Micro Technology Unlimited

K-1008-5C

VISIBLE MEMORY DEMONSTRATIONS
VISIBLE MEMORY TO AIM-65 BASIC INTERFACE
VISIBLE MEMORY SUBROUTINE PACKAGE

JANUARY, 1980

AIM-65 VISIBLE MEMORY SUPPORT PACKAGE

The goal of this software package is to give the AIM-65 owner full access to the capabilities of the Visible Memory graphic display board both through AIM-65 BASIC and AIM-65 machine language. Additionally we have provided software "hooks" for the Visible Memory Screen Print program (K-1009-1) which will provide a print-out of the screen image on the AIM-65 printer dot-for-dot with no printer modifications required. Use of this package will give the AIM-65 graphics capabilities unmatched by any of the packaged computers on the market.

All of the software contained herein (except the BASIC demonstration program) will run on a 4K AIM (6 additional 2114 RAM chips installed). However we suggest additional memory for the maximum utilization of BASIC with the Visible Memory. In particular, our K-1016 16K memory will expand the amount of memory usable for BASIC programs by a factor of nearly 13 over that available on a 4K AIM. The BASIC interface program is supplied assembled for both 4K and 20K AIM's although only the 20K listing is available.

The enclosed cassette has the following programs recorded on it in AIM-65 format with a gap length of 08 except BDEMO which has a gap length of 24.

NAME	ADDRESSES	DESCRIPTION
1. SWIRL	0200-059B	Swirl demonstration program, entry at 0200, 0222
2. VLIFE	0200-05E0	Life demonstration program, entry at 0200
3. BAS20	4772-4FFF	BASIC interface program for 20K AIM, see text for entry points.
4. BAS04	0772-0FFF	BASIC interface program for 4K AIM, see text for entry points.
5. BDEMO	-	Demonstration program in BASIC. Will not load on a 4K AIM. See text and listing.
6. SDTXT	0B69-0FFF	Simplified display text subroutine, entry at 0B69.
7. GRAPH	0682-0FFF	Full display text and graphics routines. See text and appendix for entry points.

Please read the description of each program for loading and running instructions. All programs have been tested and many of them have been successfully used on the KIM-1 for over two years. Any remaining bugs should be subtle and we would appreciate hearing about those that the customer finds. Please include a complete description of the conditions that cause the bug to appear and document the bug if possible with AIM printouts, etc.

Since the high speed AIM-65 tape format is somewhat more sensitive to cassette recorder differences than KIM-1 format, it may be necessary to try more than one cassette recorder or experiment with the head alignment. It is suggested that a local copy be made of each program the first time it is successfully loaded to protect against loss and tape degradation. Place the enclosed copyright label on the backup tape. Unreadable cassettes will be replaced but since they will be recorded on the same machine, the only variable will be tape dropouts.

RUNNING THE SWIRL DEMONSTRATION

Swirl is a demonstration program written in 6502 machine language that generates a variety of interesting spiral and spiderweb like patterns on the screen. Two parameters determine the appearance of the pattern and a third either includes or suppresses lines connecting the computed points. The user may set these parameters manually and then have a single pattern computed and held or another routine may be invoked which uses a random number generator to select the parameters thus giving an endless series of different patterns.

The program is based on the differential equation for a circle which tends toward an 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. Negative values of FREQ cause the spiral to rotate clockwise rather than counterclockwise. DAMP is also a double precision signed binary fraction but it must be positive for proper operation. If it is negative, the oscillation will build up instead of dying out until the fixed point arithmetic routines overflow creating a random 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. Note that all double precision values are stored with the least significant byte at the lower address, i.e., backwards like addresses in the 6502.

To run the program, first load it into AIM memory from the enclosed cassette. The file name is SWIRL and it loads from 0200 to 059B. Page zero locations from 0000 to 002E are also used for temporary storage as well as a few stack locations. The remainder of memory is unchanged. Before executing the program the default address for the Visible Memory <u>must</u> be changed if the user has it at an address different from 8000-9FFF. If it is different, use the AIM monitor to store the page address of the Visible Memory in location 0280.

Default values for all of the parameters have been supplied. To see the default pattern, start execution at address 0200. This is accomplished by first issuing the * command to the AIM and giving an answer of 0200. Then type G and a carriage return. 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 AIM monitor by pressing the reset button.

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 start execution at location 200C which will not set the default parameters. 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 (02 into 0001 and 11 into 0002) and DAMP to 7FC0 (C0 into 0003 and 7F into 0004) and execute SWIRL again. Interrupt 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 (locations 0005 and 0006) 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 0222) 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!

DESCRIPTION OF THE VLIFE DEMONSTRATION

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

A. Central cell is alive
   1. 0 or 1 live neighbors, the central cell dies of starvation
   2. 2 or 3 live neighbors, the central cell lives on
   3. 4 or more live neighbors, the central cell dies of overcrowding

B. Central cell is not alive
   1. Fewer than or more than 3 live neighbors, the central cell remains dead
   2. Exactly 3 live neighbors, a birth is recorded in the central cell.

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 AIM, 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 possible as well 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.

RUNNING THE VLIFE DEMONSTRATION

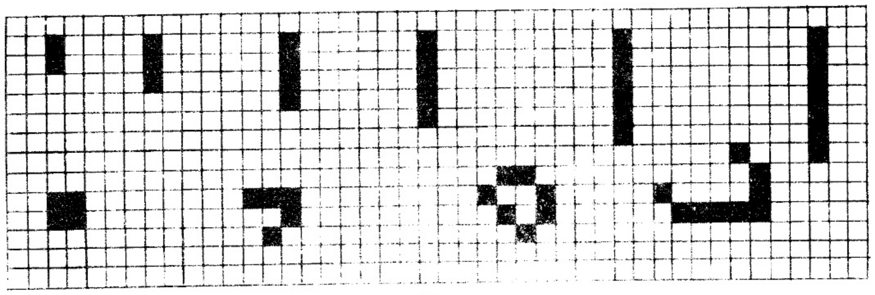
To run the program, first load it into AIM memory from the enclosed cassette. The file name is VLIFE and it loads from 0200 to 05E0. Page zero locations from 0000 to 0099 are also used for temporary storage as well as a few stack locations. The remainder of memory is unchanged. Before executing the program _the address of the Visible Memory must be specified_. Using the AIM monitor, store the page address of the Visible Memory in location 0245. If the Visible Memory is addressed at 8000, this is not necessary as an 80 has already been stored there in the cassette program.

The Life demonstration program has a single entry point which initializes things, draws a default initial configuration of cells, and then goes through the life history of that configuration. At any time the sequence of generations may be interrupted by pressing and holding the "S" key on the AIM keyboard until the generations stop and a single dot of the display (the "graphic cursor") starts flashing. At this point several of the keyboard keys are active which allow the entry of user defined initial patterns.

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 the canned demonstration completely (leave it as a surprise!). To do this, start the program at 0200 using the AIM monitor and then immediately hold down the S key until the graphic cursor is seen in the center of the screen. The letters "LIFE" should be drawn, one generation of life executed, and then the cursor should be visible. 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 AIM (do _not_ hold the shift key down). 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 "X" key will kill the cell under the cursor. The graphic cursor may be moved around by using the U key for Up, D for Down, L for Left, and R for Right. Also the W key will Wipe out the entire screen. Finally, the G key will Go to executing life on whatever pattern is showning on the screen at the time. The evolution may be stopped at any time with the S key and the colony of cells edited as desired.

Initial patterns may also be entered using the AIM monitor to write directly into the Visible Memory. Other methods include reading the pattern from cassette tape using the AIM monitor or generating the pattern with another program (such as SWIRL), loading LIFE, and executing it (try one of the checkerboards generated by the memory test program in the Visible Memory manual). The entry point LIFE (0265) starts the evolution process and should be used if initial patterns are generated in any of these ways.

Try the initial patterns shown below and note their fate.

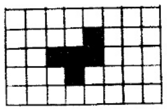


The patterns that evolve from those on the previous page are fundamental and well known to every Life fan. They are so common in the result of many initial patterns that they have been given descriptive names. See if you can match the following names with the corresponding final patterns: Block, Honeyfarm,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

Another interesting pastime is to note the life history (number and configuration 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. If the screen print program (K-1009-1) has been purchased, it may be executed at any time with the AIM monitor to provide a written record of particularly interesting Life histories. To use it, stop the evolution with the S key and reset the AIM. Then print the screen. The evolution may be restarted by going to 0265.

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 the canned demonstration pattern completely, try the very simple initial pattern shown below. As it expands and differentiates, it will leave a litter of the fundamental patterns discussed earlier.



To execute the demonstration, simply go to 0200. An initial pattern will be generated and the Life algorithm will be executed on it. The initial pattern generated by DEMO may be changed by altering the table of coordinates that starts at LIST (0335). Note that the simplified 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.

BAS04 AND BAS20 BASIC INTERFACE PROGRAMS

Of all of the programs in this package the BASIC interface program will probably be of most interest. The purpose of the BASIC interface program is two-fold. Most importantly it gives the user an easy way to write application programs which use high resolution graphics, particularly for complex drawings and plotting mathematical and other detailed curves. Text may also be freely mixed in with the graphics without the restrictions normally encountered in packaged computer systems. The second benefit of using this package is more convenient conversation with the BASIC interpreter itself while entering and editing programs. The wide (53 characters) screen will hold most BASIC statements on one line while the 22 line display format allows whole routines to be viewed at once without having to constantly print them out. Also, the execution of a LIST command or an executing program may be temporarily suspended so the screen may be read and then resumed by using two special keyboard keys. Finally, the text dis- play part of the interface also works with other AIM-65 functions such as the monitor, editor, and assembler making the overall system much easier to use. Pro- visions are made in the software to link to the MTU screen print program (K-1009-1) so that screen prints can be initiated by BASIC statements and even complete program listings with 53 character lines printed automatically.

Two versions of the interface program are supplied on the enclosed cassette. The file called BAS04 is to be used on AIM's with only 4K of read/write memory. BAS20 is to be used on AIM's with 20K of contiguous memory from 0000 to 4FFF. This 20K memory capacity may be conveniently provided by an MTU K-1016 16K memory board. Both versions of the program provide identical functions; the only difference is the memory locations occupied. For convenience in the following discussions. The 20K version addresses are given. For the corresponding 4K version addresses, simply replace the first digit of the address with a zero.

BAS04 AND BAS20 LOADING INSTRUCTIONS

1. Be sure your system has a Visible Memory and that it is working properly. When connected to a monitor and first powered-up a random dot pattern should appear on the screen. Note the address of the Visible Memory for use later (8000 is recommended).

2. Rewind the cassette and instruct the AIM to load either BAS04 for a 4K AIM or BAS20 for a 20K AIM.

3. After successfully loading the program (and saving it on your own tape with copyright label), start execution at one of the following addresses that match the location of your Visible Memory:

2000	Start at	0772 (BAS04)	or	4772 (BAS20)
4000	Start at	0776 (BAS04)	or	4776 (BAS20)
6000	Start at	077A (BAS04)	or	477A (BAS20)
8000	Start at	077E (BAS04)	or	477E (BAS20)

4. The screen should blank and the top line should indicate that a break instruction has been executed which can be ignored. Also while this package is in control of the AIM-65 output, you should ignore the AIM-65 display. The printer will continue to operate normally. From now on until power is shut off or the "PUTBAK" program is executed all AIM output will go to the screen.

5. Type the 5 key to enter the BASIC interpreter.

6. Answer the MEMORY SIZE? question with 1905 for a 4K AIM or 18289 for a 20K AIM. DO NOT KEY IN RETURN WITHOUT THE SIZE NUMBER OR THE INTERFACE PROGRAM WILL BE WIPED OUT AND THE AIM WILL CRASH IN SUCH A WAY THAT POWER-OFF IS THE ONLY WAY TO RECOVER.

7. Answer the TERMINAL WIDTH? question with 53.

8. BASIC will respond with: 1375 BYTES FREE for a 4K AIM or 17757 for a 20K AIM.

9. If you have a 20K AIM, you may then load in the demonstration program (BDEMO) from cassette with the LOAD command (see the AIM BASIC manual). The program listing will appear while loading but the tape record numbers will be interspersed; just ignore these. If you have a 4K AIM, the demonstration program listing elsewhere in this manual will have to be consulted and just a couple of the demonstrations typed in at once since memory space is limited.

10. You may list the entire program by typing LIST 0-9999 and carriage return. To temporarily stop the listing, hit the F2 key. To resume listing type F3. To terminate the listing type F1. These keys also function while a BASIC program is running and producing printed output. Be careful however not to type F2 while BASIC is waiting for a command, it may crash the system.

11. Run the demonstration program by typing RUN followed by a carriage return. The program will run for approximately 1.5 hours with a long pause between each demonstration so that the screen can be examined. Most of the time is spent in the prime number mosaic demonstration. An infinite loop has been programmed following the prime number mosiac so it will be necessary to hit F1 to interrupt the program and return to BASIC.

BAS04 AND BAS20 TEXT DISPLAY INTERFACE

The text display routine that is now in control of your AIM can be used with the monitor, editor, and assembler as well as BASIC. It is particularly effective with the editor since normal text and assembler source statements now have a wide line to work with. All AIM functions should operate normally with the exception of the tape load and dump functions which do funny things with the AIM display to show the record number and therefore do not display properly on the screen. Tape dump and load with BASIC works satisfactorily but when using the AIM monitor it is recommended that the text display interface be disabled by executing PUTBAK (479E or 079E) before doing the tape operations. If this is not done, the program will crash 1 out of 10 times. The interface may be restored by executing the same address used in step 3 above. Note that when doing a tape dump with BASIC the MORE? question is not asked and that tape writing is complete when the cursor returns to the left of the screen a couple of lines below the string of record numbers that will be displayed. Also note that a tape read error will exit BASIC and put you in the AIM monitor. Type a 6 to get back to BASIC.

The interface program uses the upper invisible 192 bytes of the Visible Memory (not affected by screen clear) and locations 0106 and 0107 for storage. In order to preserve the information in 0106 and 0107 _the fourth breakpoint address must not be defined_. This should not cause any particular hardship.

If the AIM Print Package (K-1009-1) has been purchased, it can be linked with this program (instructions in the K-1009-1 manual). When this is done a printout of the Visible Memory may be initiated in one of two ways. One way is from a BASIC program with the USR(10) function for a quick screen print or a USR(11) for a quality screen print. Another way is by setting location 0106 (262 decimal) to a 1 for quick print or 2 for quality print. Then every time the display fills up with text (i.e., a scroll would take place), the screen is printed. Following printing, the screen is cleared and the cursor homed to top left in preparation for another screenful of text. Storing anything else in location 0106 restores normal scrolling display action. This "print flag" is active regardless of the source of the text and is also effective with the AIM text editor. In preparation for the first page, a control/L character may be typed to home the cursor. To get the last page printed, control/J characters may be entered until the print is triggered. If the K-1009 package is not linked in, screen printing is just ignored.

BAS04 AND BAS20 PLOTTING ROUTINES

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

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

```
1 POKE 4,169: POKE 5,71  (20k AIM)        POKE 4,169: POKE 5,7  (4k AIM)
2 X1%=0: Y1%=0: X2%=0: Y2%=0
```

Using statement numbers 1 and 2 insure that these statements are executed first whenever a RUN[1] command is given to BASIC. This causes the USR entry into the interface program to be defined in locations 4 and 5 and the integer variables X1%, Y1%, X2%, and Y2% to be placed first in the variable table where the machine language plotting routines can easily find them.

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

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

USR(0)	Clear the screen
USR(1)	Plot a white point at X1%,Y1%
USR(2)	Plot a white line from X1%,Y1% to X2%,Y2%
USR(3)	Erase the point at X1%,Y1%
USR(4)	Erase the line running from X1%,Y1% to X2%,Y2%
USR(5)	Returns the color of the point at X1%,Y1% black=0, white=1
USR(6)	Clear cursor on screen
USR(7)	Set cursor on screen
USR(8)	Flip the state of the point at X1%,Y1%
USR(9)	Flip all points along the line from X1%,Y1% to X2%,Y2%.
USR(10)	Perform a quick print of what is on the screen(requires K-1009-1)
USR(11)	Perform a quality print of what is on the screen(requires K-1009-1)

Note that USR(x) is a function subprogram, not a statement. A convenient method of using it to plot is to code the BASIC statement: Z=USR(x) where x is the argument corresponding to the desired plot function and Z is a dummy variable not used for anything else, (except for USR(5) which puts the state of the pixel in Z). The line plot and erase routines copy X2% into X1% and Y2% into Y1% when they execute. This allows a chain of end-to-end lines to be plotted or erased by simply changing X2% and Y2% for each successive endpoint after the first.

EXAMPLE PROGRAM SEGMENTS

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

1. To clear the screen before plotting:

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

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

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

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

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

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

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

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

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

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

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

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

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

7. To print the image that exists on the screen at this time (K-1009-1 required)

```
10 Z=USR(10)    (quick print)
```

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

BAS04 AND BAS20 TEXT DISPLAY ROUTINE

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

The text display routine in the package, SDTXT, keeps two variables of its own which identify the location of the text cursor on the screen. The character number is stored in location XFE9 (where X is the first digit of the Visible Memory address) and varies from 0 for the left screen edge to 52 decimal for the right screen edge. The line number is kept in location XFEA and varies from 0 for the top line to 21 decimal for the bottom line. For the Visible Memory at 8000, the decimal addresses of the cursor are 40937 for the character number and 40938 for the line number.

BASIC also has its own character number which is stored in location 11 (17 decimal) and ranges from 0 to 52 for a terminal width of 53. Normally BASIC's character number and SDTXT's character number agree. Every carriage-return/line-feed issued by BASIC sets both character numbers to zero and increments SDTXT's line number. When the line number tries to go beyond 21 the screen contents are scrolled upward by 9 raster lines instead.

Putting text at arbitrary locations on the screen basically amounts to POKEing the desired character and line numbers into memory at the addresses given above. The text is then generated with print statements. The coordinates of the center of a character at character position C and line number L are: X=6*C+2 Y=194-9*L; C=(X-2)/6 L=(194-Y)/9. Characters extend 2 pixels either side of center widthwise and 3 pixels either side of center heightwise. A semicolon terminator should be used after each element printed to prevent BASIC from following it with a carriage return. Also, BASIC's character number at location 17. should be reset to zero before the accumulated output exceeds 53 characters or else BASIC will insert a carriage-return/line-feed anyway. Finally be aware that when numbers are printed with the semicolon terminator that a blank is printed following the number and that positive numbers are preceeded by a blank.

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

For example, if the caption "Market Index" is desired to start at X=70 Y=180 the following BASIC statements should be coded (if the VM starts at 8000):

```
10 Z=USR(6)
20 POKE 40937,11
30 POKE 40938,2
40 POKE 17,0
50 Z=USR(7)
60 PRINT "Market Index";
```

Character number 11 and line number 2 are closest to the desired starting point of X=70 Y=180. Note that lower case letters are available and may be part of a literal field with no problems (however they cannot be typed in with the AIM keyboard!). The demonstration program can be consulted for additional examples of text output.

BASIC DEMONSTRATION PROGRAM

The BASIC demonstration program supplied with this software package is designed to illustrate the use of the plotting and text display functions. It is intended to be easy to read and understand rather than illustrate techniques for program compression and speed enhancement. The program is composed of five different demonstrations that execute in sequence with a long pause between each demonstration. The fifth ends with an infinite loop which must be interrupted to return to BASIC. If a 4K AIM is used, each demonstration will have to be typed in and run individually to fit into memory.

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

The second program illustrates vector plotting by creating a 31 point star. Since the string of endpoints is connected, the line drawing routine's property of updating X1% and Y1% from X2% and X2% values is utilized to advantage. However the first point is a special case. To handle the first point, a variable called FP is initially set to 1. As each new endpoint is computed, the value of FP is interrogated. If it is found to be non-zero, the endpoints are forced to be equal which effectively moves the "pen" without drawing a line from where it was. After the first point is plotted, FP is set to zero thus allowing vectors to be drawn between all successive points.

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

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

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

BASIC DEMONSTRATION PROGRAMS

```
1 POKE 4,169: POKE 5,71
2 X1%=0: Y1%=0: X2%=0: Y2%=0
3 REM PREVIOUS STATEMENTS REQUIRED TO DEFINE
4 REM GRAPHIC COORDINATES AND SET USRLOC
5 CX=40937: CY=CX+1: REM TEXT CSR ADDR FOR VM AT 8000
10 REM CLEAR THE SCREEN
11 Z=USR(0)


100 REM DEMONSTRATION OF POINT PLOT
110 REM PLOT A CIRCLE IN DEAD CENTER OF SCREEN USING
120 REM 250 POINTS
130 FOR I=0 TO 250
140 A=6.28318*I/250
150 X1%=100*COS(A)+160
160 Y1%=100*SIN(A)+100
170 Z=USR(1)
180 NEXT I
190 GOSUB 9000


200 REM DEMONSTRATION OF VECTOR PLOT
210 Z=USR(0): REM CLEAR SCREEN
220 FP=1: REM SET FIRST POINT FLAG
230 FOR I=0 TO 31
240 A=13*I*6.2831828/31
250 X2%=150*COS(A)+160
260 Y2%=100*SIN(A)+100
270 IF FP<>1 THEN GOTO 290
280 X1%=X2%: Y1%=Y2%: FP=0
290 Z=USR(2)
300 NEXT I
310 GOSUB 9000


400 REM DEMONSTRATION OF VECTOR ERASE
410 FP=1
420 FOR I=0 TO 31
430 A=13*I*6.2831828/31
440 X2%=150*COS(A)+160
450 Y2%=100*SIN(A)+100
460 IF FP<>1 THEN GOTO 480
470 X1%=X2%: Y1%=Y2%: FP=0
480 Z=USR(4)
490 NEXT I
460 IF FP<>1 THEN GOTO 480
470 X1%=X2%: Y1%=Y2%: FP=0
480 Z=USR(4)
490 NEXT I
500 GOSUB 9000
```

```
600 REM DEMONSTRATION OF AXIS PLOT, LABEL, AND TITLE
610 Z=USR(0)
620 REM INSERT Y AXIS LABELLING FIRST
630 FOR Y=-10 TO 10 STEP 2
640 REM REPOSITION TEXT CURSOR
650 Z=USR(6)
660 POKE CX,0: POKE CY,(-Y+10)
670 Z=USR(7)
680 PRINT Y/10;: REM PRINT Y AXIS LABEL
690 NEXT Y
700 REM PRINT X AXIS CAPTION
710 Z=USR(6): POKE CX,49: POKE CY,10: Z=USR(7)
720 PRINT "Time";
730 REM PRINT X AXIS CAPTION AND FIGURE CAPTION
740 Z=USR(6): POKE CX,0: POKE CY,21: Z=USR(7)
741 POKE 17,0: REM RESET BASIC'S CHAR POINTER TO 0
750 PRINT "Amplitude";
760 PRINT "     Waveform of Great Diapason C4 16FT";
770 Z=USR(6)
800 REM PLOT X AND Y AXES
810 X1%=20: X2%=294: Y1%=105: Y2%=105: REM HOR AXIS
820 Z=USR(2)
830 X1%=20: X2%=20: Y1%=11: Y2%=199: REM VERT AXIS
840 Z=USR(2)
900 REM PLOT TIC MARKS ON Y AXIS
910 FOR Y=-1 TO 1 STEP .2
920 X1%=18: X2%=20
930 Y1%=15+90*(Y+1): Y2%=Y1%
940 Z=USR(2)
950 NEXT Y
1000 REM PLOT THE WAVEFORM USING VECTORS
1010 FP=1
1020 XF=270/(4*3.14159): REM X SCALE FACTOR
1030 YF=60: REM Y SCALE FACTOR
1040 FOR X=0 TO 4*3.14159 STEP 4*3.14159/270
1050 Y=SIN(X)+.49*SIN(2*X+3.9)+.3*SIN(3*X+5.81)
1060 Y=Y+.24*SIN(4*X+3.8)+.18*SIN(5*X+.97)
1070 Y=Y+.12*SIN(6*X+4.3)+.04*SIN(7*X+3.54)
1080 Y=Y+.07*SIN(8*X+.87)+.03*SIN(9*X+5.3)
1090 X2%=20+XF*X: Y2%=105+YF*Y
1100 IF FP<>1 THEN GOTO 1120
1110 X1%=X2%: Y1%=Y2%: FP=0
1120 Z=USR(2)
1130 NEXT X
1140 GOSUB 9000
```

```
2000 REM SIEVE OF ERATOSTHENES DEMONSTRATION
2001 REM THIS PROGRAM FINDS ALL OF THE PRIME NUMBERS
2002 REM FROM 3 TO 128001 USING THE VISIBLE MEMORY.
2003 REM EACH PIXEL ON THE SCREEN REPRESENTS AN ODD
2004 REM INTEGER STARTING WITH 3 IN THE LOWER LEFT
2005 REM CORNER.  THE PROGRAM FIRST TURNS ALL PIXELS
2006 REM ON AND THEN TURNS THOSE OFF THAT DO NOT
2007 REM REPRESENT PRIME NUMBERS.  THOSE THAT ARE
2008 REM LEFT ON AFTER EXECUTION ARE PRIME.  IS THE
2009 REM RESULTING PATTERN RANDOM?  ARE THE PRIME
2010 REM NUMBERS UNIFORMLY DISTRIBUTED?  THE ABILITY
2011 REM TO READ BACK FROM THE VISIBLE MEMORY IS
2012 REM USED IN THIS PROGRAM.
2100 Z=USR(0)
2110 REM QUICKLY TURN ALL PIXELS ON
2120 FOR I=0 TO 199
2130 X1%=0: X2%=319: Y1%=I: Y2%=I: Z=USR(2)
2140 NEXT I
2200 FOR I=3 TO SQR(128001) STEP 2
2210 N=I
2220 GOSUB 8000
2230 IF USR(5)=0 THEN GOTO 2300
2240 FOR J=3 TO 128001 STEP 2
2250 N=I*J
2260 IF N>128001 THEN GOTO 2300
2270 GOSUB 8000
2280 Z=USR(3)
2290 NEXT J
2300 NEXT I
2310 GOTO 2310: REM WAIT FOREVER
8000 REM FUNCTION TO CONVERT ODD INTEGER TO X,Y
8010 N1=(N-3)/2
8020 N2=N1/320
8030 X1%=N1-INT(N2)*320
8040 Y1%=N2
8050 RETURN


9000 REM DELAY ROUTINE TO HOLD IMAGE IN SCREEN
9010 FOR D=1 TO 10000: NEXT D: RETURN
9999 END
```

SDTXT MACHINE LANGUAGE DESCRIPTION

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.

SDTXT is supplied in two forms in the K-1008-5C software package The cassette has an assembled version of the program which resides in the top part of a 4K AIM's memory (0B69 to 0FFF) and which has all of its temporary storage in page zero at locations 00F5 to 00FF. The back part of this manual has listing of the page zero variables and addresses of major routines which correspond to the assembled cassette version. The K-1008-5CS is a cassette which has the source statements for SDTXT in a form suitable for reassembly anywhere desired with the AIM-65 assembler ROM. It is available with a signed license agreement.

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 all of this storage has been placed at the top of page zero in the version on the K-1008-5C cassette.

As given. 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, a decision will have to be made about where the temporary storage is to be placed. One possibility if it is not possible to dedicate part of page 0 is to use some of the 192 invisible bytes at the top of the Visible Memory itself. A short pre-entry routine can save the 4 bytes that must be in page zero on the stack when SDTXT is executing and then restore them before returning. The BASIC interface package (BAS04 and BAS20) can be consulted to see how this might be accomplished.

USE OF 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 (location 0B69 in the assembled version on cassette). 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 must be set to the page number of the first VM location before SDTXT is used. For example, if the VM is jumpered for addresses 8000-9FFF, then VMORG, which is in location 00FF on the assembled version on the cassette, should be 80_{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. Alternatively, the user's initialization routine can simply zero CSRX (00FD) and CSRY (00FE) for the upper left corner.

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 21_{10} 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 registers are preserved but A is destroyed. .

SDTXT 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 AIM 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 (Form-Feed which is a screen clear) and LF (Line-Feed 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 (such as provided on a K-1012 PROM/IO board) is used for the serial communication. If the timed loop (software UART) serial routines in the AIM 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 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 retrieve the ASCII codes.

If the editing functions of the built-in AIM editor are sufficient, then use of the BAS04 or BAS20 program is recommended. These are described elsewhere in this manual.

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 2.4K 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.

Although the best way to use the graphics support routines is to assemble the ones needed with the user's application program, the pre-assembled version on cassette has been organized so that unwanted routines can simply be ignored. The assembled package has been divided into three "tiers" of routines. Tier 1 gives all of the graphics, character drawing, and text display functions and needs the full 2.4K (0682-0FFF). Tier 2 omits the text display routine but retains individual character display and all of the graphics functions and thus needs only 1.6K (09C2-0FFF). Tier 3 gives only graphics functions and thus is smaller still at .6K (0DA8-0FFF). Operating at the tier 1 level requires the entire program code from 0682 to 0FFF. When operating at the tier 2 level, the tier 1 routines (0682-09C1) may be overwritten by the user program. When operating at the tier 3 level, both tier 1 and 2 routines (0682-0DA7) may be overwritten.

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. The pre-assembled version on cassette places all temporary and permanent storage at the end of page 0 (locations 00DE-00FF).

SIGNIFICANCE OF THE PARAMETERS

Information to most of the graphics routines is passed via parameters in memory rather than in the registers. VMORG (00FF in the pre-assembled cassette version) 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 8000 - 9FFF then VMORG should be set to 80_{16}. Once set it will never be changed by any of these routines. Failure to set VMORG will usually cause total program wipeout.

Most graphic routines use one or two sets of coordinates. X1CORD and Y1CORD (00F7 and 00F9) define one set of coordinates and X2CORD and Y2CORD (00FB and 00FD) 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 X must be between 0 and 319 inclusive and Y must be between 0 and 199 inclusive (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 AIM 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 (0DA8) will verify and correct if necessary X1CORD and Y1CORD. A call to CKCRD2 (0DC1) will check and correct X2CORD and Y2CORD. Correction, if necessary, is accomplished by subtracting the maximum allowable value of a coordinate until an in range result is obtained. The check routines do not alter any of the registers thus allowing calls to them to be inserted anywhere without problems.

If the text display routine (DTEXT, 069E) is used, the text margins (TMAR, BMAR, LMAR, and RMAR; 00EF, 00F1, 00F3, 00F5) must be defined. These margin settings are double precision numbers and are the actual coordinates of the margin. Thus a right margin 3/4 of the way accross the screen would be set to 240 decimal which is an E0 in location 00F5 and a 00 in location 00F6. Text may be written up to <u>and including</u> 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 and if scrolling is triggered, a minimum of two lines must fit.

USE OF THE GRAPHIC POINT PLOT ROUTINES

All of the point oriented routines work with the point defined by X1CORD,Y1CORD. All of the point 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 (0E78)	- Sets the pixel at X1CORD,Y1CORD to a one (white dot)
CLPIX (0E8B)	- Clears the pixel at X1CORD,Y1CORD to zero (black dot)
FLPIX (0E9E0	- Changes the state of the pixel at X1CORD,Y1CORD from black to white or white to black
WRPIX (0EB1)	- Stores bit 0 of the accumulator into the pixel at X1CORD,Y1CORD
RDPIX (0EC2)	- Copies the state of the pixel at X1CORD,Y1CORD into all bits of the accumulator

An internal subroutine frequently used by other routines in this package is PIXADR (0E1E). 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 (00EB) and the bit number is stored in BTPT (00DE). Note that for the purpose of this routine that bit 0 is <u>leftmost</u> 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, DRAW (0EEF) 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 (0EEB) 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 (0A1F) 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 and DCHAR returns from subroutine immediately if one is received. 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 (069E) is a much 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 (except that the cursor position is in X and Y coordinates, not line and character numbers). 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 (0682) is a convenience routine that sets the margins for full screen operation, clears the screen and sets the cursor to the upper left corner. With a full screen, DTEXT can display 18 lines of 53 characters. The appendix at the back of this manual gives a summary of the ASCII control codes recognized by DTEXT.

When calls to DTEXT are intermingled with calls to the graphic routines, CSRINS (08D7) and CSRDEL (08DB) 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 Y1CORD. 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.

APPENDIX

Because of the great bulk involved and relatively light demand for them, the assembled listings for the programs in this package are not included in the standard manual. Instead, the following appendix gives a listing of all of the page zero storage for each of the programs and the address and description of each major routine in each program. For those who wish to have the complete printed assembly listing of each program for modification purposes should order the K-1008-5L which is approximately $25.00 and requires signing and returning a non-disclosure form.

For customers who wish to incorporate SDTXT, DTEXT, or the graphics routines in an application program, the source code for these is available on cassette in a form acceptable to the AIM-65 assembler. Order the K-1008-5CS package which is $20.00 and requires signing a non-disclosure statement. Please note that this is little more than a cassette and assembly instructions; the standard K-1008-5C package is still needed to effectively use the results. Also note that making multiple copies of these routines, whether reassembled or not, for sale or use on multiple machines is in violation of the copyright. Anyone wishing to license this software should contact MTU for details.

SWIRL

Page Zero Storage

0000	LINES	Lines connect dots if non-zero
0001	FREQ	Double precision frequency value, low byte first
0003	DAMP	Double precision damping factor, low byte first
0005	COSINT	Initial value of calculated cosine, double precision
0007	COS	Calculated cosine value (X coordinate of point)
0009	SIN	Calculated sine value (Y coordinate of point)
000B	VMORG	Page number of first Visible Memory location
000C	RANDNO	Seed for random number, double precision, must not be 0
000E	ADP1	Temporary address pointer used by graphics routines
0010	ADP2	Temporary address pointer used by graphics routines
0012	X1CORD	X coordinate number 1, initial endpoint
0014	Y1CORD	Y coordinate number 1, initial endpoint
0016	X2CORD	X coordinate number 2, final endpoint
0018	Y2CGRD	Y coordinate number 2, final endpoint
001B - 0025		Temporary storage for graphics routines
0027	PROD	Product area for math routines, quad precision, LSB first
	MPLR	Multiplier is lower 2 bytes before multiplication
002B	MPCD	Multiplicand for math routines, double precision
002D	MPSAVE	2 bytes temporary storage for math routines

Major Routines

0200	SWIRLI	Initialize and display canned demo entry point
020C	SWIRL	Display swirl with user specified parameters entry point
0222	RSWIRL	Swirl with randomly selected parameters entry point
026A	INPRMS	7 bytes which is canned demo parameters, mirror of 0-6
0271	SWINIT	Initialize routine to set SIN, COS, clear screen, set VMORG
028D	SCALE	Scale SIN and COS and put into X2CORD and Y2CORD
02D4	POINT	Compute next COS,SIN values from current values and FREQ and DAMP parameters, difference equation for circle used.
0331	C2TOC1	Move X2CORD to X1CORD and Y2CORD to Y1CORD
0342	PIXADR	Compute byte and bit address of point at X1CORD,Y1CORD
029C	STPIX	Sets pixel at X1CORD,Y1CORD
03AF	CLEAR	Clears display memory
03D1	DRAW	Draws line from X1CORD,Y1CORD to X2CORD,Y2CORD

04DA	SGNMPY	Signed multiply of MPLR by MPCD, product in PROD
0508	UNSMPY	Unsigned multiply of MPLR by MPCD, product in PROD
0544	RAND	Uniformly distributed random number generator, result in A
0560	RNDEXP	Exponentially distributed random number generator, rslt in A
057A	RANGCK	Range check of random FREQ and DAMP parameters

VLIFE

Page Zero Storage

0000	VMORG	Page number of first Visible Memory location
0001	NCYSV	Temporary storage for NCNTC
	LSTKEY	Last key number seen pressed on the AIM keyboard
0002	NCNT	Count of live neighbors
	DBCNT	AIM keyboard debounce counter
0003	LNCNT	Cell line counter
	REALST	State of cell under graphic cursor
0004	NGEN	Byte used to accumulate new cells
0005	ADP1	Temporary address pointer
0007	ADP2	Temporary address pointer
0009	BTPT	Bit number within a Visible Memory byte
000A	X1CORD	X coordinate number 1, initial endpoint
000C	Y1CORD	Y coordinate number 1, initial endpoint
000E	X2CORD	X coordinate number 2, final endpoint
0010	Y2CORD	Y coordinate number 2, final endpoint
0012	TEMP	Temporary storage, 2 bytes
0014	FLASHC	Time delay counter for cursor flash
0016	MSK-1	Table of 10 masks for cell selection in a byte
0020	-	Storage to buffer 3 scan lines of cells, 122 bytes

Major Routines

0200	DEMO	Draws a figure defined by LIST and then goes to LIFE
0244	INIT	Initialize VMORG, mask table, and life buffers
0265	LIFE	Execute life on Visible Memory contents endlessly
028C	LIFE3	Test if the "S" key is pressed on the AIM keyboard
0296	LFBUF	Computes next generation of 320 cells in middle buffer
02DA	NCNTC	Counts neighbors of cell addressed by Y=byte, X=bit in middle buffer
033F	ROLL	Rolls a scan line from the VM, through the buffers, and back to the VM
035C	PRIME	Primes the 3 buffers to get started
036B	CLEAR	Clear the display memory
0385	PIXADR	Computes byte address in ADP1 and bit number in BTPT of X1CORD,Y1CORD
03DF	STPIX	Sets a pixel at X1CORD,Y1CORD
03F0	CLPIX	Clears a pixel at X1CORD,Y1CORD
0403	WRPIX	Writes bit 0 of A at X1CORD,Y1CORD
040B	CSRINS	Insert graphic cursor at X1CORD,Y1CORD and save pixel in REALST
0411	CSRDEL	Delete graphic cursor at X1CORD,Y1CORD and restore pixel
0417	RDPIX	Read pixel at X1CORD,Y1CORD into all of A
0440	SDRAW	Simplified DRAW from X1CORD,Y1CORD to X2CORD,Y2CORD only horizontal, vertical, and 45 degree lines allowed
0480	LIST	List of coordinates for initial cell colony, coordinate pairs connected by lines if X has MSB on
0512	KYPT	Wait for keyboard entry to control setting cells
05B9	SCNKEY	Scan AIM keyboard and return with key ID in A, =80 if none

BAS20 BASIC INTERFACE

Page Zero Storage (This is kept at the end of the Visible Memory except during actual execution. Addresses given assume the VM at 8000)

9FE4	00E0	ADP1	Address pointer 1
9FE6	00E2	ADP2	Address pointer 2
9FE8	00E4	CURP2	Copy of AIM-65 cursor pointer
9FE9	00E5	CSRX	Text cursor character number
9FEA	00E6	CSRY	Text cursor line number
9FEB	00E7	X1CORD	X coordinate number 1, initial endpoint
9FED	00E9	Y1CORD	Y coordinate number 1, initial endpoint
9FEF	00EB	X2CORD	X coordinate number 2, final endpoint
9FF1	00ED	Y2CORD	Y coordinate number 2, final endpoint
9FF3	00EF	TEMP DCNT1	Temporary storage, 2 bytes
9FF4	00F1	BTPT	Bit number within a byte
9FF5-9FFF	00F2-00FB		Temporary storage for line draw

Permanent storage

0106	PRTFLG	Print flag to auto print when the screen fills up
0107	VMORG	Page address of VM set by INITx routine

Major Routines (BAS20 assumed, for BAS04, first hex digit of address is 0)

4772	INIT2	Initialization entry point for Visible Memory at 2000-3FFF
4776	INIT4	Initialization entry point for Visible Memory at 4000-5FFF
477A	INIT6	Initialization entry point for Visible Memory at 6000-7FFF
477E	INIT8	Initialization entry point for Visible Memory at 8000-9FFF
479E	PUTBAK	Restore normal AIM-65 operation entry point
47A9	DISPCH	USR function in BASIC goes here, deciphers the argument
4813	DSPTAB	Dispatch table of action addresses for USR arguments 0-11
482B	TVOUT	AIM monitor goes here to print characters
48CA	POSWAP	Swap page 0 00E0-00FB with end of Visible Memory whose page address is kept in location 0107
492B	CLEAR	Clear Visible Memory screen routine
493F	PIXADR	Computes byte address in ADP1 and bit number in BTPT of X1CORD,Y1CORD
499A	STPIX	Sets a pixel at X1CORD,Y1CORD
49A9	CLPIX	Clears a pixel at X1CORD,Y1CORD
49B8	FLPIX	Flips the pixel at X1CORD,Y1CORD
49C7	WRPIX	Writes into X1CORD,Y1CORD according to Acc., 0=off, 1=on, 2=flip
49CF	RDPIX	Read pixel at X1CORD,Y1CORD into all of Acc.
49F0	CKCRD	Check and correct if needed X1CORD, Y1CORD, X2CORD, Y2CORD
4A2F	ERASE	Erase line between X1CORD,Y1CORD and X2CORD,Y2CORD
4A33	DRAW	Draw line between X1CORD,Y1CORD and X2CORD,Y2CORD
4A37	FLIP	Flip line between X1CORD,Y1CORD and X2CORD,Y2CORD
4B40	SDTXT	Display character at cursor position and move cursor CR, LF, FF, BS recognized
4B8F	-	Jump if cursor against right edge of screen
4BAC	SDTXCR	Carriage return processor
4BB6	SDTXCL	Backspace processor
4BC4	SDTXFF	Form-feed (clear screen and home cursor) processor
4BD0	SDTXLF	Line feed processor
4BE7	-	Link to quick screen print
4BEE	-	Link to quality screen print

4C42	CSRSET	Display a cursor at the cursor position
4C4D	CSRCLR	Clear the cursor at the cursor position
4C69	CSRTAD	Compute byte and bit address of upper left corner of char.
4CAF	CKCUSR	Check and correct if necessary CSRX and CSRY
4D16	FMOVE	Fast memory move, ADP1=source, ADP2=dest, DCNT1=byte count
4D42	FCLR	Fast memory clear, ADP2=address, DCNT1=byte count
4D60	CHTB	Font table, 7 bytes per character, starts with ASCII blank

SDTXT ROUTINES

age Zero Storage

00F5	BTPT	Bit number within Visible Memory byte
00F6	DCNT1	Double precision counter
00F8	MRGT1	Temporary storage for graphic merge function
00F9	ADP1	Address pointer 1
00FB	ADP2	Address pointer 2
00FD	CSRX	Cursor position, character number
00FE	CSRY	Cursor position, line number
00FF	VMORG	First page number of Visible Memory

ajor Routines

0B69	SDTXT	Main text display entry point, character to display in Acc.
0BB8	-	Jump if cursor against right edge of screen
0BC2	SDTX10	Control character interpreter
0BD5	SDTXCR	Carriage return processor
0BDF	SDTXCL	Backspace processor
0BED	SDTXFF	Form-feed (clear screen and home cursor) processor
0C09	SDTXLF	Line feed processor
0C62	CSRSET	Display a cursor at the cursor position
0C6A	CSRCLR	Clear the cursor at the cursor position
0C85	CSRTAD	Compute byte and bit address of upper left corner of char.
0CCA	MERGE	Merge a row of 5 dots with VM starting at byte address in ADP2 and bit number in BTPT. 5 dots to merge left justified in Acc.
0D16	FMOVE	Fast memory move, ADP1=source, ADP2=dest, DCNT1=byte count
0D42	FCLR	Fast memory clear, ADP2=address, DCNT1=byte count
0D60	CHTB	Font table, 7 bytes per character, starts with ASCII blank

GRAPHIC ROUTINES

Page Zero Storage

00DE	BTPT	Bit number within a Visible Memory byte
00DF - 00EA		Temporary storage for line drawing routines and DTEXT
00EB	ADP1	Address pointer 1
00ED	ADP2	Address pointer 2
00EF	TMAR	Y coordinate of top margin for DTEXT
00F1	BMAR	Y coordinate of bottom margin for DTEXT
00F3	LMAR	X coordinate of left margin for DTEXT
00F5	RMAR	X coordinate of right margin for DTEXT
00F7	X1CORD	X coordinate number 1, initial endpoint
00F9	Y1CORD	Y coordinate number 1, initial endpoint
00FB	X2CORD	X coordinate number 2, final endpoint
00FD	Y2CORD	Y coordinate number 2, final endpoint
00FF	VMORG	Page address of Visible Memory

Major Routines

** Tier 1 routines **

0682	DTXTIN	Initialize margins for full screen, clear screen, home cursor for DTEXT
069E	DTEXT	Main display text entry point, ASCII character in A
06C9	DTEXT3	Vector jump routine for control code interpreter
06D2	CRR	Move cursor right in response to ASCII DC2
06D8	CRL	Move cursor left in response to ASCII DC1
06DE	CRU	Move cursor up in response to ASCII DC3
06E4	CRD	Move cursor down in response to ASCII DC4
06EA	BASUP	Baseline up 3 scan lines in response to ASCII SI
06FE	BASDN	Baseline down 3 scan lines in response to ASCII SO
0712	CARRET	Carriage return (cursor to left edge) in response to CR
0723	LNFED	Line feed (cursor down with scroll) in response to LF
07BC	FMFED	Form feed (clear screen & home cursor) in response to FF
07D8	LNCLR	Clear VM inside the margins routine
0831	RECTP	Establish useful data about the margins to simplify limit checking (in temporary storage 00DF-00EA)
088B	DNTST	Test if cursor can go down, A negative if not
089E	UPTST	Test if cursor can go up, A negative if not
08B1	LFTST	Test if cursor can go left, A negative if not
08C4	RTTST	Test if cursor can go right, A negative if not
08D7	CSRINS	Insert a text cursor at X1CORD,Y1CORD
08FC	CSRR	Cursor right, do nothing if against right margin
0913	CSRL	Cursor left, do nothing if against left margin
092A	CSRU	Cursor up, do nothing if against top margin
0941	CSRD	Cursor down, do nothing if against bottom margin
0958	CCTAB	Control character dispatch table
0976	MERGEL	Merge left A with VM at ADP1,BTPT, bits to left preserved
099C	MERGER	Merge right A with VM at ADP1,BTPT, bits to right preservec

** Tier 2 routines **

09C2	MERGE5	Merge 5 dots left just in A with VM at ADP1,BTPT
09FE	-	Merge bit pattern tables
0A1F	DCHAR	Draw character upper left corner at X1CORD,Y1CORD, char in Acc.
0A97	CHTB	Font table, 8 bytes per shape, sign bit of first byte specifies 2 scan line descender
0D97	SADP2L	Shift ADP2 left one bit
0D9C	DN1SCN	Double add 40 to ADP1

** Tier 3 routines **

0DA8	CKCRD1	Check and correct if necessary X1CORD and Y1CORD
0DC1	CKCRD2	Check and correct if necessary X2CORD and Y2CORD
0DFE	CLEAR	Clear Visible Memory screen
0E1E	PIXADR	Compute address of pixel at X1CORD,Y1CORD into ADP1 and BTPT
0E78	STPIX	Sets a pixel at X1CORD,Y1CORD
0E8B	CLPIX	Clears a pixel at X1CORD,Y1CORD
0E9E	FLPIX	Flips the pixel at X1CORD,Y1CORD
0EB1	WRPIX	Writes into X1CORD,Y1CORD the least significant bit of A
0EC2	RDPIX	Read pixel at X1CORD,Y1CORD into all of A
0EEB	ERASE	Erase line between X1CORD,Y1CORD and X2CORD,Y2CORD
0EEF	DRAW	Draw line between X1CORD,Y1CORD and X2CORD,Y2CORD

SUMMARY OF ASCII CONTROL CODES RECOGNIZED BY DTEXT

HEX	ASCII SYMBOL	
0D	CR	Carriage return, sets cursor to left margin
0A	LF	Line feed, moves cursor down 1, if at bottom margin, scroll up instead
08	BS	Backspace, same as cursor left
0C	FF	Form-feed, clear screen, home cursor in upper left corner
0F	SI	Shift in, move baseline up 3 scan lines for superscripts
0E	SO	Shift out, move baseline down 3 scan lines for subscripts
11	DC1	Move cursor left one character, nothing if at left margin
12	DC2	Move cursor right one character, nothing if at right margin
13	DC3	Move cursor up one line, nothing if at top margin
14	DC4	Move cursor down one line, nothing if at bottom margin