SECTION 21: COLOR, GRAPHICS, AND SPRITES

The TI Home Computer gives you the capability of displaying a wide variety of colored graphics and sprites, enabling you to make your programs lively and interesting. You can place the screen in one of four modes: text, graphics, multicolor, and bit-map (available only on the TI-99/4A).

In graphics mode, you can use the standard ASCII characters and define new characters. You can make characters and their backgrounds a variety of colors. The screen is 32 columns by 24 lines. This is the mode used by the Editor/Assembler except when editing, TI BASIC, and most applications.

In multicolor mode, you can set the colors of a number of small boxes. The screen is 64 columns by 48 lines.

In text mode, you can use the standard ASCII characters and define new characters. All characters are one color, and the background is one color. The screen is 40 columns by 24 lines. This is the mode used by the Editor.

In bit-map mode (available only on the TI-99/4A because of its use of the TMS9918A video processor instead of the TMS9918 video processor), you can set any pixel (the smallest dot on the screen) on or off and make the pixels and the background a variety of colors. The screen is 256 columns by 192 lines.

In all modes except text, up to 32 sprites (moving graphics) can be created and set in motion without further program control.

21.1 VDP WRITE-ONLY REGISTERS

Before using the different modes, certain preliminary information is necessary. The following describes the eight VDP write-only registers.

VDP Register 0 — The default for Register 0 is >00 for the Editor/Assembler, TI BASIC, and TI Extended BASIC.

- Bits 0 - 5 — Reserved. Must be 000000.
- Bit 6 — Mode bit 3, called M3. If this bit is set, the display is in bit-map mode.
- Bit 7 — External video enable/disable. A value of 1 enables video input and a value of 0 disables video input.

VDP Register 1 — The default for Register 1 is >E0 in the Editor/Assembler, TI BASIC, and TI Extended BASIC. **Note:** Before changing this Register, put a copy of the new value you wish it to have at address >83D4. When a key is pressed, a copy of the value at this address is placed in Register 1.

- Bit 0 — 4/16K selection. A value of 0 selects 4K RAM operation, and a value of 1 selects 16K RAM operation.
- Bit 1 — Blank enable/disable. A value of 0 causes the active display (the entire screen) to be blank, and a value of 1 allows display on the screen. With a value of 0, the screen only shows the border color.
- Bit 2 — Interrupt enable/disable. A value of 0 disables VDP interrupt, and a value of 1 enables VDP interrupt.
- Bit 3 — Mode bit 1, called M1. If this bit is set, the display is in text mode.
- Bit 4 — Mode bit 2, called M2. If this bit is set, the display is in multicolor mode.
- Bit 5 — Reserved. Must be 0.

Bit 6 Sprite size selection. A value of 0 selects standard size sprites, and a value of 1 selects double-size sprites.

Bit 7 Sprite magnification selection. A value of 0 selects unmagnified sprites, and a value of 1 selects magnified sprites.

VDP Register 2 The default for Register 2 is >00 in the Editor/Assembler, TI BASIC, and TI Extended BASIC.

Defines the base address of the Screen Image Table. The Screen Image Table base address is equal to the value of this register times >400.

VDP Register 3 The default for Register 3 is >0E in the Editor/Assembler, >0C in TI BASIC, and >20 in TI Extended BASIC.

Defines the base address of the Color Table. The Color Table base address is equal to the value of this register times >40.

VDP Register 4 The default for Register 4 is >01 in the Editor/Assembler and >00 in TI BASIC and TI Extended BASIC.

Defines the base address of the Pattern Descriptor Table. The Pattern Descriptor Table base address is equal to the value of this register times >800.

VDP Register 5 The default for Register 5 is >06 in the Editor/Assembler, TI BASIC, and TI Extended BASIC.

Defines the base address of the Sprite Attribute List. The Sprite Attribute List base address is equal to the value of this register times >80.

VDP Register 6 — The default for Register 6 is >00 in the Editor/Assembler, TI BASIC, and TI Extended BASIC.

Defines the base address of the Sprite Descriptor Table. The Sprite Descriptor Table base address is equal to the value of this register times >800.

VDP Register 7 — The default for Register 7 is >F5 in the Editor/Assembler and >17 in TI BASIC and TI Extended BASIC.

Bits 0 - 3 — The color code of the foreground color in text mode.

Bits 4 - 7 — The color code for the background color in all modes.

The mode bits, M1, M2, and M3, are in bits 3 and 4 of Register 1 and bit 6 of Register 0. They determine the mode of the display. If they are all reset, the display is in graphics mode. If M1, in bit 3 of Register 1, is set, the display is in text mode. If M2, in bit 4 of Register 1, is set, the display is in multicolor mode. If M3, in bit 6 of Register 0, is set, the display is in bit-map mode, available only on the TI-99/4A.

21.2 GRAPHICS MODE

In graphics mode, you can use the standard ASCII characters and define patterns or characters and their foreground and background colors. The display is 32 columns by 24 lines. You can use sprites. Color and graphics are available by defining each of the 256 characters and setting their foreground and background colors. The standard ASCII characters are predefined by the system software.

21.2.1 Pattern Descriptor Table

The Pattern Descriptor Table contains descriptions of the 256 patterns or characters. By changing these descriptions, you can alter the appearance of the character on the screen. The description of each of the 256 patterns or characters takes eight bytes of information. The description of the subprogram CHAR in the User's Reference Guide discusses character definition.

In the Editor/Assembler, the Pattern Description Table starts at address >0800. Thus, the description of character >00 occupies addresses >0800 through >0807, character >01 occupies addresses >0808 through >080F, and character >FF occupies addresses >0FF8 through >0FFF.

21.2.2 Color Table

The Color Table contains the descriptions of the foreground and background colors of the characters. The most-significant four bits of the byte specify the foreground color and the least-significant four bits specify the background color. Each byte specifies the color for a group of eight characters. The 16 colors available on the TI Home Computer and their hexadecimal codes are listed on the next page.

Color	Hexadecimal Code	Color	Hexadecimal Code
Transparent	0	Medium red	8
Black	1	Light red	9
Medium green	2	Dark yellow	A
Light green	3	Light yellow	B
Dark blue	4	Dark green	C
Light blue	5	Magenta	D
Dark red	6	Gray	E
Cyan	7	White	F

In the Editor/Assembler, the Color Table starts at address >0380. Thus, the byte at address >0380 specifies the colors of characters >00 through >07, the byte at address >0381 specifies the colors of characters >08 through >0F, and the byte at address >039F specifies the colors of characters >F8 through >FF.

For example, placing a value of >17 at address >0384 sets the colors of characters >20 through >27 to black on cyan.

21.2.3 Screen Image Table

The Screen Image Table specifies the characters that occupy each of the screen positions. Each byte specifies the character at one screen position. The 768 screen positions are arranged on the screen in 24 rows of 32 columns.

In the Editor/Assembler, the Screen Image Table starts at address >0000. The first 32 addresses (>0000 through >001F) contain the characters for the first row, the second 32 addresses (>0020 through >003F) contain the characters for the second row, and so on.

For example, if the value >41 (normally the code for the ASCII character A) is at address >0022, the character described at addresses >0A08 through >0A0F of the Pattern Descriptor Table appears in the third column of the second row, assuming the Pattern Descriptor Table starts at address >0800.

21.3 MULTICOLOR MODE

In multicolor mode, the display is divided into 48 rows, each containing 64 "boxes" that are four pixels by four pixels. Each of the 3072 boxes thus defined can be one of the 16 colors available. You can use sprites in multicolor mode.

You should initialize the Screen Image Table so that the first >80 bytes contain >00 through >1F repeated four times, the next >80 bytes contain >20 though >3F repeated four times, and so on, so that the last >80 bytes contain >A0 through >BF repeated four times.

The Pattern Descriptor Table, instead of containing patterns, contains colors. Each pattern in the Pattern Descriptor Table contains eight bytes. In multicolor mode, each group of eight bytes contains 16 color descriptions, each giving the color of one box. The colors are as given in Section 21.2.2. The left four bits of each byte describe the color of one box and the right four bits describe the color of the next box on the same row.

The first byte in the Pattern Descriptor Table defines the colors of the first two boxes in the first row. The second byte defines the colors of the first two boxes in the second row. The third byte defines the colors of the first two boxes in the third row. This continues until the colors of the first two boxes in each of the first eight rows have been defined.

The next eight-byte segment similarly defines the colors of the third and fourth boxes in each of the first eight rows. This definition continues until the first 32 eight-byte segments have described all the boxes in the first eight rows. Subsequent groups of eight rows are described in a similar manner by subsequent groups of 32 eight-byte segments.

The following diagram represents the screen and how it is divided in multicolor mode. The Screen Image Table address is the offset from the beginning. The Screen Image Table value is what you should insert in the memory location.

Screen Image Table Address	Row	Columns 1 2 3 4 ... 63 64	Screen Image Table Value
>0000 - >001F	1	| | | | | ... | | |	>00 - >1F
>0000 - >001F	2	| | | | | ... | | |	
>0020 - >003F	3	| | | | | ... | | |	>00 - >1F
>0020 - >003F	4	| | | | | ... | | |	
>0040 - >005F	5	| | | | | ... | | |	>00 - >1F
>0040 - >005F	6	| | | | | ... | | |	
>0060 - >007F	7	| | | | | ... | | |	>00 - >1F
>0060 - >007F	8	| | | | | ... | | |	
>0080 - >009F	9	| | | | | ... | | |	>20 - >3F
>0080 - >009F	10	| | | | | ... | | |	
.	.	.	.
.	.	.	.
.	.	.	.
>02E0 - >02FF	47	| | | | | ... | | |	>A0 - >BF
>02E0 - >02FF	48	| | | | | ... | | |	

The following table shows the Screen Image Table character code, the addresses in the Pattern Descriptor Table, assuming that it starts at address >0800, and the portions of the screen that those characters and addresses describe.

Screen Image Table Character Code	Pattern Descriptor Table Address	Row and Columns Described Row	Columns
>00	>0800	1	1 and 2
>00	>0801	2	1 and 2
>00	>0802	3	1 and 2
>00	>0803	4	1 and 2
>00	>0804	5	1 and 2
>00	>0805	6	1 and 2
>00	>0806	7	1 and 2
>00	>0807	8	1 and 2
>01	>0808	1	3 and 4
>01	>080A	2	3 and 4
.		.	.
.		.	.
.		.	.
>BF	>0DFF	48	63 and 64

21.4 TEXT MODE

In text mode, the display is 40 columns by 24 lines. You cannot use sprites. The tables used to generate the patterns are the same as the Screen Image Table and Pattern Descriptor Table used in graphics mode. However, since 960 screen positions are used instead of 768, the Screen Image Table is longer. The definitions ignore the last two bits in each entry so that each character has a 6-by-8 pixel definition. The Editor is in text mode.

The two colors available in text mode are defined in VDP write-only Register 7. The leftmost four bits describe the color of the pixels that are on and the rightmost four bits describe the color of the pixels that are off.

For example, if the Screen Image Table starts at address >0000 and >41 is at address >0202, the ASCII symbol A is placed on the 35th column of the 13th row. In graphics mode, however, this address and value would place the A on the third column of the 17th row.

21.5 BIT-MAP MODE

In the TI-99/4A Home Computer, the bit-map mode is available for defining the display. In bit-map mode, you can independently define each of the 768 (32-by-24) positions of the screen. Additionally, more color information is available for each 8-by-8 pixel pattern. You can use sprites, but not their automatic motion feature.

In the bit-map mode, the patterns that occupy screen positions are described in the Screen Image Table, the pattern descriptions are in the Pattern Descriptor Table, and the colors of the characters are described in the Color Table.

21.5.1 Screen Image Table

The Screen Image Table lists the names of the patterns, from the Pattern Descriptor Table, that are to be generated. Each name is a single byte from >00 to >FF.

The table is divided into three sections, with each section describing 256 entries. The first section of 256 entries uses descriptions taken from the first 256 entries in the Pattern Generator Table and the Color Table. The second section of 256 entries uses descriptions taken from the second 256 entries in the Pattern Generator Table and the Color Table, and the third section of 256 entries uses descriptions taken from the third 256 entries in the Pattern Generator Table and the Color Table.

The first 32 entries describe the patterns that are placed on the first row of the screen, the second 32 entries describe the patterns on the second row of the screen, and so on. The Screen Image Table should usually be placed starting at address >1800 by setting VDP write-only Register 2 to >06.

21.5.2 Pattern Descriptor Table

The Pattern Descriptor Table is divided into three sections of 256 entries each and thus contains the 768 possible patterns. Each description is eight bytes long. The description of the subprogram CHAR in the User's Reference Guide discusses character definition.

The descriptions in the first third of the table, 256 entries or 2048 bytes, describe the characters in the first third of the screen. The descriptions in the second third of the table describe the characters in the second third of the screen and the descriptions in the last third of the table describe the characters in the last third of the screen.

The Pattern Descriptor Table is >1800 bytes long. You must start it either at address >0000 or >2000 by placing either >00 or >04 in VDP write-only Register 4. If the Pattern Descriptor Table starts at address >0000, the Color Table must start at address >2000, and vice versa.

21.5.3 Color Table

The Color Table contains the descriptions of the colors of the characters in the Pattern Descriptor Table. The color codes are as described in Section 21.2.2. Eight bytes are used to describe the colors of each character. The first nybble of each byte describes the color of the pixels that are on in one row of eight pixels, and the second nybble describes the color of the pixels that are off in the same row of eight pixels.

The color descriptions in the first third of the table, 256 entries or 2048 bytes, describe the colors of the characters in the first third of the screen. The descriptions in the second third of the table describe the colors of the characters in the second third of the screen and the descriptions in the last third of the table describe the colors of the characters in the last third of the screen.

The Color Table is >1800 bytes long. You must start it either at address >0000 or >2000 by placing either >00 or >04 in VDP write-only Register 3. If the Color Table starts at address >0000, the Pattern Descriptor Table must start at address >2000, and vice versa.

21.5.4 Bit-Map Mode Discussion

In using the bit-map mode, it is usually easiest to initialize the Screen Image Table to >00 through >FF repeated three times, and then alter the entries in the Pattern Descriptor Table and the Color Table.

To alter a pixel on the screen, you must calculate the byte and bit to be changed in the Pattern Descriptor Table. To alter the foreground and background colors of a row of eight pixels, you must calculate the byte that must be changed in the Color Table. The following program segment allows you to find those values.

The program segment assumes that the X-value of the pixel is in Workspace Register 0 (R0) and the Y-value of the pixel is in Workspace Register 1 (R1). The offset of the byte that you must change in the Pattern Generator Table is returned in Workspace Register 4 (R4), and the bit that must be altered is returned in Workspace Register 5 (R5). The offset of the byte that you must change in the Color Table is also returned in Workspace Register 4.

```
MOV     R1,R4       R1 is the Y value.
SLA     R4,5
SOC     R1,R4
ANDI    R4,>FF07
MOV     R0,R5       R0 is the X value.
ANDI    R5,7
A       R0,R4       R4 is the byte offset.
S       R5,R4       R5 is the bit offset.
```

21.5.5 Bit-Map Mode Example

Suppose the entry for a character in the Pattern Descriptor Table is >FF9999FF182442C3. This defines the character shown below.

Character	Pattern
|*|*|*|*|*|*|*|*|	FF
|*| | |*|*| | |*|	99
|*| | |*|*| | |*|	99
|*|*|*|*|*|*|*|*|	FF
| | | |*|*| | | |	18
| | |*| | |*| | |	24
| |*| | | | |*| |	42
|*|*| | | | |*|*|	C3

If the entry in the Color Table is >464646464D4D4D4D, the pattern is as follows, with B representing dark blue (>4), R representing dark red (>6), and M representing magenta (>D).

Character	Pattern	Colors
|B|B|B|B|B|B|B|B|	FF	46
|B|R|R|B|B|R|R|B|	99	46
|B|R|R|B|B|R|R|B|	99	46
|B|B|B|B|B|B|B|B|	FF	46
|M|M|M|B|B|M|M|M|	18	4D
|M|M|B|M|M|B|M|M|	24	4D
|M|B|M|M|M|M|B|M|	42	4D
|B|B|M|M|M|M|B|B|	C3	4D

On a magenta background, the magenta portions of the character blends with the background. With the pixel markings removed, the character appears as follows, with * representing dark blue (>4) and = representing dark red (>6).

```
* * * * * * * *
* = = * * = = *
* = = * * = = *
* * * * * * * *
      * *
    *     *
  *         *
* *         * *
```

21.6 SPRITES

Sprites are moving graphics that can occupy space on the screen independently and in addition to the characters which normally make up the screen. You can define and place in motion up to 32 sprites of any shape and several different sizes. After you start sprites moving, their motion continues without further program control. You can use sprites in graphics, multicolor, and bit-map mode. In bit-map mode, however, automatic motion cannot be used. Sprites are defined by setting up tables that indicate their position, their pattern, their color, their size, and their motion.

21.6.1 Sprite Attribute List

The Sprite Attribute List defines the position and color of each of the 32 possible sprites, numbered 0 through 31. As sprites move, the entries in the Sprite Attribute List are changed.

For sprites, the screen is divided into 192 (>C0) rows of 256 (>100) columns. Each of these locations is called a pixel, the smallest dot that can be displayed on the screen. The top row of pixels is designated >FF, followed by >00, >01, and so forth up to >BE. The left column of pixels is designated >00, followed by >01, >02, and so forth up to >FF.

Each sprite definition takes up four bytes in the Sprite Attribute List. The first byte is the vertical (Y) position of the sprite and starts at >FF, followed by >00 through >BE. The second byte is the horizontal (X) position of the sprite, which can be from >00 through >FF. The third byte is the pattern code, which can be from >00 through >FF. The fourth byte is the early clock attribute, which controls the location of the sprite, and color of the sprite.

Y-locations with values of >C0 through >FE are effectively off the bottom of the screen. However, a Y-location of >D0 causes that sprite and all following it in the Sprite Attribute List to be undefined. For example, if the Sprite Attribute List starts at address >0300 and no sprites are defined, the value >D0 should be placed at address >0300. If the fifth sprite is the last one active, a value of >D0 should be placed at address >0314. You can leave all 32 sprites active with the ones you do not wish to appear located off the bottom of the screen. However, it is recommended that you cause the final unused sprites to be undefined with a Y-location of >D0.

The third byte of each entry of the Sprite Attribute Table defines the character pattern to use for the sprite. The pattern can be from >00 to >FF and corresponds to a character defined in the Sprite Descriptor Table. For example, in the Editor/Assembler addresses >400 through >407 contain the entry for character >80.

The four most-significant bits in the fourth byte control the early clock of the sprite. If the last of these four bits is 0, the early clock is off. Then the sprite's location is its upper left-hand corner, and it fades in and out on the right edge of the screen. If the last of these four bits is 1, the early clock is on. Then the sprite's location is shifted 32 pixels to the left, allowing it to fade in and out on the left edge of the screen.

The color of the sprite is specified in the four least-significant bits of the fourth byte of the sprite description. The values used are the same as those given in Section 21.2.2.

In the Editor/Assembler, the Sprite Attribute List starts at address >0300. If you wish to use automatic motion, the Sprite Attribute List must start at that address. If you put the default base address (>0000) in VDP Register 6, the Sprite Descriptor Table (described in Section 21.6.2) starts at address >0000. Since the area >0000 through >03FF is used for the Screen Image Table, Color Table, and Sprite Attribute List, character codes starting at >80, at address >0400, are then normally used for sprites. When you use sprite motion, only the character codes from >80 through >EF can be used because the Sprite Motion Table starts at address >0780.

21.6.2 Sprite Descriptor Table

The Sprite Descriptor Table describes the sprites' patterns in the same way as in the Pattern Descriptor Table. However, sprites can be double-size or magnified by writing a value to the two least-significant bits in VDP Register 1. The following description tells the different sizes and magnifications possible.

Value	Description
00	Standard size sprites. Each sprite is 8 by 8 pixels, the same as a standard character on the screen.
01	Magnified sprites. Each sprite is 16 by 16 pixels, equal to four standard characters on the screen. The pattern definition is the same as for standard size sprites, but each pixel occupies four pixels on the screen.
10	Double-size sprites. Each sprite is 16 by 16 pixels, equal to four standard characters on the screen. Each sprite is defined by four consecutive patterns from the Sprite Descriptor Table. For example, each of the character codes >80, >81, >82, or >83 causes a double-size sprite to use characters >80, >81, >82, and >83 for the sprite. The first of these characters is the upper left-hand corner of the sprite, the second is the lower left-hand corner, the third is the upper right-hand corner, and the fourth is the lower right-hand corner.
11	Double-size magnified sprites. Each sprite is 32 by 32 pixels, equal to 16 standard characters on the screen. Sprites are defined as described under double-size sprites, and each pixel occupies four pixels on the screen.

In the Editor/Assembler, the Sprite Descriptor Table starts at address >0000 for pattern code >00. However, addresses >0400 and above are usually used for the block because the lower addresses are used for the Screen Image Table, Color Table, and Sprite Attribute List. The pattern defined starting at address >0400 is referred to as pattern code >80 in the Sprite Attribute Table.

21.6.3 Sprite Motion Table

The Sprite Motion Table defines the motion of sprites. It must start at address >0780. In order to move sprites, you must set up a number of conditions.

First, interrupts must be enabled during the execution of the program. Therefore, every time the program accesses the VDP RAM, interrupt handling must be disabled, which is the default. If you have enabled interrupt handling with the LIMI 2 instruction, you must disable it with a LIMI 0 instruction so that the interrupt handling routine does not alter the VDP write address.

Second, an indication of the number of sprites which have motion must be put in CPU RAM address >837A. For example, if sprites 2 and 4 are moving, the number 5 must be put at that address to allow for the motion of sprites 0, 1, 2, 3, and 4.

Third, descriptions of the motion of the sprite must be put in the Sprite Motion Table which always starts at VDP address >0780. Each sprite's motion takes up four bytes in the table. The first byte defines the vertical (Y) motion of the sprite. The second byte defines the horizontal (X) motion of the sprite. The third and fourth bytes are used by the interrupt routine.

The velocity in the first and second bytes can range from >00 to >FF. Velocities from >00 to >7F are positive velocities (down for vertical motion and right for horizontal motion), and velocities from >FF to >80 are taken as two's-complement negative velocities (up for vertical motion and left for horizontal motion).

A value of >01 causes the sprite to move one pixel every 16 VDP interrupts, or about once every 16/60ths of a second.

Since sprites are set up by loading data into VDP RAM and the TI BASIC interpreter allows interrupts, you can run sprites by successive use of the statement CALL POKEV (see Section 17.1.6). However, caution must be taken not to interfere with the TI BASIC interpreter, which does not recognize the existence of sprites. It is possible that the sprites may cause the TI BASIC interpreter to stop functioning. In TI Extended BASIC, this problem does not exist.

21.7 GRAPHICS AND SPRITE EXAMPLES

The first two of the following three assembly language programs are similar in their effect. The first places several bubble shapes on the screen and moves them up the screen. It does not use sprites, so the motion is not smooth. The second program defines the shapes as sprites, so the motion is quite smooth. In addition, pressing any key toggles the sprites from standard size to magnified sprites and back. The third program is a demonstration of automatic sprite motion.

Each of these programs must be assembled with the R option, which automatically generates Workspace Registers, and run with the LOAD AND RUN option of the Editor/Assembler.

21.7.1 Graphics Example

In the following program, several characters shaped like bubbles are placed on the screen and moved up the screen. These characters are not sprites, so the motion is not smooth. Run the program with the LOAD AND RUN option of the Editor/Assembler, using the program name BUBBLE. To leave the program, the computer must be turned off because no provision has been made for returning to the Editor/Assembler.

```
        DEF   BUBBLE
        REF   VMBW,VMBR,VSBW
*
BBLE    DATA  >3C7E,>CFDF,>FFFF,>7E3C
COLOR   DATA  >F333
BBL     BYTE  >A0
SPACE   BYTE  >A8
LOC     DATA  >01DA,>020D,>0271,>02A5,>02D6,>02E1,>0000
MYREG   BSS   >20
*
* Set up colors.
*
BUBBLE
        LWPI  MYREG
        LI    R0,>394       Color Table 20 and 21.
        LI    R1,COLOR      Load colors >F3 and >33.
        LI    R2,2          Two bytes to load.
        BLWP  @VMBW         Move to VDP RAM.
```

```
*
* Set up character definition.
*
          LI      R0,>D00     Character >A0 location.
          LI      R1,BBLE     Definition of bubble character.
          LI      R2,8        8 bytes to move.
          BLWP    @VMBW
*
* Clear screen
*
          CLR     R0          Start at VDP RAM >0000.
LOOP1     MOVB    @SPACE,R1   Move space character.
          BLWP    @VSBW       Move one space at a time.
          INC     R0          Points to next location on screen.
          CI      R0,>300     Out of screen.
          JNE     LOOP1
*
* Place bubbles on the screen.
*
          MOVB    @BBL,R1     Load character code for bubble.
          LI      R2,LOC      Load pointer to address for bubble.
LOOP2     MOV     *R2+,R0     Load real address.
          MOV     R0,R0       Check if finished loading.
          JEQ     SCROLL      Finished.  Start scrolling the screen.
          BLWP    @VSBW       Write bubble on the screen.
          JMP     LOOP2
*
* Scroll Screen.
*
VDPBF1    BSS     >20
VDPBF2    BSS     >20
*
SCROLL
          CLR     R0          VDP source address.
          LI      R1,VDPBF1   CPU buffer address.
          LI      R2,>20      Number of bytes to move.
          BLWP    @VMBR       Move >20 from VDP RAM.
*
          LI      R0,>20      VDP address >20.
          LI      R1,VDPBF2   CPU buffer address.
          LI      R2,>20      Number of bytes to move.
```

```
LOOP3   BLWP   @VMBR        Copy the line.
        AI     R0, >20      Move to lower VDP memory.
        BLWP   @VMBW        Write back to the lower line.
        AI     R0,>40       Read next line.
        CI     R0,>300      Check if end of screen.
        JL     LOOP3        If not, copy more.
*
        LI     R0,>2E0      Write the last line.
        LI     R1,VDPBF1    CPU buffer where the first line is.
        BLWP   @VMBW        Move CPU to VDP.
*
        JMP    SCROLL       Keep scrolling.
*
        END
```

21.7.2 Sprite Example

In the following program, several sprites shaped like bubbles are placed on the screen and moved up the screen. Because sprites are used, the motion is quite smooth. Run the program with the LOAD AND RUN option of the Editor/Assembler, using the program name SBBLE. To leave the program, the computer must be turned off because no provision has been made for returning to the Editor/Assembler.

```
        DEF    SBBLE
        REF    VMBW,VMBR,VSBW,VSBR
        REF    VWTR,KSCAN
*
BBLE    DATA   >3C7E,>CFDF,>FFFF,>7E3C
BBL     BYTE   >80
SPACE   BYTE   >20
SLIST   DATA   >70D0,>800F,>8068,>800F,>9888,>800F
        DATA   >A828,>800F,>B0B0,>800F,>B808,>800F
        DATA   >D000
MYREG   BSS    >20
```

```
*
* Set up character definition.
*
SBBLE
        LWPI    MYREG
        LI      R0,>400         Sprite character >80.
        LI      R1,BBLE         Definition of character.
        LI      R2,8            8 bytes to move.
        BLWP    @VMBW
*
* Define sprites.
*
        LI      R0,>300         Address of Sprite Attribute List.
        LI      R1,SLIST        Pointer to the list.
        LI      R2,26           Move 26 bytes to VDP RAM at >300.
        BLWP    @VMBW           Move the list.
*
* Scroll screen.
*
KEYBRD  EQU     >8375
STATUS  EQU     >837C
*
SET     DATA    >2000
*
SCROLL
        CLR     R5              Counter for sprite size.
LOOP
        LI      R0,>300         Pointer to the first Y-location.
READ    BLWP    @VSBR           Read one byte into R1.
        SRL     R1,8            Make it a word operation.
        CI      R1,>00D0        Check to see if it is finished.
        JEQ     KEY             Check on key input.
        DEC     R1              Decrement Y-location.
        JNE     MOVE            Move up one pixel.
        LI      R1,>00C8        Adjust the pointer.
MOVE    SLA     R1,8            Change it back to byte operation.
        BLWP    @VSBW           Write back to the list.
        AI      R0,>4           Points to the next location.
        JMP     READ            Read next Y-pointer.
```

```
*
KEY
        CLR     @KEYBRD         Clear keyboard.
        BLWP    @KSCAN          Call key scan routine.
        MOV     @STATUS,R3      Move status byte.
        COC     @SET,R3         Check for status bit.
        JEQ     CHANGE          Key pressed, change size of sprites.
        JMP     LOOP            Otherwise, keep scrolling.
*
CHANGE
        MOV     R5,R5           Is R5 null?
        JNE     SMALL           Make sprites small.
LARGE
        INC     R5              Change R5.
        LI      R0,>01E1        Change R1 to >E1.
        BLWP    @VWTR           Modify VDP register.
        JMP     LOOP            Go back to loop.
SMALL
        CLR     R5              Clear R5.
        LI      R0,>01E0        Change R1 to >E0.
        BLWP    @VWTR           Modify VDP register.
        JMP     LOOP            Go back to loop.
*
        END
```

21.7.3 Automatic Sprite Motion Example

This program is an example of using automatic motion. It places four magnified sprites in the middle of the screen and moves them in different directions at different speeds. Note that the LIMI 2 instruction is given to allow interrupts to occur. Without interrupts, sprites cannot be moved. Then, the LIMI 0 instruction is given to prevent the rest of the program from inadvertantly changing VDP RAM registers which are being used by the sprites' motion.

Run the program with the LOAD AND RUN option of the Editor/Assembler, using the program name MOVE. To leave the program, the computer must be turned off because no provision has been made for returning to the Editor/Assembler.

```
         DEF   MOVE
         REF   VMBW,VWTR,VMBR,VSBW
*
NUM      EQU   >837A
SAL      EQU   >300
COLTAB   EQU   >384
PATTN    EQU   >400
SPEED    EQU   >780
*
COLOR    DATA  >FF00
BALL     DATA  >3C7E,>FFFF,>FFFF,>7E3C
SDATA    DATA  >6178,>8006
         DATA  >6178,>8003
         DATA  >6178,>8004
         DATA  >6178,>800B,>D000
SPDATA   DATA  >0404,>0000,>F808,>0000
         DATA  >0CF4,>0000,>F0F0,>0000
MYREG    BSS   >20
*
MOVE
         LWPI  MYREG       Load my own registers.
         LI    R0,COLTAB
         MOVB  @COLOR,R1
         BLWP  @VSBW       Load background color as white.
*
         LI    R0,PATTN
         LI    R1,BALL
         LI    R2,8
         BLWP  @VMBW       Load ball pattern.
*
         LI    R0,SAL
         LI    R1,SDATA
         LI    R2,17
         BLWP  @VMBW       Load Sprite Attribute List.
*
         LI    R0,SPEED
         LI    R1,SPDATA
         LI    R2,16
         BLWP  @VMBW       Load speed of sprites.
```

```
*
        LI      R0,>81E1
        BLWP    @VWTR           Load VDP Register to magnify sprites.
*
        LI      R1,4
        SLA     R1,8
        MOVB    R1,@NUM         Specify number of sprites.
*
LOOP
        LI      R0,SAL
        LI      R3,4            Repeat 4 times.
        LI      R2,2
*
LOOP2
        LIMI    2               Enable interrupt.
        LIMI    0               Disable interrupt.
*
        LI      R1,MYREG+14     Read it into Register 7.
        BLWP    @VMBR
        AI      R7,-24
        CI      R7,>B8C8        Check if 0 < Y < 184, 24 < X < 224.
        JH      ADJUST
NEXT    AI      R0,4            Look at next sprite.
        DEC     R3
        JEQ     LOOP
        JMP     LOOP2
ADJUST  LI      R1,SDATA        Reload Sprite Attribute List.
        LI      R2,2
        BLWP    @VMBW
        JMP     NEXT
        END
```