NEGATE CHARACTER CODE
5B28 38                 SEC                 ; SUBSTRACT CHARACTER CODE FROM ADP2 AND
5B29 65EC               ADC    ADP2         ; PUT RESULT IN ADP1 FOR A FINAL RESULT OF
5B2B 85EA               STA    ADP1         ; 7*CHARACTER CODE
5B2D A5ED               LDA    ADP2+1
5B2F 69FF               ADC    #X'FF
5B31 85EB               STA    ADP1+1
5B33 A5EA               LDA    ADP1         ; ADD IN ORIGIN OF CHARACTER TABLE
5B35 18                 CLC
5B36 6921               ADC    #CHTB&X'FF
5B38 85EA               STA    ADP1
5B3A A5EB               LDA    ADP1+1
5B3C 695D               ADC    #CHTB/256
5B3E 85EB               STA    ADP1+1       ; ADP1 NOW HAS ADDRESS OF TOP ROW OF
                                            ; CHARACTER SHAPE
               ;        COMPUTE BYTE AND BIT ADDRESS OF FIRST SCAN LINE OF
               ;        CHARACTER AT CURSOR POSITION

5B40 20355C             JSR    CSRTAD       ; COMPUTE BYTE AND BIT ADDRESSES OF FIRST
                                            ; SCAN LINE OF CHARACTER AT CURSOR POS.

               ;        SCAN OUT THE 7 CHARACTER ROWS

5B43 A000               LDY    #0           ; INITIALIZE Y INDEX=FONT TABLE POINTER
5B45 B1EA      SDTX2:   LDA    (ADP1),Y     ; GET A DOT ROW FROM THE FONT TABLE
5B47 20805C             JSR    MERGE        ; MERGE IT WITH GRAPHIC MEMORY AT (ADP2)
5B4A 20275C             JSR    DN1SCN       ; ADD 40 TO ADP2 TO MOVE DOWN ONE SCAN
                                            ; LINE IN GRAPHIC MEMORY
```

```
5B4D C8                 INY                 ; BUMP UP POINTER INTO FONT TABLE
5B4E C007               CPY   #7            ; TEST IF DONE
5B50 D0F3               BNE   SDTX2         ; GO DO NEXT SCAN LINE IF NOT
5B52 AD045B             LDA   CSRX          ; DO A CURSOR RIGHT
5B55 C934               CMP   #NCHR-1       ; TEST IF LAST CHARACTER ON THE LINE
5B57 1006               BPL   SDTX3         ; SKIP CURSOR RIGHT IF SO
5B59 201A5C             JSR   CSRCLR        ; CLEAR OLD CURSOR
5B5C EE045B             INC   CSRX          ; MOVE CURSOR ONE POSITION RIGHT
5B5F 4CF85B SDTX3:      JMP   SDTXRT        ; GO INSERT CURSOR, RESTORE REGISTERS,
                                            ; AND RETURN


             ;         INTERPRET CONTROL CODES

5B62 C9ED   SDTX10:     CMP   #X'0D-X'20    ; TEST IF CR
5B64 F00F               BEQ   SDTXCR        ; JUMP IF SO
5B66 C9EA               CMP   #X'0A-X'20    ; TEST IF LF
5B68 F047               BEQ   SDTXLF        ; JUMP IF SO
5B6A C9E8               CMP   #X'08-X'20    ; TEST IF BS
5B6C F012               BEQ   SDTXCL        ; JUMP IF SO
5B6E C9EC               CMP   #X'0C-X'20    ; TEST IF FF
5B70 F01E               BEQ   SDTXFF        ; JUMP IF SO
5B72 4CF85B             JMP   SDTXRT        ; GO RETURN IF UNRECOGNIZABLE CONTROL

5B75 201A5C SDTXCR:     JSR   CSRCLR        ; CARRIAGE RETURN, FIRST CLEAR CURSOR
5B78 A900               LDA   #0            ; ZERO CURSOR HORIZONTAL POSITION
5B7A 8D045B             STA   CSRX
5B7D 4CF85B             JMP   SDTXRT        ; GO SET CURSOR AND RETURN

5B80 201A5C SDTXCL:     JSR   CSRCLR        ; CURSOR LEFT, FIRST CLEAR CURSOR
5B83 AD045B             LDA   CSRX          ; GET CURSOR HORIZONTAL POSITION
5B86 C900               CMP   #0            ; TEST IF AGAINST LEFT EDGE
5B88 F003               BEQ   SDTX20        ; SKIP UPDATE IF SO
5B8A CE045B             DEC   CSRX          ; OTHERWISE DECREMENT CURSOR X POSITION
5B8D 4CF85B SDTX20:     JMP   SDTXRT        ; GO SET CURSOR AND RETURN

5B90 AD065B SDTXFF:     LDA   VMORG         ; FORM FEED, CLEAR SCREEN TO ZEROES
5B93 85ED               STA   ADP2+1        ; TRANSFER VISIBLE MEMORY ORIGIN ADDRESS
5B95 A900               LDA   #0            ; TO ADP2
5B97 85EC               STA   ADP2
5B99 A940               LDA   #NLOC&X'FF    ; SET COUNT OF LOCATIONS TO CLEAR IN DCNT1
5B9B 8D015B             STA   DCNT1
5B9E A91F               LDA   #NLOC/256
5BA0 8D025B             STA   DCNT1+1
5BA3 20015D             JSR   FCLR          ; CLEAR THE SCREEN
5BA6 A900               LDA   #0
5BA8 8D045B             STA   CSRX          ; PUT CURSOR IN UPPER LEFT CORNER
5BAB 8D055B             STA   CSRY
5BAE 4CF85B             JMP   SDTXRT        ; GO SET CURSOR AND RETURN

5BB1 201A5C SDTXLF:     JSR   CSRCLR        ; LINE FEED, FIRST CLEAR CURSOR
5BB4 AD055B             LDA   CSRY          ; GET CURRENT LINE POSITION
5BB7 C915               CMP   #NLIN-1       ; TEST IF AY BOTTOM OF SCREEN
5BB9 1005               BPL   SDTX40        ; GO SCROLL IF SO
5BBB EE055B             INC   CSRY          ; INCREMENT LINE NUMBER IF NOT AT BOTTOM
5BBE D038               BNE   SDTXRT        ; GO INSERT CURSOR AND RETURN
```

```
5BC0 A900      SDTX40:  LDA    #0            ; SET UP ADDRESS POINTERS FOR MOVE
5BC2 85EC               STA    ADP2          ; ADP1 - SOURCE FOR MOVE = FIRST BYTE OF
5BC4 AD065B             LDA    VMORG         ; SECOND LINE OF TEXT
5BC7 85ED               STA    ADP2+1        ; ADP2 = DESTINATION FOR MOVE = FIRST BYTE
5BC9 18                 CLC                  ; IN VISIBLE MEMORY
5BCA 6901               ADC    #CHHI*40/256
5BCC 85EB               STA    ADP1+1
5BCE A968               LDA    #CHHI*40&X'FF
5BD0 85EA               STA    ADP1
5BD2 A988               LDA    #NSCRL&X'FF   ; SET NUMBER OF LOCATIONS TO MOVE
5BD4 8D015B             STA    DCNT1         ; LOW PART
5BD7 A91D               LDA    #NSCRL/256    ; HIGH PART
5BD9 8D025B             STA    DCNT1+1
5BDC 20D35C             JSR    FMOVE         ; EXECUTE MOVE USING AN OPTIMIZED, HIGH
                                             ; SPEED MEMORY MOVE ROUTINE

                                             ; CLEAR LAST LINE OF TEXT
5BDF A988               LDA    #NLIN-1*CHHI*40&X'FF  ; SET ADDRESS POINTER
5BE1 85EC               STA    ADP2          ; LOW BYTE
5BE3 A91D               LDA    #NLIN-1*CHHI*40/256
5BE5 18                 CLC
5BE6 6D065B             ADC    VMORG
5BE9 85ED               STA    ADP2+1        ; HIGH BYTE
5BEB A9B8               LDA    #NCLR&X'FF    ; SET LOW BYTE OF CLEAR COUNT
5BED 8D015B             STA    DCNT1
5BF0 A901               LDA    #NCLR/256     ; SET HIGH BYTE OF CLEAR COUNT
5BF2 8D025B             STA    DCNT1+1
5BF5 20015D             JSR    FCLR          ; CLEAR THE DESIGNATED AREA

               ;        NO EFFECTIVE CHANGE IN CURSOR POSITION

5BF8 20125C    SDTXRT:  JSR    CSRSET        ; RETURN SEQUENCE, INSERT CURSOR
5BFB 68                 PLA                  ; RESTORE REGISTERS FROM THE STACK
5BFC A8                 TAY
5BFD 68                 PLA
5BFE AA                 TAX
5BFF 68                 PLA
5C00 60                 RTS                  ; RETURN
```

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

5C01 20355C         CSRBAD:  JSR    CSRTAD       ; COMPUTE ADDRESS OF TOP OF CHARACTER CELL
                                                 ; FIRST
5C04 A5EC                    LDA    ADP2         ; ADD 320 TO RESULT = 8 SCAN LINES
5C06 18                      CLC
5C07 6940                    ADC    #320&X'FF
5C09 85EC                    STA    ADP2
5C0B A5ED                    LDA    ADP2+1
5C0D 6901                    ADC    #320/256
5C0F 85ED                    STA    ADP2+1
5C11 60                      RTS

                    ;        SET CURSOR AT CURRENT POSITION

5C12 20015C         CSRSET:  JSR    CSRBAD       ; GET BYTE AND BIT ADDRESS OF CURSOR
5C15 A9F8                    LDA    #X'F8        ; DATA = UNDERLINE CURSOR
5C17 4C805C         CSRST1:  JMP    MERGE        ; MERGE CURSOR WITH GRAPHIC MEMORY
                                                 ; AND RETURN

                    ;        CLEAR CURSOR AT CURRENT POSITION

5C1A 20015C         CSRCLR:  JSR    CSRBAD       ; GET BYTE AND BIT ADDRESS OF CURSOR
5C1D A900                    LDA    #0           ; DATA = BLANK DOT ROW
5C1F 4C805C                  JMP    MERGE        ; REMOVE DOT ROW FROM GRAPHIC MEMORY
                                                 ; AND RETURN

                    ;        SHIFT ADP2 LEFT ONE BIT POSITION

5C22 06EC           SADP2L:  ASL    ADP2
5C24 26ED                    ROL    ADP2+1
5C26 60                      RTS

                    ;        MOVE DOWN ONE SCAN LINE      DOUBLE ADDS 40 TO ADP2

5C27 A5EC           DN1SCN:  LDA    ADP2         ; ADD 40 TO LOW BYTE
5C29 18                      CLC
5C2A 6928                    ADC    #40
5C2C 85EC                    STA    ADP2
5C2E A900                    LDA    #0           ; EXTEND CARRY TO UPPER BYTE
5C30 65ED                    ADC    ADP2+1
5C32 85ED                    STA    ADP2+1
5C34 60                      RTS                 ; RETURN

                    ;        COMPUTE BYTE ADDRESS CONTAINING FIRST SCAN LINE OF
                    ;        CHARACTER AT CURSOR POSITION AND PUT IN ADP2
                    ;        BIT ADDRESS (BIT 0 IS LEFTMOST) AT BTPT
                    ;        BYTE ADDRESS =VMORG*256+CHHI*40*CSRY+INT(CSRX*6/8)
                    ;        SINCE CHHI IS A CONSTANT 9, THEN CHHI*40=360
                    ;        BIT ADDRESS=REM(CSRX*5/8)
```

```

5C35 A900      CSRTAD:  LDA    #0           ; AERO UPPER ADP2
5C37 85ED               STA    ADP2+1
5C39 AD055B             LDA    CSRY         ; FIRST COMPUTE 360*CSRY
5C3C 0A                 ASLA                ;   COMPUTE 9*CSRY DIRECTLY IN A
5C3D 0A                 ASLA
5C3E 0A                 ASLA
5C3F 6D055B             ADC    CSRY
5C42 85EC               STA    ADP2         ;   STORE 9*CSRY IN LOWER ADP2
5C44 20225C             JSR    SADP2L       ;   18*CSRY IN ADP2
5C47 20225C             JSR    SADP2L       ;   36*CSRY IN ADP2
5C4A 65EC               ADC    ADP2         ;   ADD IN 9*CSRY TO MAKE 45*CSRY
5C4C 85EC               STA    ADP2
5C4E A900               LDA    #0
5C50 65ED               ADC    ADP2+1
5C52 85ED               STA    ADP2+1       ;   45*CSRY IN ADP2
5C54 20225C             JSR    SADP2L       ;   90*CSRY IN ADP2
5C57 20225C             JSR    SADP2L       ;   180*CSRY IN ADP2
5C5A 20225C             JSR    SADP2L       ;   360*CSRY IN ADP2
5C5D AD045B             LDA    CSRX         ; NEXT COMPUTE 6*CSRX WHICH IS A 9 BIT
5C60 0A                 ASLA                ; VALUE
5C61 6D045B             ADC    CSRX
5C64 0A                 ASLA
5C65 8D005B             STA    BTPT         ;   SAVE RESULT TEMPORARILY
5C68 6A                 RORA                ;   DIVIDE BY 8 AND TRUNCATE FOR INT
5C69 4A                 LSRA                ;   FUNCTION
5C6A 4A                 LSRA                ;   NOW HAVE INT(CSRX*6/8)
5C6B 18                 CLC                 ; DOUBLE ADD TO ADP2
5C6C 65EC               ADC    ADP2
5C6E 85EC               STA    ADP2
5C70 A5ED               LDA    ADP2+1
5C72 6D065B             ADC    VMORG        ; ADD IN VMORG*256
5C75 85ED               STA    ADP2+1       ; FINISHED WITH ADP2
5C77 AD005B             LDA    BTPT         ; COMPUTE REM(CSRX*6/8) WHICH IS LOW 3
5C7A 2907               AND    #7           ; BITS OF CSRX*6
5C7C 8D005B             STA    BTPT         ; KEEP IN BTPT
5C7F 60                 RTS                 ; FINISHED

               ;        MERGE A ROW OF 5 DOTS WITH GRAPHIC MEMORY STARTING AT BYTE
               ;        ADDRESS AND BIT NUMBER IN ADP2 AND BTPT
               ;        5 DOTS TO MERGE LEFT JUSTIFIED IN A
               ;        PRESERVES X AND Y

5C80 8D035B    MERGE:   STA    MRGT1        ; SAVE INPUT DATA
5C83 98                 TYA                 ; SAVE Y
5C84 48                 PHA
5C85 AC005B             LDY    BTPT         ; OPEN UP A 5 BIT WINDOW IN GRAPHIC MEMORY
5C88 B9C35C             LDA    MERGT, Y     ; LEFT BITS
5C8B A000               LDY    #0           ; ZERO Y
5C8D 31EC               AND    (ADP2),Y
5C8F 91EC               STA    (ADP2),Y
5C91 AC005B             LDY    BTPT
5C94 B9CB5C             LDA    MERGT+8,Y    ; RIGHT BITS
5C97 A001               LDY    #1
5C99 31EC               AND    (ADP2),Y
```

```
5C9B 91EC                STA    (ADP2),Y
5C9D AD035B              LDA    MRGT1          ; SHIFT DATA RIGHT TO LINE UP LEFTMOST
5CA0 AC005B              LDY    BTPT           ; DATA BIT WITH LEFTMOST GRAPHIC FIELD
5CA3 F004                BEQ    MERGE2         ; SHIFT BTPT TIMES
5CA5 4A        MERGE1:   LSRA
5CA6 88                  DEY
5CA7 D0FC                BNE    MERGE1
5CA9 11EC      MERGE2:   ORA    (ADP2),Y       ; OVERLAY WITH GRAPHIC MEMORY
5CAB 91EC                STA    (ADP2),Y
5CAD A908                LDA    #8             ; SHIFT DATA LEFT TO LINE UP RIGHTMOST
5CAF 38                  SEC                   ; DATA BIT WITH RIGHTMOST GRAPHIC FIELD
5CB0 ED005B              SBC    BTPT           ; SHIFT (8-BTPT) TIMES
5CB3 A8                  TAY
5CB4 AD035B              LDA    MRGT1
5CB7 0A        MERGE3:   ASLA
5CB8 88                  DEY
5CB9 D0FC                BNE    MERGE3
5CBB C8                  INY
5CBC 11EC                ORA    (ADP2),Y       ; OVERLAY WITH GRAPHIC MEMORY
5CBE 91EC                STA    (ADP2),Y
5CC0 68                  PLA                   ; RESTORE y
5CC1 A8                  TAY
5CC2 60                  RTS                   ; RETURN

5CC3 0783C1E0  MERGT:     .BYTE  X'07,X'83,X'C1,X'E0  ; TABLE OF MASKS FOR OPENING UP
5CC7 F0F8FCFE             .BYTE  X'F0,X'F8,X'FC,X'FE  ; A 5 BIT WINDOW ANYWHERE
5CCB FFFFFFFF             .BYTE  X'FF,X'FF,X'FF,X'FF  ; IN GRAPHIC MEMORY
5CCF 7F3F1F0F             .BYTE  X'7F,X'3F,X'1F,X'0F

               ;         FAST MEMORY MOVE ROUTINE
               ;         ENTER WITH SOURCE ADDRESS IN ADPT1 AND DESTINATION ADDRESS IN
               ;         ADPT2 AND MOVE COUNT (DOUBLE PRECISION) IN DCNT1.
               ;         MOVE PROCEEDS FROM LOW TO HIGH ADDRESSES AT APPROXIMATELY 16US
               ;         PER BYTE.
               ;         EXIT WITH ADDRESS POINTERS AND COUNT IN UNKNOWN STATE.
               ;         PRESERVES X AND Y REGISTERS.

5CD3 8A        FMOVE:    TXA                   ; SAVE X AND Y ON THE STACK
5CD4 48                  PHA
5CD5 98                  TYA
5CD6 48                  PHA
5CD7 CE025B    FMOVE1:   DEC    DCNT1+1        ; TEST IF LESS THAN 256 LEFT TO MOVE
5CDA 3015                BMI    FMOVE3         ; JUMP TO FINAL MOVE IF SO
5CDC A000                LDY    #0             ; MOVE A BLOCK OF 256 BYTES QUICKLY
5CDE B1EA      FMOVE2:   LDA    (ADP1),Y       ; TWO BYTES AT A TIME
5CE0 91EC                STA    (ADP2),Y
5CE2 C8                  INY
5CE3 B1EA                LDA    (ADP1),Y
5CE5 91EC                STA    (ADP2),Y
5CE7 C8                  INY
5CE8 D0F4                BNE    FMOVE2         ; CONTINUE UNTIL DONE
5CEA E6EB                INC    ADP1+1         ; BUMP ADDRESS POINTERS TO NEXT PAGE
5CEC E6ED                INC    ADP2+1
5CEE 4CD75C              JMP    FMOVE1         ; GO MOVE NEXT PAGE
5CF1 AE015B    FMOVE3:   LDX    DCNT1          ; GET REMAINING BYTE COUNT INTO X
```

```
5CF4 B1EA      FMOVE4:  LDA    (ADP1),Y     ; MOVE A BYTE
5CF6 91EC               STA    (ADP2),Y
5CF8 C8                 INY
5CF9 CA                 DEX
5CFA D0F8               BNE    FMOVE4       ; CONTINUE UNTIL DONE
5CFC 68                 PLA                 ; RESTORE INDEX REGISTERS
5CFD A8                 TAY
5CFE 68                 PLA
5CFF AA                 TAX
5D00 60                 RTS                 ; AND RETURN

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

5D01 98        FCLR:    TYA                 ; SAVE Y
5D02 48                 PHA
5D03 A000      FCLR1:   LDY    #0
5D05 CE025B             DEC    DCNT1+1      ; TEST IF LESS THAN 256 LEFT TO MOVE
5D08 300B               BMI    FCLR3        ; JUMP INTO FINAL CLEAR IF SO
5D0A 98                 TYA                 ; CLEAR A BLOCK OF 256 QUICKLY
5D0B 91EC      FCLR2:   STA    (ADP2),Y     ; CLEAR A BYTE
5D0D C8                 INY
5D0E D0FB               BNE    FCLR2
5D10 E6ED               INC    ADP2+1       ; BUMP ADDRESS POINTER TO NEXT PAGE
5D12 4C035D             JMP    FCLR1        ; GO CLEAR NEXT PAGE
5D15 98        FCLR3:   TYA                 ; CLEAR REMAINING PARTIAL PAGE
5D16 91EC      FCLR4:   STA    (ADP2),Y
5D18 C8                 INY
5D19 CE015B             DEC    DCNT1
5D1C D0F8               BNE    FCLR4
5D1E 68                 PLA                 ; RESTORE Y
5D1F A8                 TAY
5D20 60                 RTS                 ; RETURN
```

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

5D21 000000    CHTB:    .BYTE       X'00,X'00,X'00    ; BLANK
5D24 00000000           .BYTE  X'00,X'00,X'00,X'00
5D28 202020             .BYTE       X'20,X'20,X'20    ; !
5D2B 20200020           .BYTE  X'20,X'20,X'00,X'20
5D2F 505050             .BYTE       X'50,X'50,X'50    ; "
5D32 00000000           .BYTE  X'00,X'00,X'00,X'00
5D36 5050F8             .BYTE       X'50,X'50,X'F8    ; #
5D39 50F85050           .BYTE  X'50,X'F8,X'50,X'50
5D3D 2078A0             .BYTE       X'20,X'78,X'A0    ; X'
5D40 7028F020           .BYTE  X'70,X'28,X'F0,X'20
5D44 C8C810             .BYTE       X'C8,X'C8,X'10    ; %
5D47 20409898           .BYTE  X'20,X'40,X'98,X'98
5D4B 40A0A0             .BYTE       X'40,X'A0,X'A0    ; &
5D4E 40A89068           .BYTE  X'40,X'A8,X'90,X'68
5D52 303030             .BYTE       X'30,X'30,X'30    ; '
5D55 00000000           .BYTE  X'00,X'00,X'00,X'00
5D59 204040             .BYTE       X'20,X'40,X'40    ; (
5D5C 40404020           .BYTE  X'40,X'40,X'40,X'20
5D60 201010             .BYTE       X'20,X'10,X'10    ; )
5D63 10101020           .BYTE  X'10,X'10,X'10,X'20
5D67 20A870             .BYTE       X'20,X'A8,X'70    ; *
5D6A 2070A820           .BYTE  X'20,X'70,X'A8,X'20
5D6E 002020             .BYTE       X'00,X'20,X'20    ; +
5D71 F8202000           .BYTE  X'F8,X'20,X'20,X'00
5D75 000000             .BYTE       X'00,X'00,X'00    ; ,
5D78 30301020           .BYTE  X'30,X'30,X'10,X'20
5D7C 000000             .BYTE       X'00,X'00,X'00    ; -
5D7F F8000000           .BYTE  X'F8,X'00,X'00,X'00
5D83 000000             .BYTE       X'00,X'00,X'00    ; .
5D86 00003030           .BYTE  X'00,X'00,X'30,X'30
5D8A 080810             .BYTE       X'08,X'08,X'10    ; /
5D8D 20408080           .BYTE  X'20,X'40,X'80,X'80
5D91 609090             .BYTE       X'60,X'90,X'90    ; 0
5D94 90909060           .BYTE  X'90,X'90,X'90,X'60
5D98 206020             .BYTE       X'20,X'60,X'20    ; 1
5D9B 20202070           .BYTE  X'20,X'20,X'20,X'70
5D9F 708810             .BYTE       X'70,X'88,X'10    ; 2
5DA2 204080F8           .BYTE  X'20,X'40,X'80,X'F8
5DA6 708808             .BYTE       X'70,X'88,X'08    ; 3
5DA9 30088870           .BYTE  X'30,X'08,X'88,X'70
5DAD 103050             .BYTE       X'10,X'30,X'50    ; 4
5DB0 90F81010           .BYTE  X'90,X'F8,X'10,X'10
5DB4 F880F0             .BYTE       X'F8,X'80,X'F0    ; 5
5DB7 080808F0           .BYTE  X'08,X'08,X'08,X'F0
5DBB 708080             .BYTE       X'70,X'80,X'80    ; 6
5DBE F0888870           .BYTE  X'F0,X'88,X'88,X'70
```

```
  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 5DDE 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 5DEC 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    ; GREATER THAN
  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 5E0F 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 F08080F8           .BYTE  X'F0,X'80,X'80,X'F8
  491 5E2B F88080             .BYTE       X'F8,X'80,X'80    ; F
  492 5E2E F0808080           .BYTE  X'F0,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 8890A0             .BYTE       X'88,X'90,X'A0    ; K
  502 5E51 C0A09088           .BYTE  X'C0,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
```

```
5E82 F0A09088           .BYTE  X'F0,X'A0,X'90,X'88
5E86 788080             .BYTE       X'78,X'80,X'80    ; S
5E89 700808F0           .BYTE  X'70,X'08,X'08,X'F0
5E8D F82020             .BYTE       X'F8,X'20,X'20    ; T
5E90 20202020           .BYTE  X'20,X'20,X'20,X'20
5E94 888888             .BYTE       X'88,X'88,X'88    ; U
5E97 88888870           .BYTE  X'88,X'88,X'88,X'70
5E9B 888888             .BYTE       X'88,X'88,X'88    ; V
5E9E 50502020           .BYTE  X'50,X'50,X'20,X'20
5EA2 888888             .BYTE       X'88,X'88,X'88    ; W
5EA5 A8A8D888           .BYTE  X'A8,X'A8,X'D8,X'88
5EA9 888850             .BYTE       X'88,X'88,X'50    ; X
5EAC 20508888           .BYTE  X'20,X'50,X'88,X'88
5EB0 888850             .BYTE       X'88,X'88,X'50    ; Y
5EB3 20202020           .BYTE  X'20,X'20,X'20,X'20
5EB7 F80810             .BYTE       X'F8,X'08,X'10    ; Z
5EBA 204080F8           .BYTE  X'20,X'40,X'80,X'F8
5EBE 704040             .BYTE       X'70,X'40,X'40    ; LEFT BRACKET
5EC1 40404070           .BYTE  X'40,X'40,X'40,X'70
5EC5 808040             .BYTE       X'80,X'80,X'40    ; BACKSLASH
5EC8 20100808           .BYTE  X'20,X'10,X'08,X'08
5ECC 701010             .BYTE       X'70,X'10,X'10    ; RIGHT BRACKET
5ECF 10101070           .BYTE  X'10,X'10,X'10,X'70
5ED3 205088             .BYTE       X'20,X'50,X'88    ; CARROT
5ED6 00000000           .BYTE  X'00,X'00,X'00,X'00
5EDA 000000             .BYTE       X'00,X'00,X'00    ; UNDERLINE
5EDD 000000F8           .BYTE  X'00,X'00,X'00,X'F8
5EE1 C06030             .BYTE       X'C0,X'60,X'30    ; GRAVE ACCENT
5EE4 00000000           .BYTE  X'00,X'00,X'00,X'00
5EE8 000020             .BYTE       X'00,X'00,X'20    ; A (LC)
5EEB 5088F888           .BYTE  X'50,X'88,X'F8,X'88
5EEF 0000F0             .BYTE       X'00,X'00,X'F0    ; B (LC)
5EF2 487048F0           .BYTE  X'48,X'70,X'48,X'F0
5EF6 000078             .BYTE       X'00,X'00,X'78    ; C (LC)
5EF9 80808078           .BYTE  X'80,X'80,X'80,X'78
5EFD 0000F0             .BYTE       X'00,X'00,X'F0    ; D (LC)
5F00 484848F0           .BYTE  X'48,X'48,X'48,X'F0
5F04 0000F8             .BYTE       X'00,X'00,X'F8    ; E (LC)
5F07 80E080F8           .BYTE  X'80,X'E0,X'80,X'F8
5F0B 0000F8             .BYTE       X'00,X'00,X'F8    ; F (LC)
5F0E 80E08080           .BYTE  X'80,X'E0,X'80,X'80
5F12 000078             .BYTE       X'00,X'00,X'78    ; G (LC)
5F15 80988878           .BYTE  X'80,X'98,X'88,X'78
5F19 000088             .BYTE       X'00,X'00,X'88    ; H (LC)
5F1C 88F88888           .BYTE  X'88,X'F8,X'88,X'88
5F20 000070             .BYTE       X'00,X'00,X'70    ; I (LC)
5F23 20202070           .BYTE  X'20,X'20,X'20,X'70
5F27 000038             .BYTE       X'00,X'00,X'38    ; J (LC)
5F2A 10105020           .BYTE  X'10,X'10,X'50,X'20
5F2E 000090             .BYTE       X'00,X'00,X'90    ; K (LC)
5F31 A0C0A090           .BYTE  X'A0,X'C0,X'A0,X'90
5F35 000080             .BYTE       X'00,X'00,X'80    ; L (LC)
5F38 808080F8           .BYTE  X'80,X'80,X'80,X'F8
5F3C 000088             .BYTE       X'00,X'00,X'88    ; M (LC)
5F3F D8A88888           .BYTE  X'D8,X'A8,X'88,X'88
```

```
5F43 000088              .BYTE       X'00,X'00,X'88    ; N (LC)
5F46 C8A89888            .BYTE  X'C8,X'A8,X'98,X'88
5F4A 000070              .BYTE       X'00,X'00,X'70    ; O (LC)
5F4D 88888870            .BYTE  X'88,X'88,X'88,X'70
5F51 0000F0              .BYTE       X'00,X'00,X'F0    ; P (LC)
5F54 88F08080            .BYTE  X'88,X'F0,X'80,X'80
5F58 000070              .BYTE       X'00,X'00,X'70    ; Q (LC)
5F5B 88A89068            .BYTE  X'88,X'A8,X'90,X'68
5F5F 0000F0              .BYTE       X'00,X'00,X'F0    ; R (LC)
5F62 88F0A090            .BYTE  X'88,X'F0,X'A0,X'90
5F66 000078              .BYTE       X'00,X'00,X'78    ; S (LC)
5F69 807008F0            .BYTE  X'80,X'70,X'08,X'F0
5F6D 0000F8              .BYTE       X'00,X'00,X'F8    ; T (LC)
5F70 20202020            .BYTE  X'20,X'20,X'20,X'20
5F74 000088              .BYTE       X'00,X'00,X'88    ; U (LC)
5F77 88888870            .BYTE  X'88,X'88,X'88,X'70
5F7B 000088              .BYTE       X'00,X'00,X'88    ; V (LC)
5F7E 88885020            .BYTE  X'88,X'88,X'50,X'20
5F82 000088              .BYTE       X'00,X'00,X'88    ; W (LC)
5F85 88A8D888            .BYTE  X'88,X'A8,X'D8,X'88
5F89 000088              .BYTE       X'00,X'00,X'88    ; X (LC)
5F8C 50205088            .BYTE  X'50,X'20,X'50,X'88
5F90 000088              .BYTE       X'00,X'00,X'88    ; Y (LC)
5F93 50202020            .BYTE  X'50,X'20,X'20,X'20
5F97 0000F8              .BYTE       X'00,X'00,X'F8    ; Z (LC)
5F9A 102040F8            .BYTE  X'10,X'20,X'40,X'F8
5F9E 102020              .BYTE       X'10,X'20,X'20    ; LEFT BRACE
5FA1 60202010            .BYTE  X'60,X'20,X'20,X'10
5FA5 202020              .BYTE       X'20,X'20,X'20    ; VERTICAL BAR
5FA8 20202020            .BYTE  X'20,X'20,X'20,X'20
5FAC 402020              .BYTE       X'40,X'20,X'20    ; RIGHT BRACE
5FAF 30202040            .BYTE  X'30,X'20,X'20,X'40
5FB3 10A840              .BYTE       X'10,X'A8,X'40    ; TILDA
5FB6 00000000            .BYTE  X'00,X'00,X'00,X'00
5FBA A850A8              .BYTE       X'A8,X'50,X'A8    ; RUBOUT
5FBD 50A850A8            .BYTE  X'50,X'A8,X'50,X'A8

0000                     .END
NO ERROR LINES
```

```
                           .PAGE 'DOCUMENTATION, EQUATES, STORAGE'

                  ;        THIS PACKAGE PROVIDES FUNDAMENTAL GRAPHICS ORIENTED
                  ;        SUBROUTINES NEEDED FOR EFFECTIVE USE OF THE VISIBLE MEMORY AS
                  ;        A GRAPHIC DISPLAY DEVICE.  MAJOR SUBROUTINES INCLUDED ARE AS
                  ;        FOLLOWS:
                  ;           CLEAR - CLEARS THE ENTIRE VISIBLE MEMORY AS DEFINED BY
                  ;                   NPIX/8
                  ;           PIXADR- RETURNS BYTE AND BIT ADDRESS OF PIXEL AT X1CORD,
                  ;                   Y1CORD
                  ;           CKCRD1- PERFORM A RANGE CHECK ON X1CORD,Y1CORD
                  ;           CKCRD2- PERFORM A RANGE CHECK ON X2CORD,Y2CORD
                  ;           STPIX - SET PIXEL AT X1CORD,Y1CORD TO A ONE (WHITE DOT)
                  ;           CLPIX - CLEAR PIXEL AT X1CORD,Y1CORD TO ZERO (BLACK DOT)
                  ;           FLPIX - FLIP THE PIXEL AT X1CORD,Y1CORD
                  ;           WRPIX - UPDATE PIXEL AT X1CORD,Y1CORD ACCORDING TO THE
                  ;                   STATE OF THE ACCUMULATOR
                  ;           RDPIX - COPY THE STATE OF THE PIXEL AT X1CORD,Y1CORD INTO
                  ;                   THE ACCUMULATOR
                  ;           DRAW -  DRAW THE BEST STRAIGHT LINE FROM X1CORD,Y1CORD
                  ;                   TO X2CORD,Y2CORD. X2CORD,Y2CORD COPIED TO
                  ;                   X1CORD,Y1CORD AFTER DRAWING
                  ;           ERASE - SAME AS DRAW EXCEPT A BLACK LINE IS DRAWN
                  ;           DCHAR - DISPLAYS A CHARACTER WHOSE UPPER LEFT CORNER IS
                  ;                   X1CORD,Y1CORD. CHARACTER MATRIX IS 5 WIDE BY 9
                  ;                   HIGH INCLUDING LOWER CASE DESCENDERS BUT NOT
                  ;                   INCLUDING CHARACTER AND LINE SPACING.
                  ;           DTEXT - ACCEPTS ASCII CHARACTERS AND FORMATS THEM INTO
                  ;                   TEXT. A STANDARD (BUT EASILY MODIFIED) CHARACTER
                  ;                   FIELD 6 WIDE BY 11 HIGH ALLOWS UP TO 18 LINES OF 53
                  ;                   CHARACTERS. SUBSCRIPT AND SUPERSCRIPT VIA CONTROL
                  ;                   CHARACTERS IS IMPLEMENTED.
                  ;           DTXTIN- INITIALIZE PARAMETERS FOR USE OF DTEXT ON FULL
                  ;                   SCREEN.
                  ;
                  ;        ALL SUBROUTINES DEPEND ON ONE OR TWO PAIRS OF COORDINATES.
                  ;        EACH COORDINATE IS A DOUBLE PRECISION, UNSIGNED NUMBER WITH
                  ;        THE LOW BYTE FIRST (I.E.  LIKE MEMORY ADDRESSES IN THE 6502)
                  ;        THE ORIGIN OF THE COORDINATE SYSTEM IS AT THE LOWER LEFT
                  ;        CORNER OF THE SCREEN THEREFORE THE ENITRE SCREEN IS IN THE
                  ;        FIRST QUADRANT.  ALLOWABLE RANGE OF THE X COORDINATE IS 0 TO
                  ;        319 (DECIMAL) AND THE RANGE OF THE Y COORDINATE IS 0 TO 199.
                  ;        FOR MAXIMUM SPEED ALL SUBROUTINES ASSUME THAT THE COORDINATE
                  ;        VALUES ARE IN RANGE.  IF THEY ARE NOT, WILD STORING INTO ANY
                  ;        PART OF KIM RAM IS POSSIBLE.  FOR DEBUGGING, CALLS TO CKCRD1
                  ;        AND CKCRD2 SHOULD BE PERFORMED PRIOR TO GRAPHIC ROUTINE CALLS
                  ;        IN ORDER TO DETECT AND CORRECT ERRONEOUS COORDINATE VALUES.

                  ;        GENERAL EQUATES

   0140           NX       =      320          ; NUMBER OF BITS IN A ROW
   00C8           NY       =      200          ; NUMBER OF ROWS  (CHANGE FOR HALF SCREEN
                                               ; OPERATION)
   FA00           NPIX     =      NX*NY        ; NUMBER OF PIXELS
   000B           CHHIW    =      11           ; HEIGHT OF CHARACTER WINDOW
```

```
0006           CHWIDW    =      6            ; WIDTH OF CHARACTER WINDOW
0009           CHHIM     =      9            ; HEIGHT OF CHARACTER MATRIX
0005           CHWIDM    =      5            ; WIDTH OF CHARACTER MATRIX

               ;         BASE PAGE TEMPORARY STORAGE (MAY BE DESTROYED BETWEEN CALLS)

0000                     .=     X'EA

00EA           ADP1:     .=.+   2            ; ADDRESS POINTER 1
00EC           ADP2:     .=.+   2            ; ADDRESS POINTER 2

               ;         PERMANENT RAM STORAGE  (MUST BE PRESERVED BETWEEN CALLS)
               ;******* THESE PARAMETERS MUST BE SET BEFORE USING GRAPHIC ************
               ;******************* ROUTINES THAT REFERENCE THEM *********************

00EE                     .=     X'100        ; PUT IN STACK AREA FOR CONVENIENCE

0100           VMORG:    .=.+   1            ; PAGE NUMBER OF FIRST VISIBLE MEMORY
                                             ; LOCATION
0101           X1CORD:   .=.+   2            ; COORDINATE PAIR 1 AND CURSOR LOCATION
0103           Y1CORD:   .=.+   2
0105           X2CORD:   .=.+   2            ; COORDINATE PAIR 2
0107           Y2CORD:   .=.+   2
0109           TMAR:     .=.+   2            ; TOP MARGIN FOR DTEXT
010B           BMAR:     .=.+   2            ; BOTTOM MARGIN FOR DTEXT
010D           LMAR:     .=.+   2            ; LEFT MARGIN FOR DTEXT
010F           RMAR:     .=.+   2            ; RIGHT MARGIN FOR DTEXT

               ;         GENERAL TEMPORARY STORAGE (CAN BE DESTROYED BETWEEN CALLS)

0111           BTPT:     .=.+   1            ; BIT NUMBER
0112           DELTAX:   .=.+   2            ; DELTA X FOR LINE DRAW
0114           DELTAY:   .=.+   2            ; DELTA Y FOR LINE DRAW
0116           ACC:      .=.+   2            ; ACCUMULATOR FOR LINE DRAW
0118           XDIR:     .=.+   1            ; X MOVEMENT DIRECTION, ZERO=+
0119           YDIR:     .=.+   1            ; Y MOVEMENT DIRECTION, ZERO=+
011A           XCHFLG:   .=.+   1            ; EXCHANGE X AND Y FLAG, EXCHANGE IF NOT 0
011B           COLOR:    .=.+   1            ; COLOR OF LINE DRAWN -1=WHITE
011C           TEMP:     .=.+   2            ; TEMPORARY STORAGE
0112           TLBYT     =      DELTAX       ; TOP LEFT BYTE ADDRESS FOR TEXT WINDOW
0118           TLBIT     =      XDIR         ; TOP LEFT BIT ADDRESS FOR TEXT WINDOW
0114           TRBYT     =      DELTAY       ; TOP RIGHT BYTE ADDRESS FOR TEXT WINDOW
0119           TRBIT     =      YDIR         ; TOP RIGHT BIT ADDRESS FOR TEXT WINDOW
0116           BRBYT     =      ACC          ; BOTTOM RIGHT BYTE ADDRESS FOR TXT WINDOW
```

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

011E                          .=     X'5500        ; PUT AT END OF 16K EXPANSION

5500 A000           CLEAR:    LDY    #0            ; INITIALIZE ADDRESS POINTER
5502 84EA                     STY    ADP1          ; AND ZERO INDEX Y
5504 AD0001                   LDA    VMORG
5507 85EB                     STA    ADP1+1
5509 18                       CLC                  ; COMPUTE END ADDRESS
550A 691F                     ADC    #NPIX/8/256
550C AA                       TAX                  ; KEEP IT IN X
550D 98             CLEAR1:   TYA                  ; CLEAR A BYTE
550E 91EA                     STA    (ADP1),Y
5510 E6EA                     INC    ADP1          ; INCREMENT ADDRESS POINTER
5512 D002                     BNE    CLEAR2
5514 E6EB                     INC    ADP1+1
5516 A5EA           CLEAR2:   LDA    ADP1          ; TEST IF DONE
5518 C940                     CMP    #NPIX/8&X'FF
551A D0F1                     BNE    CLEAR1        ; LOOP IF NOT
551C E4EB                     CPX    ADP1+1
551E D0ED                     BNE    CLEAR1        ; LOOP IF NOT
5520 60                       RTS                  ; RETURN

```

```
                         .PAGE  'PIXADR - BYTE AND BIT ADDRESS OF A PIXEL'
                ;        PIXADR - FIND THE BYTE ADDRESS AND BIT NUMBER OF PIXEL AT
                ;                 X1CORD,Y1CORD
                ;        PUTS BYTE ADDRESS IN ADP1 AND BIT MUMBER (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

5521 AD0101     PIXADR:  LDA    X1CORD       ; COMPUTE BIT ADDRESS FIRST
5524 85EA                STA    ADP1         ; ALSO TRANSFER X1CORD TO ADP1
5526 2907                AND    #X'07        ; WHICH IS SIMPLY THE LOW 3 BITS OF X
5528 8D1101              STA    BTPT
552B AD0201              LDA    X1CORD+1     ; FINISH TRANSFERRING X1CORD TO ADP1
552E 85EB                STA    ADP1+1
5530 46EB                LSR    ADP1+1       ; DOUBLE SHIFT ADP1 RIGHT 3 TO GET
5532 66EA                ROR    ADP1         ; INT(XCORD/8)
5534 46EB                LSR    ADP1+1
5536 66EA                ROR    ADP1
5538 46EB                LSR    ADP1+1
553A 66EA                ROR    ADP1
553C A9C7                LDA    #199         ; TRANSFER (199-Y1CORD) TO ADP2
553E 38                  SEC                 ; AND TEMPORARY STORAGE
553F ED0301              SBC    Y1CORD
5542 85EC                STA    ADP2
5544 8D1C01              STA    TEMP
5547 A900                LDA    #0
5549 ED0401              SBC    Y1CORD+1
554C 85ED                STA    ADP2+1
554E 8D1D01              STA    TEMP+1
5551 06EC                ASL    ADP2         ; COMPUTE 40*(199-Y1CORD)
5553 26ED                ROL    ADP2+1       ;  2*(199-Y1CORD)
5555 06EC                ASL    ADP2
5557 26ED                ROL    ADP2+1       ;  4*(199+Y1CORD)
5559 A5EC                LDA    ADP2         ;  ADD IN TEMPORARY SAVE OF (199-Y1CORD)
555B 18                  CLC                 ;  TO MAKE 5*(199-Y1CORD)
555C 6D1C01              ADC    TEMP
555F 85EC                STA    ADP2
5561 A5ED                LDA    ADP2+1
5563 6D1D01              ADC    TEMP+1
5566 85ED                STA    ADP2+1       ;  5*(199-Y1CORD)
5568 06EC                ASL    ADP2         ;  10*(199-Y1CORD)
556A 26ED                ROL    ADP2+1
556C 06EC                ASL    ADP2         ;  20*(199-Y1CORD)
556E 26ED                ROL    ADP2+1
5570 06EC                ASL    ADP2         ;  40*(199-Y1CORD)
5572 26ED                ROL    ADP2+1
5574 A5EC                LDA    ADP2         ; ADD IN INT(X1CORD/8) COMPUTED EARLIER
5576 18                  CLC
5577 65EA                ADC    ADP1
5579 85EA                STA    ADP1
557B A5ED                LDA    ADP2+1
```

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

```

```
                        .PAGE  'INDIVIDUAL PIXEL SUBROUTINES'
               ;        STPIX - SETS THE PIXEL AT X1CORD,Y1CORD TO A ONE (WHITE DOT)
               ;        DOES NOT ALTER X1CORD OR Y1CORD
               ;        PRESERVES X AND Y
               ;        ASSUMES IN RANGE CORRDINATES

5585 202155    STPIX:   JSR    PIXADR       ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
                                            ; INTO ADP1
5588 98                 TYA                 ; SAVE Y
5589 48                 PHA
558A AC1101             LDY    BTPT         ; GET BIT NUMBER IN Y
558D B9EC55             LDA    MSKTB1,Y     ; GET A BYTE WITH THAT BIT =1, OTHERS =0
5590 A000               LDY    #0           ; ZERO Y
5592 11EA               ORA    (ADP1),Y     ; COMBINE THE BIT WITH THE ADDRESSED VM
5594 91EA               STA    (ADP1),Y     ; BYTE
5596 68                 PLA                 ; RESTORE Y
5597 A8                 TAY
5598 60                 RTS                 ; AND RETURN

               ;        CLPIX - CLEARS THE PIXEL AT X1CORD,Y1CORD TO A ZERO (BLACK DOT
               ;        DOES NOT ALTER X1CORD OR Y1CORD
               ;        PRESERVES X AND Y
               ;        ASSUMES IN RANGE COORDINATES

5599 202155    CLPIX:   JSR    PIXADR       ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
                                            ; INTO ADP1
559C 98                 TYA                 ; SAVE Y
559D 48                 PHA
559E AC1101             LDY    BTPT         ; GET BIT NUMBER IN Y
55A1 B9F455             LDA    MSKTB2,Y     ; GET A BYTE WITH THAT BIT =0, OTHERS =1
55A4 A000               LDY    #0           ; ZERO Y
55A6 31EA               AND    (ADP1),Y     ; REMOVE THE BIT FROM THE ADDRESSED VM
55A8 91EA      CLPIX1:  STA    (ADP1),Y     ; BYTE
55AA 68                 PLA                 ; RESTORE Y
55AB A8                 TAY
55AC 60                 RTS                 ; AND RETURN

               ;        FLPIX - FLIPS THE PIXEL AT X1CORD,Y1CORD
               ;        DOES NOT ALTER X1CORD OR Y1CORD
               ;        PRESERVES X AND Y
               ;        ASSUMES IN RANGE COORDINATES

55AD 202155    FLPIX:   JSR    PIXADR       ; GET BYTE ADDRESS AND BIT NUMBER OF PIXEL
                                            ; INTO ADP1
55B0 98                 TYA                 ; SAVE Y
55B1 48                 PHA
55B2 AC1101             LDY    BTPT         ; GET BIT NUMBER IN Y
55B5 B9EC55             LDA    MSKTB1,Y     ; GET A BYTE WITH THAT BIT =1, OTHERS =0
55B8 A000               LDY    #0           ; ZERO Y
55BA 51EA               EOR    (ADP1),Y     ; FLIP THAT BIT IN THE ADDRESSED VM BYTE
55BC 91EA               STA    (ADP1),Y
55BE 68                 PLA                 ; RESTORE Y
55BF A8                 TAY
55C0 60                 RTS                 ; AND RETURN

```

```
               ;        WRPIX - SETS THE PIXEL AT X1CORD,Y1CORD ACCORDING TO THE STATE
               ;        OF BIT 0 (RIGHTMOST) OF A
               ;        DOES NOT ALTER X1CORD OR Y1CORD
               ;        PRESERVES X AND Y AND A
               ;        ASSUMES IN RANGE CORRDINATES

55C1 2CD155    WRPIX:   BIT    WRPIXM       ; TEST LOW BIT OF A
55C4 48                 PHA
55C5 F005               BEQ    WRPIX1       ; JUMP IF A ZERO TO BE WRITTEN
55C7 208555             JSR    STPIX        ; OTHERWISE WRITE A ONE
55CA 68                 PLA                 ; RESTORE A AND RETURN
55CB 60                 RTS
55CC 209955    WRPIX1:  JSR    CLPIX        ; CLEAR THE PIXEL
55CF 68                 PLA                 ; RESTORE A AND RETURN
55D0 60                 RTS

55D1 01        WRPIXM:  .BYTE  1            ; BIT TEST MASK FOR BIT 0

               ;        RDPIX - READS THE PIXEL AT X1CORD,Y1CORD AND SETS A TO ALL
               ;        ZEROES IF IT IS A ZERO OR TO ALL ONES IF IT IS A ONE
               ;        LOW BYTE OF ADP1 IS EQUAL TO A ON RETURN
               ;        DOES NOT ALTER X1CORD OR Y1CORD
               ;        PRESERVES X AND Y
               ;        ASSUMES IN RANGE CORRDINATES

55D2 202155    RDPIX:   JSR    PIXADR       ; GET BYTE AND BIT ADDRESS OF PIXEL
55D5 98                 TYA                 ; SAVE Y
55D6 48                 PHA
55D7 A000               LDY    #0           ; GET ADDRESSED BYTE FROM VM
55D9 B1EA               LDA    (ADP1),Y
55DB AC1101             LDY    BTPT         ; GET BIT NUMBER IN Y
55DE 39EC55             AND    MSKTB1,Y     ; CLEAR ALL BUT ADDRESSED BIT
55E1 F002               BEQ    RDPIX1       ; SKIP AHEAD IF IT WAS A ZERO
55E3 A9FF               LDA    #X'FF        ; SET TO ALL ONES IF IT WAS A ONE
55E5 85EA      RDPIX1:  STA    ADP1         ; SAVE A TEMPORARILY IN ADP1 WHILE
55E7 68                 PLA                 ; RESTORING Y
55E8 A8                 TAY
55E9 A5EA               LDA    ADP1
55EB 60                 RTS                 ; RETURN

               ;        MASK TABLES FOR INDIVIDUAL PIXEL SUBROUTINES
               ;        MSKTB1 IS A TABLE OF 1 BITS CORRESPONDING TO BIT NUMBERS
               ;        MSKTB2 IS A TABLE OF 0 BITS CORRESPONDING TO BIT NUMBERS

55EC 80402010  MSKTB1:  .BYTE  X'80,X'40,X'20,X'10
55F0 08040201           .BYTE  X'08,X'04,X'02,X'01
55F4 7FBFDFEF  MSKTB2:  .BYTE  X'7F,X'BF,X'DF,X'EF
55F8 F7FBFDFE           .BYTE  X'F7,X'FB,X'FD,X'FE

```

```
                        .PAGE  'COORDINATE CHECK ROUTINES'
               ;        CKCRD1 - CKECK 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

55FC 48        CKCRD1:  PHA                 ; SAVE ALL REGISTERS
55FD 8A                 TXA
55FE 48                 PHA
55FF 98                 TYA
5600 48                 PHA
5601 A200               LDX    #X1CORD-X1CORD ; CHECK X1CORD
5603 A000               LDY    #XLIMIT-LIMTAB
5605 202B56             JSR    CK
5608 A202               LDX    #Y1CORD-X1CORD ; CHECK Y1CORD
560A A002               LDY    #YLIMIT-LIMTAB
560C 202B56             JSR    CK
560F 68        CKCRDR:  PLA                 ; RESTORE REGISTERS
5610 A8                 TAY
5611 68                 PLA
5612 AA                 TAX
5613 68                 PLA
5614 60                 RTS                 ; AND RETURN


               ;        CKCRD2 - SAME AS CKCRD1 EXCEPT CHECKS X2CORD,Y2CORD

5615 48        CKCRD2:  PHA                 ; SAVE ALL REGISTERS
5616 8A                 TXA
5617 48                 PHA
5618 98                 TYA
5619 48                 PHA
561A A204               LDX    #X2CORD-X1CORD ; CHECK X2CORD
561C A000               LDY    #XLIMIT-LIMTAB
561E 202B56             JSR    CK
5621 A206               LDX    #Y2CORD-X1CORD ; CHECK Y2CORD
5623 A002               LDY    #YLIMIT-LIMTAB
5625 202B56             JSR    CK
5628 4C0F56             JMP    CKCRDR       ; GO RESTORE REGISTERS AND RETURN

562B BD0201    CK:      LDA    X1CORD+1,X   ; CHECK UPPER BYTE
562E D95556             CMP    LIMTAB+1,Y   ; AGAINST UPPER BYTE OF LIMIT
5631 9020               BCC    CK4          ; OK IF LESS THAN UPPER BYTE OF LIMIT
5633 F016               BEQ    CK3          ; GO CHECK LOWER BYTE IF EQUAL TO
                                            ; UPPER BYTE OF LIMIT
5635 BD0101    CK2:     LDA    X1CORD,X     ; SUBTRACT THE LIMIT
5638 38                 SEC                 ; LOWER BYTE FIRST
5639 F95456             SBC    LIMTAB,Y
563C 9D0101             STA    X1CORD,X
563F BD0201             LDA    X1CORD+1,X
5642 F95556             SBC    LIMTAB+1,Y
5645 9D0201             STA    X1CORD+1,X
5648 4C2B56             JMP    CK           ; AND THEN GO CHECK RANGE AGAIN
```

```
564B BD0101    CK3:     LDA    X1CORD,X      ; CHECK LOWER BYTE OF X
564E D95456             CMP    LIMTAB,Y
5651 B0E2               BCS    CK2           ; GO ADJUST IF TOO LARGE
5653 60        CK4:     RTS                  ; RETURN

               LIMTAB:                       ; TABLE OF LIMITS
5654 4001      XLIMIT:  .WORD  NX
5656 C800      YLIMIT:  .WORD  NY

```

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

5658 A900      ERASE:   LDA    #X'00        ; SET LINE COLOR TO BLACK
565A F002               BEQ    DRAW1        ; GO DRAW THE LINE

565C A9FF      DRAW:    LDA    #X'FF        ; SET LINE COLOR TO WHITE
565E 8D1B01    DRAW1:   STA    COLOR
5661 8A                 TXA                 ; SAVE X AND Y
5662 48                 PHA
5663 98                 TYA
5664 48                 PHA

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

5665 A900               LDA    #0           ; FIRST ZERO DIR
5667 8D1801             STA    XDIR
566A AD0501             LDA    X2CORD       ; NEXT COMPUTE TWOS COMPLEMENT DIFFERENCE
566D 38                 SEC
566E ED0101             SBC    X1CORD
5671 8D1201             STA    DELTAX
5674 AD0601             LDA    X2CORD+1
5677 ED0201             SBC    X1CORD+1
567A 8D1301             STA    DELTAX+1
567D 1014               BPL    DRAW2        ; SKIP AHEAD IF DIFFERENCE IS POPSITIVE
567F CE1801             DEC    XDIR         ; SET XDIR TO -1
5682 38                 SEC                 ; NEGATE DELTAX
5683 A900               LDA    #0
5685 ED1201             SBC    DELTAX
5688 8D1201             STA    DELTAX
568B A900               LDA    #0
568D ED1301             SBC    DELTAX+1
5690 8D1301             STA    DELTAX+1

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

5693 A900      DRAW2:   LDA    #0           ; FIRST ZERO YDIR
5695 8D1901             STA    YDIR
5698 AD0701             LDA    Y2CORD       ; NEXT COMPUTE TWOS COMPLEMENT DIFFERENCE
569B 38                 SEC
569C ED0301             SBC    Y1CORD
569F 8D1401             STA    DELTAY
56A2 AD0801             LDA    Y2CORD+1
56A5 ED0401             SBC    Y1CORD+1
56A8 8D1501             STA    DELTAY+1
56AB 1014               BPL    DRAW3        ; SKI AHEAD IF DIFFERENCE IS POSITIVE
56AD CE1901             DEC    YDIR         ; SET YDIR TO -1
56B0 38                 SEC                 ; NEGATE DELTAX
56B1 A900               LDA    #0
```

```
56B3 ED1401              SBC    DELTAY
56B6 8D1401              STA    DELTAY
56B9 A900                LDA    #0
56BB ED1501              SBC    DELTAY+1
56BE 8D1501              STA    DELTAY+1

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

56C1 A900      DRAW3:    LDA    #0           ; FIRST ZERO XCHFLG
56C3 8D1A01              STA    XCHFLG
56C6 AD1401              LDA    DELTAY       ; COMPARE DELTAY WITH DELTAX
56C9 38                  SEC
56CA ED1201              SBC    DELTAX
56CD AD1501              LDA    DELTAY+1
56D0 ED1301              SBC    DELTAX+1
56D3 901B                BCC    DRAW4        ; SKIP EXCHANGE IF DELTAX IS GREATER THAN
                                             ; DELTAY
56D5 AE1401              LDX    DELTAY       ; EXCHANGE DELTAX AND DELTAY
56D8 AD1201              LDA    DELTAX
56DB 8D1401              STA    DELTAY
56DE 8E1201              STX    DELTAX
56E1 AE1501              LDX    DELTAY+1
56E4 AD1301              LDA    DELTAX+1
56E7 8D1501              STA    DELTAY+1
56EA 8E1301              STX    DELTAX+1
56ED CE1A01              DEC    XCHFLG       ; SET XCHFLG TO -1
56F0 AD1201    DRAW4:    LDA    DELTAX       ; INITIALIZE ACC TO DELTAX
56F3 8D1601              STA    ACC
56F6 AD1301              LDA    DELTAX+1
56F9 8D1701              STA    ACC+1
56FC AD1B01              LDA    COLOR        ; PUT A DOT AT THE INITIAL ENDPOINT
56FF 20C155              JSR    WRPIX        ; X1CORD,Y1CORD

               ;         HEAD OF MAIN DRAWING LOOP
               ;         TEST IF DONE

5702 AD1A01    DRAW45:   LDA    XCHFLG       ; TEST IF X AND Y EXCHANGED
5705 D012                BNE    DRAW5        ; JUMP AHEAD IF SO
5707 AD0101              LDA    X1CORD       ; TEST FOR X1CORD=X2CORD
570A CD0501              CMP    X2CORD
570D D01F                BNE    DRAW7        ; GO FOR ANOTHER ITERATION IF NOT
570F AD0201              LDA    X1CORD+1
5712 CD0601              CMP    X2CORD+1
5715 D017                BNE    DRAW7        ; GO FOR ANOTHER ITERATION IF NOT
5717 F010                BEQ    DRAW6        ; GO RETURN IF SO
5719 AD0301    DRAW5:    LDA    Y1CORD       ; TEST FOR Y1CORD=Y2CORD
571C CD0701              CMP    Y2CORD
571F D00D                BNE    DRAW7        ; GO FOR ANOTHER ITERATION IF NOT
5721 AD0401              LDA    Y1CORD+1
5724 CD0801              CMP    Y2CORD+1
5727 D005                BNE    DRAW7        ; GO FOR ANOTHER ITERATION IF NOT
5729 68        DRAW6:    PLA                 ; RESTORE INDEX REGISTERS
```

```
572A A8                 TAY
572B 68                 PLA
572C AA                 TAX
572D 60                 RTS                 ; AND RETURN

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

572E AD1A01     DRAW7:  LDA    XCHFLG       ; TEST IF X AND Y EXCHANGED
5731 D006               BNE    DRAW8        ; JUMP IF SO
5733 208957             JSR    BMPX         ; BUMP X IF NOT
5736 4C3C57             JMP    DRAW9
5739 20A357     DRAW8:  JSR    BMPY         ; BUMP Y IF SO
573C 206157     DRAW9:  JSR    SBDY         ; SUBTRACT DY FROM ACC TWICE
573F 206157             JSR    SBDY
5742 1014               BPL    DRAW12       ; SKIP AHEAD IF ACC IS NOT NEGATIVE
5744 AD1A01             LDA    XCHFLG       ; EST IF X AND Y EXCHANGED
5747 D006               BNE    DRAW10       ; JUMP IF SO
5749 20A357             JSR    BMPY         ; BUMP Y IF NOT
574C 4C5257             JMP    DRAW11
574F 208957     DRAW10: JSR    BMPX         ; BUMP X IF SO
5752 207557     DRAW11: JSR    ADDX         ; ADD DX TO ACC TWICE
5755 207557             JSR    ADDX

5758 AD1B01     DRAW12: LDA    COLOR        ; OUTPUT THE NEW POINT
575B 20C155             JSR    WRPIX
575E 4C0257             JMP    DRAW45       ; GO TEST IF DONE

                ;       SUBROUTINES FOR DRAW

5761 AD1601     SBDY:   LDA    ACC          ; SUBTRACT DELAY FROM ACC AND PUT RESULT
5764 38                 SEC                 ; IN ACC
5765 ED1401             SBC    DELTAY
5768 8D1601             STA    ACC
576B AD1701             LDA    ACC+1
576E ED1501             SBC    DELTAY+1
5771 8D1701             STA    ACC+1
5774 60                 RTS


5775 AD1601     ADDX:   LDA    ACC          ; ADD DELTAX TO ACC AND PUT RESULT IN ACC
5778 18                 CLC
5779 6D1201             ADC    DELTAX
577C 8D1601             STA    ACC
577F AD1701             LDA    ACC+1
5782 6D1301             ADC    DELTAX+1
5785 8D1701             STA    ACC+1
5788 60                 RTS


5789 AD1801     BMPX:   LDA    XDIR         ; BUMP X1CORD BY +1 OR -1 ACCORDING
578C D009               BNE    BMPX2        ; XDIR
578E EE0101             INC    X1CORD       ; DOUBLE INCREMENT X1CORD IF XDIR=0
5791 D003               BNE    BMPX1
```

```
5793 EE0201              INC    X1CORD+1
5796 60         BMPX1:   RTS
5797 AD0101     BMPX2:   LDA    X1CORD       ; DOUBLE DECREMENT X1CORD IF XDIR<>0
579A D003                BNE    BMPX3
579C CE0201              DEC    X1CORD+1
579F CE0101     BMPX3:   DEC    X1CORD
57A2 60                  RTS


57A3 AC1901     BMPY:    LDY    YDIR         ; BUMP Y1CORD BY +1 OR -1 ACCORDING TO
57A6 D009                BNE    BMPY2        ; YDIR
57A8 EE0301              INC    Y1CORD       ; DOUBLE INCREMENT Y1CORD IF YDIR=0
57AB D003                BNE    BMPY1
57AD EE0401              INC    Y1CORD+1
57B0 60         BMPY1:   RTS
57B1 AD0301     BMPY2:   LDA    Y1CORD       ; DOUBLE DECREMENT Y1CORD IF YDIR<>0
57B4 D003                BNE    BMPY3
57B6 CE0401              DEC    Y1CORD+1
57B9 CE0301     BMPY3:   DEC    Y1CORD
57BC 60                  RTS

```

```
                        .PAGE  'DCHAR - DRAW A CHARACTER'
               ;        DCHAR - DRAW A CHARACTER WHOSE UPPER LEFT CORNER IS AT
               ;        X1CORD,Y1CORD
               ;        X1CORD AND Y1CORD ARE NOT ALTERED
               ;        THIS ROUTINE DISPLAYS A 5 BY 9 DOT MATRIX CHARACTER AT THE
               ;        SPECIFIED LOCATION.  THE 5 BY 9 BLOCK IS CLEARED AND THEN THE
               ;        CHARACTER IS WRITTEN INTO IT.
               ;        THE 5 BY 9 MATRIX INCLUDES 2 LINE DESCENDERS ON LOWER CASE
               ;        CHARACTERS.
               ;        BOTH INDEX REGISTERS AND THE ACCUMULATOR ARE PRESERVED.
               ;        THE CHARACTER CODE TO BE DISPLAYED SHOULD BE IN A.
               ;        ASCII CONTROL CODES ARE IGNORED AND NO DRAWING IS DONE
               ;        THIS ROUTINE ASSUMES IN RANGE COORDINATES INCLUDING WIDTH AND
               ;        HEIGHT OF CHARACTER.

57BD 48        DCHAR:   PHA                 ; SAVE REGISTERS
57BE 8A                 TXA
57BF 48                 PHA
57C0 98                 TYA
57C1 48                 PHA
57C2 BA                 TSX                 ; GET IMPUT CHARACTER BACK
57C3 BD0301             LDA    X'103,X
57C6 297F               AND    #X'7F        ; INSURE 7 BIT ASCII INPUT
57C8 38                 SEC
57C9 E920               SBC    #X'20        ; TEST IF A CONTROL CHARACTER
57CB 3062               BMI    DCHAR5       ; DO A QUICK RETURN IF SO

               ;        CALCULATE FONT TABLE ADDRESS FOR CHAR

57CD 48                 PHA                 ; SAVE VERIFIED, ZERO ORIGIN CHAR CODE
57CE 202155             JSR    PIXADR       ; GET BYTE AND BIT ADDRESS OF FIRST SCAN
                                            ; LINE OF CHARACTER INTO ADP1 AND BTPT
57D1 68                 PLA                 ; RESTORE ZERO ORIGIN CHARACTER CODE
57D2 85EC               STA    ADP2         ; PUT IT INTO ADP2
57D4 A900               LDA    #0
57D6 85ED               STA    ADP2+1
57D8 20DC5A             JSR    SADP2L       ; COMPUTE 8*CHARACTER CODE IN ADP2
57DB 20DC5A             JSR    SADP2L
57DE 20DC5A             JSR    SADP2L
57E1 A5EC               LDA    ADP2         ; ADD IN ORIGIN FOR CHARACTER TABLE
57E3 18                 CLC
57E4 6976               ADC    #CHTB&X'FF
57E6 85EC               STA    ADP2
57E8 A5ED               LDA    ADP2+1
57EA 695C               ADC    #CHTB/256
57EC 85ED               STA    ADP2+1       ; ADP2 NOW HAS ADDRESS OF TOP ROW OF
                                            ; CHARACTER SHAPE


57EE A000               LDY    #0           ; INITIALIZE Y INDEX = FONT TABLE POINTER
57F0 A200               LDX    #0           ; INITIALIZE X = SCAN LINE COUNTER

               ;        CLEAR THE FIRST TWO SCAN LINES OF DESCENDING CHARACTERS
               ;        FOR LOWER CASE "J", PUT IN THE DOT AS A SPECIAL CASE

```

```
57F2 B1EC             LDA    (ADP2),Y     ; GET THE FIRST ROW FROM THE TABLE
57F4 F01C             BEQ    DCHAR3       ; SKIP AHEAD IF NOT A DESCENDING CHARACTER
57F6 A5EC             LDA    ADP2         ; IF DESCENDING, TEST IF LOWER CASE J
57F8 C9C6             CMP    #X'6A-X'20*8+CHTB&X'FF
57FA D004             BNE    DCHAR1       ; CLEAR FIRST SCAN LINE IF NOT
57FC A920             LDA    #X'20        ; LOAD THE DOT FOR THE J IF A J
57FE D002             BNE    DCHAR2
5800 A900    DCHAR1:  LDA    #0           ; DO THE FIRST SCAN LINE
5802 208558  DCHAR2:  JSR    MERGE5
5805 20E15A           JSR    DN1SCN       ; GO DOWN 1 SCAN LINE
5808 E8               INX                 ; COUNT SCAN LINES DONE
5809 A900             LDA    #0           ; CLEAR THE SECOND SCAN LINE
580B 208558           JSR    MERGE5
580E 20E15A           JSR    DN1SCN       ; GO DOWN ANOTHER SCAN LINE
5811 E8               INX                 ; COUNT SCAN LINES DONE

             ;        SCAN QUT THE BODY OF THE CHARACTER

5812 C8      DCHAR3:  INY                 ; GO TO NEXT SCAN LINE OF THE FRONT
5813 B1EC             LDA    (ADP2),Y     ; GET THE SCAN LINE
5815 208558           JSR    MERGE5       ; MERGE IT WITH GRAPHIC MEMORY AT (ADP1)
5818 20E15A           JSR    DN1SCN       ; GO DOWN 1 SCAN LINE
581B E8               INX                 ; COUNT SCAN LINES OUTPUTTED
581C C007             CPY    #7           ; TEST IF WHOLE CHARACTER SCANNED OUT
581E D0F2             BNE    DCHAR3       ; GO SCAN OUT ANOTHER ROW IF NOT
5820 E009    DCHAR4:  CPX    #9           ; TEST IF THE WHOLE CHARACTER CELL SCANNED
5822 F00B             BEQ    DCHAR5       ; JUMP OUT IF SO
5824 A900             LDA    #0           ; CLEAR TRAILING SCAN LINES ON
5826 208558           JSR    MERGE5       ; NON-DESDENDING CHARACTERS
5829 20E15A           JSR    DN1SCN       ; TO NEXT LINE
582C E8               INX                 ; COUNT LINES
582D D0F1             BNE    DCHAR4       ; LOOP UNTIL DONE

             ;        RESTORE REGISTERS AND RETURN

582F 68      DCHAR5:  PLA
5830 A8               TAY
5831 68               PLA
5832 AA               TAX
5833 68               PLA
5834 60               RTS

```

```
                        .PAGE  'GRAPHIC MERGE ROUTINES'
               ;        MERGEL - MERGE LEFT ROUTINE
               ;        MERGES ACCUMULATOR CONTENTS WITH A BYTE OF GRAPHIC MEMORY
               ;        ADDRESSED BY ADP1 AND BTPT.
               ;        BITS TO THE LEFT OF (BTPT) ARE PRESERVED IN GRAPHIC MEMORY.
               ;        BIT (BTPT) AND BITS TO THE RIGHT ARE SET EQUAL TO
               ;        CORRESPONDING BIT POSITIONS IN THE ACCUMULATOR.
               ;        NO REGISTERS ARE BOTHERED.

5835 48        MERGEL:  PHA                 ; SAVE REGISTERS
5836 8A                 TXA
5837 48                 PHA
5838 98                 TYA
5839 48                 PHA
583A BA                 TSX                 ; GET INPUT BACKK
583B BD0301             LDA    X'103,X
583E AC1101             LDY    BTPT         ; GET BIT NUMBER INTO Y
5841 39D058             AND    MERGTR-1,Y   ; CLEAR BITS TO BE PRESERVED IN MEMORY
5844 9D0301             STA    X'103,X      ; FROM A
5847 A000               LDY    #0           ; CLEAR BITS FROM MEMORY TO BE CHANGED
5849 AE1101             LDX    BTPT
584C B1EA               LDA    (ADP1),Y     ; GET MEMORY BYTE
584E 3DC858             AND    MERGTL,X     ; CLEAR THE BITS
5851 BA                 TSX                 ; DO THE MERGING
5852 1D0301             ORA    X'103,X
5855 91EA               STA    (ADP1),Y
5857 68                 PLA                 ; RESTORE REGISTERS
5858 A8                 TAY
5859 68                 PLA
585A AA                 TAX
585B 68                 PLA
585C 60                 RTS                 ; RETURN

               ;        MERGR - MERGE RIGHT ROUTINE
               ;        MERGES ACCUMULATOR CONTENTS WITH A BYTE OF GRAPHIC MEMORY
               ;        ADDRESSED BY ADP1 AND BTPT.
               ;        BITS TO THE RIGHT OF (BTPT) ARE PRESERVED IN GRAPHIC MEMORY.
               ;        BIT (BTPT) AND BITS TO THE LEFT ARE SET EQUAL TO CORRESPONDING
               ;        BIT POSITIONS IN THE ACCUMULATOR.
               ;        NO REGISTERS ARE BOTHERED.

585D 48        MERGER:  PHA                 ; SAVE REGISTERS
585E 8A                 TXA
585F 48                 PHA
5860 98                 TYA
5861 48                 PHA
5862 BA                 TSX                 ; GET INPUT BACKK
5863 BD0301             LDA    X'103,X
5866 AC1101             LDY    BTPT         ; GET BIT NUMBER INTO Y
5869 39C758             AND    MERGTL-1,Y   ; CLEAR BITS TO BE PRESERVED IN MEMORY
586C 9D0301             STA    X'103,X      ; FROM A
586F A000               LDY    #0           ; CLEAR BITS FROM MEMORY TO BE CHANGED
5871 AE1101             LDX    BTPT
5874 B1EA               LDA    (ADP1),Y     ; GET MEMORY BYTE
5876 3DD158             AND    MERGTR,X     ; CLEAR THE BITS
```

```
5879 BA                 TSX                   ; DO THE MERGING
587A 1D0301             ORA    X'103,X
587D 91EA               STA    (ADP1),Y
587F 68                 PLA                   ; RESTORE REGISTERS
5880 A8                 TAY
5881 68                 PLA
5882 AA                 TAX
5883 68                 PLA
5884 60                 RTS                   ; RETURN

               ;        MERGE A ROW OF 5 DOTS WITH GRAPHIC MEMORY STARTING AT BYTE
               ;        ADDRESS AND BIT NUMBER IN ADP1 AND BTPT
               ;        5 DOTS TO MERGE LEFT JUSTIFIED IN A
               ;        PRESERVES X AND Y

5885 8D1D01    MERGE5:  STA    TEMP+1         ; SAVE INPUT DATA
5888 98                 TYA                   ; SAVE Y
5889 48                 PHA
588A AC1101             LDY    BTPT           ; OPEN UP A 5 BIT WINDOW IN GRAPHIC MEMORY
588D B9D958             LDA    MERGT5,Y       ; LEFT BITS
5890 A000               LDY    #0             ; ZERO Y
5892 31EA               AND    (ADP1),Y
5894 91EA               STA    (ADP1),Y
5896 AC1101             LDY    BTPT
5899 B9E158             LDA    MERGT5+8,Y     ; RIGHT BITS
589C A001               LDY    #1
589E 31EA               AND    (ADP1),Y
58A0 91EA               STA    (ADP1),Y
58A2 AD1D01             LDA    TEMP+1         ; SHIFT DATA RIGHT TO LINE UP LEFTMOST
58A5 AC1101             LDY    BTPT           ; DATA BIT WITH LEFTMOST GRAPHIC FIELD
58A8 F004               BEQ    MERGE2         ; SHIFT BTPT TIMES
58AA 4A        MERGE1:  LSRA
58AB 88                 DEY
58AC D0FC               BNE    MERGE1
58AE 11EA      MERGE2:  ORA    (ADP1),Y       ; OVERLAY WITH GRAPHIC MEMORY
58B0 91EA               STA    (ADP1),Y
58B2 A908               LDA    #8             ; SHIFT DATA LEFT TO LINE UP RIGHTMOST
58B4 38                 SEC                   ; DATA BIT WITH RIGHTMOST GRAPHIC FIELD
58B5 ED1101             SBC    BTPT           ; SHIFT (8-BTPT) TIMES
58B8 A8                 TAY
58B9 AD1D01             LDA    TEMP+1
58BC 0A        MERGE3:  ASLA
58BD 88                 DEY
58BE D0FC               BNE    MERGE3
58C0 C8                 INY
58C1 11EA               ORA    (ADP1),Y       ; OVERLAY WITH GRAPHIC MEMORY
58C3 91EA               STA    (ADP1),Y
58C5 68                 PLA                   ; RESTORE Y
58C6 A8                 TAY
58C7 60                 RTS                   ; RETURN

58C8 0080C0E0  MERGTL:  .BYTE  X'00,X'80,X'C0,X'E0  ; MASKS FOR MERGE LEFT
58CC F0F8FCFE           .BYTE  X'F0,X'F8,X'FC,X'FE  ; CLEAR ALL BITS TO THE RIGHT OF
58D0 FF                 .BYTE  X'FF                 ; AND INCLUDING BIT N (0=MSB)
```

```
58D1 7F3F1F0F  MERGTR:  .BYTE  X'7F,X'3F,X'1F,X'0F  ; MASKS FOR MERGE RIGHT
58D5 07030100           .BYTE  X'07,X'03,X'01,X'00  ; CLEAR ALL BITS TO THE LEFT OF
                                                    ; AND INCLUDING BIT N (0=MSB)

58D9 0783C1E0  MERGT5:  .BYTE  X'07,X'83,X'C1,X'E0  ; TABLE OF MASKS FOR OPENING UP
58DD F0F8FCFE           .BYTE  X'F0,X'F8,X'FC,X'FE  ; A 5 BIT WINDOW ANYWHERE
58E1 FFFFFFFF           .BYTE  X'FF,X'FF,X'FF,X'FF  ; IN GRAPHIC MEMORY
58E5 7F3F1F0F           .BYTE  X'7F,X'3F,X'1F,X'0F

```

```
                .PAGE  'DTEXT - SOPHISTICATED TEXT DISPLAY ROUTINE'
        ;       DTEXT - SOPHISTICATED TEXT DISPLAY ROUTINE
        ;       CURSOR IS ADDRESSED IN TERMS OF X AND Y COORDINATES.
        ;       CURSOR POSITION IS IN X1CORD AND Y1CORD 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 X1CORD,Y1CORD 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 X1CORD AND YICORD. 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
```

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

               ;        DTXTIN MAY BE CALLED TO INITIALIZE DTEXT FOR USE AS A FULL
               ;        SCREEN TEXT DISPLAY ROUTINE.  SETS MARGINS FOR FULL SCREEN
               ;        OPERATION, CLEARS THE SCREEN, AND SETS THE CURSOR AT THE UPPER
               ;        LEFT CORNER OF THE SCREEN. THE USER MUST STILL SET VMORG
               ;        HOWEVER!

               ;        DTXTIN - CONVENIENT INITIALIZE ROUTINE FOR FULL SCREEN USE OF
               ;        DTEXT.

58E9 A900      DTXTIN:  LDA    #0           ; SET LEFT AND BOTTOM MARGINS TO ZERO
58EB 8D0D01             STA    LMAR
58EE 8D0E01             STA    LMAR+1
58F1 8D0B01             STA    BMAR
58F4 8D0C01             STA    BMAR+1
58F7 A9C7               LDA    #NY-1&X'FF   ; SET TOP MARGIN TO TOP OF SCREEN
58F9 8D0901             STA    TMAR
58FC A900               LDA    #NY-1/256
58FE 8D0A01             STA    TMAR+1
5901 A93F               LDA    #NX-1&X'FF   ; SET RIGHT MARGIN TO RIGHT EDGE OF SCREEN
5903 8D0F01             STA    RMAR
5906 A901               LDA    #NX-1/256
5908 8D1001             STA    RMAR+1
590B A90C               LDA    #X'0C        ; CLEAR SCREEN AND PUT CURSOR AT UPPER
                                            ; LEFT CORNER BY SENDING AN ASCII FF
                                            ; CONTROL CHARACTER TO DTEXT.  THEN FALL
                                            ; INTO DTEXT.

               ;        DTEXT - DISPLAY ASCII TEXT ROUTINE
               ;        ENTER WITH ASCII CHARACTER CODE TO DISPLAY OR INTERPRET IN A.
               ;        PRESERVES ALL REGISTERS.

590D 48        DTEXT:   PHA                 ; SAVE THE REGISTERS
590E 8A                 TXA
590F 48                 PHA
5910 98                 TYA
5911 48                 PHA
5912 BA                 TSX                 ; GET INPUT BACK
5913 BD0301             LDA    X'103,X      ; FROM THE STACK
5916 297F               AND    #X'7F        ; INSURE 7 BIT ASCII INPUT
5918 C920               CMP    #X'20        ; TEST IF A CONTROL CHARACTER
591A 300C               BMI    DTEXT1       ; JUMP AHEAD IF SO
591C 20BD57             JSR    DCHAR        ; FOR A REGULAR TEXT CHARACTER, DISPLAY IT
591F 20F05B             JSR    CSRR         ; DO A CURSOR RIGHT
5922 68        DTEXTR:  PLA                 ; RESTORE THE REGISTERS
5923 A8                 TAY
5924 68                 PLA
5925 AA                 TAX
5926 68                 PLA
5927 60                 RTS                 ; AND RETURN

5928 A200      DTEXT1:  LDX    #0           ; SET UP A LOOP TO SEARCH THE CONTROL
```

```
592A DD585C    DTEXT2:  CMP    CCTAB,X      ; CHARACTER TABLE FOR A MATCH
592D F009               BEQ    DTEXT3       ; JUMP IF A MATCH
592F E8                 INX                 ; BUMP X TO POINT TO NEXT TABLE ENTRY
5930 E8                 INX
5931 E8                 INX
5932 E01E               CPX    #CCTABE-CCTAB; TEST IF ENTIRE TABLE SEARCHED
5934 D0F4               BNE    DTEXT2       ; LOOP IF NOT
5936 F0EA               BEQ    DTEXTR       ; GO RETURN IF ENTIRE TABLE SEARCHED

5938 BD5A5C    DTEXT3:  LDA    CCTAB+2,X    ; JUMP TO THE ADDRESS IN THE NEXT TWO
593B 48                 PHA                 ; TABLE BYTES
593C BD595C             LDA    CCTAB+1,X
593F 48                 PHA
5940 60                 RTS

```

```
                                .PAGE 'SERVICE ROUTINES FOR CONTROL CHARACTERS'
                     ;          SERVICE ROUTINES FOR CONTROL CHARACTERS.  DO THE INDICATED
                     ;          FUNCTION AND JUMP TO DTEXTR TO RESTORE REGISTERS AND RETURN.

                     ;          CRR - CURSOR RIGHT

    5941 20F05B      CRR:       JSR    CSRR          ; NOVE CURSOR RIGHT
    5944 4C2259                 JMP    DTEXTR        ; GO RETURN

                     ;          CRL - CURSOR LEFT AND BACKSPACE

    5947 200A5C      CRL:       JSR    CSRL          ; MOVE CURSOR LEFT
    594A 4C2259                 JMP    DTEXTR        ; GO RETURN

                     ;          CRU - CURSOR UP

    594D 20245C      CRU:       JSR    CSRU          ; NOVE CURSOR UP
    5950 4C2259                 JMP    DTEXTR        ; GO RETURN

                     ;          CRD - CURSOR DOWD

    5953 203E5C      CRD:       JSR    CSRD          ; NOVE CURSOR DOWN
    5956 4C2259                 JMP    DTEXTR        ; GO RETURN

                     ;          BASUP - SHIFT BASELINE UP 3 SCAN LINES
                     ;          NOTE - NO RANGE CHECK ON THE Y COORDINATE IS MADE
                     ;          BASELINE SHIFTING SHOULD ONLY BE DONE AT A BLANK CHARACTER
                     ;          POSITION

    5959 20C95B      BASUP:     JSR    CSRDEL        ; DELETE CURRENT CURSOR
    595C AD0301                 LDA    Y1CORD        ; INCREMENT COORDINATE BY 3
    595F 18                     CLC
    5960 6903                   ADC    #3
    5962 8D0301                 STA    Y1CORD
    5965 9003                   BCC    BASUP1
    5967 EE0401                 INC    Y1CORD+1
    596A 20C55B      BASUP1:    JSR    CSRINS        ; DISPLAY CURSOR AT NEW LOCATION
    596D 4C2259                 JMP    DTEXTR        ; GO RETURN

                     ;          BASDN - SHIFT BASELINE DOEN 3 SCAN LINES
                     ;          NOTE - NO RANGE CHECK ON THE Y COORDINATE IS MADE
                     ;          BASELINE SHIFTING SHOULD ONLY BE DONE AT A BLANK CHARACTER
                     ;          POSITION

    5970 20C95B      BASDN:     JSR    CSRDEL        ; DELETE CURRENT CURSOR
    5973 AD0301                 LDA    Y1CORD        ; INCREMENT COORDINATE BY 3
    5976 38                     SEC
    5977 E903                   SBC    #3
    5979 8D0301                 STA    Y1CORD
    597C B003                   BCS    BASDN1
    597E CE0401                 DEC    Y1CORD+1
    5981 20C55B      BASDN1:    JSR    CSRINS        ; DISPLAY CURSOR AT NEW LOCATION
    5984 4C2259                 JMP    DTEXTR        ; GO RETURN

                     ;          CARRET - CARRIAGE RETURN
```

```

5987 20C95B    CARRET:  JSR    CSRDEL        ; DELETE CURRENT CURSOR
598A AD0D01             LDA    LMAR          ; SET X1CORD TO THE LEFT MARGIN
598D 8D0101             STA    X1CORD
5990 AD0E01             LDA    LMAR+1
5993 8D0201             STA    X1CORD+1
5996 20C55B             JSR    CSRINS        ; DISPLAY CURSOR AT NEW LOCATION
5999 4C2259             JMP    DTEXTR        ; GO RETURN

               ;        LNFED - LINE FEED ROUTINE, SCROLLS IF NOT SUFFICIENT SPACE
               ;                AT THE BOTTOM FOR A NEW LINE

599C 20695B    LNFED:   JSR    DNTST         ; TEST IF CURSOR IS TOO FAR DOWN TO ALLOW
599F 9006               BCC    LNFED1        ; MOVEMENT
59A1 203E5C             JSR    CSRD          ; IF OK, DO A SIMPLE CURSOR DOWN
59A4 4C2259             JMP    DTEXTR        ; AND GO RETURN
59A7 20C95B    LNFED1:  JSR    CSRDEL        ; DELETE CURRENT CURSOR
59AA 20ED5A             JSR    RECTP         ; SAVE CURSOR COORDINATES AND PROCESS
                                             ; CORNER DATA
59AD AD1201    LNFED0:  LDA    TLBYT         ; ADD CHHIW SCAN LINES TO ADDRESS OF TOP
59B0 18                 CLC                  ; LEFT CORNER TO ESTABLISH ADDRESS OF
59B1 69B8               ADC    #CHHIW*NX/8&X'FF  ; FIRST SCAN LINE TO SCROLL
59B3 85EC               STA    ADP2          ; AND PUT INTO ADP2
59B5 AD1301             LDA    TLBYT+1
59B8 6901               ADC    #CHHIW*NX/8/256
59BA 85ED               STA    ADP2+1

               ;        MOVE LEFT PARTIAL BYTE

59BC AD1201    LNFED2:  LDA    TLBYT         ; MOVE CURRENT TOP LEFT BYTE ADDRESS INTO
59BF 85EA               STA    ADP1          ; ADP1
59C1 AD1301             LDA    TLBYT+1
59C4 85EB               STA    ADP1+1
59C6 AD1801             LDA    TLBIT         ; MOVE LEFT BIT ADDRESS TO BTPT
59C9 8D1101             STA    BTPT
59CC A000               LDY    #0
59CE B1EC               LDA    (ADP2),Y      ; MOVE A PARTIAL BYTE FROM (ADP2)
59D0 203558             JSR    MERGEL        ; TO (ADP1) ACCORDING TO BTPT

               ;        MOVE FULL BYTES IN THE MIDDLE

59D3 E6EA      LNFED3:  INC    ADP1          ; INCREMENT ADP1
59D5 D002               BNE    LNFED4
59D7 E6EB               INC    ADP1+1
59D9 E6EC      LNFED4:  INC    ADP2          ; INCREMENT ADP2
59DB D002               BNE    LNFED5
59DD E6ED               INC    ADP2+1
59DF A5EA      LNFED5:  LDA    ADP1          ; TEST IF EQUAL TO CURRENT TOP RIGHT BYTE
59E1 CD1401             CMP    TRBYT         ; ADDRESS
59E4 D007               BNE    LNFED6        ; SKIP AHEAD IF NOT
59E6 A5EB               LDA    ADP1+1
59E8 CD1501             CMP    TRBYT+1
59EB F007               BEQ    LNFED7        ; GO TO RIGHT PARTIAL BYTE PROCESSING IF =
59ED B1EC      LNFED6:  LDA    (ADP2),Y      ; MOVE A BYTE
59EF 91EA               STA    (ADP1),Y
```

```
59F1 4CD359             JMP    LNFED3       ; GO PROCESS NEXT BYTE

               ;        MOVE RIGHT PARTIAL BYTE

59F4 AD1901    LNFED7:  LDA    TRBIT        ; MOVE RIGHT BIT ADDRESS TO BTPT
59F7 8D1101             STA    BTPT
59FA B1EC               LDA    (ADP2),Y     ; MOVE A PARTIAL BYTE FROM (ADP2) TO
59FC 205D58             JSR    MERGER       ; (ADP1) ACCORDING TO BTPT
59FF A5EC               LDA    ADP2         ; TEST IF ADP2 = BRBYT
5A01 CD1601             CMP    BRBYT
5A04 D009               BNE    LNFED8       ; JUMP AHEAD IF NOT
5A06 A5ED               LDA    ADP2+1
5A08 CD1701             CMP    BRBYT+1
5A0B D002               BNE    LNFED8       ; JUMP AHEAD IF NOT
5A0D F01F               BEQ    LNFEDB       ; FINISHED WITH MOVE PART OF SCROLL, GO
                                            ; CLEAR AREA LEFT AT BOTTOM OF RECTANGLE

               ;        PREPARE TO START NEXT LINE

5A0F AD1201    LNFED8:  LDA    TLBYT        ; ADD NX/8 TO TOP LEFT BYTE ADDRESS
5A12 18                 CLC
5A13 6928               ADC    #NX/8
5A15 8D1201             STA    TLBYT
5A18 9003               BCC    LNFED9
5A1A EE1301             INC    TLBYT+1
5A1D AD1401    LNFED9:  LDA    TRBYT        ; ADD NX/8 TO TOP RIGHT BYTE ADDRESS
5A20 18                 CLC
5A21 6928               ADC    #NX/8
5A23 8D1401             STA    TRBYT
5A26 9085               BCC    LNFED0
5A28 EE1501             INC    TRBYT+1
5A2B 4CAD59             JMP    LNFED0       ; GO MOVE NEXT SCAN LINE

               ;        CLEAR REGION AT BOTTOM OF RECTANGLE FOR NEW LINE OF TEXT
               ;        AND REINSERT CURSOR

5A2E 20735A    LNFEDB:  JSR    LNCLR        ; DO THE CLEARING
5A31 AD0501             LDA    X2CORD       ; RESTORE CURSOR COORDINATES
5A34 8D0101             STA    X1CORD
5A37 AD0601             LDA    X2CORD+1
5A3A 8D0201             STA    X1CORD+1
5A3D AD0701             LDA    Y2CORD
5A40 8D0301             STA    Y1CORD
5A43 AD0801             LDA    Y2CORD+1
5A46 8D0401             STA    Y1CORD+1
5A49 20C55B             JSR    CSRINS       ; INSERT CURSOR AT THE SAME POSITION
5A4C 4C2259             JMP    DTEXTR       ; GO RETURN

               ;        FMFED - FORM FEED ROUTINE, CLEARS THE SCREEN BETWEEN THE
               ;                MARGINS AND PLACES CURSOR AT UPPER LEFT CORNER OF
               ;                RECTANGLE DEFINED BY THE MARGINS.
               ;        NOTE: ROUTINE MODIFIES BOTH ADDRESS POINTERS AND BOTH SETS OF
               ;        COORDINATES.

5A4F 20ED5A    FMFED:   JSR    RECTP        ; PROCESS MARGIN DATA INTO CORNER
```

```
                                                   ; BYTE AND BIT ADDRESSES
5A52 20735A              JSR    LNCLR        ; CLEAR THE AREA DEFINED BY THE CORNERS
5A55 AD0D01              LDA    LMAR         ; POSITION CURSOR AT TOP AND LEFT MARGINS
5A58 8D0101              STA    X1CORD
5A5B AD0E01              LDA    LMAR+1
5A5E 8D0201              STA    X1CORD+1
5A61 AD0901              LDA    TMAR
5A64 8D0301              STA    Y1CORD
5A67 AD0A01              LDA    TMAR+1
5A6A 8D0401              STA    Y1CORD+1
5A6D 20C55B              JSR    CSRINS       ; INSERT CURSOR
5A70 4C2259              JMP    DTEXTR       ; FINISGED WITH FORM FEED
```

```
                        .PAGE 'MISCELLANEOUS INTERNAL SUBROUTINES'
               ;        LNCLR - SUBROUTINE TO CLEAR AREA INSIDE OF THE MARGINS
               ;        DEFINED BY TLBYT,TLBIT; TRBYT,TRBIT; BRBYT
               ;        USED BY FORM FEED AND SCROLL TO CLEAR BETWEEN THE MARGINS
               ;        CLEAR LEFT PARTIAL BYTE
               ;        USES INDEX Y

5A73 AD1201    LNCLR:   LDA    TLBYT        ; MOVE CURRENT TOP LEFT BYTE ADDRESS INTO
5A76 85EA               STA    ADP1         ; ADP1
5A78 AD1301             LDA    TLBYT+1
5A7B 85EB               STA    ADP1+1
5A7D AD1801             LDA    TLBIT        ; MOVE LEFT BIT ADDRESS TO BTPT
5A80 8D1101             STA    BTPT
5A83 A900               LDA    #0           ; CLEAR LEFT PARTIAL BYTE
5A85 203558             JSR    MERGEL

               ;        CLEAR FULL BYTES IN THE MIDDLE

5A88 E6EA      LNCLR1:  INC    ADP1         ; INCREMENT ADP1
5A8A D002               BNE    LNCLR2
5A8C E6EB               INC    ADP1+1
5A8E A5EA      LNCLR2:  LDA    ADP1         ; TEST IF EQUAL TO CURRENT TOP RIGHT BYTE
5A90 CD1401             CMP    TRBYT        ; ADDRESS
5A93 D007               BNE    LNCLR3       ; SKIP AHEAD IF NOT
5A95 A5EB               LDA    ADP1+1
5A97 CD1501             CMP    TRBYT+1
5A9A F007               BEQ    LNCLR4       ; GO TO RIGHT PARTIAL BYTE PROCESSING IF =
5A9C A900      LNCLR3:  LDA    #0           ; ZERO A BYTE
5A9E A8                 TAY
5A9F 91EA               STA    (ADP1),Y
5AA1 F0E5               BEQ    LNCLR1       ; LOOP UNTIL ALL FULL BYTES ON THIS LINE
                                            ; HAVE BEEN CLEARED

               ;        CLEAR RIGHT PARTIAL BYTE

5AA3 AD1901    LNCLR4:  LDA    TRBIT        ; MOVE RIGHT BIT ADDRESS TO BTPT
5AA6 8D1101             STA    BTPT
5AA9 A900               LDA    #0           ; CLEAR RIGHT PARTIAL BYTE
5AAB 205D58             JSR    MERGER
5AAE A5EA               LDA    ADP1         ; TEST IF ADP1 = BRBYT
5AB0 CD1601             CMP    BRBYT
5AB3 D008               BNE    LNCLR5       ; JUMP AHEAD IF NOT
5AB5 A5EB               LDA    ADP1+1
5AB7 CD1701             CMP    BRBYT+1
5ABA D001               BNE    LNCLR5       ; JUMP AHEAD IF NOT
5ABC 60                 RTS                 ; FINISHED WITH CLEAR IF SO

               ;        PREPARE TO STAR NEXT LINE

5ABD AD1201    LNCLR5:  LDA    TLBYT        ; ADD NX/8 TO TOP LEFT BYTE ADDRESS
5AC0 18                 CLC
5AC1 6928               ADC    #NX/8
5AC3 8D1201             STA    TLBYT
5AC6 9003               BCC    LNCLR6
5AC8 EE1301             INC    TLBYT+1
```

```
5ACB AD1401    LNCLR6:  LDA    TRBYT         ; ADD NX/8 TO TOP RIGHT BYTE ADDRESS
5ACE 18                 CLC
5ACF 6928               ADC    #NX/8
5AD1 8D1401             STA    TRBYT
5AD4 909D               BCC    LNCLR         ; GO PROCESS NEXT LINE
5AD6 EE1501             INC    TRBYT+1
5AD9 4C735A             JMP    LNCLR

               ;        SADP2L - SHIFT ADP2 LEFT 1 BIT POSITION
               ;        NO REGISTERS BOTHERED

5ADC 06EC      SADP2L:  ASL    ADP2          ; SHIFT LOW PART
5ADE 26ED               ROL    ADP2+1        ; SHIFT HIGH PART
5AE0 60                 RTS                  ; RETURN

               ;        DN1SCN - SUBROUTINE TO ADD NX/8 TO ADP1 TO EFFECT A DOWN
               ;        SHIFT OF ONE SCAN LINE
               ;        INDEX REGISTERS PRESERVED

5AE1 A5EA      DN1SCN:  LDA    ADP1          ; ADD NX/8 TO LOW ADP1
5AE3 18                 CLC
5AE4 6928               ADC    #NX/8
5AE6 85EA               STA    ADP1
5AE8 9002               BCC    DN1SC1
5AEA E6EB               INC    ADP1+1        ; INCREMENT HIGH PART IF CARRY FROM LOW
5AEC 60        DN1SC1:  RTS                  ; RETURN

               ;        SUBROUTINE TO ESTABLISH USEFUL DATA ABOUT THE RECTANGLE
               ;        DEFINED BY THE TEXT MARGINS IN TERMS OF BYTE AND BIT ADDR.
               ;        TLBYT AND TLBIT DEFINE THE UPPER LEFT CORNER, TRBYT AND TRBIT
               ;        DEFINE UPPER RIGHT CORNER, BRBYT DEFINES BOTTOM RIGHT CORNER

5AED AD0101    RECTP:   LDA    X1CORD        ; SAVE CURRENT CURSOR POSITION IN
5AF0 8D0501             STA    X2CORD        ; X2CORD AND Y2CORD
5AF3 AD0201             LDA    X1CORD+1
5AF6 8D0601             STA    X2CORD+1
5AF9 AD0301             LDA    Y1CORD
5AFC 8D0701             STA    Y2CORD
5AFF AD0401             LDA    Y1CORD+1
5B02 8D0801             STA    Y2CORD+1
5B05 AD0D01             LDA    LMAR          ; ESTABLISH BYTE AND BIR ADDRESSES OF
5B08 8D0101             STA    X1CORD        ; TOP LEFT CORNER
5B0B AD0E01             LDA    LMAR+1
5B0E 8D0201             STA    X1CORD+1
5B11 AD0901             LDA    TMAR
5B14 8D0301             STA    Y1CORD
5B17 AD0A01             LDA    TMAR+1
5B1A 8D0401             STA    Y1CORD+1
5B1D 202155             JSR    PIXADR
5B20 A5EA               LDA    ADP1
5B22 8D1201             STA    TLBYT
5B25 A5EB               LDA    ADP1+1
5B27 8D1301             STA    TLBYT+1
5B2A AD1101             LDA    BTPT
5B2D 8D1801             STA    TLBIT
```

```
5B30 AD0F01             LDA    RMAR          ; ESTABLISH BYTE AND BIT ADDRESSES OF TOP
5B33 8D0101             STA    X1CORD        ; RIGHT CORNER
5B36 AD1001             LDA    RMAR+1
5B39 8D0201             STA    X1CORD+1
5B3C 202155             JSR    PIXADR
5B3F A5EA               LDA    ADP1
5B41 8D1401             STA    TRBYT
5B44 A5EB               LDA    ADP1+1
5B46 8D1501             STA    TRBYT+1
5B49 AD1101             LDA    BTPT
5B4C 8D1901             STA    TRBIT
5B4F AD0B01             LDA    BMAR          ; ESTABLISH BYTE ADDRESS OF BOTTOM RIGHT
5B52 8D0301             STA    Y1CORD        ; CORNER; BIT ADDRESS IS SAME AS BIT
5B55 AD0C01             LDA    BMAR+1        ; ADDRESS OF TOP RIGHT CORNER
5B58 8D0401             STA    Y1CORD+1
5B5B 202155             JSR    PIXADR
5B5E A5EA               LDA    ADP1
5B60 8D1601             STA    BRBYT
5B63 A5EB               LDA    ADP1+1
5B65 8D1701             STA    BRBYT+1
5B68 60                 RTS                  ; RETURN
```

```
                        .PAGE  'CURSOR-BORDER LIMIT TEST ROUTINES'
               ;        CURSOR-BORDER LIMIT TEST ROUTINES
               ;        TESTS IF ENOUGH SPACE TO ALLOW CURSOR MOVEMENT IN ANY OF 4
               ;        RETURNS WITH POSITIVE OR ZERO RESULT IF ENOUGH
               ;        SPACE AND A NEGATIVE RESULT IF NOT ENOUGH SPACE.
               ;        SUBROUTINES USE A AND X

5B69 AD0301    DNTST:   LDA    Y1CORD       ; COMPUTE Y1CORD-BMAR-(2*CHHIW-2)
5B6C 38                 SEC
5B6D ED0B01             SBC    BMAR         ; SIGN OF RESULT
5B70 AA                 TAX                 ; - NOT OK
5B71 AD0401             LDA    Y1CORD+1     ; Z OK
5B74 ED0C01             SBC    BMAR+1       ; + OK
5B77 48                 PHA
5B78 8A                 TXA
5B79 38                 SEC
5B7A E914               SBC    #2*CHHIW-2
5B7C 68                 PLA
5B7D E900               SBC    #0
5B7F 60                 RTS

5B80 AD0901    UPTST:   LDA    TMAR         ; COMPUTE TMAR-Y1CORD-CHHIW
5B83 38                 SEC
5B84 ED0301             SBC    Y1CORD       ; SIGN OF RESULT
5B87 AA                 TAX                 ; - NOT OK
5B88 AD0A01             LDA    TMAR+1       ; Z OK
5B8B ED0401             SBC    Y1CORD+1     ; + OK
5B8E 48                 PHA
5B8F 8A                 TXA
5B90 38                 SEC
5B91 E90B               SBC    #CHHIW
5B93 68                 PLA
5B94 E900               SBC    #0
5B96 60                 RTS

5B97 AD0101    LFTST:   LDA    X1CORD       ; COMPUTE X1CORD-LMAR-CHWIDW
5B9A 38                 SEC
5B9B ED0D01             SBC    LMAR         ; SIGN OF RESULT
5B9E AA                 TAX                 ; - NOT OK
5B9F AD0201             LDA    X1CORD+1     ; Z OK
5BA2 ED0E01             SBC    LMAR+1       ; + OK
5BA5 48                 PHA
5BA6 8A                 TXA
5BA7 38                 SEC
5BA8 E906               SBC    #CHWIDW
5BAA 68                 PLA
5BAB E900               SBC    #0
5BAD 60                 RTS

5BAE AD0F01    RTTST:   LDA    RMAR         ; COMPUTE RMAR-X1CORD-(2*CHWIDW-2)
5BB1 38                 SEC
5BB2 ED0101             SBC    X1CORD       ; SIGN OF RESULT
5BB5 AA                 TAX                 ; - NOT OK
5BB6 AD1001             LDA    RMAR+1       ; Z OK
5BB9 ED0201             SBC    X1CORD+1     ; + OK
```

```
5BBC 48                  PHA
5BBD 8A                  TXA
5BBE 38                  SEC
5BBF E90A                SBC    #2*CHWIDW-2
5BC1 68                  PLA
5BC2 E900                SBC    #0
5BC4 60                  RTS
```

```
                       .PAGE  'CURSOR MANIPULATION ROUTINES'
               ;       CSRINS - INSERT A CURSOR AT THE CURRENT CURSOR POSITION
               ;                WHICH IS DEFINED BY X1CORD,Y1CORD
               ;       CSRDEL - REMOVE THE CURSOR WHICH IS ASSUMED TO BE AT THE
               ;                CURRENT CURSOR POSITION
               ;       CURSOR IS DISPLAYED AS AN UNDERLINE CHHIM+1 SCAN LINES BELOW
               ;       ACTUAL CHARACTER COORDINATES WHICH SPECIFY THE LOCATION OF THE
               ;       UPPER LEFT CORNER OF THE CHARACTER
               ;       INDEX REGISTERS PRESERVED

5BC5 A9F8      CSRINS: LDA    #X'F8        ; SET A FOR INSERTING THE CURSOR
5BC7 D002              BNE    CSR
5BC9 A900      CSRDEL: LDA    #0           ;  SET A FOR DELETING THE CURSOR

5BCB 48        CSR:    PHA                 ; SAVE A
5BCC AD0301            LDA    Y1CORD       ; TEMPORARILY SUBTRACT CHHIM FROM Y1CORD
5BCF 38                SEC
5BD0 E909              SBC    #CHHIM
5BD2 8D0301            STA    Y1CORD
5BD5 B003              BCS    CSR1
5BD7 CE0201            DEC    Y1CORD-1
5BDA 202155    CSR1:   JSR    PIXADR       ; COMPUTE ADDRESS OF CURSOR MARK
5BDD 68                PLA                 ; RESTORE SAVED A
5BDE 208558            JSR    MERGE5       ; MERGE CURSOR DATA WITH DISPLAY MEMORY
5BE1 AD0301            LDA    Y1CORD       ; RESTORE YICORD BY ADDING CHHIM BACK
5BE4 18                CLC
5BE5 6909              ADC    #CHHIM
5BE7 8D0301            STA    Y1CORD
5BEA 9003              BCC    CSR2
5BEC EE0401            INC    Y1CORD+1
5BEF 60        CSR2:   RTS                 ; RETURN

               ;       CSRR - MOVE CURSOR RIGHT ROUTINE
               ;       DO NOTHING IF AGAINST RIGHT MARGIN
               ;       USES X AND A

5BF0 20AE5B    CSRR:   JSR    RTTST        ; TEST IF CURSOR CAN GO RIGHT
5BF3 3014              BMI    CSRR2        ; GO RETURN IF NOT ENOUGH ROOM
5BF5 20C95B            JSR    CSRDEL       ; DELETE THE PRESENT CURSOR
5BF8 AD0101            LDA    X1CORD       ; ADD CHARACTER WINDOW WIDTH TO X
5BFB 18                CLC                 ; COORDINATE
5BFC 6906              ADC    #CHWIDW
5BFE 8D0101            STA    X1CORD
5C01 9003              BCC    CSRR1
5C03 EE0201            INC    X1CORD+1
5C06 20C55B    CSRR1:  JSR    CSRINS       ; DISPLAY CURSOR AT THE NEW LOCATION
5C09 60        CSRR2:  RTS                 ; RETURN

               ;       CSRL - MOVE CURSOR LEFT
               ;       DO NOTHING IF AGAINST LEFT MARGIN
               ;       USES A AND X

5C0A 20975B    CSRL:   JSR    LFTST        ; TEST IF CURSOR IS TOO FAR LEFT
5C0D 3014              BMI    CSRL2        ; JUMP IF IT IS TOO FAR LEFT
5C0F 20C95B            JSR    CSRDEL       ; DELETE THE PRESENT CURSOR
```

```
5C12 AD0101              LDA    X1CORD        ; SUBTRACT CHARACTER WINDOW WIDTH FROM
5C15 38                  SEC                  ; X COORDINATE
5C16 E906                SBC    #CHWIDW
5C18 8D0101              STA    X1CORD
5C1B B003                BCS    CSRL1
5C1D CE0201              DEC    X1CORD+1
5C20 20C55B    CSRL1:    JSR    CSRINS        ; DISPLAY CURSOR AT THE NEW LOCATION
5C23 60        CSRL2:    RTS                  ; RETURN

               ;         CSRU - CURSOR UP F
               ;         DO NOTHING IF AGAINST TOP MARGIN
               ;         USES A AND X

5C24 20805B    CSRU:     JSR    UPTST         ; TEST IF CURSOR IS TOO FAR UP
5C27 3014                BMI    CSRU2         ; JUMP IF IT IS TOO HIGH
5C29 20C95B              JSR    CSRDEL        ; DELETE THE PRESENT CURSOR
5C2C AD0301              LDA    Y1CORD        ; ADD CHARACTER WINDOW HEIGHT TO Y
5C2F 18                  CLC                  ; COORDINATE
5C30 690B                ADC    #CHHIW
5C32 8D0301              STA    Y1CORD
5C35 9003                BCC    CSRU1
5C37 EE0401              INC    Y1CORD+1
5C3A 20C55B    CSRU1:    JSR    CSRINS        ; DISPLAY CURSOR AT THE NEW LOCATION
5C3D 60        CSRU2:    RTS                  ; RETURN

               ;         CSRD - CURSOR DOWN
               ;         DO NOTHING IF AGAINST
               ;         USES X AND A

5C3E 20695B    CSRD:     JSR    DNTST         ; TEST IF CURSOR IS TOO FAR DOWN
5C41 3014                BMI    CSRD2         ; JUMP IF NOT ENOUGH SPACE
5C43 20C95B              JSR    CSRDEL        ; DELETE THE CURRENT CURSOR
5C46 AD0301              LDA    Y1CORD        ; SUBTRACT CHARACTER WINDOW HEIGHT FROM
5C49 38                  SEC                  ; Y COORDINATE
5C4A E90B                SBC    #CHHIW
5C4C 8D0301              STA    Y1CORD
5C4F B003                BCS    CSRD1
5C51 CE0401              DEC    Y1CORD+1
5C54 20C55B    CSRD1:    JSR    CSRINS        ; DISPLAY CURSOR AT THE NEW LOCATION
5C57 60        CSRD2:    RTS                  ; RETURN
```

```
                        .PAGE  'CONTROL CHARACTER DISPATCH TABLE'
               ;        CONTROL CHARACTER DISPATCH TABLE FOR DTEXT
               ;        FIRST BYTE IS ASCII CONTROL CHARACTER CODE
               ;        AND THIRD BYTES ARE ADDRESS OF SERVICE ROUTINE

5C58 0D        CCTAB:   .BYTE  X'0D         ; CR
5C59 8659               .WORD  CARRET-1     ; CARRIAGE RETURN
5C5B 0A                 .BYTE  X'0A         ; LF
5C5C 9B59               .WORD  LNFED-1      ; LINE FEED
5C5E 08                 .BYTE  X'08         ; BS
5C5F 4659               .WORD  CRL-1        ; BACKSPACE
5C61 0C                 .BYTE  X'0C         ; FF
5C62 4E5A               .WORD  FMFED-1      ; FORMFEED (CLEAR SCREEN)
5C64 0F                 .BYTE  X'0F         ; SI
5C65 5859               .WORD  BASUP-1      ; BASELINE SHIFT UP
5C67 0E                 .BYTE  X'0E         ; SO
5C68 6F59               .WORD  BASDN-1      ; BASELINE SHIFT DOWN
5C6A 11                 .BYTE  X'11         ; DC1
5C6B 4659               .WORD  CRL-1        ; CURSOR LEFT
5C6D 12                 .BYTE  X'12         ; DC2
5C6E 4059               .WORD  CRR-1        ; CURSOR RIGHT
5C70 13                 .BYTE  X'13         ; DC3
5C71 4C59               .WORD  CRU-1        ; CURSOR UP
5C73 14                 .BYTE  X'14         ; DC4
5C74 5259               .WORD  CRD-1        ; CURSOR DOWN
               CCTABE:                      ; END OF LIST

```

```
                         .PAGE    'CHARACTER FONT TABLE'
               ;         CHARACTER FONT TABLE 5 WIDE BY 7 HIGH PLUS 2 DESCENDING
               ;         ENTRIES IN ORDER STARTING AT ASCII BLANK
               ;         96 ENTRIES
               ;         EACH ENTRY CONTAINS 8 BYTES
               ;         SIGN BIT OF FIRST BYTE IS A DESCENDER FLAG, CHARACTER DESCENDS
               ;         2 ROWS IF IT IS A ONE
               ;         NEXT 7 BYTES ARE CHARACTER MATRIX, TOP ROW FIRST, LEFTMOST DOT
               ;         IS LEFTMOST IN BYTE

5C76 00000000  CHTB:     .BYTE  X'00,X'00,X'00,X'00    ; BLANK
5C7A 00000000            .BYTE  X'00,X'00,X'00,X'00
5C7E 00202020            .BYTE  X'00,X'20,X'20,X'20    ; !
5C82 20200020            .BYTE  X'20,X'20,X'00,X'20
5C86 00505050            .BYTE  X'00,X'50,X'50,X'50    ; "
5C8A 00000000            .BYTE  X'00,X'00,X'00,X'00
5C8E 005050F8            .BYTE  X'00,X'50,X'50,X'F8    ; #
5C92 50F85050            .BYTE  X'50,X'F8,X'50,X'50
5C96 002078A0            .BYTE  X'00,X'20,X'78,X'A0    ; X'
5C9A 7028F020            .BYTE  X'70,X'28,X'F0,X'20
5C9E 00C8C810            .BYTE  X'00,X'C8,X'C8,X'10    ; %
5CA2 20409898            .BYTE  X'20,X'40,X'98,X'98
5CA6 0040A0A0            .BYTE  X'00,X'40,X'A0,X'A0    ; &
5CAA 40A89068            .BYTE  X'40,X'A8,X'90,X'68
5CAE 00303030            .BYTE  X'00,X'30,X'30,X'30    ; '
5CB2 00000000            .BYTE  X'00,X'00,X'00,X'00
5CB6 00204040            .BYTE  X'00,X'20,X'40,X'40    ; (
5CBA 40404020            .BYTE  X'40,X'40,X'40,X'20
5CBE 00201010            .BYTE  X'00,X'20,X'10,X'10    ; )
5CC2 10101020            .BYTE  X'10,X'10,X'10,X'20
5CC6 0020A870            .BYTE  X'00,X'20,X'A8,X'70    ; *
5CCA 2070A820            .BYTE  X'20,X'70,X'A8,X'20
5CCE 00002020            .BYTE  X'00,X'00,X'20,X'20    ; +
5CD2 F8202000            .BYTE  X'F8,X'20,X'20,X'00
5CD6 80000000            .BYTE  X'80,X'00,X'00,X'00    ; ,
5CDA 30301020            .BYTE  X'30,X'30,X'10,X'20
5CDE 00000000            .BYTE  X'00,X'00,X'00,X'00    ; -
5CE2 F8000000            .BYTE  X'F8,X'00,X'00,X'00
5CE6 00000000            .BYTE  X'00,X'00,X'00,X'00    ; .
5CEA 00003030            .BYTE  X'00,X'00,X'30,X'30
5CEE 00080810            .BYTE  X'00,X'08,X'08,X'10    ; /
5CF2 20408080            .BYTE  X'20,X'40,X'80,X'80
5CF6 00609090            .BYTE  X'00,X'60,X'90,X'90    ; 0
5CFA 90909060            .BYTE  X'90,X'90,X'90,X'60
5CFE 00206020            .BYTE  X'00,X'20,X'60,X'20    ; 1
5D02 20202070            .BYTE  X'20,X'20,X'20,X'70
5D06 00708810            .BYTE  X'00,X'70,X'88,X'10    ; 2
5D0A 204080F8            .BYTE  X'20,X'40,X'80,X'F8
5D0E 00708808            .BYTE  X'00,X'70,X'88,X'08    ; 3
5D12 30088870            .BYTE  X'30,X'08,X'88,X'70
5D16 00103050            .BYTE  X'00,X'10,X'30,X'50    ; 4
5D1A 90F81010            .BYTE  X'90,X'F8,X'10,X'10
5D1E 00F880F0            .BYTE  X'00,X'F8,X'80,X'F0    ; 5
5D22 080808F0            .BYTE  X'08,X'08,X'08,X'F0
5D26 00708080            .BYTE  X'00,X'70,X'80,X'80    ; 6
```

```
5D2A F0888870            .BYTE  X'F0,X'88,X'88,X'70
5D2E 00F80810            .BYTE  X'00,X'F8,X'08,X'10    ; 7
5D32 20408080            .BYTE  X'20,X'40,X'80,X'80
5D36 00708888            .BYTE  X'00,X'70,X'88,X'88    ; 8
5D3A 70888870            .BYTE  X'70,X'88,X'88,X'70
5D3E 00708888            .BYTE  X'00,X'70,X'88,X'88    ; 9
5D42 78080870            .BYTE  X'78,X'08,X'08,X'70
5D46 00303000            .BYTE  X'00,X'30,X'30,X'00    ; :
5D4A 00003030            .BYTE  X'00,X'00,X'30,X'30
5D4E 00303000            .BYTE  X'00,X'30,X'30,X'00    ; ;
5D52 30301020            .BYTE  X'30,X'30,X'10,X'20
5D56 00102040            .BYTE  X'00,X'10,X'20,X'40    ; LESS THAN
5D5A 80402010            .BYTE  X'80,X'40,X'20,X'10
5D5E 000000F8            .BYTE  X'00,X'00,X'00,X'F8    ; =
5D62 00F80000            .BYTE  X'00,X'F8,X'00,X'00
5D66 00402010            .BYTE  X'00,X'40,X'20,X'10    ; GREATER THAN
5D6A 08102040            .BYTE  X'08,X'10,X'20,X'40
5D6E 00708808            .BYTE  X'00,X'70,X'88,X'08    ; ?
5D72 10200020            .BYTE  X'10,X'20,X'00,X'20
5D76 00708808            .BYTE  X'00,X'70,X'88,X'08    ; @
5D7A 68A8A8D0            .BYTE  X'68,X'A8,X'A8,X'D0
5D7E 00205088            .BYTE  X'00,X'20,X'50,X'88    ; A
5D82 88F88888            .BYTE  X'88,X'F8,X'88,X'88
5D86 00F04848            .BYTE  X'00,X'F0,X'48,X'48    ; B
5D8A 704848F0            .BYTE  X'70,X'48,X'48,X'F0
5D8E 00708880            .BYTE  X'00,X'70,X'88,X'80    ; C
5D92 80808870            .BYTE  X'80,X'80,X'88,X'70
5D96 00F04848            .BYTE  X'00,X'F0,X'48,X'48    ; D
5D9A 484848F0            .BYTE  X'48,X'48,X'48,X'F0
5D9E 00F88080            .BYTE  X'00,X'F8,X'80,X'80    ; E
5DA2 F08080F8            .BYTE  X'F0,X'80,X'80,X'F8
5DA6 00F88080            .BYTE  X'00,X'F8,X'80,X'80    ; F
5DAA F0808080            .BYTE  X'F0,X'80,X'80,X'80
5DAE 00708880            .BYTE  X'00,X'70,X'88,X'80    ; G
5DB2 B8888870            .BYTE  X'B8,X'88,X'88,X'70
5DB6 00888888            .BYTE  X'00,X'88,X'88,X'88    ; H
5DBA F8888888            .BYTE  X'F8,X'88,X'88,X'88
5DBE 00702020            .BYTE  X'00,X'70,X'20,X'20    ; I
5DC2 20202070            .BYTE  X'20,X'20,X'20,X'70
5DC6 00381010            .BYTE  X'00,X'38,X'10,X'10    ; J
5DCA 10109060            .BYTE  X'10,X'10,X'90,X'60
5DCE 008890A0            .BYTE  X'00,X'88,X'90,X'A0    ; K
5DD2 C0A09088            .BYTE  X'C0,X'A0,X'90,X'88
5DD6 00808080            .BYTE  X'00,X'80,X'80,X'80    ; L
5DDA 808080F8            .BYTE  X'80,X'80,X'80,X'F8
5DDE 0088D8A8            .BYTE  X'00,X'88,X'D8,X'A8    ; M
5DE2 A8888888            .BYTE  X'A8,X'88,X'88,X'88
5DE6 008888C8            .BYTE  X'00,X'88,X'88,X'C8    ; N
5DEA A8988888            .BYTE  X'A8,X'98,X'88,X'88
5DEE 00708888            .BYTE  X'00,X'70,X'88,X'88    ; O
5DF2 88888870            .BYTE  X'88,X'88,X'88,X'70
5DF6 00F08888            .BYTE  X'00,X'F0,X'88,X'88    ; P
5DFA F0808080            .BYTE  X'F0,X'80,X'80,X'80
5DFE 00708888            .BYTE  X'00,X'70,X'88,X'88    ; Q
5E02 88A89068            .BYTE  X'88,X'A8,X'90,X'68
```

```
5E06 00F08888            .BYTE  X'00,X'F0,X'88,X'88    ; R
5E0A F0A09088            .BYTE  X'F0,X'A0,X'90,X'88
5E0E 00788080            .BYTE  X'00,X'78,X'80,X'80    ; S
5E12 700808F0            .BYTE  X'70,X'08,X'08,X'F0
5E16 00F82020            .BYTE  X'00,X'F8,X'20,X'20    ; T
5E1A 20202020            .BYTE  X'20,X'20,X'20,X'20
5E1E 00888888            .BYTE  X'00,X'88,X'88,X'88    ; U
5E22 88888870            .BYTE  X'88,X'88,X'88,X'70
5E26 00888888            .BYTE  X'00,X'88,X'88,X'88    ; V
5E2A 50502020            .BYTE  X'50,X'50,X'20,X'20
5E2E 00888888            .BYTE  X'00,X'88,X'88,X'88    ; W
5E32 A8A8D888            .BYTE  X'A8,X'A8,X'D8,X'88
5E36 00888850            .BYTE  X'00,X'88,X'88,X'50    ; X
5E3A 20508888            .BYTE  X'20,X'50,X'88,X'88
5E3E 00888850            .BYTE  X'00,X'88,X'88,X'50    ; Y
5E42 20202020            .BYTE  X'20,X'20,X'20,X'20
5E46 00F80810            .BYTE  X'00,X'F8,X'08,X'10    ; Z
5E4A 204080F8            .BYTE  X'20,X'40,X'80,X'F8
5E4E 00704040            .BYTE  X'00,X'70,X'40,X'40    ; LEFT BRACKET
5E52 40404070            .BYTE  X'40,X'40,X'40,X'70
5E56 00808040            .BYTE  X'00,X'80,X'80,X'40    ; BACKSLASH
5E5A 20100808            .BYTE  X'20,X'10,X'08,X'08
5E5E 00701010            .BYTE  X'00,X'70,X'10,X'10    ; RIGHT BRACKET
5E62 10101070            .BYTE  X'10,X'10,X'10,X'70
5E66 00205088            .BYTE  X'00,X'20,X'50,X'88    ; CARROT
5E6A 00000000            .BYTE  X'00,X'00,X'00,X'00
5E6E 00000000            .BYTE  X'00,X'00,X'00,X'00    ; UNDERLINE
5E72 000000F8            .BYTE  X'00,X'00,X'00,X'F8

5E76 00C06030            .BYTE  X'00,X'C0,X'60,X'30    ; GRAVE ACCENT
5E7A 00000000            .BYTE  X'00,X'00,X'00,X'00
5E7E 00006010            .BYTE  X'00,X'00,X'60,X'10    ; A (LC)
5E82 70909068            .BYTE  X'70,X'90,X'90,X'68
5E86 008080F0            .BYTE  X'00,X'80,X'80,X'F0    ; B (LC)
5E8A 888888F0            .BYTE  X'88,X'88,X'88,X'F0
5E8E 00000078            .BYTE  X'00,X'00,X'00,X'78    ; C (LC)
5E92 80808078            .BYTE  X'80,X'80,X'80,X'78
5E96 00080878            .BYTE  X'00,X'08,X'08,X'78    ; D (LC)
5E9A 88888878            .BYTE  X'88,X'88,X'88,X'78
5E9E 00000070            .BYTE  X'00,X'00,X'00,X'70    ; E (LC)
5EA2 88F08078            .BYTE  X'88,X'F0,X'80,X'78
5EA6 00304040            .BYTE  X'00,X'30,X'40,X'40    ; F (LC)
5EAA E0404040            .BYTE  X'E0,X'40,X'40,X'40
5EAE 80708888            .BYTE  X'80,X'70,X'88,X'88    ; G (LC)
5EB2 98680870            .BYTE  X'98,X'68,X'08,X'70
5EB6 008080B0            .BYTE  X'00,X'80,X'80,X'B0    ; H (LC)
5EBA C8888888            .BYTE  X'C8,X'88,X'88,X'88
5EBE 00200060            .BYTE  X'00,X'20,X'00,X'60    ; I (LC)
5EC2 20202070            .BYTE  X'20,X'20,X'20,X'70
5EC6 80701010            .BYTE  X'80,X'70,X'10,X'10    ; J (LC)
5ECA 10109060            .BYTE  X'10,X'10,X'90,X'60
5ECE 00808090            .BYTE  X'00,X'80,X'80,X'90    ; K (LC)
5ED2 A0C0A090            .BYTE  X'A0,X'C0,X'A0,X'90
5ED6 00602020            .BYTE  X'00,X'60,X'20,X'20    ; L (LC)
5EDA 20202020            .BYTE  X'20,X'20,X'20,X'20
```

```
5EDE 000000D0           .BYTE  X'00,X'00,X'00,X'D0    ; M (LC)
5EE2 A8A8A8A8           .BYTE  X'A8,X'A8,X'A8,X'A8
5EE6 000000B0           .BYTE  X'00,X'00,X'00,X'B0    ; N (LC)
5EEA C8888888           .BYTE  X'C8,X'88,X'88,X'88
5EEE 00000070           .BYTE  X'00,X'00,X'00,X'70    ; O (LC)
5EF2 88888870           .BYTE  X'88,X'88,X'88,X'70
5EF6 80F08888           .BYTE  X'80,X'F0,X'88,X'88    ; P (LC)
5EFA 88F08080           .BYTE  X'88,X'F0,X'80,X'80
5EFE 80788888           .BYTE  X'80,X'78,X'88,X'88    ; Q (LC)
5F02 88780808           .BYTE  X'88,X'78,X'08,X'08
5F06 000000B0           .BYTE  X'00,X'00,X'00,X'B0    ; R (LC)
5F0A C8808080           .BYTE  X'C8,X'80,X'80,X'80
5F0E 00000078           .BYTE  X'00,X'00,X'00,X'78    ; S (LC)
5F12 807008F0           .BYTE  X'80,X'70,X'08,X'F0
5F16 004040E0           .BYTE  X'00,X'40,X'40,X'E0    ; T (LC)
5F1A 40405020           .BYTE  X'40,X'40,X'50,X'20
5F1E 00000090           .BYTE  X'00,X'00,X'00,X'90    ; U (LC)
5F22 90909068           .BYTE  X'90,X'90,X'90,X'68
5F26 00000088           .BYTE  X'00,X'00,X'00,X'88    ; V (LC)
5F2A 88505020           .BYTE  X'88,X'50,X'50,X'20
5F2E 000000A8           .BYTE  X'00,X'00,X'00,X'A8    ; W (LC)
5F32 A8A8A850           .BYTE  X'A8,X'A8,X'A8,X'50
5F36 00000088           .BYTE  X'00,X'00,X'00,X'88    ; X (LC)
5F3A 50205088           .BYTE  X'50,X'20,X'50,X'88
5F3E 80888888           .BYTE  X'80,X'88,X'88,X'88    ; Y (LC)
5F42 50204080           .BYTE  X'50,X'20,X'40,X'80
5F46 000000F8           .BYTE  X'00,X'00,X'00,X'F8    ; Z (LC)
5F4A 102040F8           .BYTE  X'10,X'20,X'40,X'F8
5F4E 00102020           .BYTE  X'00,X'10,X'20,X'20    ; LEFT BRACE
5F52 60202010           .BYTE  X'60,X'20,X'20,X'10
5F56 00202020           .BYTE  X'00,X'20,X'20,X'20    ; VERTICAL BAR
5F5A 20202020           .BYTE  X'20,X'20,X'20,X'20
5F5E 00402020           .BYTE  X'00,X'40,X'20,X'20    ; RIGHT BRACE
5F62 30202040           .BYTE  X'30,X'20,X'20,X'40
5F66 0010A840           .BYTE  X'00,X'10,X'A8,X'40    ; TILDA
5F6A 00000000           .BYTE  X'00,X'00,X'00,X'00
5F6E 00A850A8           .BYTE  X'00,X'A8,X'50,X'A8    ; RUBOUT
5F72 50A850A8           .BYTE  X'50,X'A8,X'50,X'A8

0000                    .END
NO ERROR LINES
```