VIPER

January - February
Volume 5, Number 5

Journal of the VIP Hobby Computer Assn.

The VIPER was founded by ARESCO, Inc. in June 1978

****

PIPS for VIPS IV - Part 5

We're getting down to the home stretch as once again VIPER continues with the serialization of Tom Swan's PIPS for VIPS IV. This issue deals with Page Switching and Graphics in machine language, both very useful techniques for game programs for 1802/1861 computers. The next issue of VIPER will complete the PIPS series.

In this issue we also have a very nice memory expansion hardware project for VIP users.

Cassettes of PIPS IV are still available and the cost for the ten programs on the tape is $5. You may send your check to VIPHCA at the usual address. Each program on the tape is complete, with the appropriate CHIP-8 interpreter, where necessary. The program titles are:

1. Cos Melodeon
2. Holiday tree
3. Attack of Micromen
4. Line up
5. Sweeper
6. Move on
7. Box
8. Shape maker
9. Graphics #1
10. Graphics #2

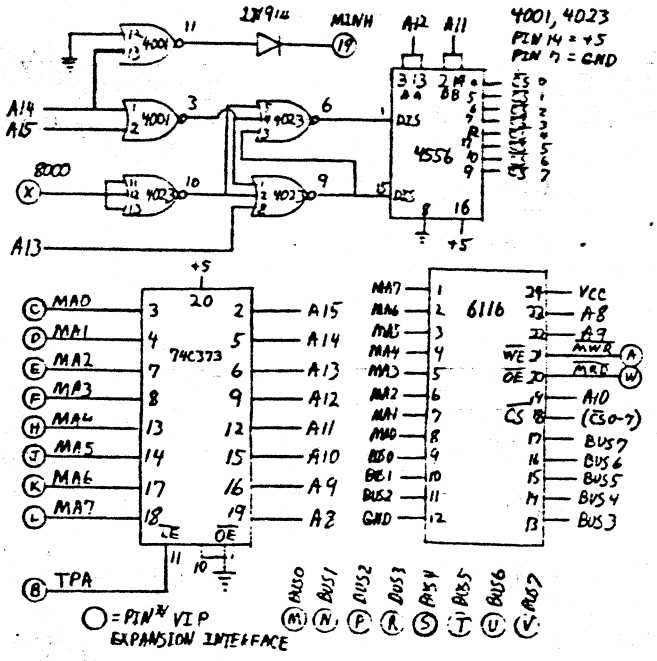
VIP 16K CMOS MEMORY

The 6116 CMOS memory chip is a natural for 1800 based micros. It's high impedance inputs do not load the data or address lines, MWR and MRD interface directly to WE and OE, and the single active low chip enable make large memories a snap. The all CMOS circuitry shown here has been working flawlessly for almost two years on both my VIP and a 4MHz wire-wrap system.

The address decoding could be made simpler, but I wanted ease in re-addressing and compatibility with my home-brew micro. As implemented here, the 16K resides at 0000 to 3fff hex in the VIP memory space with the on-board memory now at 4000 to 4fff hex by holding #19MINH high until A14 goes high. As long as both A15 and A14 are low and #X8000 is low, depending on the state of A13, either pin 1 or pin 15 of the 4556 will go low. This will allow either the first or second half of the 4556 to decode a low CS from A12 and A11.

All data lines and address lines A0-A7 tie directly from the VIP expansion interface card edge connector to all 8 of the 6116s. A8-A15 are latched on the trailing edge of TPA by the 74C373. The whole thing fits on a 4"X 4½" wire-wrap plug board. I do not recommend bare copper edge contacts due to corrosion problems.

Claude Hesselman -- 2545 Bainbridge Blvd. Chesapeake, Virginia 23324



ENOUGH TALK -- LET'S PROGRAM

All programs that use the graphics subs may be written as subroutines themselves or as whole programs. Usually the starting address of your program will be $0300 but this may be changed by inserting a new start address into the CALL PROG instruction at $0021. Your machine language program may end in one of three ways:

1) RETN (SEP R5) -- ends program
2) DEC R3 -- Halts by decrementing program counter
3) BR HERE -- Halts by branching to itself

When your program returns via SEP R5, control passes to $0024 where you will find the halt instruction #2 (DEC R3). Some program crashes are prevented by this setup. If somehow you were to execute too many returns in your program, an automatic halt would be executed at $0024. You may want to keep this in mind when debugging a program. If you end upat $0024 at the wrong time, then you have executed too many RETN's.

Another possibility would be to program a jump at $0024 to your operating system reset entry point. You may have to turn off the video and initialize a few registers to program this feature, but there is plenty of room at $0024 - $007E. VIP owners with 4K of memory (in their computers, that is) may program the following sequence for an automatic return to the VIP operating system in

ROM. Your program must end with a $D5 byte (SEP R5) to use this auto restart feature.

```
0024  D4 01 30 22 61 F8 0F B1
002C  F8 FF A1 F8 80 B2 F8 28
0034  A2 D2 00 00 00 00 00 00
```

For more details on the above routine, see my article "Super Duper 3-Way Memory Diagnostic" in the February '80 issue of the VIPER newsletter. Note that in the above, the CALL CLEAR at 0024 is optional. Also if you have less memory (and again I mean of course your computer), you may enter the page address of your highest on card RAM page into location $002A.

Register R3 is always the program counter and needs no initializing in your routines. Registers R7 and R8 may be used by you freely without concern that they will be changed somewhere. R7, however, is needed as a pointer to shape tables by the subs SHAPE and POINT. This may seem to be slim pickins', but isn't really. Because parameters (X,Y coordinates, etc.) are usually passed to subs in other registers, you may also count these among those available to you. For example, POINT requires the X,Y values to be placed in RE. Before you call POINT, or before it is called by another sub, RE is available for you to use. So is RF. They will be changed by POINT, but there is no reason why you could not use them for some immediate purpose.

The registers which must not be changed by you are: R0, R1, R2, R4, R5, R6, and RB.1. Register R3 may be changed, but only in the context of its use as the program counter. Inserting an

address into R3.0, for example, will cause execution to begin at that new address, a popular 1802 technique.

Don't forget about the big stack at your disposal. If you are using register RA and need to CALL LINE where you see in the Graphics Subs Reference that RA is needed by that subroutine, you may save RA on the stack with the following steps. (N stands for the register you want to preserve. N would be set equal to A in our example.)

```
(SEX   R2) - Only if X ≠ 2 here
 GHI   RN  - Get high part of register
 STXD      - Push onto stack
 GLO   RN  - Get low part of register
 STXD      - Push onto stack
 CALL SUB  - CALL subroutine
 IRX       - Point to saved data
 LDXA      - Pop low part of register
 PLO   RN  - Restore RN.0 value
 LDX       - Pop high part of register
 PHI   RN  - Restore RN.1 value
```

The order of the above is very important. In this graphics system, R2 is <u>always</u> addressing a free location on the top of the stack and must be left in that condition when you are finished using the stack. You may use the top of the stack as an additional eight-bit variable without decrementing R2, but any CALLs to subroutines will destroy that value addressed by R2. Assuming X was not changed by your program, the following would add R8.0 and R8.1 together (ignoring overflow).

```
GHI   R8  ;Push R8.1 onto stack for use
STR   R2  ;  as 8-bit variable addressed by R2
GLO   R8  ;Add R8.0 via R(X)
ADD       ;  assuming X = 2 here
PHI   R8  ;Put answer in R8.1
```

Knowing for sure that the above will not disturb the contents of the stack makes programming a bit less worrisome. Because 256 bytes have been reserved for the stack, even the largest of programs are unlikely to ever cause a stack overflow. If you would rather the stack be located at the end of programming space with it growing toward you and your programming running toward it, you must change the initialization of R2 at $0004 - $0009. This would clearly result in more efficient use of memory, and programs could begin at $0200 (by inserting that address into CALL PROG at $0021). Not knowing if you will be using page switching or not, however, I located the stack at $0200 and made it larger than probably necessary. Those of you with very small systems may make the appropriate modifications to utilize a smaller stack in trade for more programming room.

In addition to these suggestions, small systems owners may have to change the byte at $0011 to the address of the four memory pages to be used for the display refresh. Normally for VIP systems, the following values apply.

```
2K systems - change $0011 to $04
3K systems - change $0011 to $08
4K systems - change $0011 to $0C
```

Even if you have the smallest VIP, you may run many of the following high resolution graphics programs but unless you modify the software as suggested above, you are restricted to programs that use no more than 256 bytes at $0300 - $03FF without page switching enabled. The listing and the programs on tape are set to be used by standard VIPs containing at least 4K of memory.

* * *

OK let's draw a box. Not an exciting prospect but a simple program to try. (This is not included on the tape -- this section is for you to learn some graphics techniques leading hopefully to your own programs.)

First, load the Graphics Subs Package #1 into memory at 0000 - 01FF. Then enter the following programs and Shape Table. There is no need to save it unless you want to.

DRAW SHAPE

```
0300  F8 03          LDI  $03     ;Load high address SHAPE TABLE
  02  B7             PHI  R7      ;  into R7.1
  03  F8 0A          LDI  $0A     ;Load low address SHAPE TABLE
  05  A7             PLO  R7      ;  into R7.0
  06  D4 00 84       CALL SHAPE   ;Draw the shape
  09  23             DEC  R3      ;Halt here
```

SHAPE TABLE

```
030A  00 00                       ;Starting X,Y coordinate
  0C  3F 00                       ;To upper right corner
  0E  3F 7F                       ;To lower right corner
  10  00 7F                       ;To lower left corner
  12  00 00                       ;To upper left corner
  14  FF FF                       ;Stop indicator
```

When you run the program you should see a quickly drawn box outlining your display area. The shape table you entered at $030A contains only the endpoints of each line -- that is, the corners of the box. There are five X,Y pairs in the table because the first pair of bytes defines the starting coordinates from which a line will be connected to the following X,Y pairs in the table.

You may draw <u>any</u> shape using this same routine by only entering a new Shape Table. (Be sure to end the table with the $FFFF stop indicator, or SHAPE won't know when to stop connecting lines!) Here is a different Shape Table to try: (To save space, it is typed in rows of eight bytes, but these are all X,Y pairs just as for the box.)

SHAPE TABLE #2

```
030A    0020 0850 1020 2820
  12    2020 2050 1850 3050
  1A    3020 3F20 3F36 3036
  22    FFFF
```

To see the difference between the SHAPE and the PLOT subroutines, change the call instruction in $0306 to:

```
$0306   D4 00 98  CALL PLOT
```

Now instead of connecting the lines, the X,Y pairs in the SHAPE TABLE are plotted individually. That's all there is to drawing or plotting shape tables -- the subroutines take care of most of the work for you.

Something to remember: never enter X,Y coordinates beyond the range of the display resolution you are using. The highest allowed X coordinate is $3F and the maximum Y is $7F. If you are not using the highest display resolution, Y coordinates must be adjusted to avoid overwriting non-display memory areas. A two page resolution cannot show a Y coordinate greater than $3F and one page displays, as in CHIP-8 programs, are limited to $1F in the vertical direction.

To experiment with various resolutions, try calling the CHANGE sub with the following sequence:

```
0300  F8 0N           LDI  $0N     ;Enter resolution index
  02  AE              PLO  RE      ;  into RE.0
  03  D4 00 BD        CALL CHANGE  :Change interrupt routines
  06       (Now  program a routine to draw a shape table)
```

This is not a complete program. You must enter either a 00, 01, or 02 byte at 0301 depending on which resolution you wish to view. Then you have to write a program similar to the one we just saw to draw a shape. You can copy the old DRAW SHAPE program starting at $0306, but be sure to change the address of the shape table given to R7.0!

* * *

It's also easy to plot a single point or draw a line anywhere by using the POINT and LINE subs. Again, you should use X,Y

coordinates within allowable ranges. The following routine will put a point at any position on the display.

```
0300  F8 12          LDI  #12     ;Place X coordinate in
  02  AE             PLO  RE      ;  register RE.0
  03  F8 3B          LDI  #3B     ;Place Y coordinate in
  05  BE             PHI  RE      ;  register RE.1
  06  D4 01 08       CALL POINT   ;Display the point
  09  23             DEC  R3      ;Halt
```

This program displays a single dot of light at the X,Y coordinates ($12,$3B). You may enter any X,Y pairs into locations $0301 and $0304. These values are placed in register RE, then the subroutine POINT is called. When values are given in this way to a subroutine, they are sometimes called "parameters" and the action of giving them to the routine called "passing the parameters." Notice that setting RE meets the input requirements for POINT as stated in the Graphics Subs Reference. (RB.1 was already set in the initializing part of the program at $0012. You never have to worry about setting RB.1)

Similarily, here is a routine that will connect any two points X_0,Y_0 to X_1,Y_1 with a straight line.

```
0300  F8 00          LDI  $00     ;Load X0 coordinate into
  02  AC             PLO  RC      ;  register RC.0
  03  F8 50          LDI  $50     ;Load Y0 coordinate into
  05  BC             PHI  RC      ;  register RC.1
  06  F8 2F          LDI  $2F     ;Load X1 coordinate into
  08  AD             PLO  RD      ;  register RD.0
  09  F8 47          LDI  $47     ;Load Y1 coordinate into
  0B  BD             PHI  RD      ;  register RD.1
  0C  D4 01 47       CALL LINE    ;Connect X0,Y0 to X1,Y1
  0F  23             DEC  23      ;Halt
```

When you run the program, you will notice the "stair-step effect" mentioned earlier. The higher the resolution, the less this effect will detract from the picture. Just as with POINT, LINE requires that you pass the parameters it needs in specified registers. In this case the parameters are both end coordinates of the line and the registers are RC and RD. At the end of the sub, RC will equal the previous value of RD which is lost. Therefore to connect a series of points, RC only needs to be set once. That's exactly how the SHAPE sub works by the way.

You never have to worry about which direction you are drawing lines. There is never any wrap around for instance, and all lines will appear whole and unbroken in the display area. You do have to stay within the valid X,Y ranges as discussed before.

Another thing to remember is that a line may not be selectively erased (when $012D in POINT is set to XOR $F3) in the reverse direction. Sometimes yes, sometimes no, but when you are programming this way, it's best to plan to selectively erase lines in the same direction they were drawn or they may leave bits of themselves on the display when the "stair steps" are drawn the other way.

That covers the basics of using the graphics subs in machine language programs. Some of the following programs use various manipulations to produce patterns on the display that would be difficult to accomplish without these basic graphics capabilities especially of POINT and LINE. I have not given an example for

using CLEAR -- this should be obvious. A call to CLEAR simply erases the display. FLOP was already covered, and we will soon see some programming examples that make use of that sub to animate complex graphics.

LINE UP!

The following program shows off the capabilities of the LINE sub by producing a changing display of randomly drawn lines that often look three dimensional. The program has been included on the tape but you may want to read through this section and the listing for ideas on how to write your own graphics programs.

Four sections make up this program. First, R8.0 is set to any initial value as a "seed" for the random number generator. Next, RC is set to the center display coordinates ($1F,$3F) from where the first line will be drawn. R7.0 is set to the number of lines that will be drawn before the screen is cleared and a new pattern is started. You may enter any value in location $030A to draw from 1 to 256 lines before restarting.

Section two begins at $030C with CALLs to RAND for new random numbers returned in register R8.0. Register RD is set to two different random coordinates to which a line will be drawn. Because RC ends up equal to the old value of RD after LINE is done drawing a single line, new lines always begin where old ones left off. The loop terminates at 031F when R7.0, used as the loop counter, finally goes to zero.

Section three from $0321 to $032C contains a little timer which waits for a few seconds then CLEARs the display and branches to start the whole thing over again.

Section four is the random number generator. Rather than waste time here discussing the merits or the failings of one generator over another, you will notice that unless $00F7 is set to $18 (INC R8) in the interrupt routine, the patterns repeat rather quickly. Even so, they eventually must repeat and as I am writing with the program running in front of me, they got locked into a loop for a half dozen cycles! (The taped version of this program has the modified interrupt.)

That suggests that a program such as this could be used as a quick, admittedly subjective, appraisal of a random number generator. Rather than put a new routine through a series of complex numerical tests, at least its apparent randomness could be tested by using graphics on a TV set. Animations need not only be for fun -- there is a world full of possibilities!

LINE UP

```
                    ;Requires Graphics Subs Package #1
                    ;at $0000 to $01FF

0300  F8 AE    LINEUP: LDI  $AE     ;Random number seed may be
  02  A8              PLO  R8      ;  any value
  03  F8 1F           LDI  $1F     ;Initialize RC to X,Y
  05  AC              PLO  RC      ;  coordinates at center
  06  F8 3F           LDI  $3F     ;  of screen (X=1F, Y=3F)
  08  BC              PHI  RC
  09  F8 10       1H: LDI  $10     ;Set R7.0 to number of lines
  0B  A7              PLO  R7      ;  to draw - any value OK
```

```
030C  D4 03 2D     2H: CALL RAND   ;Generate random number
  0F  88             GLO  R8      ;Get the random value from R8.0
0310  FA 3F          ANI  $3F     ;Limit to within 00-3F range
  12  AD             PLO  RD      ;Put in RD.0 as X coordinate
  13  D4 03 2D       CALL RAND    ;Do the same for Y, limiting
  16  88             GLO  R8      ;  the random value to
  17  FA 7F          ANI  $7F     ;  00-7F range and
  19  BD             PHI  RD      ;  placing in RD.1
  1A  D4 01 47       CALL LINE    ;Draw line from RC to RD
  1D  27             DEC  R7      ;Decrement counter
  1E  87             GLO  R7      ;Test count
  1F  3A 0C          BNZ  2B      ;If ≠ 0, jump back to do another

                ;Wait before restarting

0321  F8 80          LDI  $80     ;Load timer value into
  23  BF             PHI  RF      ;  register RF.1
  24  2F          3H: DEC  RF      ;Decrement the register
  25  9F             GHI  RF      ;Test RF.1
  26  3A 24          BNZ  3B      ;Loop until = zero
  28  D4 01 30       CALL CLEAR   ;Erase display
  2B  30 09          BR   1B      ;Restart random line draw

                ;Random number generator

  2D  88          RAND: GLO  R8   ;Push old random value
  2E  52             STR  R2      ;  onto stack
  2F  FE             SHL          ;Multiply by 4 using
0330  FE             SHL          ;  bit shifts
  31  F4             ADD          ;Add to old random value
  32  FC 02          ADI  $2      ;Add constant
  34  A8             PLO  R8      ;Put in R8.0
  35  D5             RETN         ;Exit

                ;Above random generator is poor unless
                ;  the following change is made to INT4

00F7  18             INC  R8      ;R8← R8+1 each interrupt
```

PLOTTER

You do not need to have a fancy floating point BASIC with a SIN(X) function to plot curves on a video display though I have seen few assembly language programs including such a feature.

The following program will plot a sine wave of frequency (f) simply by varying Y over a range of X, X+1, X+2...X+N. (Actually PLOTTER uses XN, XN-1, XN-2... as this was a little easier to program.)

The big secret is the sine look up table at 0400 - 04FF. Using f+X as an index address into the table, the appropriate Y value is found. Because the highest value in the table is $FF, Y is divided by two with the shift instruction at $0312 thus assuring that Y will never be greater than $7F, the display range limit.

Probably for scientific work the sine would have to be multiplied by a constant, then divided to bring it into the display range for Y. This is because the sine table actually contains fractions between 0 and 1. For purposes of demonstrating curves on a VIP, there is no need to complicate the program.

You may change the starting frequency at $0301 and the time between plots at $031E. You could also insert a branch at $0312 and further divide SIN(X) to flatten the curve even more. For example:

```
$0312  30 30    BR   $30
  --
$0330  F6       SHR
       F6       SHR
       BE       PHI  RE
       30 14    BR   $14
```

A constant may have to be added to SIN(X) to bring the display to the center of the screen but again this is only intended as a jumping off demo for your own programming ideas.

Guess I'll "SINE" off now.

PLOTTER

```
                ;Requires Graphics Subs Package #1 at $0000 - $01FF
                ;SINTAB is at $0400 - $04FF (may be relocated
                ;  for your own uses.)

0300  F8 12  PLOTTER: LDI  $12     ;Load R7.1 with starting
  02  B7          1H: PHI  R7      ;  frequency
  03  F8 3F           LDI  $3F     ;Load R7.0 with starting
  05  A7              PLO  R7      ;  X coordinate
  06  F8 00           LDI  $00     ;Load R8.0 with 00.  R8.0
  08  A8              PLO  R8      ;  will index SINTAB
  09  88          2H: GLO  R8      ;Add f to R8.0 forming
  0A  52              STR  R2      ;  SINTAB index
  0B  97              GHI  R7      ;   "       "       "
  0C  F4              ADD          ;   "       "       "
  0D  A8              PLO  R8      ;Save value for next time in R8.0
  0E  F8 04           LDI  $4      ;Set R8.1 to the page address
0310  B8              PHI  R8      ;  of SINTAB
  11  08              LDN  R8      ;Get SIN(X), and divide by 2
  12  F6              SHR          ;  to limit Y coordinate
  13  BE              PHI  RE      ;Put Y in RE.1
  14  87              GLO  R7      ;
  15  AE              PLO  RE      ;Put X in RE.0 (Note X is backwards)
  16  D4 01 08        CALL POINT   ;Plot X,Y in RE
  19  27              DEC  R7      ;Count loops by decrementing X
  1A  87              GLO  R7      ;Test count
  1B  3A 09           BNZ  2B      ;If not zero, branch to continue
  1D  F8 40           LDI  $40     ;Else wait here between
  1F  BF              PHI  RF      ;  sine wave displays
0320  2F          3H: DEC  RF      ;   "       "       "
  21  9F              GHI  RF      ;   "       "       "
  22  3A 20           BNZ  3B      ;   "       "       "
  24  D4 01 30        CALL CLEAR   ;Clear the display
  27  97              GHI  R7      ;Subtract one from
  28  FF 01           SMI  $1      ;  frequency in R7.1
  2A  3A 02           BNZ  1B      ;Loop to do another
  2C  30 00           BR   PLOTTER;Reset on R7.1=0
```

NOTE: PLOTTER and the SINE TABLE are not included on the cassette tape. Sorry, friends, but when I made up the master tape, I ran out of room! You need to first load Graphics Subs Package #1 into the two pages from $0000 to $01FF. Then, using the Memory Write mode of the VIP operating system, load in the PLOTTER program and the SINE TABLE at the indicated addresses. Record five pages from $0000 when you are done. This program requires a minimum of 3K RAM on board your VIP.

SINE TABLE FOR PLOTTER

```
0400  80 83 86 89 8C 8F 92 95 98 9C 9F A2 A5 A8 AB AE
  10  B0 B3 B6 B9 BC BF C1 C4 C7 C9 CC CE D1 D3 D5 D8
  20  DA DC DE E0 E2 E4 E6 E8 EA EC ED EF F0 F2 F3 F5
  30  F6 F7 F8 F9 FA FB FC FC FD FE FE FF FF FF FF FF
  40  FF FF FF FF FF FF FE FE FD FC FC FB FA F9 F8 F7
  50  F6 F5 F3 F2 F0 EF ED EC EA E8 E6 E4 E2 E0 DE DC
  60  DA D8 D5 D3 D1 CE CC C9 C7 C4 C1 BF BC B9 B6 B3
  70  B0 AE AB A8 A5 A2 9F 9C 98 95 92 8F 8C 89 86 83
  80  7F 7C 79 76 73 70 6D 6A 67 63 60 5D 5A 57 54 51
  90  4F 4C 49 46 43 40 3E 3B 38 36 33 31 2E 2C 2A 27
  A0  25 23 21 1F 1D 1B 19 17 15 13 12 10 0F 0D 0C 0A
  B0  09 08 07 06 05 04 03 03 02 01 01 00 00 00 00 00
  C0  00 00 00 00 00 00 01 01 02 03 03 04 05 06 07 08
  D0  09 0A 0C 0D 0F 10 12 13 15 17 19 1B 1D 1F 21 23
  E0  25 27 2A 2C 2E 31 33 36 38 3B 3E 40 43 46 49 4C
  F0  4F 51 54 57 5A 5D 60 63 67 6A 6D 70 73 76 79 7C
```

SWEEPER

Here is a program that uses the LINE subroutine to produce random graphics patterns on the display. Before, I mentioned that the POINT sub could be modified to erase bits selectively without having to clear the display. SWEEPER is rather dull without this

change and I suggest you make the following modifications to Graphics Subs Package #1.

```
012D  F3    XOR   ;Change from $F1 (OR) in POINT
```

With the above change, SWEEPER produces a hypnotic pliable montage of swirls and curves that really shows off the high resolution capabilities of the VIP. As in the last program, a random number generator is called for the endpoints of lines and in this case, an index to the direction of movement. The generator is the same one, but has been relocated to $036E. For a different sequence, you may enter new "seeds" at 0301, but to help the generator appear more random, the following change to the interrupt routine is recommended, just as for the LINE UP program.

```
00F7  18    INC  RF  ;Replace $E2 with $18 for better "seed"
```

The above two changes are already included in the taped version of SWEEPER.

* * *

SWEEPER is a more complex graphics program than LINE UP, but even so, may be run on a 2K VIP system. It works by first selecting a random line with the instructions from $0303 to $0319. Just as in LINE UP, RC and RD are set to random values which have been limited to the proper display range. Two subroutines which do the work of getting a random number and limiting the X values

to $3F and the Y values to $7F have been included at $0377 and $037F. CALLing either sub will cause the appropriate random coordinate to be returned in register RE.1

The programming from $031A to $0324 forms a random indexed address into the move table of the LINE sub. (Rather than copying the table over for this program, we may as well use the one that is already there!) The bytes in that table are either $FF(-1), 00, or 01 and will be used later to produce movement in random directions.

From $0325 to $0334, RC is stored on the stack, LINE is called and the values of RD and RC are restored to their orignal values. Following this, the indexed MOVTAB address is placed in RF and the values from that table are added to each of the four line endpoint coordinates with the instructions from $0335 to $035C. If any of the coordinates happens to extend outside of the allowable ranges, a branch to $035E is performed where the loop will either terminate on R7.0 = 00, or a new sweep will begin.

That's all it takes to produce the patterns. If you are new to machine language, these programs may appear formidable, but they are really very simple and straightforward. Hopefully, you are getting some ideas on using the graphics subs in your own programs. I wish at this point it were possible for me to stop and ask, "Any questions?" But of course that's not possible. If you do have a question, however, feel free to send it in to the VIPER. That's what we are here for!

```
0300  F8 89          LDI  $89      ;(any value @ 0301 OK)
  02  A8             PLO  R8       ;Seed R8.0 for RAND
  03  D4 03 77  1H:  CALL GETX     ;Get a random X coordinate
  06  9E             GHI  RE
  07  AC             PLO  RC       ;X0 <- random X in RC.0
  08  D4 03 7F       CALL GETY
  0B  9E             GHI  RE
  0C  BC             PHI  RC       ;Y0 <- random Y in RC.1
  0D  D4 03 77       CALL GETX
0310  9E             GHI  RE
  11  AD             PLO  RD       ;X1 <- random X in RD.0
  12  D4 03 7F       CALL GETY
  15  9E             GHI  RE
  16  BD             PHI  RD       ;Y1 <- random Y in RD.1
  17  F8 10          LDI  $10
  19  A7             PLO  R7       ;R7.0 <- loop count of 10 (or other)
  1A  D4 03 6E  2H:  CALL RAND     ;Get random index
  1D  88             GLO  R8
  1E  FA 07          ANI  $07      ;Limit number to 00 - 07, then
0320  FE             SHL           ;  using shifting, multiply by
  21  FE             SHL           ;  4, the # bytes of MOVTAB entries
  22  FC C6          ADI  $C6      ;Add address MOVTAB
  24  73             STXD          ;Push for later indexing MOVTAB
  25  9C        3H:  GHI  RC       ;Save RC on the stack
  26  73             STXD          ;  for later restoring
  27  8C             GLO  RC
  28  73             STXD
  29  D4 01 47       CALL LINE     ;Draw line from RC to RD
  2C  9C             GHI  RC       ;Restore RD value by transferring
  2D  BD             PHI  RD       ;  from RC.  (RC = old RD
  2E  8C             GLO  RC       ;  following LINE)
  2F  AD             PLO  RD
0330  60             IRX           ;Point to save RC on stack
  31  72             LDXA          ;Pop stack into RC to
  32  AC             PLO  RC       ;  restore old RC value
  33  72             LDXA          ;(Note extra increment of R2 here!)
  34  BC             PHI  RC
  35  F0             LDX           ;Pop indexed MOVTAB address
  36  AF             PLO  RF       ;  and place in RF.0.  Then set
  37  F8 01          LDI  MOVTAB.1; RF.1=01 to form address of
  39  BF             PHI  RF       ;  direction vectors in MOVTAB
  3A  EF             SEX  RF       ;X+F for simple additions coming up
  3B  8C             GLO  RC
  3C  F4             ADD
  3D  FA 3F          ANI  $3F
  3F  32 5E          BZ   5F
```

GET Y

```
037F  D4 03 6E   GETY: CALL RAND    ;Same as GETX
  82  88               GLO  R8
  83  FA 7F            ANI  $7F     ;But limit to 00 - $7F range
  85  BE               PHI  RE
  86  D5               RETN         ;Return
```

COMPUTER ANIMATED ANIMATION

Earlier in this book, graphic animations were compared to the process of cartooning. When page switching is used to animate a display, the effect is much like that of a series of pages flipping before your eye tricking your brain into seeing a moving figure.

One of the most fascinating graphics techniques is to let the computer do the work of the animating. Because the intermediate "pages" of an animation are derived fairly easily though tediously, it is preferable to let the computer draw them out of objects which you give to it.

The following subroutine is only one example of how a computer can be used not only to display an animated pattern, but to actually be responsible for creating the animation. You supply two or more shape tables containing equal numbers of X,Y coordinate pairs. The subroutine MERGE will cause each of your shapes to melt into each other in an endless cycle. All of the intermediate pictures -- the actual animations -- are calculated by MERGE. You only tell the computer which figure to start with and what you want that figure to eventually become.

MERGE has been located at $0400 but could be easily rewritten

```
0341  AC              PLO  RC      ;Add direction vectors addressed
  42  60              IRX          ;  by RF to X and Y coordinates
  43  9C              GHI  RC      ;  in RC and RD for new line
  44  F4              ADD          ;  endpoints.  The coordinates are
  45  FA 7F           ANI  $7F     ;  each limited to the correct
  47  32 5E           BZ   5F      ;  display ranges, but when any
  49  BC              PHI  RC      ;  coordinate reaches the edge of
  4A  60              IRX          ;  the display, the branch to
  4B  8D              GLO  RD      ;  $035E is taken and a new line
  4C  F4              ADD          ;  will be drawn until the loop
  4D  FA 3F           ANI  $3F     ;  terminates.
  4F  32 5E           BZ   5F
0351  AD              PLO  RD
  52  60              IRX
  53  9D              GHI  RD
  54  F4              ADD
  55  FA 7F           ANI  $7F
  57  32 5E           BZ   5F
  59  BD              PHI  RD
  5A  22              DEC  R2      ;Protect address
  5B  E2              SEX  R2      ;Reset X=2
  5C  30 25           BR   3B      ;Loop to continue this line
  5E  27          5H: DEC  R7      ;Count # loops
  5F  87              GLO  R7      ;Test count
0360  3A 1A           BNZ  2B      ;Loop on R7.0 ≠ 0
  62  F8 FF           LDI  $FF     ;Load RF.1 with timer value
  64  BF              PHI  RF
  65  2F          5H: DEC  RF      ;Decrement register RF (over 16 bits)
  66  9F              GHI  RF      ;Test high eight bits of RF
  67  3A 65           BNZ  5B      ;If not zero yet, loop back to 0365
  69  D4 01 30        CALL CLEAR   ;Else erase display
  6C  30 03           BR   1B      ;Branch to restart new pattern

036E  88          RAND: GLO  R8    ;Get seed from R8.0
  6F  52              STR  R2      ;Push seed
  70  FE              SHL          ;Multiply by 4 using
  71  FE              SHL          ;  shifting to the left
  72  F4              ADD          ;Add to seed
  73  FC 02           ADI  $2      ;Add constant
  75  A8              PLO  R8      ;Put in R8.0 to pass back
  76  D5              RETN         ;Return
```

GET X

```
0377  D4 03 6E  GETX: CALL RAND    ;Generate a random number
  7A  88              GLO  R8      ;Limit that number to
  7B  FA 3F           ANI  $3F     ;  00 - $3F range
  7D  BE              PHI  RE      ;Place in RE.1 to pass back
  7E  D5              RETN         ;Return
```

to fit anywhere you want. You must use a copy of the Graphics Subs Package #2 with the FLOP Sub enabled. (Please read the section on FLOP if you are unsure how to do this.) If you have the tape, you only need to load the program into your computer.

MERGE is a subroutine, remember, and will not do anything completely on its own. Also listed here (and included with MERGE on tape) is a MERGE CONTROLLER, the main program which goes into locations $0300 - 0335. This is needed to set up the proper pointers that MERGE needs in order to do its job.

You may use MERGE in your own programs, but for now I suggest you experiment with merging the shapes presented here and later by trying to enter some of your own shapes.

Shape tables begin at $0450 and may extend up to $06FF. MERGE requires the work space at $0700 to $07FF, and the two display refresh pages are located from $0800 to $0FFF. You need at least 4K of memory on board your VIP to use this subroutine.

The format for a shape table is exactly as described before with two important differences:

1) All shape tables must be exactly the same length
2) An extra $FFFF stop indicator must follow the <u>last</u> shape table

To make all shape tables come out to the same size, you may retrace lines or simply duplicate the last X,Y pair of shorter

tables to make them longer.

After entering a series of shape tables, you need to tell the program how big the tables are and where you want the cycle to begin repeating. Here's how you do that.

```
031A  NN      ;Enter number bytes in shape table including
                the $FFFF stop indicator
0325  NN      ;High part address of first table to repeat
--
0328  NN      ;Low part address of first table to repeat
```

You may want all of your shapes to merge in one big complete circle, but you may have a title or something for the first picture which you do not want to be repeated on each cycle. That's why I've included the capability to have the restart begin with any shape table.

The listing here begins with the MERGE CONTROLLER then lists the MERGE sub. If you are hand loading the program, you should not run it without first entering shape tables as just explained. The sample MERGE programs BOX and MOVE ON may be loaded and run directly from tape. The shape tables for these programs are given following the listing.

MERGE CONTROLLER

```
0300  D4 00 7F  BEGIN: CALL FLOP
  03  D4 01 30        CALL CLEAR
  06  F8 04           LDI  $04
  08  B7              PHI  R7
  09  B8              PHI  R8
  0A  F8 50           LDI  $50
  0C  A7              PLO  R7      ;R7← $0450
  0D  A8              PLO  R8      ;R8← $0450
  0E  D4 00 84        CALL SHAPE
0311  D4 00 7F        CALL FLOP
  14  98           1H: GHI  R8
  15  B7              PHI  R7
  16  88              GLO  R8
  17  A7              PLO  R7      ;R7← R8
  18  88              GLO  R8
  19  FC 60           ADI  $60     ;Size of matrices
  1B  A8              PLO  R8
  1C  98              GHI  R8
  1D  7C 00           ADCI $00
  1F  B8              PHI  R8      ;R8← R8 + matrix size
0320  08              LDN  R8
  21  FE              SHL          ;Test end cycle
  22  3B 2A           BNF  2F
  24  F8 05           LDI  $05
  26  B8              PHI  R8
  27  F8 10           LDI  $10
  29  A8              PLO  R8      ;R8← Restart matrix address
  2A  F8 80        2H: LDI  $40
  2C  BF              PHI  RF
  2D  2F           2H: DEC  RF      ;Wait before continuing
  2E  9F              GHI  RF
  2F  3A 2D           BNZ  2B
  31  D4 04 00        CALL MERGE
  34  30 14           BR   1B      ;Loop forever
```

MERGE SUB

```
                ;Requires Graphics Subs with page switching enabled
                ;Also needs a control routine.  Do not run alone.

0400  F8 07     MERGE: LDI  $07    ;Set up pointer to image buffer
  02  BE             PHI  RE      ;  at $0500
  03  F8 00          LDI  $0      ;   "    "      "     "     "
  05  AE             PLO  RE      ;   "    "      "     "     "
  06  47          1H: LDA  R7     ;Transfer matrix #1 to
  07  5E             STR  RE      ;  the image buffer
  08  1E             INC  RE      ;  Does full 256 bytes for
  09  8E             GLO  RE      ;  simplicity
  0A  3A 06          BNZ          ;   "          "          "
```

```
040C  F8 07        2H: LDI  $07    ;Set up R7 as pointer to
  0E  B7               PHI  R7     ;  image buffer
  0F  F8 00            LDI  $0     ;   "       "
0411  A7               PLO  R7     ;   "       "
  12  D4 01 30         CALL CLEAR  ;Clear the off screen
  15  D4 00 84         CALL SHAPE  ;Draw a shape
  18  D4 00 7F         CALL FLOP   ;View the shape
  1B  F8 07            LDI  $07    ;Set up RE & RF as work
  1D  BE               PHI  RE     ;  pointers to the
  1E  F8 00            LDI  $0     ;  matrices
0420  AE               PLO  RE     ;   "       "
  21  AD               PLO  RD     ;(RD.0 is a flag)
  22  98               GHI  R8     ;   "       "
  23  BF               PHI  RF     ;   "       "
  24  88               GLO  R8     ;   "       "
  25  AF               PLO  RF     ;   "       "
  26  EE               SEX  RE     ;X=E for coming comparisons
  27  0F           3H: LDN  RF     ;Begin merge loop
  28  F7               SM          ;Subtract M(R(F)) - M(R(E))
  29  F8 FF            LDI  $FF    ;Load -1 in case adjustment needed
  2B  3B 33            BM   4F     ;Go adjust on M(R(E))> M(R(F))
  2D  0F               LDN  RF     ;( M(R(E)) is≤ M(R(F)) here)
  2E  F5               SD          ;Subtract M(R(E)) - M(R(F))
  2F  F8 01            LDI  $1     ;Load +1 in case adjustment needed
0431  33 38            BPZ  5F     ;Skip adjust on M(R(E))=M(R(F))
  33  F4           4H: ADD         ;Add D to M(R(E)) and
  34  5E               STR  RE     ;  put new value into buffer
  35  F8 01            LDI  $1     ;Set flag to indicate a
  37  AD               PLO  RD     ;  change was made
  38  1E           5H: INC  RE     ;Point to next coordinate in
  39  1F               INC  RF     ;  buffer and matrix #2
  3A  0E               LDN  RE     ;Test for end of matrix
  3B  FE               SHL
  3C  3B 27            BNF  3B     ;If not end, continue merging
  3E  8D               GLO  RD     ;Else test flag
  3F  3A 0C            BNZ  2B     ;Go display if a change was made
0441  D5               RETN        ;Else return on no changes made
                                   ;  that is, image buffer = matrix #2
```

MOVE ON

NOTE: If you are hand loading, first enter the MERGE and MERGE CONTROLLER routines, then enter the following information and shape tables.

```
031A  60          ;Length of each shape table
 --
0325  05          ;Page address at which the cycle repeats
 --
0328  10          ;Low byte address of the repeat cycle

0450  00 3E 00 22 03 2E 06 22 06 3E 08 3E 08 22 0E 22
  60  0E 3E 08 3E 0E 3E 0E 22 10 22 13 3E 16 22 1E 22
  70  18 22 18 30 1C 30 18 30 18 3E 36 3E 36 22 30 22
  80  30 3E 38 3E 38 22 3E 3E 3E 22 3E 3E 3E 22 3E 22
  90  3E 22 3E 22 3E 22 3E 22 3E 22 3E 22 3E 22 3E 22
  A0  3E 22 3E 22 3E 22 3E 22 3E 22 3E 22 3E 22 FF FF
  B0  03 3E 03 22 00 22 0E 22 08 22 08 3E 0E 3E 0E 22
  C0  0E 3E 10 3E 10 22 13 2E 16 22 16 3E 26 3E 26 30
  D0  20 30 20 22 28 22 28 3E 2B 2C 2E 3E 2E 22 30 22
  E0  30 3E 30 22 36 22 36 30 30 30 36 30 36 3E 38 3E
  F0  38 22 3E 3E 3E 22 3E 22 3E 22 3E 22 3E 22 3E 22
0500  3E 22 3E 22 3E 22 3E 22 3E 22 3E 22 3E 22 FF FF
  10  0E 09 0E 20 15 20 15 09 15 20 14 20 14 24 13 24
  20  14 24 14 27 13 27 14 27 14 2A 13 2A 14 2A 14 2C
  30  1D 2C 21 28 25 28 29 2C 2A 2E 2A 3F 2B 3F 2B 43
  40  1F 43 1B 3F 17 3F 17 43 10 43 10 21 10 3A 17 3A
  50  17 3E 17 3A 1B 3A 20 3F 29 3F 29 3F 29 3F 29 3F
  60  29 3F 29 3F 29 3F 29 3F 29 3F 29 3F 29 3F FF FF
  70  0C 4C 10 3C 13 33 16 30 24 32 24 3F 1B 3F 1B 31
  80  24 32 2C 2A 2E 23 30 2C 33 35 33 39 2C 36 27 44
  90  27 61 28 61 25 61 25 4B 20 47 1F 47 1F 3F 1F 4F
  A0  20 4F 1F 4F 1F 47 1C 47 19 4A 19 61 1A 61 17 61
  B0  17 4A 14 3F 16 31 16 31 16 31 16 31 16 31 16 31
  C0  16 31 16 31 16 31 16 31 16 31 16 31 16 31 FF FF
  D0  08 49 05 46 05 3D 1D 3D 05 3D 05 36 09 32 13 30
  E0  17 1F 2B 1F 2F 29 36 30 3C 30 3F 33 3F 37 3A 37
  F0  3A 3C 3F 3C 3F 37 3F 46 3B 49 3B 4F 37 55 33 4F
0600  33 49 3B 49 2E 49 2E 31 2E 49 08 49 0F 49 0F 4F
  10  13 55 17 4F 17 49 20 49 20 37 22 37 20 37 20 20
  20  20 30 13 30 36 30 2C 30 28 2C 27 2D 2A 2A FF FF
  30  14 49 0D 49 02 3E 02 39 0B 39 0B 30 03 38 0B 30
  40  2D 32 36 29 3B 29 3B 38 36 38 3B 38 3B 40 35 46
  50  28 46 37 55 2E 5E 2A 5E 0D 41 15 39 1B 39 28 46
  60  1F 3D 26 3D 26 39 26 3D 28 3D 28 39 2A 39 2A 3B
  70  28 3B 2A 3B 2A 3D 28 3D 2E 3D 2E 39 2C 39 2C 3D
  80  34 3D 30 3D 30 3B 32 3B 32 39 30 39 30 39 FF FF
  90  22 42 26 3E 2D 3E 2D 38 2A 38 2A 2A 37 2A 38 24
  A0  27 24 26 26 26 28 27 2A 2A 2A 2A 38 21 38 24 38
  B0  21 30 23 30 23 2C 1C 2C 1A 28 1A 2A 19 28 19 2A
  C0  18 2A 17 2C 10 2C 10 30 17 30 19 34 19 32 1A 34
  D0  1C 30 21 30 1C 30 1F 36 20 36 1F 36 1C 3C 24 3C
  E0  1D 3C 1D 3E 1F 42 22 42 22 42 22 42 22 42 FF FF
  F0  FF FF
```

BOX

NOTE: If you are hand loading, first enter the MERGE and MERGE CONTROLLER routines, then enter the following information and shape tables.

```
031A  26          ;Length of each shape table
 --
0325  04          ;Page address at which the cycle repeats
 --
0328  50          ;Low byte address of the repeat cycle

0450  08 10 0E 10 0E 2C 08 2C 09 2C 09 1E 0D 1E 09 1E
  60  09 10 16 10 16 2C 10 2C 10 10 18 10 1E 2C 1B 1C
  70  18 2C 1E 10 FF FF 08 10 0E 10 0E 10 08 10 09 10
  80  09 10 0D 10 09 10 09 10 16 10 16 10 10 10 10 10
  90  18 10 1E 10 1B 10 18 10 1E 10 FF FF 10 10 20 10
  A0  20 30 10 30 10 10 20 10 10 10 10 30 20 30 10 30
  B0  20 30 20 10 10 10 10 10 20 10 20 30 10 30 10 10
  C0  FF FF 10 10 20 10 20 30 10 30 10 10 20 10 28 18
  D0  28 38 20 30 10 30 20 30 20 10 10 10 10 10 20 10
  E0  20 30 10 30 10 10 FF FF 10 10 20 10 20 30 10 30
  F0  10 10 20 10 28 18 28 38 20 30 10 30 18 38 18 18
0500  10 10 10 10 20 10 20 30 10 30 10 10 FF FF 10 10
  10  20 10 20 30 10 30 10 10 20 10 28 18 28 38 20 30
  20  10 30 18 38 18 18 10 10 18 18 28 18 28 38 18 38
  30  18 18 FF FF FF FF
```

The two sample programs MOVE ON and BOX show how MERGE may be used to construct figures out of seemingly random lines as in MOVE ON or to animate a pattern as in BOX. By carefully positioning each point in the shape tables, the BOX is opened by the computer. Only the <u>end</u> patterns for each BOX segment were entered. The computer did all the calculating for the apparent movements of the sides opening etc.

The following shape tables are presented to help you get started MERGING your own shapes. These are not included on the tape. You must load them using the "write mode" of your VIP operating system.

Before doing this, however, you must have these things already in your computer:

1) Graphics Subs Package #2 with FLOP enabled ($0000-$01FF)
2) MERGE CONTROLLER at $0300
3) MERGE sub at $0400

If you have the tape, simply load either BOX or MOVE ON into your VIP and you're ready to begin merging by entering the following tables.

A SIMPLE SAMPLE

The following shapes are intended to show you who the <u>real</u> VIP is around here. Starting at $0450, enter both of the following shape tables.

SHAPE #1

```
0450  00 20 08 50 10 20 28 20 20 20 20 50 18 50 30 50
  60  30 20 3F 20 3F 36 30 36 30 36 FF FF
```

SHAPE #2

```
046C  00 20 08 38 08 50 08 38 10 20 28 20 28 50 18 50
  7C  18 20 30 20 30 50 3F 50 3F 20 FF FF FF FF
```

Notice that both tables are of the identical size but that the last table ends with a second $FFFF stop indicator. Notice also that Shape #1 is the same one from the chapter describing the SHAPE sub but with the last X,Y pair (30,36) repeated one time to make both tables the same number of bytes long.

Now, before flipping to run, enter the following information:

```
031A  1C        ;# bytes each table in hexadecimal
 --
0325  04        ;High and low bytes of the address $0450 at which
 --
0328  50        ;  the MERGE cycle is to repeat
```

The program may now be run and the above two tables will continually MERGE back and forth into each other.

Here are some other shapes to use with MERGE. Remember to enter the size of the tables and the address of the shape to be repeated as just explained in the Simple Sample.

NINE ARABS

```
031A  12        ;Length of each of nine shape tables page
 --
0325  04        ;Page address at which cycle repeats*
 --
0328  50        ;Low byte address of the repeat cycle

0450  1D 47 21 47 1F 47 1F 39 1F 3B 1E 3B 1E 3C 1E 3B
  60  FF FF 1D 39 21 39 21 40 1D 40 1D 47 21 47 1D 47
  70  1D 40 FF FF 1D 47 21 47 21 40 1D 40 21 40 21 39
  80  1D 39 21 39 FF FF 1D 39 1D 40 21 40 20 40 20 3B
  90  20 47 20 3B 20 40 FF FF 21 39 1D 39 1D 40 21 40
  A0  21 47 1D 47 21 47 21 40 FF FF 1D 39 1D 47 21 47
  B0  21 40 1D 40 21 40 21 47 1D 47 FF FF 21 47 21 39
  C0  1D 39 21 39 21 47 21 39 1D 39 21 39 FF FF 21 47
  D0  21 39 1D 39 1D 40 21 40 1D 40 1D 47 21 47 FF FF
  E0  21 39 1D 39 1D 40 21 40 21 39 21 47 21 39 21 40
  F0  FF FF FF FF
```

*HINT: If you want to prevent a cycle from repeating, enter the address of the last shape table in the sequence. It will continue to MERGE into itself, but will appear to be stationary on the display. For example, to stop at the ninth ARAB, enter $04 at $0325 and $E0 at $0328. To recycle from a mid point, try entering $0498 or $0474 as the recycle address.