SPPU004

Video Display Processors

Programmer's Guide

Video Display Products

TEXAS INSTRUMENTS

Video Display Processors

Programmer's Guide

Texas Instruments
Semiconductor Products
P.O. Box 309066
Dallas, Texas 75380-9066

TEXAS INSTRUMENTS

IMPORTANT NOTICE

CONTENTS

APPENDICES

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

This is the first in a series of publications concerned with programming Texas Instruments Video Display Processors. This programmer's guide will pay close attention to the fundamentals of initializing and creating a display with the TMS9118/28/29 VDPs. The book also covers their predecessors, the TMS9918A/28A/29A VDPs, and serves as a prerequisite to future publications on the next generation of Texas Instruments Advanced Video Display Processors. Device differences are noted for your convenience.

The programming approach in this publication is at the assembly language level. Most programming examples are very general for the sake of clarity. Actual working programs written in 8088, 6502, TMS7000, and TMS9995 assembly languages are included in Appendix E.*

All necessary subjects about programming a VDP are covered in this programmer's guide. If a subject is not at first discussed thoroughly enough or if more information about a particular subject is desired, let the Table of Contents guide you to a more detailed discussion of that topic.

1.1 GENERAL VDP OPERATION

The VDP fetches data from Video RAM (VRAM) and processes it into a serial stream of data used to control the beam of a CRT as it sweeps across the screen. The VDP performs this operation over and over again, much like a program executing in a loop. The VDP does, however, perform many more functions in this simulated loop (see Figure 1-1).

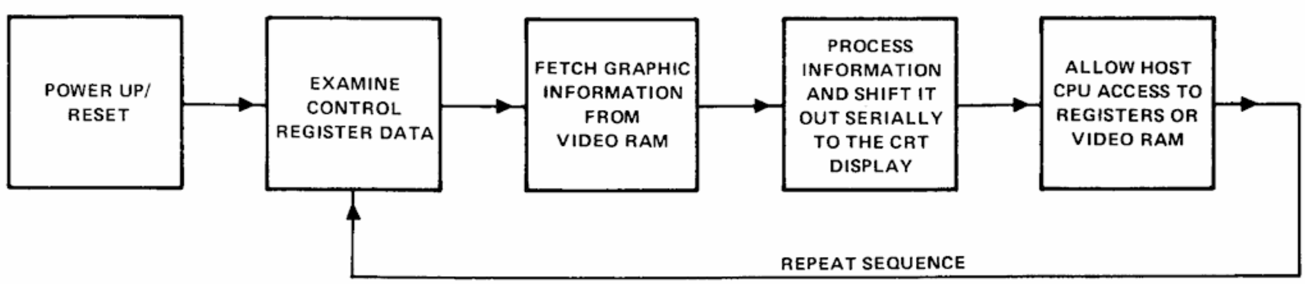


FIGURE 1-1 — VDP FLOW OF OPERATION

Much of the VDP's versatility stems from the fact that it is not restricted to fetching data from the same place in memory in the same sequence. The VDP has nine internal registers, eight of which contain option and control bits which may be programmed by the user. The ninth register is the Status Register and may be read by the user in order to determine certain things that are happening within the VDP. By programming information into the eight control registers, the VDP can be directed to fetch data from different VRAM locations in various sequences.

The VDP takes time out every few microseconds to see if the host CPU would like access to one of its internal registers or VRAM. If the VDP did not perform this function, it would not be possible to program the internal registers, read the status, or even load an artistic masterpiece into VRAM for display.

* 8088 is a registered trademark of the Intel Corporation, and 6502 is a registered trademark of MOS Technology.

1.2 REFERENCE MATERIAL

1) TMS9918A/28A/29A Video Display Processors Data Manual (MP010A)
2) TMS9118/28/29 Video Display Processors Data Manual (SPPS002)
3) TMS9928/29 and TMS9128/29 Interface to Color Monitors Application Report (SPPA004)
4) TMS9118/TMS9128/TMS9129 Evaluation Module User's Guide (SPPU003)
5) Dual Video Display Processor Application Report (SPPA005)

2. FEATURES

2.1 DISPLAY PLANES

The VDP displays an image on the screen that can best be thought of as a set of 35 display planes stacked on top of one another (see Figure 2-1). Looking at a monitor or television screen, we can visualize the highest priority plane as the closest to us and the lowest priority plane as the plane farthest away.

If patterns on different planes happen to be occupying the same spot on the screen, then the pattern on the highest priority plane will show through at that spot. For a particular pattern on a plane to show through, any pattern on higher priority planes directly in front of it must be set to the VDP color 'transparent'. See the TMS9118/28/29 Video Display Processors Data Manual (SPPS002) for more details.

The 35 prioritized planes are shown in Figure 2-1, with each of the first 32 planes containing a single sprite. A sprite is a definable object whose position on the screen is relative to X,Y coordinates. The X,Y coordinates are composed of two bytes in VRAM. By changing the data in these two bytes, a sprite can be moved smoothly around the screen to an X,Y position of one pixel. Sprites are available in two sizes, either 8x8 pixels or 16x16 pixels. These sprites can also be magnified to 16x16 or 32x32 pixels.

Behind the 32 Sprite Planes is the Pattern Plane. This plane is used to display either graphics or text. The VDP can display patterns on this plane in one of four possible modes: Text, Graphics I, Graphics II, or Multicolor.

2.2 DISPLAY MODES

Text Mode breaks the screen down into 6x8 pixel blocks specifically designed for displaying text. In Graphics I Mode, the screen is broken up into 32 horizontal blocks by 24 vertical blocks. Each block in Graphics I Mode contains 8x8 pixels, yielding a total screen resolution of 256x192 pixels. In Graphics II Mode, the screen breakdown and resolution are the same as in Graphics I Mode, but more complicated color and pattern displays are possible. Multicolor Mode is a low-resolution display mode which divides the screen into 64 horizontal blocks by 48 vertical blocks. Each block in Multicolor Mode contains 4x4 pixels and may be one of the sixteen colors available.

Behind the Pattern Plane is the Backdrop, which is larger in area than the other planes so that it forms a border around the other planes. The color of the Backdrop is defined by four bits in VDP Register 7.

The 35th and lowest priority plane is the External VDP Plane. If the output of a second VDP (slave) is detected by the main VDP (master), then all 35 planes generated by the second VDP will show through on this 35th plane. For an entity on the 35th plane to show through, all planes in front of the 35th plane must be transparent at that point.

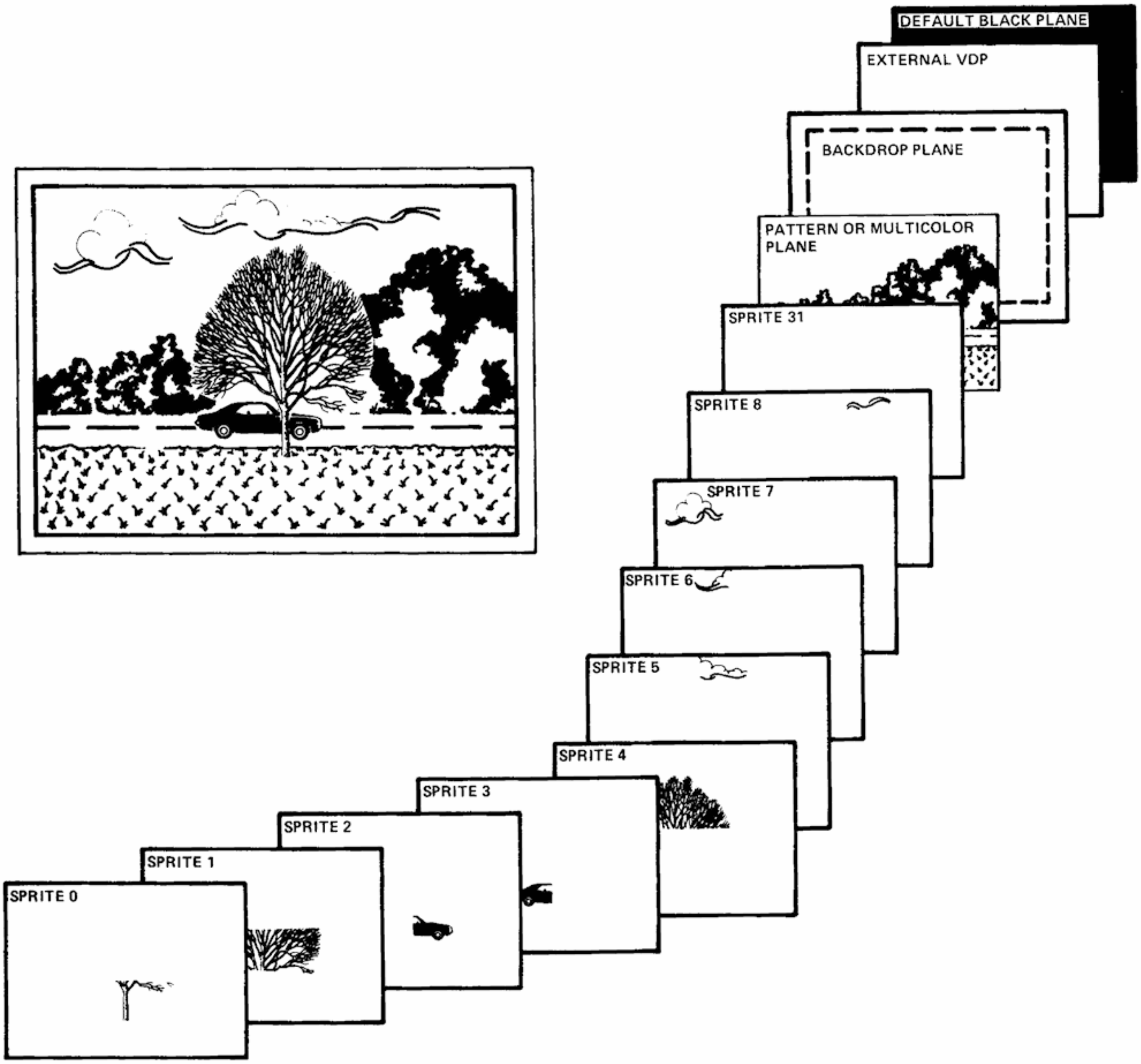


FIGURE 2-1 — VDP DISPLAY PLANES

2.3 AVAILABLE COLORS

The VDP can display 16 colors (including transparent) as shown in Table 2-1. The VDP can also display fifteen different gray levels on monochrome monitors.

TABLE 2-1 — VDP COLOR ASSIGNMENTS

COLOR NUMBER (IN HEX)	ACTUAL COLOR	COLOR NUMBER (IN HEX)	ACTUAL COLOR
0	Transparent	8	Medium Red
1	Black	9	Light Red
2	Medium Green	A	Dark Yellow
3	Light Green	B	Light Yellow
4	Dark Blue	C	Dark Green
5	Light Blue	D	Magenta (Purple)
6	Dark Red	E	Gray
7	Cyan (Aqua Blue)	F	White

3. COMMUNICATION BREAKDOWN

The circuit shown in Figure 3-1 is actually part of the Texas Instruments TMS9118/28/29 Evaluation Module (available for demonstration at your local TI Field Sales Office). We will use this circuit to help describe how the CPU and VDP communicate. This circuit is a complete working system.

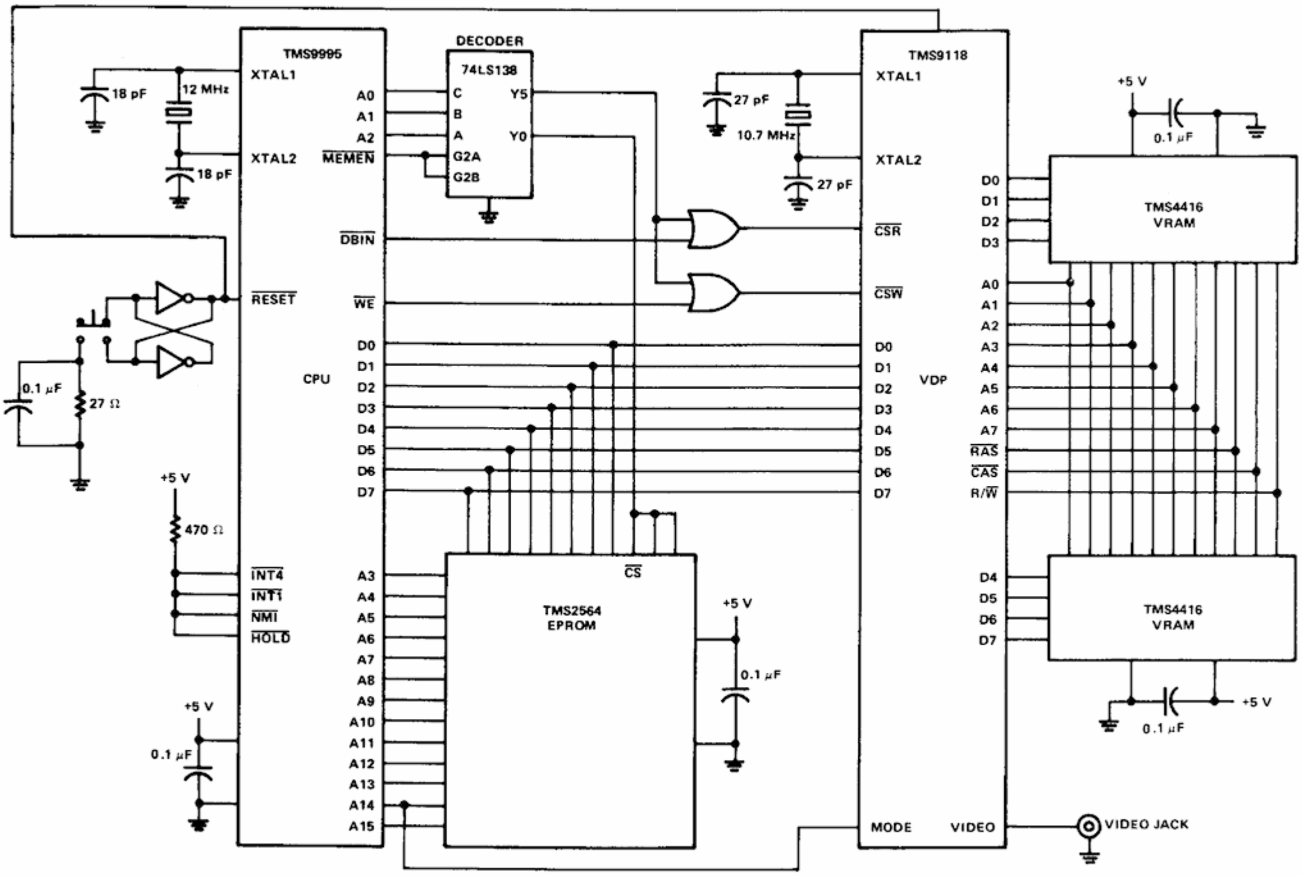


FIGURE 3-1 — CPU TO VDP INTERFACE

3.1 CPU TO VDP INTERFACE

The CPU communicates with the VDP through an eight-bit bidirectional data bus, three control lines, and an interrupt line. The three control signals are $\overline{\text{CSR}}$ (chip select read), $\overline{\text{CSW}}$ (chip select write), and MODE. $\overline{\text{CSR}}$ and $\overline{\text{CSW}}$ determine whether the VDP gets information off the data bus or puts information onto it. If $\overline{\text{CSR}}$ is active, the VDP will output information for the CPU onto the data bus. If $\overline{\text{CSW}}$ is active, the VDP will get information sent by the CPU off the data bus.

The MODE signal determines the VDP's source or destination for a data transfer. If the MODE signal is low, the VDP will do a VRAM operation. If the MODE signal is high, the VDP will do a VDP register operation.

One of the easiest ways to design the hardware interface is to set aside two addresses in the host CPU memory map for VDP communication. In the circuit shown in Figure 3-1, the two addresses set aside are Hex C000 and Hex C002. Performing a CPU operation at location Hex C000 will make the MODE signal low. Performing an operation at Hex C002 will make the MODE signal high. $\overline{CSR}$ and $\overline{CSW}$ are controlled by the CPU read/write logic. If a read operation is performed, $\overline{CSR}$ will be active (low), and if a write operation is performed, $\overline{CSW}$ will be active (low).

NOTE

> The addresses you will use in a particular VDP system will probably be different than Hex C000 and Hex C002, but the function will be the same.

In order to have the full capability of each VDP graphics mode, our VDP must have 16K bytes of VRAM available. This is also the most popular amount of VRAM found in VDP systems. VRAM is located in the VDP memory map from Hex 0000 to Hex 3FFF. As described earlier, VRAM can only be accessed through the VDP by reading or writing from memory locations Hex C000 and Hex C002.

Another important note to make concerns the examples using address and data lines. Examples in this guide refer to the most significant data line bit (MSB) as D0 and the least significant data line bit (LSB) as D7. This also holds true for the 14 bit address bus, with A0 being the MSB and A13 being the LSB.

3.2 SOFTWARE OPERATIONS

The CPU can be programmed to conduct one of four operations:

1) Write a byte of data to VRAM
2) Read a byte of data from VRAM
3) Write to one of the eight VDP internal registers, or set up the VRAM address by writing to the 14-bit Address Register
4) Read the VDP Status Register.

Each of these operations requires one or more data transfers to take place from the CPU to the VDP. The VDP determines which of these four data transfers is being performed by the state of the three control signals (CSR, CSW, and MODE) as shown in Table 3-1.

TABLE 3-1 — CPU TO VDP DATA TRANSFERS

OPERATION	$\overline{CSW}$	$\overline{CSR}$	MODE	PORT ADDRESS
Write to VRAM	0	1	0	C000
Read from VRAM	1	0	0	C000
Write to VDP register	0	1	1	C002
Read VDP Status Register	1	0	1	C002

NOTE

Memory-mapped addresses >C000 and >C002 are arbitrary addresses chosen for this guide.

4. TALKING TO THE VDP

4.1 WRITING TO THE VDP REGISTERS

The VDP has eight write-only registers and one read-only Status Register. The write-only registers contain information that controls the operation of the VDP, including the way VRAM is allocated. The Status Register contains interrupt and sprite information.

A VDP write-only register is loaded using two eight-bit data transfers from the CPU. The first byte written is the data, and the second byte is the register number and tells the VDP where to put the data just sent to it. The MSB of the second byte must be a 1, the next four bits must be 0s, and the lowest three bits are the actual register number (from 0 to 7). Table 4-1 shows the format for the eight write-only registers.

TABLE 4-1 — WRITE TO VDP REGISTERS

OPERATION	MSB 0	1	2	3	4	5	6	LSB 7	$\overline{CSR}$	$\overline{CSW}$	MODE
Data Write (Byte 1)	D0	D1	D2	D3	D4	D5	D6	D7	1	0	1
Register Select (Byte 2)	1	0	0	0	0	Rn	Rn	Rn	1	0	1

EXAMPLE 4-1.

Let's say we wish to initialize Register 0 (R0) with a value of Hex 00. The first byte written to address Hex C002 will be Hex 00, the second byte will be Hex 80 (remember from Table 4-1 that the MSB must be set to 1). If we had wanted to write Hex 00 to Register 7 (R7), then the second byte transferred would have been Hex 87. If Hex 00 was to be written to Register 7 (R7), then the second byte transferred would be Hex 87.

4.2 READING THE VDP STATUS REGISTER

The Status Register contents can be read with a single byte transfer, just by doing a read from address Hex C002 (see Table 4-2).

TABLE 4-2 — READ FROM STATUS REGISTER

OPERATION	MSB 0	1	2	3	4	5	6	LSB 7	$\overline{CSR}$	$\overline{CSW}$	MODE
Data Read (Byte 1)	D0	D1	D2	D3	D4	D5	D6	D7	0	1	1

4.3 WRITING AND READING VRAM

The VDP is connected to VRAM via a 14-bit autoincrementing Address Register. Once the address to read from or write to is set up (two-byte data transfer), we can read or write a byte of data using a one-byte transfer. Continuing to read or write to the VDP causes the address to increment automatically. Therefore, reading or writing a sequential chunk of data can be performed very quickly. The MODE signal is high (MODE1) for the first two data transfers (address setup), and low (MODE0) for the third when actually reading from or writing to VRAM.

The following sequences illustrate the proper steps for writing to and reading from VRAM. Refer to Table 4-3 and Table 4-4 for details.

Write to VRAM

1) Transfer lower eight bits of address to MODE HIGH.
2) Transfer upper eight bits of address to MODE HIGH. (The two MSBs must be set to 0 and 1, respectively.)
3) Write a byte to MODE LOW.
4) Write next byte.

TABLE 4-3 — WRITE TO VRAM

OPERATION	MSB 0	1	2	3	4	5	6	LSB 7	$\overline{CSR}$	$\overline{CSW}$	MODE
Setup Address (Byte 1)	A6	A7	A8	A9	A10	A11	A12	A13	1	0	1
Setup Address (Byte 2)	0	1	A0	A1	A2	A3	A4	A5	1	0	1
Write Data (Byte 3)	D0	D1	D2	D3	D4	D5	D6	D7	1	0	0

Read from VRAM

1) Transfer lower eight bits of address to MODE HIGH.
2) Transfer upper eight bits of address to MODE HIGH. (The two MSBs must be set to 0.)
3) Read a byte from MODE LOW.
4) Read next byte.

TABLE 4-4 — READ FROM VRAM

OPERATION	MSB 0	1	2	3	4	5	6	LSB 7	$\overline{CSR}$	$\overline{CSW}$	MODE
Setup Address (Byte 1)	A6	A7	A8	A9	A10	A11	A12	A13	1	0	1
Setup Address (Byte 2)	0	0	A0	A1	A2	A3	A4	A5	1	0	1
Read Data (Byte 3)	D0	D1	D2	D3	D4	D5	D6	D7	0	1	0

EXAMPLE 4-2.

Write To VRAM

Suppose we wish to write Hex 00 to VRAM location Hex 20A0. The first byte transferred to address Hex C002 would be the lower address byte or Hex A0. The second byte transferred to address Hex C002 is the upper eight address bits with the two MSBs set to 0 and 1, respectively. Therefore, Hex 60 would be sent as the second byte instead of Hex 20. Now that the address is set up, a byte of data can be written to Hex 20A0 by a doing a write to address Hex C000.

EXAMPLE 4-3.

Read From VRAM

Suppose we wish to read the byte of VRAM located at Hex 20A0. The first byte transferred to address Hex C002 would be the lower address byte or Hex A0. The second byte transferred to address Hex C002 is the upper eight address bits or Hex 20. The address is now set up, and location Hex 20A0 can be read by doing a read from address Hex C000.

5. DESCRIPTION OF THE VDP REGISTERS

5.1 VDP WRITE-ONLY REGISTERS

The eight VDP registers are described in the following paragraphs. Registers 0 and 1 contain bits to enable or disable various features and modes. Registers 2 through 6 contain values that specify the starting locations of various tables in VRAM. These VRAM tables are used to generate displays on the Pattern Plane and Sprite Planes. Register 7 contains the color of text (if in Text Mode) and contains the Backdrop color for all modes.

In some of the registers not all eight bits are used. To insure software compatibility with the next generation Advanced Video Display Processor, the unused bits must be set to 0s.

NOTE

Bit 0 is the MSB, and Bit 7 is the LSB.

5.1.1 Register 0 (Contains Two VDP Control Bits)

REGISTER 0

MSB D0							LSB D7
0	0	0	0	0	0	M3	EXT. VID.

Bit 6 = M3 (Pattern Mode Bit 3)

This is one of three bits that, when set, determine the display mode the VDP is in. The other two mode bits are located in Register 1.

M1	M2	M3	Display Mode
0	0	0	Graphics I Mode
0	0	1	Graphics II Mode
0	1	0	Multicolor Mode
1	0	0	Text Mode

Bit 7 = External VDP Plane Enable/Disable
0-Disables External VDP Plane
1-Enables External VDP Plane

5.1.2 Register 1 (Contains Eight VDP Control Bits)

REGISTER 1

LSB D0							MSB D7
4/16 K	BLK. SCRN	IE	M1	M2	0	SPR. SIZE	SPR. MAG.

Bit 0 = 4/16K Selection

0-Selects 4K bytes of VRAM
1-Selects 16K bytes of VRAM.

NOTE

This bit is used only on the TMS9918A/28A/29A. When using TMS9118/28/29 this bit is a "Don't Care" and can be set to either state. The TMS9118/28/29 Family assumes 16K of VRAM is present.

Bit 1 = Display Blank Enable/Disable

0-Causes the active display area to blank
1-Enables the active display

Blanking causes the Sprite and Pattern Planes to blank but still allows the Backdrop color to show through. Blanking the display does not destroy any tables in VRAM.

Bit 2 = IE (Interrupt Enable)

0-Disables VDP interrupt
1-Enables VDP interrupt

If the VDP interrupt is connected in hardware and enabled by this bit, it will occur at the end of the active screen display area, just before vertical retrace starts. Exceptionally smooth, clean pattern drawing and sprite movement can be achieved by writing to the VDP during the period this interrupt is active.

Bit 3,4 = M1,M2 (Pattern Mode Bits 1 and 2)

Refer to Bit 6 of Register 0 for a description of these bits.

Bit 5 = Reserved Bit (must be set to 0)

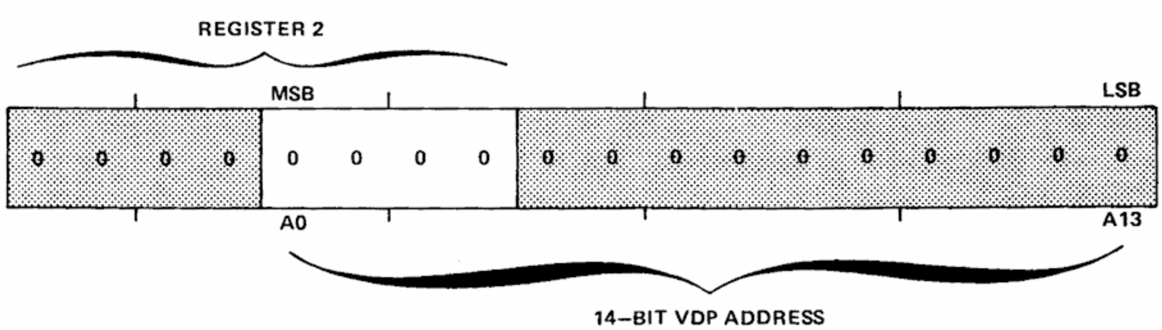
Bit 6 = Sprite Size Select

0-Selects Size 0 sprites (8x8 pixels)
1-Selects Size 1 sprites (16x16 pixels)

Bit 7 = Sprite Magnify Option

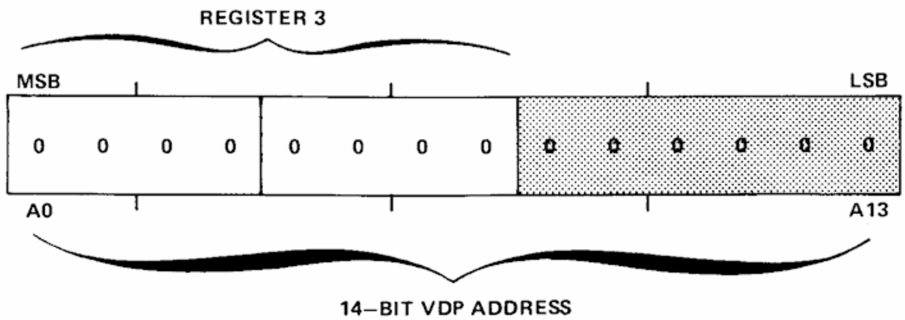
0-Selects no magnification
1-Selects a magnification of 1, thus 8x8 sprites become 16x16 and 16x16 sprites become 32x32.

5.1.3 Register 2



Register 2 tells the VDP where the starting address of the Name Table is located in VRAM. The range of its contents is from 0-F. The contents of the register form the upper four bits of the 14-bit VDP address, therefore making the location of the Name Table in VRAM equal to (Register 2) * 400 (Hex).

5.1.4 Register 3

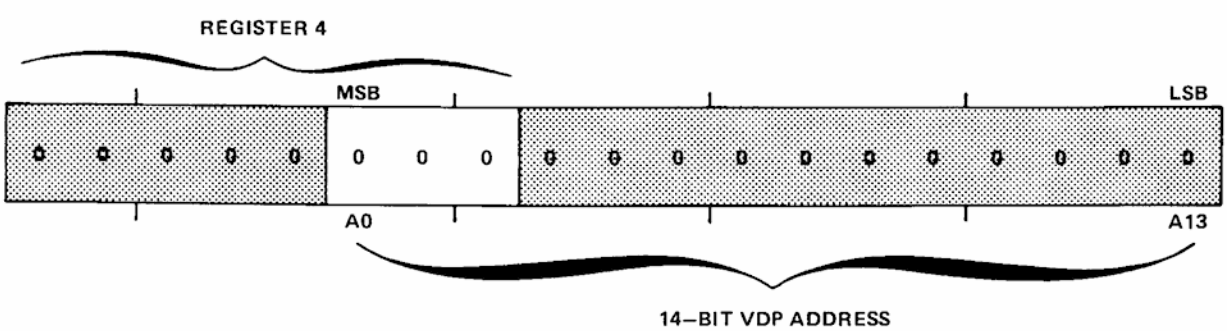


Register 3 tells the VDP where the starting address of the Color Table is located in VRAM. The range of its contents is from 0-FF. The contents of the register form the upper eight bits of the 14-bit VDP address, therefore making the location of the Color Table in VRAM equal to (Register 3) * 40 (Hex).

NOTE

Register 3 functions differently when the VDP is in Graphics II Mode. In this mode the Color Table can only be located in one of two places in VRAM, either Hex 0000 or Hex 2000. If Hex 0000 is where you wish the Color Table to be located, then the MSB in Register 3 has to be a 0. If Hex 2000 is the location choice for your Color Table, then the MSB in Register 3 must be a 1. In either case, all the LSBs in Register 3 must be set to 1s. Therefore, in Graphics II Mode the only two values that work correctly in Register 3 are Hex 7F and Hex FF.

5.1.5 Register 4

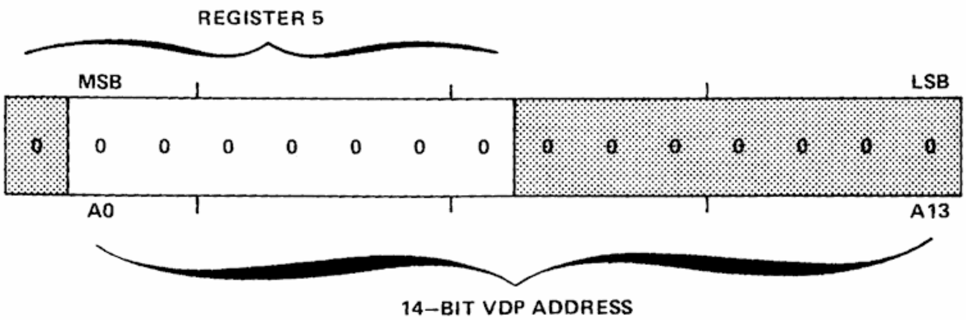


Register 4 tells the VDP where the starting address of the Pattern Table is located in VRAM. The range of its contents is from 0-7. The contents of the register form the upper three bits of the 14 bit VDP address, therefore making the location of the Pattern Table in VRAM equal to (Register 4) * 800 (Hex).

NOTE

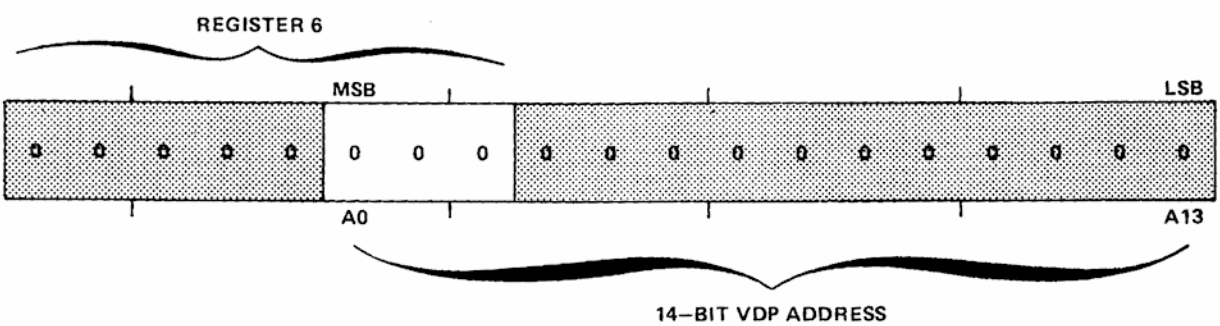
Register 4 functions differently when the VDP is in Graphics II Mode. In this mode the Pattern Table can only be located in one of two places in VRAM, either Hex 0000 or Hex 2000. If Hex 0000 is where you wish the Pattern Table to be located, then the MSB in Register 4 has to be a 0. If Hex 2000 is the location choice for your Pattern Generator Table, then the MSB in Register 4 must be a 1. In either case, all the LSBs in Register 4 must be set to 1s. Therefore, in Graphics II Mode the only two values that work correctly in Register 4 are Hex 03 and Hex 07.

5.1.6 Register 5



Register 5 tells the VDP where the starting address of the Sprite Attribute Table is located in VRAM. The range of its contents is from 0-7F. The contents of the register form the upper seven bits of the 14 bit VDP address, therefore making the location of the Sprite Attribute Table in VRAM equal to (Register 5) * 80 (Hex).

5.1.7 Register 6



Register 6 tells the VDP where the starting address of the Sprite Pattern Table is located in VRAM. The range of its contents is from 0-7. The contents of the register form the upper three bits of the 14 bit VDP address, therefore making the location of the Sprite Pattern Table in VRAM equal to (Register 6) * 800 (Hex).

5.1.8 Register 7

REGISTER 7

MSB D0							LSB D7
CLR 1	CLR 1	CLR 1	CLR 1	CLR 0	CLR 0	CLR 0	CLR 0

The upper four bits of Register 7 contain the color of bits on in Text Mode. The lower four bits contain the color of bits off in Text Mode and the Backdrop color in all modes.

5.2 READ-ONLY STATUS REGISTER

STATUS REGISTER

MSB D0							LSB D7
FRAME FLAG	5TH SPR.	C	FIFTH SPRITE NUMBER				

The VDP Status Register contains the Interrupt Flag, Coincidence Flag, Fifth Sprite Flag, and the Fifth Sprite Number (if one exists). Each of these is explained in the following paragraphs.

5.2.1 Interrupt Flag (F)

The F flag in the Status Register is set equal to 1 at the end of the raster scan of the last line of the active display, just before the Backdrop color at the bottom of the screen begins. It is reset to a 0 after the Status Register is read or whenever the VDP is externally reset (hardware reset). If the Interrupt Enable bit located in VDP Register 1 is active (1), then the VDP interrupt output line (INT) will be active (0) whenever the F status flag is 1.

NOTE

The Status Register needs to be read frame-by-frame in order to clear the interrupt and receive the new interrupt for the next frame.

5.2.2 Coincidence Flag (C)

The C status flag will be set to a 1 if two or more sprites coincide. Coincidence occurs if any two sprites on the screen have at least one overlapping pixel. Sprites set to the VDP color transparent, as well as those partially or completely off the screen, are considered. Sprites beyond the Attribute Table terminator of Hex D0 are not considered. The C flag is cleared whenever the VDP Status Register is read or the VDP is externally reset.

5.2.3 Fifth Sprite Flag (5S) and Number

The Fifth Sprite Flag is set to a 1 whenever there are five or more sprites active on a horizontal line. The Fifth Sprite Flag is cleared to a 0 after the Status Register is read or whenever the VDP is externally reset. The number of the lowest priority sprite on the horizontal line is loaded into the lower five bits of the Status Register whenever the Fifth Sprite Flag is set and is valid whenever the Fifth Sprite Flag is a 1. The setting of the Fifth Sprite Flag will not generate an interrupt.

6. INITIALIZING THE VDP

After powerup of our VDP system, the first thing to be done is register initialization. In order to do this we need to know a few things, such as which pattern display mode to use and where in VRAM we are going to place the tables required. Figure 6-1 shows a procedure for initializing all eight VDP registers. The next section is a brief description of the popular uses for each mode.

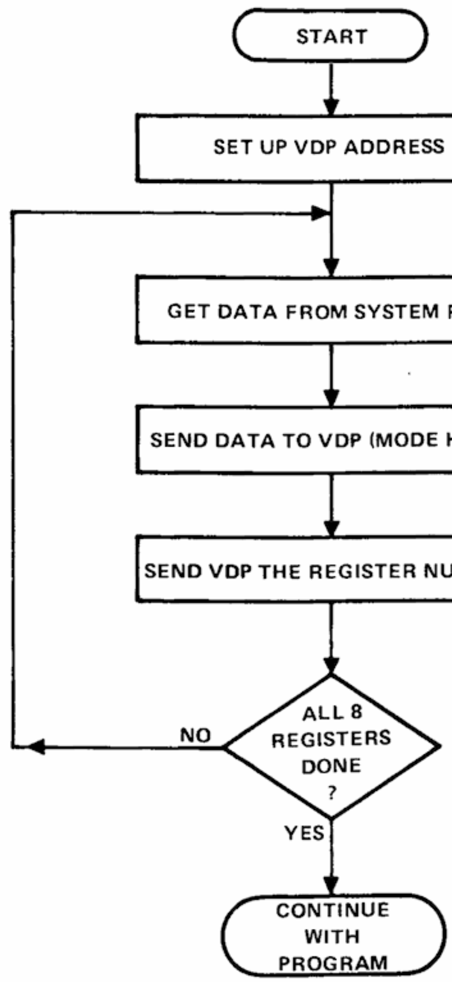


FIGURE 6-1 — REGISTER INITIALIZATION

6.1 CHOOSING THE RIGHT MODE

Most applications displaying text use either Text or Graphics I Mode. Video games needing a high-resolution display normally use Graphics I or Graphics II Mode. Graphics I Mode is a bit more popular for games because a colorful, detailed, high-resolution picture can be generated using very little data. Graphics II Mode is used when a high-resolution picture needs extremely fine detail and color or when you wish to organize memory in a bit-map arrangement for calculating pixels, lines, circles, etc. The Graphics II Mode bit-map arrangement is also very popular for personal computer business graphics. Multicolor Mode is popular for games requiring only a low-resolution display. Sprites are available in all modes except Text and are primarily used for objects that move and change shape (animation).

Detailed descriptions of Graphics I, Graphics II, Text, and Multicolor Mode appear in Section 8. Refer to these sections to decide which display mode is best suited to your particular application.

Some typical table values used to initialize the registers in each graphic mode are shown and described in the following figures. The resulting VRAM Memory Map is shown after the table values. Actual assembly language programs written for various CPUs and using the following register initialization values are included in Appendix E.

The typical register initialization values are given here only as one example. Those of you preferring a different VRAM memory map can either calculate the values as described in Section 5 or refer to the Register Address Look-Up Tables provided in Appendix A.

6.1.1 Graphics I Mode Initialization

TABLE 6-1 — GRAPHIC I MODE INITIALIZATION

REGISTER	MSB LSB	HEX	DESCRIPTION
REG 0	00000000	00	Graphics I Mode,No External Video
REG 1	11000000	C0	16K,Enable Display,Disable Int.,8x8 Sprites,Mag.Off
REG 2	00000101	05	Address of Name Table in VRAM = Hex 1400
REG 3	10000000	80	Address of Color Table in VRAM = Hex 2000
REG 4	00000001	01	Address of Pattern Table in VRAM = Hex 0800
REG 5	00100000	20	Address of Sprite Attribute Table in VRAM = Hex 1000
REG 6	00000000	00	Address of Sprite Pattern Table in VRAM = Hex 0000
REG 7	00000001	01	Backdrop Color = Black

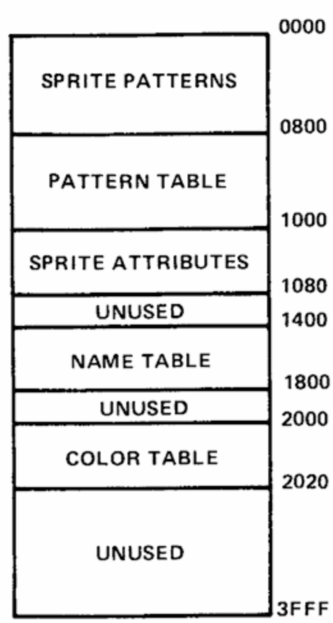


FIGURE 6-2 — GRAPHIC I MODE VRAM MEMORY MAP

6.1.2 Graphics II Mode Initialization

TABLE 6-2 — GRAPHIC II MODE INITIALIZATION

REGISTER	MSB LSB	HEX	DESCRIPTION
REG 0	00000010	02	Graphics II Mode,No External Video
REG 1	11000010	C2	16K,Enable Disp.,Disable Int.,16x16 Sprites, Mag.Off
REG 2	00001110	0E	Address of Name Table in VRAM = Hex 3800
REG 3	11111111	FF	Address of Color Table in VRAM = Hex 2000
REG 4	00000011	03	Address of Pattern Table in VRAM = Hex 0000
REG 5	01110110	76	Address of Sprite Attribute Table in VRAM = Hex 3B00
REG 6	00000011	03	Address of Sprite Pattern Table in VRAM = Hex 1800
REG 7	00001111	0F	Backdrop Color = White

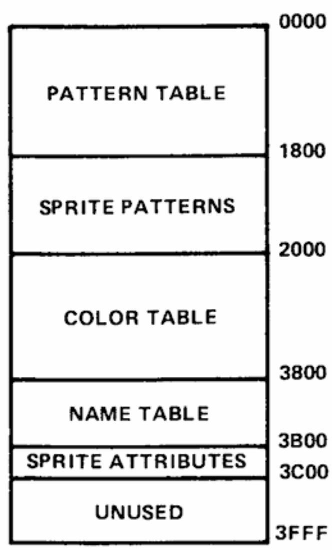


FIGURE 6-3 — GRAPHIC II MODE VRAM MEMORY MAP

6.1.3 Multicolor Mode Initialization

TABLE 6-3 — MULTICOLOR MODE INITIALIZATION

REGISTER	MSB LSB	HEX	DESCRIPTION
REG 0	00000000	00	Multicolor Mode, No External Video
REG 1	11001011	CB	16K, Enable Display, Disable Int., 16x16 Sprites, Mag. On
REG 2	00000101	05	Address of Name Table in VRAM = Hex 1400
REG 3	XXXXXXXX	XX	Color Table not used. All bits are don't cares.
REG 4	00000001	01	Address of Pattern Table in VRAM = Hex 0800
REG 5	00100000	20	Address of Sprite Attribute Table in VRAM = Hex 1000
REG 6	00000000	00	Address of Sprite Pattern Table in VRAM = Hex 0000
REG 7	00000001	04	Backdrop Color = Dark Blue

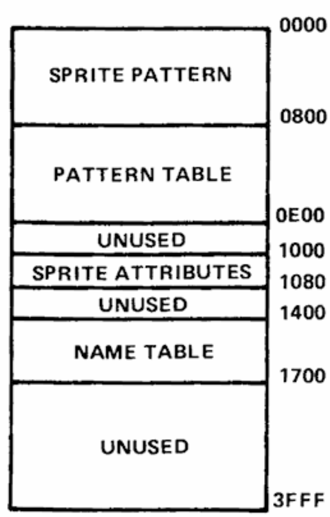


FIGURE 6-4 — MULTICOLOR MODE VRAM MEMORY MAP

6.1.4 Text Mode Initialization

TABLE 6-4 — TEXT MODE INITIALIZATION

REGISTER	MSB LSB	HEX	DESCRIPTION
REG 0	00000000	00	Text Mode,No External Video
REG 1	11010000	D0	16K,Enable Disp.,Disable Int.
REG 2	00000010	02	Address of Name Table in VRAM = Hex 0800
REG 3	XXXXXXXX	XX	Color Table not used. Color is defined in Reg.7
REG 4	00000000	00	Address of Pattern Table in VRAM = Hex 0000
REG 5	XXXXXXXX	20	
REG 6	XXXXXXXX	00	
REG 7	11110101	F5	White Text on Light Blue Background

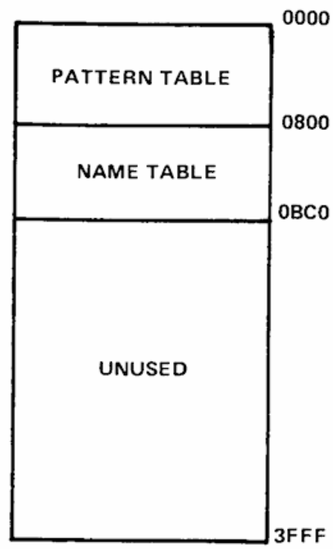


FIGURE 6-5 — TEXT MODE VRAM MEMORY MAP

7. CREATING PATTERNS

7.1 ALL PATTERNS ARE CREATED EQUAL

If you can create 8x8 pixel patterns you can create fonts for Graphics I Mode, Graphics II Mode, Text Mode, and sprites. In the following pages we will define some patterns and show how they should be entered into VRAM in order to produce a display.

1) Figure 7-1 is a sample grid which will be used to create 8x8 pixel patterns. Each small square within the grid represents one pixel on the screen.

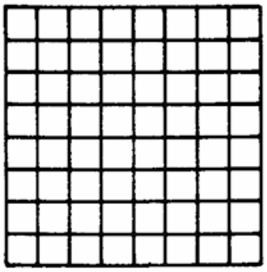


FIGURE 7-1 — 8X8 PIXEL PATTERN GRID

2) Fill in the squares within the grid to create your text, graphic, or sprite pattern. Examples of the letter "A", an arrow, and a star are shown in Figure 7-2.

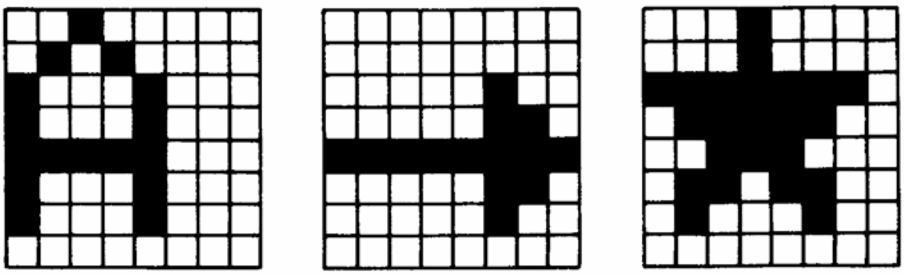


FIGURE 7-2 — EXAMPLE 8X8 PIXEL PATTERNS

NOTE

If you are defining patterns to be used in Text Mode (40 patterns per line), the patterns should be left justified within a 6x8 pixel block like the letter "A" shown in Figure 7-2. Refer to Section 8.5 for a further description.

3) Now comes the task of converting the pattern to numbers. First assign 1s to the filled in squares and 0s to the blanks. Then convert the 1s and 0s to their hexadecimal equivalents as shown in Figure 7-3.

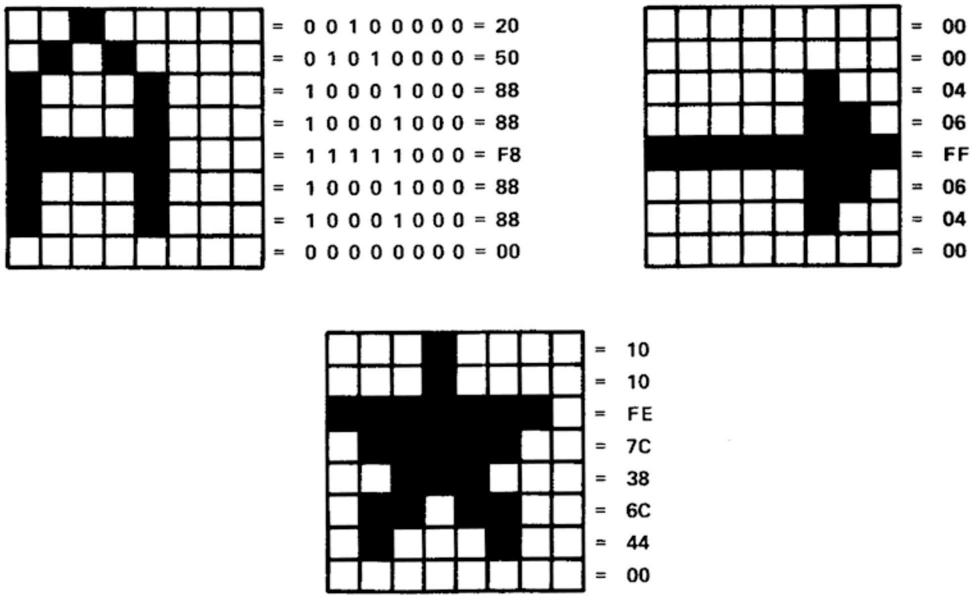


FIGURE 7-3 — HEXIDECIMAL CONVERSION

4) Now place the 8 bytes that define the pattern into the Pattern Table. Assume that the location of the Pattern Table in VRAM is defined to be Hex 800, and the arrow is to be named pattern number 00. Next place the eight bytes into the table as shown in Figure 7-4.

800	00	
801	00	
802	04	
803	06	PATTERN
804	FF	NAME 00
805	06	
806	04	
807	00	
808		
809		
80A		
80B		PATTERN
80C		NAME 01
80D		
80E		
80F		
810		
900	00	
901	00	
902	00	
903	00	PATTERN
904	00	NAME 20
905	00	
906	00	
907	00	
908		
A08	20	
A09	50	
A0A	88	
A0B	88	PATTERN
A0C	F8	NAME 41
A0D	88	
A0E	88	
A0F	00	

FIGURE 7-4 — PATTERN TABLE

7.1.1 Defining Patterns for Text

When using text in your application, it is often convenient to place the eight bytes defining your text character in its actual ASCII number location. This will simplify writing text to the screen. Writing the ASCII value directly to the Name Table causes the appropriate character to appear on the screen. As shown in Example 7-1, a space character is contained in Pattern Table position Hex 20, and the letter "A" is contained in Pattern Table position Hex 41.

EXAMPLE 7-1.

ASCII	Space	= Hex 20
	?	= Hex 3F
	A	= Hex 41
	B	= Hex 42
	C	= Hex 43
	Etc.	

NOTE

When defining patterns for Text Mode, the pattern must be defined within a 6x8 pixel grid as shown in Figure 7-5. The two LSBs are unused and therefore not displayed by the VDP.

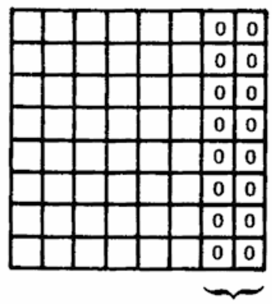


FIGURE 7-5 — 6X8 PIXEL PATTERN GRID FOR TEXT MODE

7.1.2 Defining Patterns for Sprites

1) To use Size 0 sprites (8x8 pixels), the patterns are defined exactly like the arrow and the star shape done earlier with one change. Instead of entering the code in the Pattern Table, it is now entered into the Sprite Pattern Table. Figure 7-6 shows a sprite grid and the Sprite Generator Table for an 8x8 pixel sprite pattern.

	8 X 8	
000	10	SPRITE NAME 00
001	10	
002	FE	
003	7C	
004	38	
005	6C	
006	44	
007	00	
008		SPRITE NAME 01
009		
00A		
00B		
00C		
00D		
00E		
00F		
010		

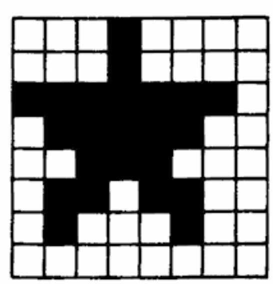


FIGURE 7-6 — 8X8 SPRITE GRID AND SPRITE TABLE

2) If you are going to use Size 1 sprites (16x16 pixels), then the patterns are still defined as 8x8 pixel patterns. It takes four 8x8 pixel patterns to form a 16x16 pixel grid as shown in Figure 7-7.

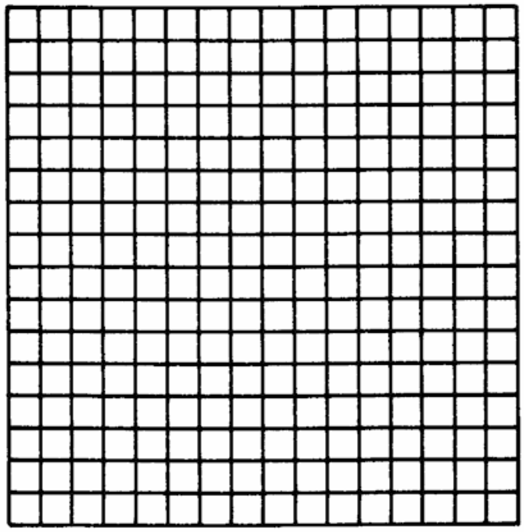


FIGURE 7-7 — 16X16 SPRITE GRID

3) Fill in the squares to create your Size 1 sprite pattern. An example is shown in Figure 7-8.

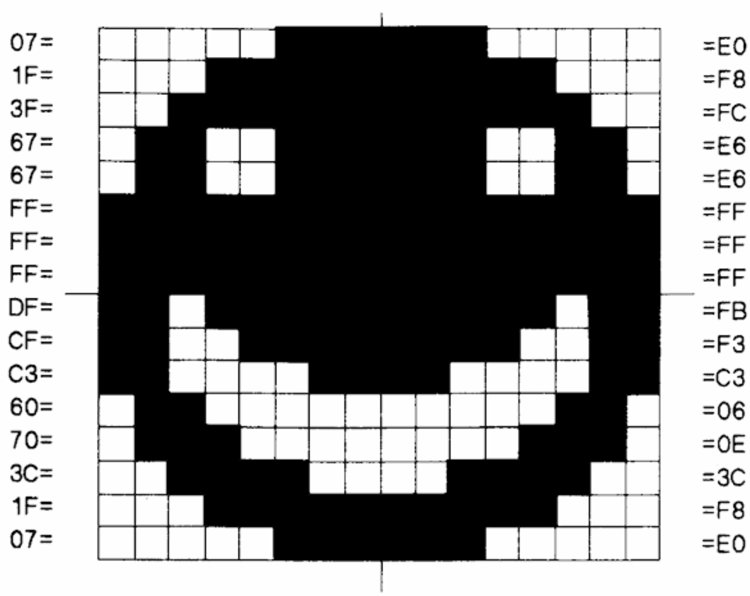


FIGURE 7-8 — SIZE 1 SPRITE PATTERN

4) Next encode the sprite pattern. This is done by splitting the sprite into four sections as shown in Figure 7-9. The four 8x8 pixel patterns should be encoded in the following order.

Pattern 1 = upper left

Pattern 2 = lower left

Pattern 3 = Upper right

Pattern 4 = Lower right

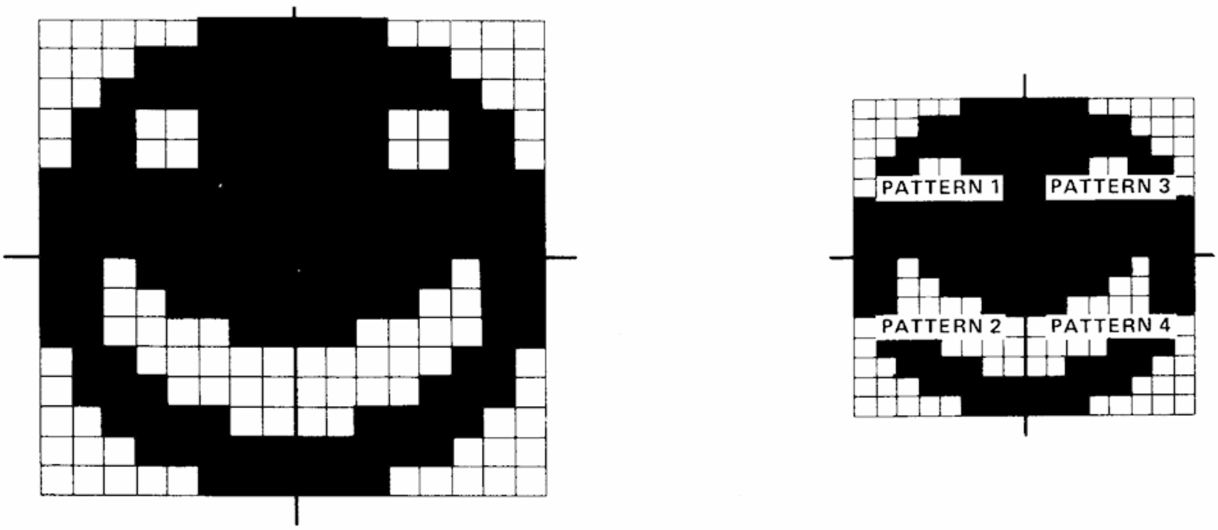


FIGURE 7-9 — SIZE 1 SPRITE ORGANIZATION

5) Place the 32 bytes of information in the Sprite Pattern Table, assuming that the table in VRAM is located at Hex 0000. Figure 7-10 shows how the Sprite Generator Table looks for our 16x16 pixel sprite.

	16 X 16		
000	07		
001	1F		
002	3F		
003	67		
004	67		
005	FF	UPPER	
006	FF	LEFT	
007	FF	CORNER	
008	DF		
009	CF		
00A	C3		
00B	60		
00C	70		
00D	3C	LOWER	
00E	1F	LEFT	SPRITE
00F	07	CORNER	NAME 00
010	E0		
011	F8		
012	FC		
013	E6		
014	E6		
015	FF	UPPER	
016	FF	RIGHT	
017	FF	CORNER	
018	FB		
019	F3		
01A	C3		
01B	06		
01C	0E		
01D	3C	LOWER	
01E	F8	RIGHT	
01F	E0	CORNER	

SPRITE
NAME 04

FIGURE 7-10 — SPRITE PATTERN TABLE

8. THE DIFFERENT DISPLAY MODES

8.1 GRAPHICS I MODE

The VDP is in Graphics I Mode when all three mode bits (M1,M2,M3) located in VDP Registers 0 and 1 are set to zero. When in this mode, the Pattern Plane has a resolution of 256 horizontal pixels by 192 vertical pixels. The screen is broken up into blocks each containing an 8x8 pixel pattern. There are 32 of these blocks horizontally and 24 of them vertically. Figure 8-1 shows the position of these 768 blocks on the screen.

ROW 0	000	001	002	003	• • •	028	029	030	031	(HEX 01F)
ROW 1	032	033	034	035	• • •	060	061	062	063	(HEX 03F)
ROW 2	064	065	066	067	• • •	092	093	094	095	(HEX 05F)
ROW 3	096	097	098	099	• • •	124	125	126	127	(HEX 07F)
•	•	•						•	•	
•	•	•			ACTIVE DISPLAY AREA			•	•	
•	•	•						•	•	
ROW 20	640	641	642	643	• • •	668	669	670	671	(HEX 29F)
ROW 21	672	673	674	675	• • •	700	701	702	703	(HEX 2BF)
ROW 22	704	705	706	707	• • •	732	733	734	735	(HEX 2DF)
ROW 23	736	737	738	739	• • •	764	765	766	767	(HEX 2FF)

FIGURE 8-1 — GRAPHICS I MODE NAME TABLE MAPPING

Three tables are required in VRAM in order to create a Graphics I Mode picture, these are the Name Table, Pattern Table, and the Color Table. If every possible bit of color and pattern detail is defined, a Graphics I Mode picture would take up 2848 (Hex B20) bytes.

8.1.1 The Pattern Table

The Pattern Table contains a library of user defined patterns that can be displayed in any of the 768 screen positions. It is 2048 bytes long and is arranged as 256 eight byte patterns. Each one of these eight byte patterns defines an 8x8 pixel area. All of the 1s within a pattern designate one color (let's call this color 1), while all of the 0s designate another color (color 0).

A unique feature of Graphics I Mode, as opposed to bit-mapped graphics, is the fact that once an 8x8 pixel pattern has been defined and stored in the Pattern Table, it can be used multiple times on the screen without being redefined.

EXAMPLE 8-1.

If only the first eight byte pattern in the Pattern Table was defined (Pattern 0), you could place this pattern in every single one of the 768 screen positions by writing Hex 00 to every byte of the 768 byte Name Table.

8.1.2 The Name Table

As illustrated in Figure 8-1, there are 768 screen locations. Each of these locations is represented by one byte of memory located in the Name Table. The first byte of the Name Table specifies which pattern will be located in the upper left hand corner of the screen. The last byte in the Name Table specifies the pattern for the lower right hand screen corner. Each byte entry in the Name Table can designate one of 256 (Hex FF) patterns. The location of the 768 byte Name Table in VRAM is defined by the base address located in VDP Register 2.

8.1.3 The Color Table

The Color Table for a Graphics I Mode picture is 32 bytes long. Its location in VRAM is determined by the eight-bit Color Table base address in VDP Register 3.

The color of the 1s and 0s within a pattern is defined by the Color Table. Each byte entry in the Color Table defines two colors. The upper nibble (four bits) defines the color of the 1s, and the lower nibble defines the color of the 0s. Since we can create 256 unique 8x8 pixel patterns but can only have 32 Color Table entries, each entry in the Color Table must define the color for more than one pattern. In fact, the first byte in the Color Table defines the color for the first eight patterns. Likewise, the second byte in the Color Table defines the color for the next eight patterns defined.

Table 8-1 illustrates the Graphics I Mode Color Table.

Figure 8-2 illustrates how the Pattern Table, Name Table, and Color Table are mapped to the screen.

TABLE 8-1 — GRAPHICS I MODE COLOR TABLE

BYTE NO.	PATTERN NO.	BYTE NO.	PATTERN NO.
0	0..7	16	128..135
1	8..15	17	136..143
2	16..23	18	144..151
3	24..31	19	152..159
4	32..39	20	160..167
5	40..47	21	168..175
6	48..55	22	176..183
7	56..63	23	184..191
8	64..71	24	192..199
9	72..79	25	200..207
10	80..87	26	208..215
11	88..95	27	216..223
12	96..103	28	224..231
13	104..111	29	232..239
14	112..119	30	240..247
15	120..127	31	248..255

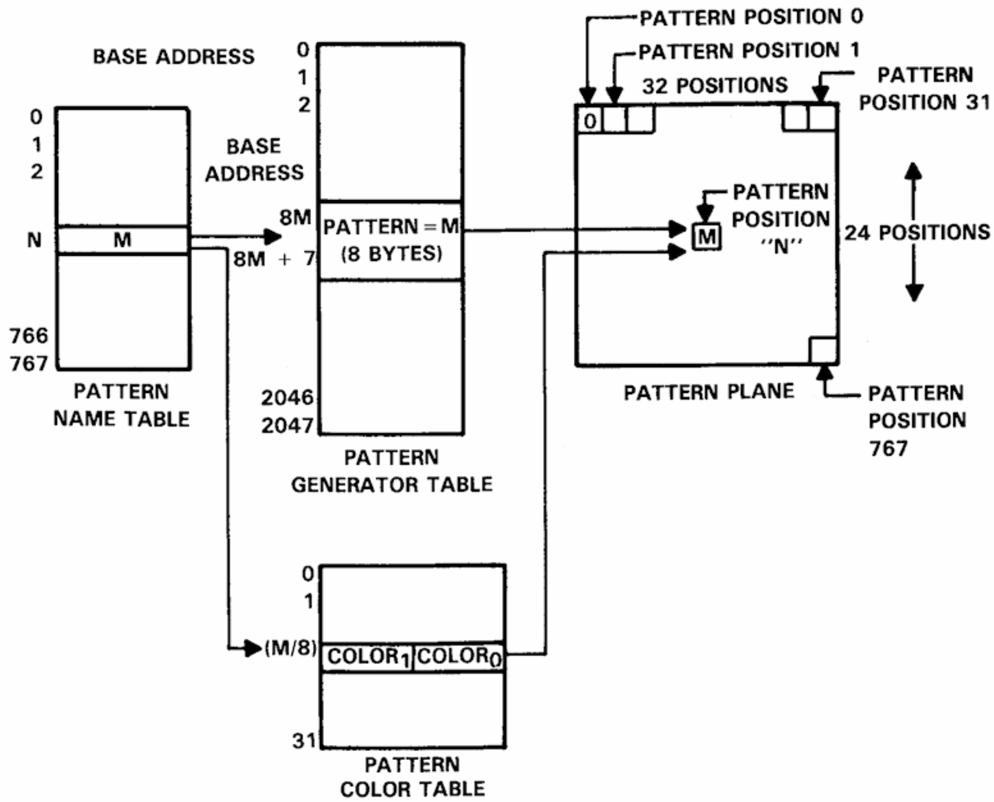


FIGURE 8-2 — GRAPHICS I MODE MAPPING

8.2 GRAPHICS II MODE

Graphics II Mode is similar to Graphics I Mode in the way the screen is organized. The resolution is still 256 horizontal pixels by 192 vertical pixels. Three tables are still required in VRAM in order to generate a display, these being the Name Table, Color Table, and Pattern Table. The Name Table is still 768 bytes long, but the length of the Color and Pattern Tables has been extended. Instead of having to choose from a library of 256 8x8 pixel patterns for display in the 768 screen locations (which means patterns have to be reused) you can define 768 8x8 pixel patterns in Graphics II Mode. This allows a unique pattern to be created for every possible screen location. Instead of one byte of color information for every eight patterns, there are now eight bytes of color information per pattern, thereby making the Pattern Table and the Color Table in Graphics II Mode the same length.

Since there are eight bytes of color information per pattern, two unique colors can be specified for each line of an 8x8 pixel pattern. This allows up to 16 colors within a pattern.

8.3 THE PATTERN TABLE

The Pattern Table is 6144 (Hex 1800) bytes long, assuming all patterns are defined, and is best thought of as three equal blocks of 2048 bytes of pattern information. Each of the three 2048 byte blocks is divided into 256 8x8 pixel pattern definitions. The first 256 patterns can only be displayed on the upper third of the screen. The second 256 patterns can only be displayed on the middle section of the screen, and the last 256 patterns can only be displayed on the lower third of the screen.

8.3.1 The Color Table

The Color Table is 6144 (Hex 1800) bytes long, assuming all colors are defined, and is segmented into three 2048 byte blocks exactly like the Pattern Table. Each 2048 byte block is divided into 256 color definitions, each being eight bytes long. The first 256 color definitions correspond directly to the first 256 patterns defined. Likewise, the second 256 color definitions correspond to the second 256 patterns, and the third 256 color definitions correspond to the last 256 patterns defined.

It takes eight bytes to define a pattern shape and eight bytes to define what color that pattern will be. Each byte in a color definition defines the color of the bits that are on or off for the corresponding line of the pattern. The upper four bits define the color of the bits on, the lower four bits define the color of the bits off in a line of the pattern. An example of how color is mapped to a pattern is shown in Figure 8-3.

ROW								
ROW 0	0	1	0	0	0	0	0	1
1	0	0	1	0	0	0	1	0
2	0	0	0	1	0	1	0	0
3	0	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0	0
5	0	0	0	0	1	0	0	0
6	0	0	0	0	1	0	0	0
7	0	0	0	0	1	0	0	0

PATTERN GENERATOR TABLE ENTRY

B	1	B	B	B	B	B	1
B	B	7	B	B	B	7	B
B	B	B	C	B	C	B	B
B	B	B	B	E	B	B	B
B	B	B	B	8	B	B	B
B	B	B	B	5	B	B	B
B	B	B	B	6	B	B	B
B	B	B	B	D	B	B	B

PATTERN

0 3	4 7	
1 (BLACK)	B (LT. YELLOW)	0 ROW
7 (CYAN)	B (LT. YELLOW)	1
C (GREEN)	B (LT. YELLOW)	2
E (GRAY)	B (LT. YELLOW)	3
8 (MED. RED)	B (LT. YELLOW)	4
5 (LT. BLUE)	B (LT. YELLOW)	5
6 (DK. RED)	B (LT. YELLOW)	6
D (MAGENTA)	B (LT. YELLOW)	7

PATTERN COLOR TABLE ENTRY

FIGURE 8-3 — PATTERN/COLOR DISPLAY MAPPING

8.4 THE NAME TABLE

As in Graphics I Mode, the Name Table of Graphics II Mode contains 768 entries which correspond to each of the 768 pattern positions on the display screen. Because each Name Table entry is only one byte long, it can only specify one of 256 patterns (Hex FF). In order to be able to specify a unique pattern for each of the 768 pattern positions, the screen is broken up into three sections as shown in Figure 8-4. Each of the screen sections is 256 bytes long, and since a byte can specify 256 different values, a unique pattern can be specified for each screen location. An example of Graphics II Mode mapping is shown in Figure 8-5.

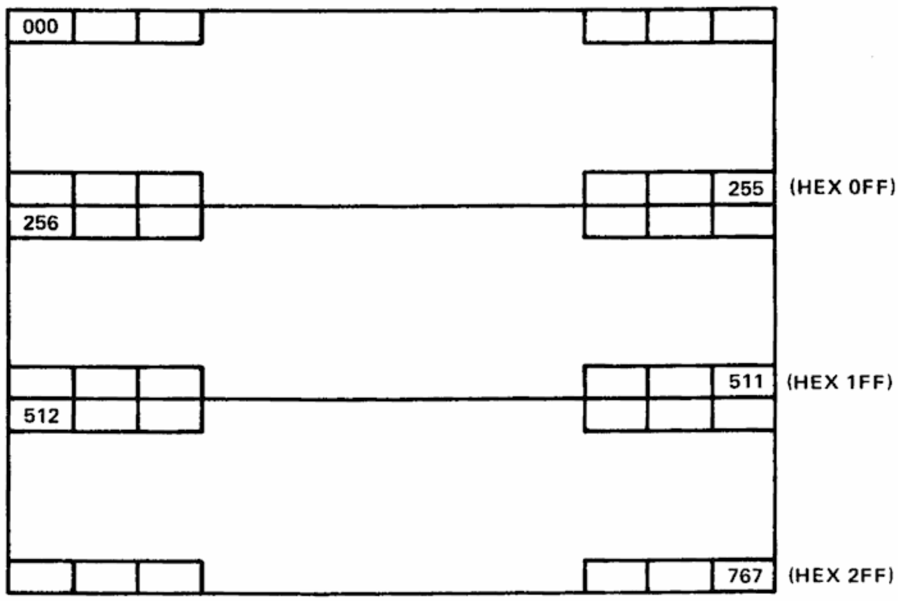


FIGURE 8-4 — GRAPHICS II MODE NAME TABLE SEGMENTED INTO THREE EQUAL BLOCKS

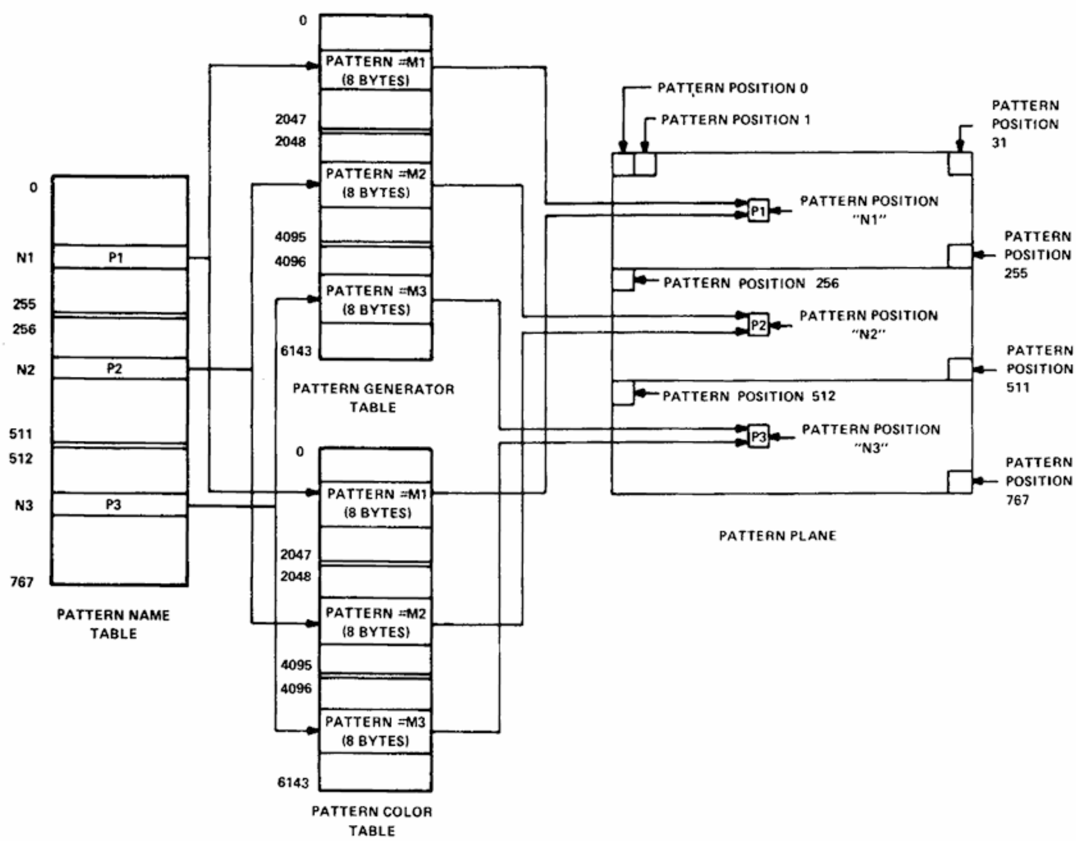


FIGURE 8-5 — GRAPHICS II MODE MAPPING

8.4.1 Graphics II Mode As a Bit-Mapped Display

A neat feature of Graphics II Mode is that it can be arranged by the programmer to act as a bit-mapped display. This is extremely useful when your application allows the use of an algorithm to calculate the position of pixels on the screen instead of hand drawing them (coding each pixel at a time). Using Graphics II Mode as a bit-map lets you address every pixel on the screen individually for plotting points, drawing lines, circles, etc. The only drawback to this arrangement is that even though the Pattern Plane is completely bit-mapped, the color assignments are not. Since a unique color cannot be specified for each pixel on the screen, one of two things can be done; use more than two colors (but be careful where you plot them on the screen) or use only two colors (pixels on could be one color, pixels off another) and not worry about where you plot.

The way to arrange Graphics II Mode as a bit-map is to write a different value to each of the 768 Name Table entries. This means that the VDP will map a unique Pattern Table entry to each screen position. By writing to a byte within an eight byte Pattern Table entry, any pixel on the screen can be turned on or off.

The simplest way to illustrate this point is to write the same value to each of the Color Table entries. A color value of Hex 4F written to all Color Table locations makes for nice blue pixels on a white background (pixels on will be blue, and pixels off will be white).

As stated earlier, to specify a bit-map a unique pattern is defined for each entry in the Name Table. An organized way to do this is to write Hex 00 to the first Name Table entry, Hex 01 to the second, Hex 02 to the third, and so forth. After reaching Hex FF the process is repeated twice more so that a dump of the 768 byte Name Table would render the values 0 - FF, 0- FF, 0-FF.

At this point we can forget about the Name Table and the Color Table and concentrate on the Pattern Table. Each bit within a Pattern Table byte entry now represents a unique pixel on the screen. Figure 8-6 illustrates how the Pattern table is currently mapped to the screen. This figure assumes that the location of the Pattern Table in VRAM starts at Hex 0000.

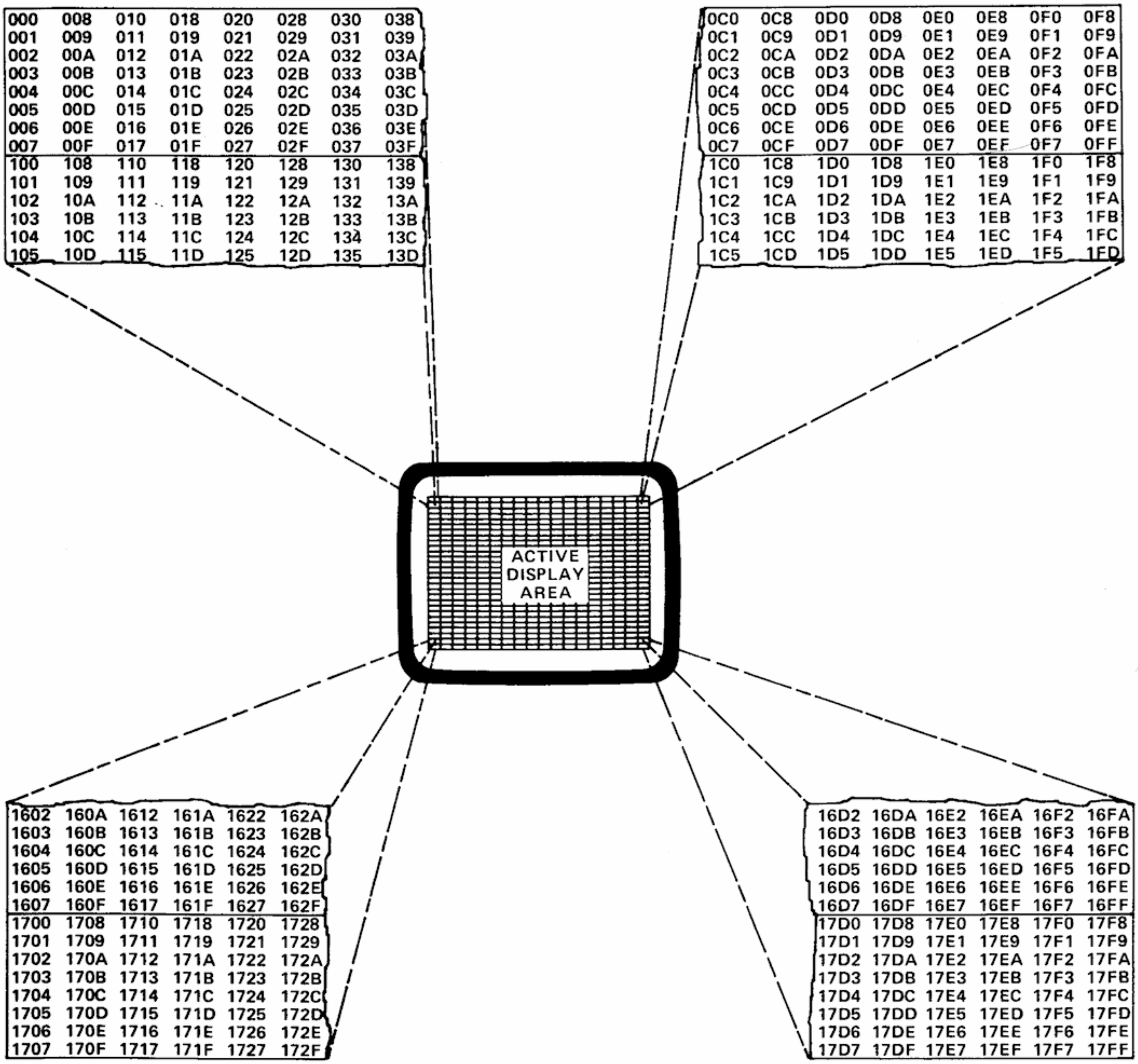


FIGURE 8-6 — GRAPHICS II PATTERN TABLE ARRANGED FOR BIT-MAPPED GRAPHICS

Looking at Figure 8-6 we can see that in order to turn on the pixel located in the upper lefthand corner of the display we would write Hex 80 to the first byte of the Pattern Table (location Hex 0000). Likewise, to turn on the pixel at the bottom righthand screen edge Hex 01 would be written to location Hex 1755.

At this point we can do ourselves a favor by writing a routine that, given any X,Y coordinates, will tell us the address of the byte we wish to write to and the data we need to write to it. The following is a step-by-step procedure of one way to calculate the address and the data.

EXAMPLE 8-2.

INPUTS: X = Hex 00-FF or Decimal 0-255
Y = Hex 00-C0 or Decimal 0-192

1) Take the integer value of (X/8) and multiply it times 8. This will give the horizontal byte offset. The actual bit we need to plot is determined by whatever remainder is left after calculating (X/8).
2) Take the integer value of (Y/8) and multiply it times Hex 100. This will give the vertical byte offset to the nearest eight bits. If there is any remainder after calculating (Y/8), add it to the vertical byte offset. This gives the vertical starting address.
3) Add the horizontal byte offset to the vertical starting address. This will give the actual address of the byte we need to write data to in order to plot our pixel.
4) Use the remainder of (X/8) to look up in a table (below) the actual data to plot. The values corresponding to different remainders are as follows:

Remainder (X/8)	Data to Write
0	Hex 80
1	Hex 40
2	Hex 20
3	Hex 10
4	Hex 08
5	Hex 04
6	Hex 02
7	Hex 01

The equation just described in the above paragraphs could be represented as follows:

BYTE ADDRESS = 8(INT(X/8)) + 256(INT(Y/8)) + R(Y/8)

WHERE R(Y/8) is equal to the remainder of (Y/8)

The actual data to write to the byte address is still obtained by taking the remainder of (X/8) and looking up the appropriate data value in the table.

8.4.2 Playing Games with VRAM Addressing

So far in Section 2.1 we have described how to use Graphics II Mode in its normal table-driven environment and how to arrange it as a bit-map. Now we are going to explain some other tricks you can play with the VDP. By experimenting with the values in VDP Registers R2 thru R6 (entering nonstandard initialization values), some interesting effects can be obtained.

You are forewarned that experimenting with VRAM addressing can cause some interesting effects but almost always produces some undesirable side effects such as losing the ability to use sprites or being only able to use a small number of sprites. Rather than dwell too long on this subject, we will describe one interesting new configuration that can be obtained and leave the rest to you.

Table 8-2 shows the register initialization values for the mode about to be described. Note that the only registers containing nonstandard values are Registers 3 and 4, which determine the Color Table and Pattern Table base address.

TABLE 8-2 — NEW MODE INITIALIZATION VALUES

REGISTER	MSB LSB	HEX	DESCRIPTION
REG 0	00000010	02	Graphics II Mode,No External Video
REG 1	11000010	C2	16K,Enable Disp.,Disable Int.,16x16 Sprites,Mag.Off
REG 2	00001110	0E	Address of Name Table in VRAM = Hex 3800
REG 3	10011111	9F	Color Table Address = Hex 2000 to Hex 2800
REG 4	00000000	00	Pattern Table Address = Hex 0000 to Hex 0800
REG 5	01110110	76	Address of Sprite Attribute Table in VRAM = Hex 3B00
REG 6	00000011	03	Address of Sprite Pattern Table in VRAM = 1800
REG 7	00001111	0F	Backdrop Color = White

What this mode does is effectively shrink the Graphics II Mode Color and Pattern Tables down from Hex 1800 bytes to Hex 800 bytes. This enables us to define up to 256 8x8 pixel patterns and 256 corresponding eight byte Color Table entries. Color is still mapped onto a pattern exactly as in Graphics II Mode.

The 768 byte Name Table is not split up into three equal sections as in Graphics II Mode but works as in Graphics I Mode. A byte of information written anywhere in the Name Table will select the appropriate pattern and the corresponding eight byte color entry and place it on the screen. In Appendix C can be found the Pattern Graphics Address Location Tables.

This mode is useful because it provides the memory savings of Graphics I Mode while allowing the color detail available in Graphics II Mode. However, a unique pattern for each screen position can no longer be defined, which is neccesary for highly detailed pictures or for bit-mapping the screen. When in this mode 32 sprites can no longer be used. If you try to put more than eight sprites on the screen at once, they will start to duplicate themselves on the screen.

8.5 TEXT MODE

The VDP is in Text Mode when mode bits M1 = 1, M2 = 0, and M3 = 0. When in this mode the screen is divided up into 40 horizontal blocks by 24 vertical blocks, each of which may contain a character shape (see Figure 8-7.). Each of these character positions is six horizontal pixels by eight vertical pixels. There are only two tables required in VRAM in order to produce a Text Mode display, these are the Name Table and the Pattern Table. No Color Table is required in VRAM because the color of the character patterns is defined by the byte of information contained in VDP Register 7. The upper four bits define the color of all the bits on, and the lower four bits define the color of all the bits off. Therefore, if you had a value of Hex F1 written to Register 7, the text color would be white (F) while the background would be black (1).

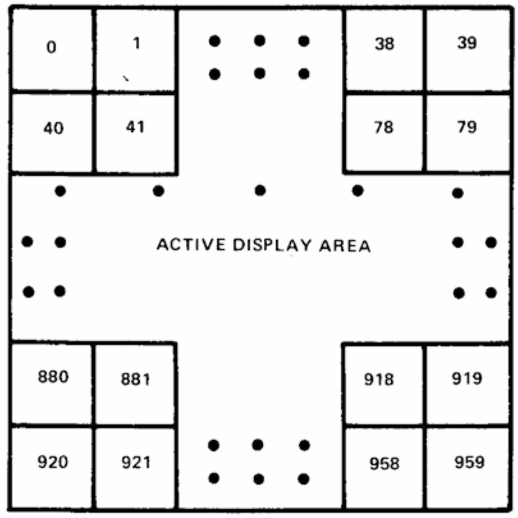


FIGURE 8-7 — TEXT MODE NAME TABLE PATTERN POSITIONS

8.5.1 The Name Table

The Name Table in Text Mode is very similar to the one in Graphics I Mode except that the screen is now 40x24 instead of 32x24 (8x8 pixel blocks). This gives 960 screen positions and 960 (40x24 = 960) entries in our Name Table. Figure 8-8 shows the Name Table positions.

Each entry in the Name Table is one byte long and therefore can specify one of 255 (Hex FF) patterns. If the first entry in the Name Table is Hex 00, then the first pattern defined (Pattern 00) would be displayed in the upper left hand corner of the screen. If the first Name Table entry contains Hex FF, then the last pattern defined (Pattern FF) would be displayed in the upper left hand corner.

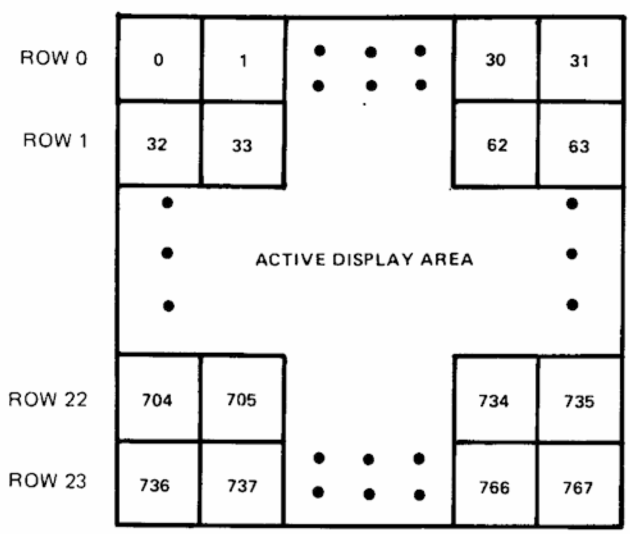


FIGURE 8-8 — PATTERN GRAPHICS NAME TABLE MAPPING

8.5.2 The Pattern Table

The Pattern Table is 2048 (Hex 800) bytes long and is composed of 256 eight-byte patterns, each of which may represent a text or graphics character. Since each screen position is only six pixels across by eight pixels down instead of eight pixels across and eight pixels down as in the graphics modes, the VDP ignores the two least significant bits of each pattern. Therefore, in Text Mode a pattern is defined as show in Figure 8-9, leaving the two LSBs set to 0s and defining our character within the remaining 6x8 pixel block.

In order to leave a space between characters on the screen, most of the patterns defined for Text Mode will only use a 5x7 grid. Special graphics characters might be defined for drawing lines, graphs, and charts that use the entire 6x8 pixel grid area. A special character set for Graphics I Mode and Graphics II Mode is included in Appendix F.

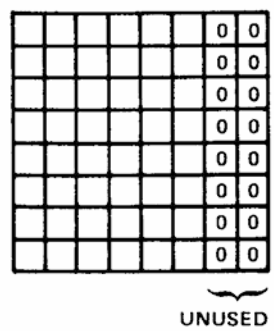


FIGURE 8-9 — 6X8 PIXEL PATTERN GRID FOR TEXT MODE

Up to 256 different patterns can be defined in the Pattern Table, though less space is required if not all 256 patterns are required. For example, if your application only required numbers 0 through 9 and upper case A through Z to be defined, then only the first 36 patterns (288 bytes) would be needed. These 36 patterns would then be selected by writing numbers ranging from 0 to 36 (Hex 24) to bytes in the Name Table. If, for instance, the letter "A" was the first pattern defined, it could be placed in every possible screen position by writing a zero to all 960 Name Table entries.

Figure 8-10 illustrates how VRAM is mapped to the Pattern Plane in Text Mode.

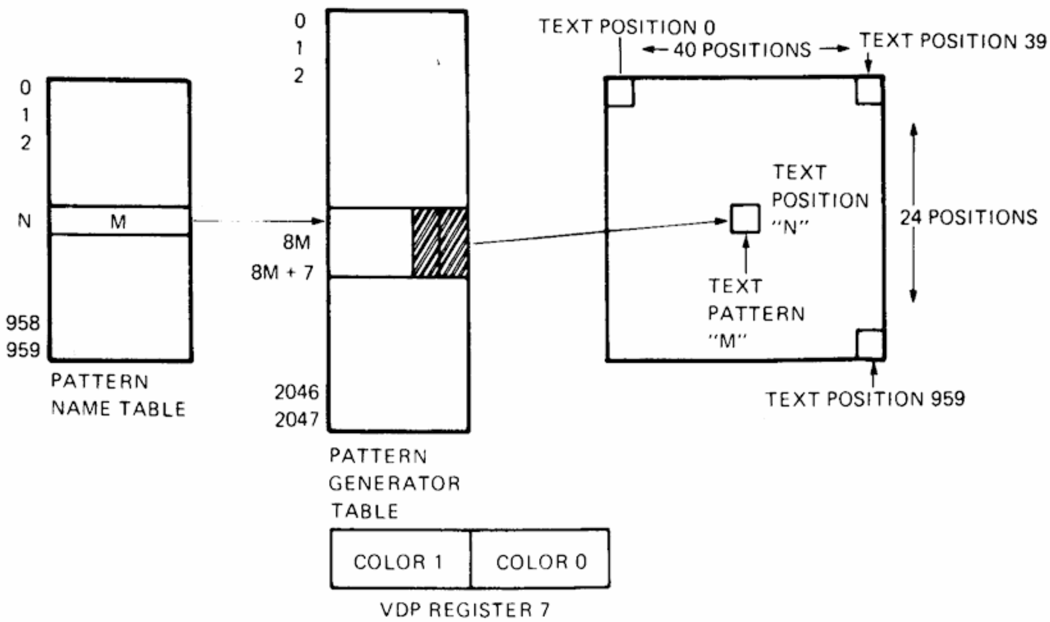


FIGURE 8-10 — MAPPING OF VRAM INTO THE PATTERN PLANE IN TEXT MODE

8.6 MULTICOLOR MODE

The VDP is in Multicolor Mode when the mode bits located in Registers 0 and 1 are equal to the following:

M1 = 0
M2 = 1
M3 = 0

Multicolor Mode provides a low-resolution display of 64 horizontal x 48 vertical color blocks. Each color block is equal to a 4x4 group of pixels and may be any of the sixteen VDP colors including transparent. The Backdrop color and Sprite Planes are also active in Multicolor Mode.

NOTE

Multicolor Mode is not supported by the Texas Instruments Advanced Video Display Processor.

Only two tables are required in VRAM in order to produce a Multicolor Mode picture, these being the Name Table and the Pattern Table. The Name Table consists of 768 entries like the other graphics modes, although the Name Table no longer points to a color list because the color of the blocks is derived from the Pattern Table. The name points to an eight-byte segment of VRAM in the Pattern Table.

Only two bytes of the eight-byte segment area are used to specify the screen image. These two bytes specify four colors, each occupying a 4x4 pixel area. The four MSBs of the first byte define the color of the upper left hand corner of the multicolor pattern. The LSBs define the color of the upper right quarter. The second byte similarly defines the lower left and right quarters of the multicolor pattern. The two bytes thus map into an 8x8 pixel multicolor pattern as shown in Figure 8-11.

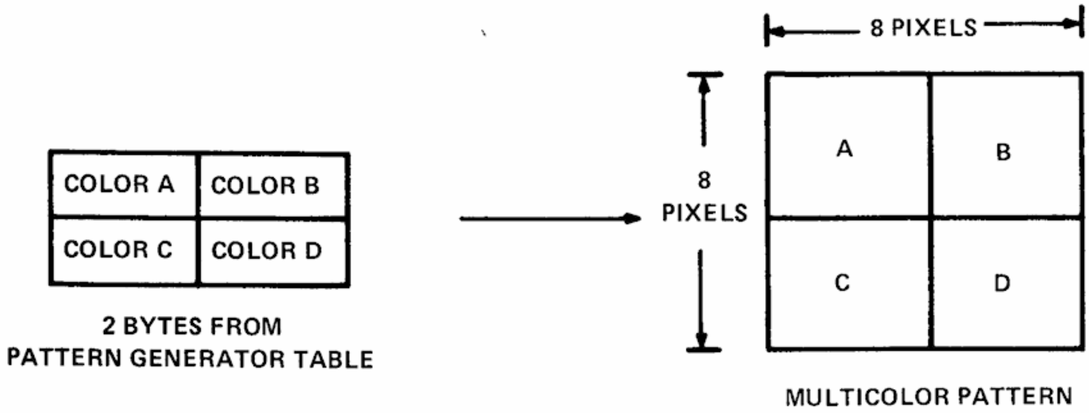


FIGURE 8-11 — MAPPING AN 8X8 PIXEL MULTICOLOR PATTERN

The location of the two bytes within the eight-byte segment pointed to by the name is dependent on the screen position where the name is mapped. For names in the top row (0-31), the two bytes are the first two in the eight-byte segments pointed to by the names. The next row of names (32-63) uses bytes number 3 and 4 within the eight-byte segment. The next row of names uses the 5th and 6th bytes, while the last row of names uses bytes 7 and 8. This series repeats for the remainder of the screen.

Let's go through a step-by-step example to help clear up any uncertainties about how Multicolor Mode works. Figure 8-12 is composed of a Multicolor Mode Name Table, Pattern Table, and a corresponding screen representation. Another screen image is also included to depict how the 767 screen positions, each composed of four 4x4 pixel blocks, fill the screen.

In our example (see Figure 8-12), a Name Table entry of Hex 02 points to locations Hex 08 and Hex 09. The first nibble of location Hex 08 contains the color red (Hex 06) and the second nibble contains the color blue (Hex 04). The first and second nibbles of the second byte contain blue and red, respectively. Therefore, screen position 0 contains the four colors specified. The calculations for this example and the others shown in Figure 8-12 are as follows.

EQUATION FOR FINDING PATTERN TABLE LOCATIONS

FIRST BYTE = 2 * ROW + NAME * 8

SECOND BYTE = FIRST BYTE + 1

ROW = MOD4[TRUNCATE(PATTERN POSITION/32)]

CALCULATIONS FOR EXAMPLES SHOWN IN FIGURE 8-12

NAME POSITION	PATTERN NAME	PATTERN TABLE ROWS
0	02	2 * 0 + 02 * 8 = >10 (Byte 1) 10 + 1 = >11 (Byte 2)
1	00	2 * 0 + 00 * 8 = >00 (Byte 1) 00 + 1 = >01 (Byte 2)
31	01	2 * 0 + 01 * 8 = >08 (Byte 1) 08 + 1 = >09 (Byte 2)
32	02	2 * 1 + 02 * 8 = >12 (Byte 1) 12 + 1 = >13 (Byte 2)
767	FF	2 * 3 + FF * 8 = >7FE (Byte 1) 01 + 7F8 = >7FF (Byte 2)

COLOR	HEX CODE	SYMBOL
DARK BLUE	04	B
DARK RED	06	R
DARK GREEN	0C	G
WHITE	0F	W

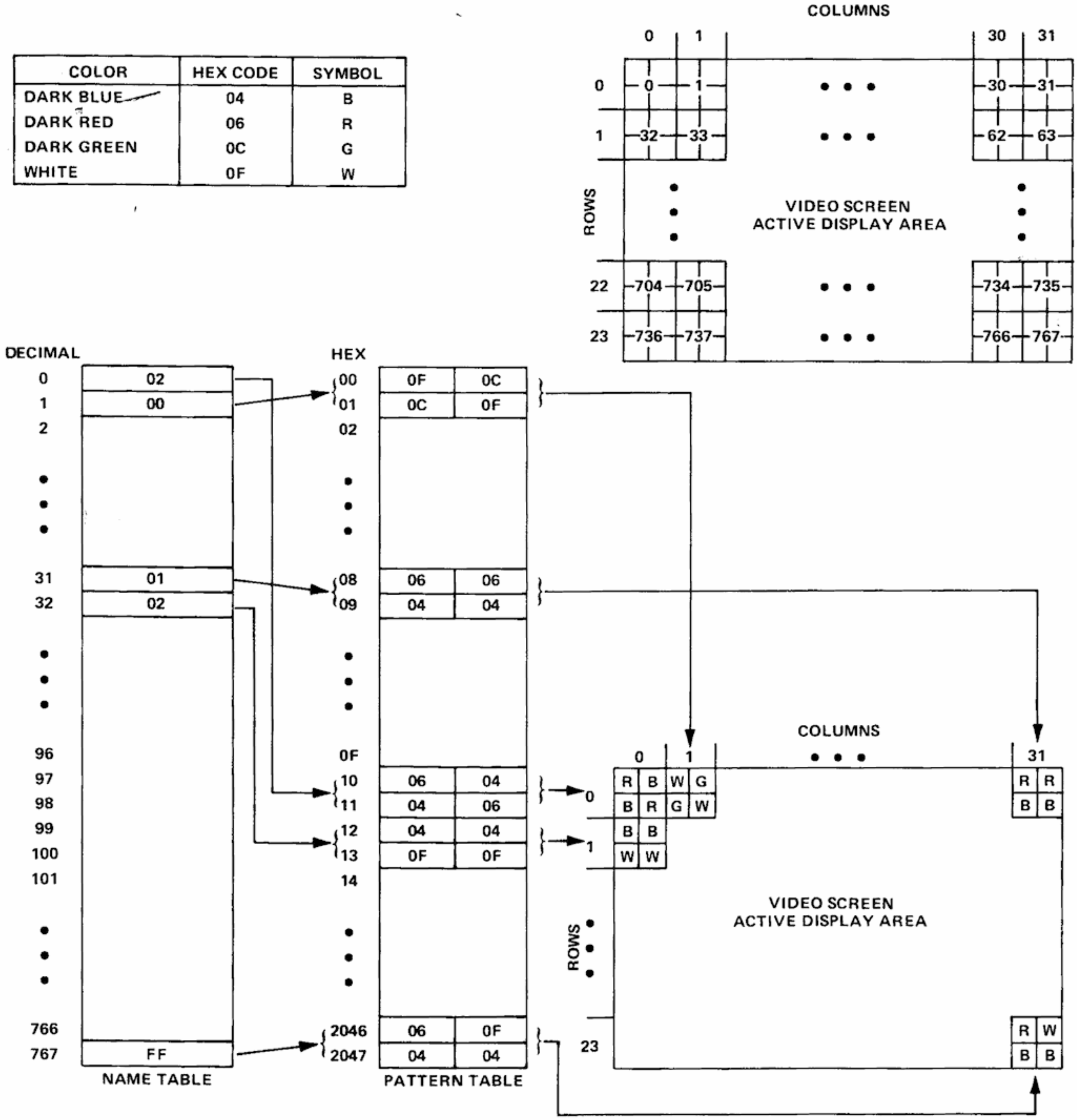


FIGURE 8-12 — MULTICOLOR MAPPING SCHEME

The mapping of VRAM contents to screen image is simpified by using duplicate names in the Name Table since the series of bytes used within the eight-byte segment specifies a 2x8 pixel color square pattern on the screen as a straightforward translation from the eight-byte segment in VRAM pointed to by the common name.

When used in this manner, 768 bytes are still used for the Name Table and 1536 bytes are used for the color information in the Pattern Table (24 rows x 32 columns x 2-bytes/pattern position). Thus a total of 1728 bytes (6144 + 768) in VRAM are required. It should be noted that the tables begin on 1K and 2K boundries and are therefore not contiguous.

9. SPRITES

Sprites are special animation-oriented patterns that can be made to move rapidly about the screen and change shape with very little programming effort. The video display has 32 Sprite Planes each of which contain a single sprite. These 32 Sprite Planes are numbered from 0 to 31 (see Section 2.1) with 0 being the highest priority or outermost Sprite Plane and 31 being the lowest priority Sprite Plane. When more than one sprite is located at the same screen coordinate the sprite on the higher priority plane will show through at that point. It should also be noted that all 32 sprites have a higher priority than the Pattern Plane and the Backdrop Plane.

Sprites come in two sizes, 8x8 pixels or 16x16 pixels. The size of all sprites is determined by the size bit in VDP Register 1. Register 1 also contains a sprite magnify bit which, when set, expands a sprite to double its normal size. Thus 8x8 sprites become 16x16, and 16x16 sprites would become 32x32. Unfortunately, when a sprite is magnified, its resolution is cut in half because the VDP maps each single pixel into a 2x2 pixel area.

Sprite patterns are defined in individual 8x8 pixel blocks exactly as patterns in Text or the graphics modes are. A Size 0 sprite (8x8 pixels) would require only one pattern to be defined. A Size 1 sprite (16x16 pixels) is made up of four 8x8 pixel patterns. All of the bits on within a sprite pattern are a single color, which can be any one of the 16 available VDP colors. Any bits off within a sprite pattern are automatically set to the VDP color transparent, which allows the Pattern Plane or Backdrop color to show through at those points. Any area within a sprite display plane outside of the sprite itself is also set to transparent. A good way to visualize this is to imagine a Sprite Plane as a pane of glass on which you can stick a single 8x8 or 16x16 pixel object.

Two tables are required in VRAM in order to produce a sprite display. The Sprite Attribute Table tells us some characteristics of each sprite, like screen location, color, and what pattern to pick for the shape of the sprite. The Sprite Pattern Table contains a library of sprite shape data to choose from.

All 32 VDP sprites may be displayed on the screen at the same time, however, a maximum of four sprites may be displayed on one horizontal line. If this rule is violated, the four highest priority sprites will be displayed normally, while the fifth and subsequent sprites will be automatically set to transparent. Furthermore, the Fifth Sprite Flag in the VDP Status Register is set to a 1, and the number of the violating fifth sprite is loaded into the Status Register. See Section 5.2 for more information on fifth sprites and the Status Register.

The VDP also provides limited sprite coincidence checking. If any two active sprites have overlapping bits, then the Coincidence Flag in the VDP Status Register will be set to a 1. It should be noted that the VDP only tells you if any two sprites are coinciding and does not specify the numbers of the sprites that are overlapping. Most applications that require knowing which sprites are coinciding continually monitor the Sprite Attribute Table for overlapping values.

9.1 THE SPRITE PATTERN TABLE

The Sprite Pattern Table has a maximum length of 2048 (Hex 800) bytes and is located in VRAM beginning on a 2K byte boundry. Its actual location in VRAM is determined by the base address in VDP Register 6.

It takes eight bytes of information to define the pattern of a Size 0 (8x8 pixel) sprite and 32 bytes (8x4) of data to define the pattern of a Size 1 (16x16 pixel) sprite. Therefore, 256 patterns can be defined for Size 0 sprites or 64 patterns for Size 1 sprites.

The Sprite Pattern Table can be as short as eight bytes if Size 0 sprites are used and 32 bytes if Size 1 sprites are used because the same sprite shape can be reused for as many sprites as desired. To select the same sprite pattern just repeat the name byte located in the Sprite Attribute Table.

9.2 THE SPRITE ATTRIBUTE TABLE

The Sprite Attribute Table contains four bytes of information for every sprite displayed. If all 32 sprites are to be displayed, then the table would have a maximum length of 128 bytes. The location of the Attribute Table in VRAM is defined by the base address contained in VDP Register 5.

The first four byte entry in the Sprite Attribute Table contains information pertaining to Sprite 0, which is the highest priority sprite. The last four byte entry in the Sprite Attribute Table contains information for the lowest priority sprite, Sprite 31. Figure 9-1 illustrates how the Sprite Attribute Table relates to the Sprite Planes.

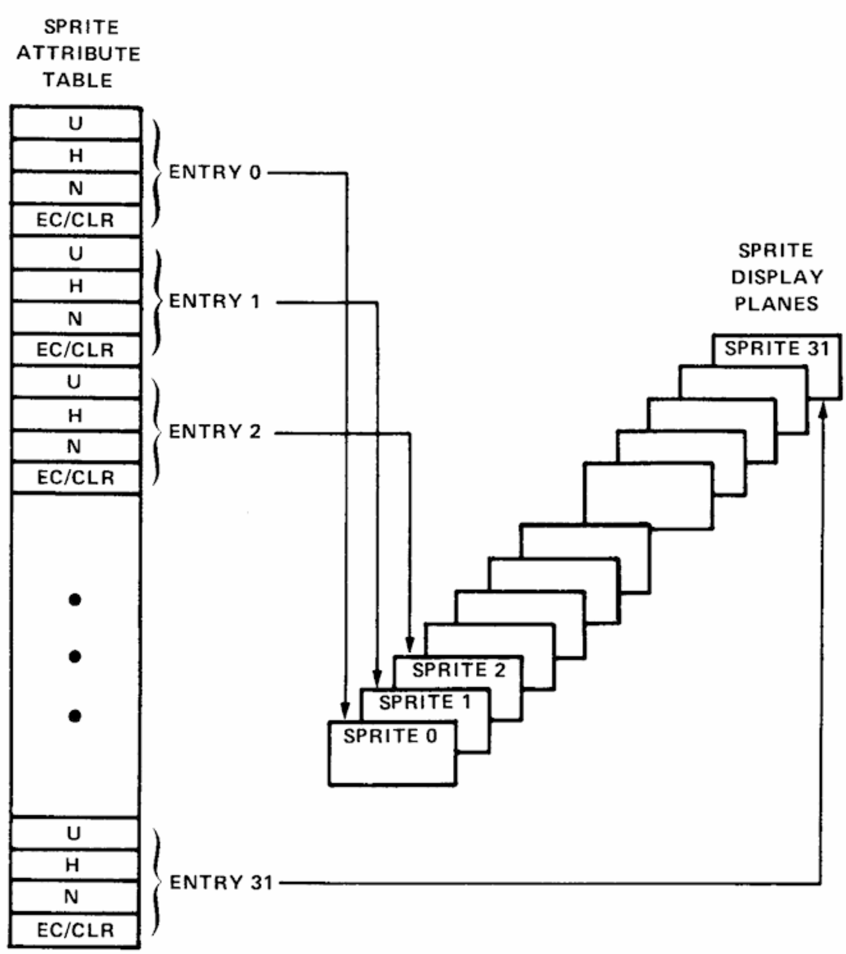


FIGURE 9-1 — SPRITE ATTRIBUTE TABLE AS RELATED TO SPRITE PLANES

Referring to Figure 9-2, let's examine one four byte attribute entry. The first two bytes determine the coordinate of the sprite on the display screen. The first byte is the vertical position and the second byte is the horizontal position. The third byte is the sprite name and specifies what pattern in the Sprite Pattern Table will be used as the sprite's shape. The fourth byte performs two functions: the lower four bits (nibble) determine the color of the sprite and the Early Clock bit (MSB) shifts the horizontal position of the sprite towards the left 32 pixels. Setting this bit high allows sprites to bleed (flow smoothly) off the left side of the screen. The other three bits in this fourth byte are unused and should be set to zeros.

	BIT NUMBER								
	MSB 0	1	2	3	4	5	6	LSB 7	
VERTICAL COORDINATE	VERT. POSN.	VERT. POSN.	VERT. POSN.	VERT. POSN.	VERT. POSN.	VERT. POSN.	VERT. POSN.	VERT. POSN.	BYTE 0
HORIZONTAL COORDINATE	HORIZ. POSN.	HORIZ. POSN.	HORIZ. POSN.	HORIZ. POSN.	HORIZ. POSN.	HORIZ. POSN.	HORIZ. POSN.	HORIZ. POSN.	BYTE 1
SPRITE NAME POINTER	NAME	NAME	NAME	NAME	NAME	NAME	NAME	NAME	BYTE 2
COLOR AND EARLY CLOCK BIT	EARLY CLOCK	0	0	0	COLOR	COLOR	COLOR	COLOR	BYTE 3

FIGURE 9-2 — SPRITE ATTRIBUTE TABLE ENTRY

9.2.1 Vertical Position

The first attribute byte of information is the vertical position of the sprite on the display screen. This coordinate determines the distance the sprite will be offset from the top of the screen in pixels. The position of a sprite is measured relative to the upper left hand corner of the sprite. A value of -1 (Hex FF) in the vertical position will butt a sprite up against the top of the screen, and a value of 191 (Hex BF) will position the sprite off the screen at the bottom as shown in Figure 9-3. Negative values can be used to bleed the sprite off the top edge of the screen. Values in the range of -32 and -1 (Hex E0 to FF) allow even the largest sprite (32x32 pixels) to bleed in from the top of the screen.

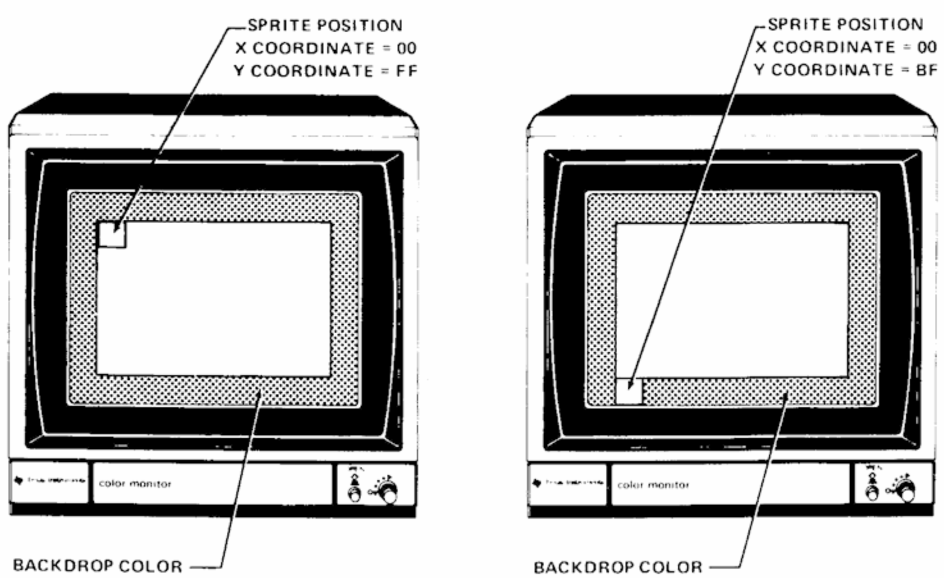


FIGURE 9-3 — VERTICAL SPRITE POSITIONING

Some applications require no sprites or less than 32 sprites to be displayed at a time. A value of Hex D0 in the vertical position of the Sprite Attribute Table will terminate sprite processing. If no sprites are to be used, Hex D0 should be the first entry in the Sprite Attribute Table. If only one sprite is to be used, then Hex D0 should be the first byte in the second sprite's attribute entry, which would be the fifth byte in the Sprite Attribute Table. Once the VDP finds a value of Hex D0 as a sprite attribute entry, it terminates processing of that sprite and all lower priority sprites.

9.2.2 Horizontal Position

The second byte of information in the Sprite Attribute Table is the horizontal coordinate. This value determines the distance the sprite will be offset in pixels from the left hand side of the screen. A value of Hex 00 will butt a sprite up against the left hand edge of the screen, while a value of 255 (Hex FF) will position the sprite completely off the right hand side of the screen as shown in Figure 9-4. Using values in the range of 255 (Hex FF) will bleed a sprite off the right hand edge of the display screen.

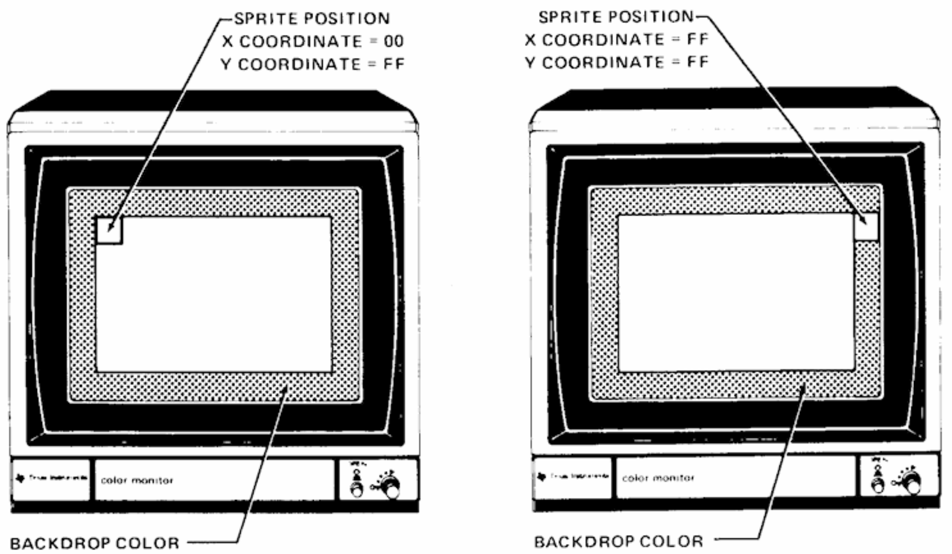


FIGURE 9-4 — HORIZONTAL SPRITE POSITIONING

In order to bleed a sprite off the left hand edge of the screen, a special bit called the Early Clock bit is used. This bit is the fourth byte of an attribute entry and is described later in this section.

9.2.3 Sprite Name

The third byte of information contained in a sprite attribute entry is the sprite name. The function of this byte is very similar to the function of a Name Table entry in the graphics modes. The value contained in this byte determines which pattern will be used as the sprite's shape. It points to a byte of information in the Pattern Table where the start of the sprite's pattern is located.

EXAMPLE 9-1.

8x8 (Size 0) Sprites

A value of Hex 00 as a Sprite Name Table entry would mean the first eight bytes in the Pattern Table would be used as the sprite's shape. A value of Hex 01 would choose the next eight bytes in the Sprite Pattern Table as the sprite's shape. Continuing on up to Hex FF gives us 256 8x8 pixel sprite shapes to choose from.

EXAMPLE 9-2.

16x16 (Size 1) Sprites

The value in the Sprite Name Table entry points to an eight-byte entry in the Sprite Pattern Table. Since a 16x16 pixel sprite is made up of four eight byte entries, our name values would be entries such as Hex 00,04,08,0C,10 etc. When the sprite Size 1 bit is set in VDP Register 1, the VDP will go to the eight-byte block pointed to by the sprite name and choose the next four eight byte entries in the Pattern Table as the sprite's shape.

Having the sprite pattern selectable by the sprite name makes for extremely simplified animation. For example, if the first four sprite patterns are defined as the graphic stages for a man walking, we could switch through these patterns and animate the man just by switching the sprite name values from 0-3 and then repeating the sequence.

9.2.4 Sprite Color and Early Clock Bit

The fourth byte in the Sprite Attribute Table entry performs two functions. The lower four bits (nibble) define the sprite color, which can be any of the 16 available VDP colors. The MSB is the Early Clock bit, which shifts the horizontal position of the sprite to the left 32 pixels (when set high). The remaining three bits are unused and should be set to 0.

The Early Clock bit is used to bleed a sprite off the screen or onto the screen from the left hand edge. When this bit is active (high), the horizontal position of the sprite is shifted to the left 32 pixels. Consider the horizontal position of a sprite being Hex 00, which butts the sprite up against the left hand edge of the screen. If the Early Clock bit is then set, even the largest sprite (32x32 pixels) would be completely off the screen. This allows values in the horizontal position in the range of 0 to 31 to bleed a sprite onto the left hand edge. Of course the Early Clock bit must be set low again in order to be able to bleed the sprite off the right-hand edge.

Now that all the information on sprites has been covered, refer to Figure 9-5 for an illustration of how sprites are mapped to the screen.

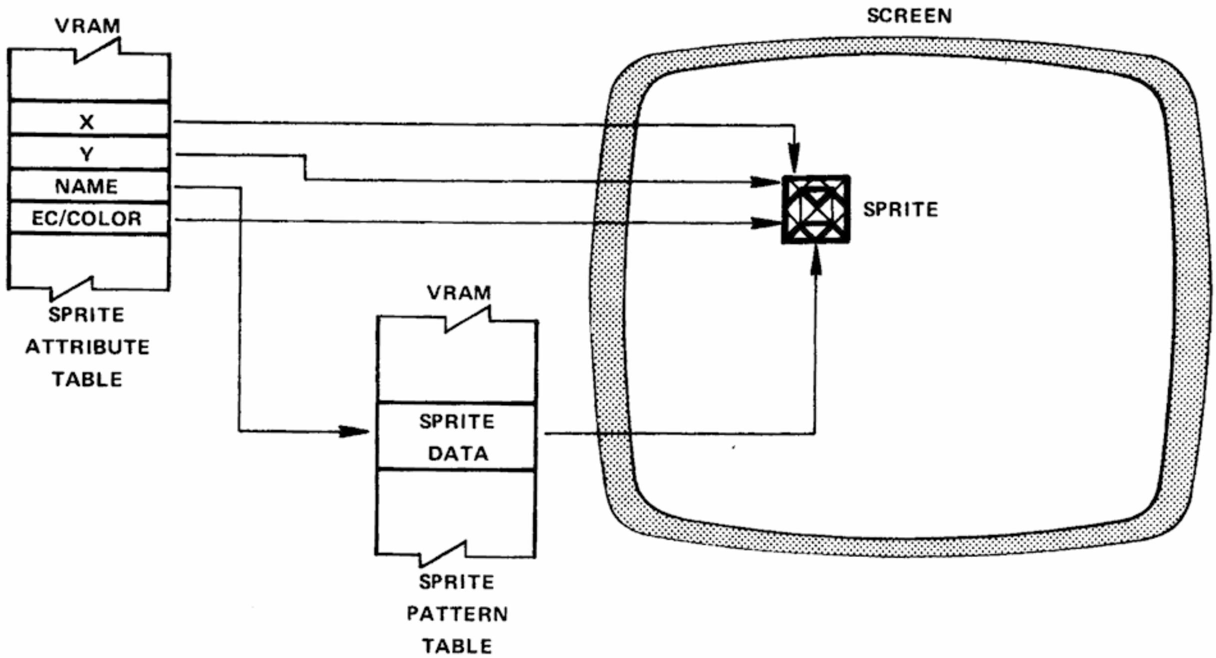


FIGURE 9-5 — SPRITE MAPPING

10. PROGRAMMING TIPS

10.1 HORIZONTAL AND VERTICAL SCROLLING

The simplest way to scroll the pattern plane display is to manipulate the values located in the Name Table. The only drawback to this method is that the screen will move in increments larger than one pixel. In Graphics I, II and Multicolor Modes the movement will be in eight pixel increments. In Text Mode the movement will be by six pixels when scrolling horizontally and eight pixels when scrolling vertically. The movement is determined by the size of a single pattern.

One major advantage to this method is that only a small number of bytes need to be moved in order to scroll the entire display. In Graphics I,II, and Multicolor Modes the Name Table is 768 bytes long, and in Text Mode the Name Table is 960 bytes long. Figure 10-1 shows the sequence for scrolling the Name Table left with screen wraparound. The Name Table in this figure has 768 entries and is designed for scrolling the screen in Graphics I or Graphics II Modes. Referring to the figure we can see that the steps involved in scrolling are as follows:

1) Read the data located in column 0 from VRAM and store it. The data consists of entries numbered 000,032,064,096 .... 736.
2) Read the data located in column 1 and write it to column 0. Read column 2 and write to column 1, and so forth, until column 31 has been read and moved to column 30.
3) Take the data stored from column 0 and write to column 31. The screen has now scrolled one column (eight pixels) to the left and wrapped around the screen.
4) Repeat this sequence to continually scroll the screen.

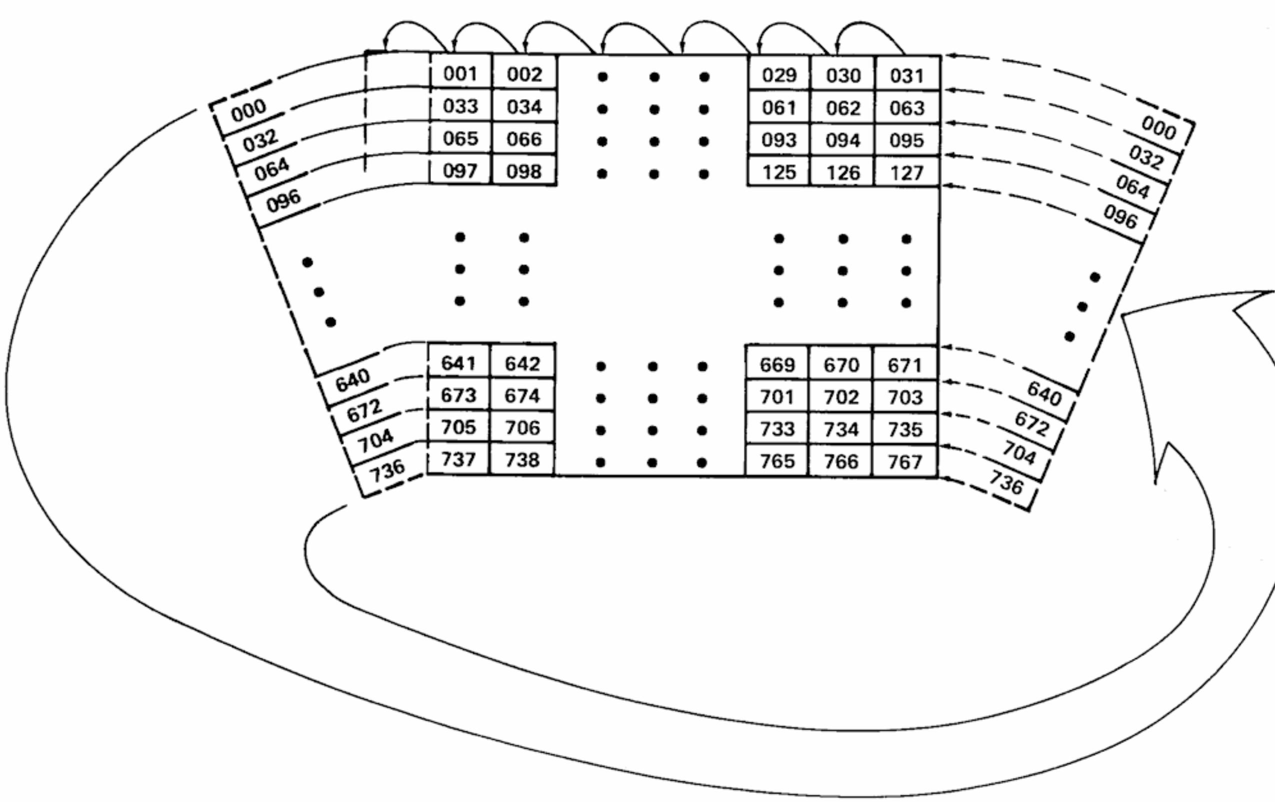


FIGURE 10-1 — SCROLLING THE NAME TABLE

10.2 ANIMATING SPRITES

The procedure for animating a sprite is relatively simple. First load the sprite pattern data for the sprites you wish to animate into the Sprite Pattern Table located in VRAM. Next load sprite attribute data into the Sprite Attribute Table located in VRAM. In this example we will talk about animating two sprites, one of which is a man walking and the other being a rotating planet. The sequence of shapes used for this exercise are shown in Figure 10-2 and Figure 10-3.

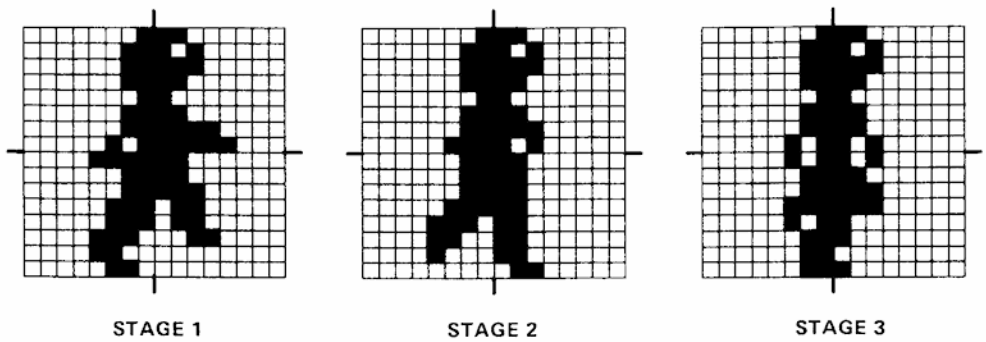


FIGURE 10-2 — ANIMATED WALKING MAN

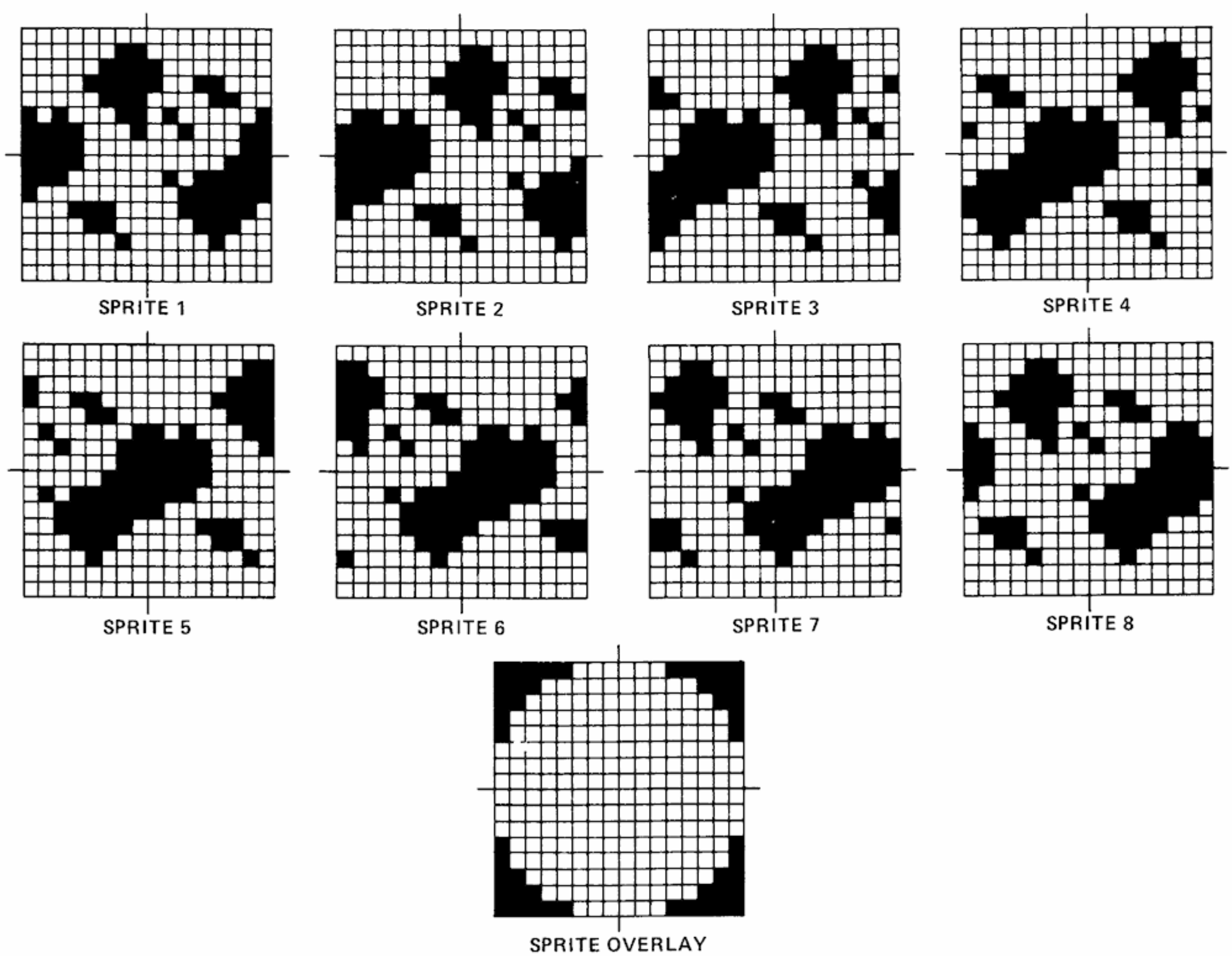


FIGURE 10-3 — ANIMATED PLANET

Referring to Figure 10-2 we can see that the walking man is a Size 1 sprite consisting of three stages of animation. The Hex data for these shapes is shown as the first three entries in Table 10-1, which is an example of what our source code listing might look like. The rotating planet is also a Size 1 sprite (see Figure 10-3) and consists of eight stages of animation. The data for these eight shapes is shown as the next eight entries in Table 10-1.

Since the rotating planet shapes were drawn as flat, square planets, a sprite overlay pattern is used in Figure 10-4 to make the planets look rounded. Figure 10-4 shows what the planets would look like with this sprite overlaid on top of them. The data for the overlay sprite is the last pattern entry in Table 10-1.

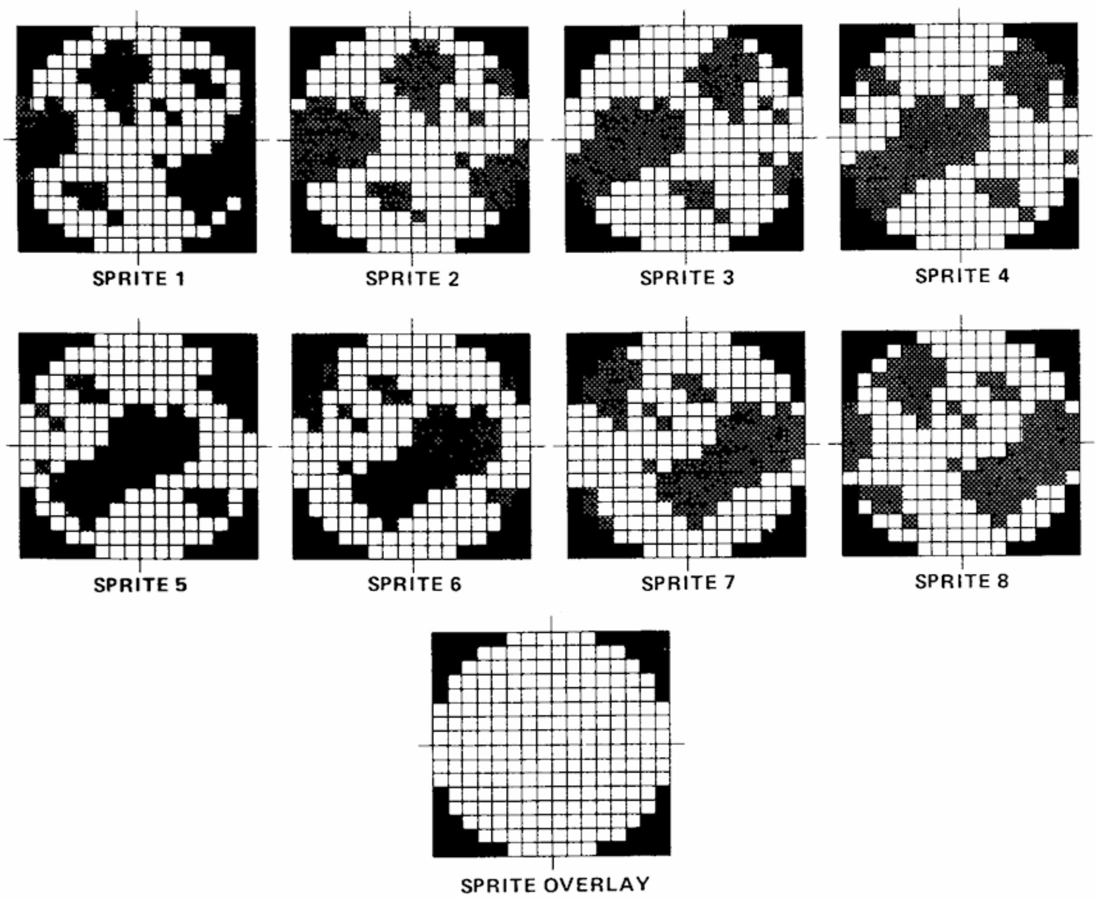


FIGURE 10-4 — ANIMATED PLANET WITH OVERLAY

Now that all the graphic data for our sprites has been defined we need to create a Sprite Attribute Table in order to have them displayed on the screen. Referring to the section of Table 10-1 labeled Sprite Attribute Table, you will see that three Sprite Attribute Table entries have been defined. The first four bytes define the first entry, which is the highest priority sprite (Sprite 0). This is the sprite used for the animated walking man.

An actual program to animate the man would change the name byte from Hex 00 to Hex 04 to Hex 08. This would shift the sprite through each of the pattern stages defined earlier. Hex 00 is the initial name value because Stage 1 of the walking man shape starts at byte 00 in the Sprite Pattern Table.

The next two Sprite Attribute Table entries (Sprites 1 and 2) are used to define the rotating planet. The spherical overlay is defined as a higher priority sprite than the rotating planet. This enables us to mask off bits around the square edges of the planet in order to make it appear round.

The spherical overlay name byte is set to Hex 2C because the pattern data for it falls on byte number 2C in the Sprite Pattern Table. This byte would remain the same in a program that animated the planet. If the horizontal and vertical positions of the rotating planet were changed during program execution, the horizontal and vertical positions of the overlay would have to be changed also.

The third Sprite Attribute Table entry (Sprite 2) is for the different stages of the rotating planet animation. The initial setting of 0C points to the pattern which falls on byte 0C in the Sprite Pattern Table. This is stage 1 of the rotating planet. During the course of a program we would shift the name byte through all eight stages of planet animation. Referring to the Pattern Table we can see that the values are 0C,10,14,18,1C,20,24,28. After shifting through all eight patterns the sequence would be repeated.

TABLE 10-1 – ANIMATION EXAMPLE DATA

```
*----------------------------------------------------*
*                                                    *
*             (* SPRITE PATTERN TABLES *)            *
*                                                    *
*----------------------------------------------------*
*
*     3 Stage Animation for "Man Walking" Sprite
*
        DATA >0103,>0303,>0103,>0305    Sprite Name = 00
        DATA >0F03,>0307,>070E,>0C06     (Man Walking. Stage 1)
        DATA >C0A0,>E0C0,>80C0,>F0F8
        DATA >C0C0,>F070,>6030,>0000
*
        DATA >0103,>0303,>0103,>0307    Sprite Name = 04
        DATA >0303,>0307,>0E0C,>0800     (Man walking. Stage 2)
        DATA >C0A0,>E0C0,>80C0,>E0A0
        DATA >C0C0,>C0C0,>C0C0,>C060
*
        DATA >0103,>0303,>0103,>0305    Sprite Name = 08
        DATA >0503,>0307,>0503,>0303     (Man walking. Stage 3)
        DATA >C0A0,>E0C0,>80C0,>C0A0
        DATA >A0C0,>E0E0,>8080,>0080
*
*     8 Stage Animation for "Rotating Planet"
*
        DATA >0003,>070F,>07A3,>F1F0    Sprite Name = 0C
        DATA >F0E0,>801C,>0C02,>0000     (Rotating Planet. Stage 1)
        DATA >0000,>8098,>0C41,>2303
        DATA >075F,>3F1E,>3E1C,>0800
*
        DATA >0000,>0103,>0168,>FCFC    Sprite Name = 10
        DATA >FCF8,>E007,>0300,>0000     (Rotating Planet. Stage 2)
        DATA >00C0,>E0E6,>C2D0,>4800
        DATA >0117,>0F06,>0E84,>0000
*
```

```
        DATA >0000,>0000,>401A,>3F3F    Sprite Name = 14
        DATA >7EFE,>F860,>6100,>0000     (Rotating Planet. Stage 3)
        DATA >0030,>78F8,>7034,>1200
        DATA >0005,>0300,>C2C0,>2000
*
        DATA >0000,>0060,>3006,>8F0F    Sprite Name = 18
        DATA >1F7F,>FE78,>7830,>0000     (Rotating Planet. Stage 4)
        DATA >0008,>1C3E,>1C8D,>C4C0
        DATA >C181,>0000,>7030,>0800
*
        DATA >0000,>0018,>0C41,>2303    Sprite Name = 1C
        DATA >075F,>3F1E,>3E1C,>0800     (Rotating Planet. Stage 5)
        DATA >0000,>040E,>06A3,>F1F0
        DATA >F0E0,>8000,>1C0C,>0000
*
        DATA >0000,>2066,>43D0,>4800    Sprite Name = 20
        DATA >0117,>0F07,>0F07,>0200     (Rotating Planet. Stage 6)
        DATA >0000,>0002,>0068,>FCFC
        DATA >FCF8,>E080,>8600,>0000
*
        DATA >0010,>3879,>7034,>1200    Sprite Name = 24
        DATA >0005,>0301,>0301,>0000     (Rotating Planet. Stage 7)
        DATA >0000,>0080,>C01A,>3F3F
        DATA >7FFE,>F8E0,>E0C0,>8000
*
        DATA >000C,>1E3E,>1C8D,>C4C0    Sprite Name = 28
        DATA >C081,>0000,>7030,>0800     (Rotating Planet. Stage 8)
        DATA >0000,>0060,>3006,>8F0F
        DATA >1F7F,>FE78,>F870,>2000

        DATA >071F,>3F7F,>7FFF,>FFFF    Sprite Name = 2C
        DATA >FFFF,>FF7F,>7F3F,>1F07     (Spherical Overlay)
        DATA >E0F8,>FCFE,>FEFF,>FFFF
        DATA >FFFF,>FFFE,>FEFC,>F8E0
*
*----------------------------------------------------*
*                                                    *
*           (* SPRITE ATTRIBUTE TABLE *)             *
*                                                    *
*----------------------------------------------------*
*
*     Attribute table entry for "Man Walking"
*     (The Name Byte will be either 00,04, or 08)
*
        BYTE >00            Y Coordinate
        BYTE >00            X Coordinate
        BYTE >00            Name
        BYTE >0F            EC/Color = White
*
*     Attribute table entry for "Spherical Overlay"
*     (The Name Byte will always be 2C)
*
        BYTE >00            Y Coordinate
        BYTE >00            X Coordinate
        BYTE >2C            Name
        BYTE >01            EC/Color = Black
*
```

```
*     Attribute table entry for "Rotating Planet"
*     (The Name Byte will either be 0C,10,14,18
*      1C,20,24, or 28)
*
        BYTE >00        Y Coordinate
        BYTE >00        X Coordinate
        BYTE >0C        Name
        BYTE >04        EC/Color = Blue
        END
```

10.3 SPRITE COINCIDENCE

The Sprite Coincidence Flag, located in the Status Register, is set whenever any two sprites have overlapping pixels. Most applications need to know not only that sprites have coincided, but which ones in particular are coinciding. A good example of this is the rotating planet sprite just described.

Since we actually defined two sprites for this shape (Sprite 1 and 2) to be located directly on top of one another on the screen, the Coincidence bit in the Status Register would be set all the time. If we wanted to monitor coincidence between the rotating planet and the walking man sprite, it would be necessary to keep track of their screen position in our program. This can be done by reading the X and Y coordinates of every Sprite Attribute Table entry and then comparing them to each other. In the case of the man and planet, if the first two bytes of attribute entry 1 (sprite 0) were the same as the first two bytes of attribute entry 2 (sprite 1), then we would know the man was positioned exactly on top of the planet.

APPENDIX A

A. REGISTER VRAM LOOKUP TABLES

Covers Registers 2-6 with special case diagrams for Registers 3 and 4 when in Graphics II Mode.

$R2 * 400_{(16)}$ = START ADDRESS

R2	START ADDRESS
00	0000
01	0400
02	0800
03	0C00
04	1000
05	1400
06	1800
07	1C00
08	2000
09	2400
0A	2800
0B	2C00
0C	3000
0D	3400
0E	3800
0F	3C00

(R3)* $40_{(16)}$ STARTING ADDRESS

R3	START ADDRESS	R3	START ADDRESS	R3	START ADDRESS	R3	START ADDRESS	R3	START ADDRESS	R3	START ADDRESS
00	0000	2B	0AC0	56	1580	81	2040	AC	2B00	D6	3580
01	0040	2C	0B00	57	15C0	82	2080	AD	2B40	D7	35C0
02	0080	2D	0B40	58	1600	83	20C0	AE	2B80	D8	3600
03	00C0	2E	0B80	59	1640	84	2100	AF	2BC0	D9	3640
04	0100	2F	0BC0	5A	1680	85	2140	B0	2C00	DA	3680
05	0140	30	0C00	5B	16C0	86	2180	B1	2C40	DB	36C0
06	0180	31	0C40	5C	1700	87	21C0	B2	2C80	DC	3700
07	01C0	32	0C80	5D	1740	88	2200	B3	2CC0	DD	3740
08	0200	33	0CC0	5E	1780	89	2240	B4	2D00	DE	3780
09	0240	34	0D00	5F	17C0	8A	2280	B5	2D40	DF	37C0
0A	0280	35	0D40	60	1800	8B	22C0	B6	2D80	E0	3800
0B	02C0	36	0D80	61	1840	8C	2300	B7	2DC0	E1	3840
0C	0300	37	0DC0	62	1880	8D	2340	B8	2E00	E2	3880
0D	0340	38	0E00	63	18C0	8E	2380	B9	2E40	E3	38C0
0E	0380	39	0E40	64	1900	8F	23C0	BA	2E80	E4	3900
0F	03C0	3A	0E80	65	1940	90	2400	BB	2EC0	E5	3940
10	0400	3B	0EC0	66	1980	91	2440	BC	2F00	E6	3980
11	0440	3C	0F00	67	19C0	92	2480	BD	2F40	E7	39C0
12	0480	3D	0F40	68	1A00	93	24C0	BE	2F80	E8	3A00
13	04C0	3E	0F80	69	1A40	94	2500	BF	2FC0	E9	1A40
14	0500	3F	0FC0*	6A	1A80	95	2540	C0	3000	EA	3A80
15	0540	40	1000	6B	1AC0	96	2580	C1	3040	EB	3AC0
16	0580	41	1040	6C	1B00	97	25C0	C2	3080	EC	3B00
17	05C0	42	1080	6D	1B40	98	2600	C3	30C0	ED	3B40
18	0600	43	10C0	6E	1B80	99	2640	C4	3100	EE	3B80
19	0640	44	1100	6F	1BC0	9A	2680	C5	3140	EF	3BCD
1A	0680	45	1140	70	1C00	9B	26C0	C6	3180	F0	3C00
1B	06C0	46	1180	71	1C40	9C	2700	C7	31C0	F1	3C40
1C	0700	47	11C0	72	1C80	9D	2740	C8	3200	F2	3C80
1D	0740	48	1200	73	1CC0	9E	2780	C9	3240	F3	3CC0
1E	0780	49	1240	74	1D00	9F	27C0	CA	3280	F4	2D00
1F	07C0	4A	1280	75	1D40	A0	2800	CB	32C0	F5	3D40
20	0800	4B	12C0	76	1D80	A1	2840	CC	3300	F6	3D80
21	0840	4C	1300	77	1DC0	A2	2880	CD	3340	F7	3DC0
22	0880	4D	1340	78	1E00	A3	28C0	CE	3380	F8	3E00
23	08C0	4E	1380	79	1E40	A4	2900	CF	33C0	F9	3E40
24	0900	4F	13C0	7A	1E80	A5	2940	D0	3400	FA	3E80
25	0940	50	1400	7B	1EC0	A6	2980	D1	3440	FB	3EC0
26	0980	51	1440	7C	1F00	A7	29C0	D2	3480	FC	3F00
27	09C0	52	1480	7D	1F40	A8	2A00	D3	34C0	FD	3F40
28	0A00	53	14C0	7E	1F80	A9	2A40	D4	3500	FE	3F80
29	0A40	54	1500	7F	1FC0	AA	2A80	D5	3540	FF	3FC0
2A	0A80	55	1540	80	2000	AB	2AC0				

REGISTER 3 ADDRESSING FOR GRAPHICS II MODE

R3	START ADDRESS
7F	0000
FF	2000

(R4) * $800_{(16)}$ = START ADDRESS

R4	START ADDRESS
00	0000
01	0800
02	1000
03	1800
04	2000
05	2800
06	3000
07	3800

REGISTER 4 ADDRESSING FOR GRAPHICS II MODE

R4	START ADDRESS
03	0000
07	2000

(R5) * $80_{(16)}$ = START ADDRESS

R5	START ADDRESS	R5	START ADDRESS	R5	START ADDRESS	R5	START ADDRESS
00	0000	21	1080	40	2000	60	3000
01	0080	22	1100	41	2080	61	3080
02	0400	23	1180	42	2100	62	3100
03	0180	24	1200	43	2180	63	3180
04	0200	25	1280	44	2200	64	3200
05	0280	26	1300	45	2280	65	3280
06	0300	27	1380	46	2300	66	3300
07	0380	28	1400	47	2380	67	3380
08	0400	29	1480	48	2400	68	3400
09	0480	2A	1500	49	2480	69	3480
0A	0500	2B	1580	4A	2500	6A	3500
0B	0580	2C	1600	4B	2580	6B	3580
0C	0600	2D	1680	4C	2600	6C	3600
0D	0680	2E	1700	4D	2680	6D	3680
0E	0700	2F	1780	4E	2700	6E	3700
0F	0780	30	1800	4F	2780	6F	3780
10	0800	31	1880	50	2800	70	3800
11	0880	32	1900	51	2880	71	3880
12	0900	33	1980	52	2900	72	3900
13	0980	34	1A00	53	2980	73	3980
14	0A00	35	1A80	54	2A00	74	3A00
15	0A80	36	1B00	55	2A80	75	3A80
16	0B00	37	1B80	56	2B00	76	3B00
17	0B80	38	1C00	57	2B80	77	3B80
18	0C00	39	1C80	58	2C00	78	3C00
19	0C80	3A	1D00	59	2C80	79	3C80
1A	0D00	3B	1D80	5A	2D00	7A	3D00
1B	0D80	3C	1E00	5B	2D80	7B	3D80
1C	0E00	3D	1E80	5C	2E00	7C	3E00
1D	0E80	3E	1F00	5D	2E80	7D	3E80
1E	0F00	3F	1F80	5E	2F00	7E	3F00
1F	0F80 *			5F	2F80	7F	3F80
20	1000						

STARTING ADDRESS = R6 *<800

R6	START ADDRESS
00	0000
01	0800
02	1000
03	1800
04	2000
05	2800
06	3000
07	3800

APPENDIX B

B. CPU TO VDP ACCESS TIMES

CONDITION	MODE	VDP DELAY	TIME WAITING FOR AN ACCESS WINDOW	TOTAL TIME
Active Display Area	Text	2 μs	0 - 1.1 μs	2 - 3.1 μs
Active Display Area	Graphics I, II	2 μs	0 - 5.95 μs	2 - 8 μs
4300 μs after Vertical Interrupt Signal	All	2 μs	0 μs	2 μs
Register I Blank Bit 0	All	2 μs	0 μs	2 μs
Active Display Area	Multicolor	2 μs	0 - 1.5 μs	2 - 3.5 μs

APPENDIX C

C. PATTERN GRAPHICS ADDRESS LOCATION TABLES

GRAPHICS I MODE ADDRESS LOCATION

ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS
1) PATTERN	NTB														PATTERN NAME TABLE BASE (VDP REG2)
NAME					ROW										PATTERN POSITION
ADDRESS										COLUMN					
2) PATTERN	COLB														PATTERN COLOR TABLE BASE (VDP REG3)
COLOR									0						ALWAYS "0" IN BIT 8
ADDRESS										NAME (0-4)					FIVE MOST SIGNIFICANT BITS OF NAME
3) PATTERN	PGB														PATTERN GENERATOR TABLE BASE (VDP REG4)
GENERATOR				NAME											ALL 8 BITS OF NAME
ADDRESS												XXX			THREE LSB'S FORM PATTERN ROW POSITION

GRAPHICS II MODE ADDRESS LOCATION

ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS
1) PATTERN	NTB														PATTERN NAME TABLE BASE (VDP REG2)
NAME					ROW										PATTERN POSITION ROW
ADDRESS										COLUMN					PATTERN POSITION COLUMN
2) PATTERN															PATTERN COLOR TABLE BASE MSB (VDP REG3)
COLOR		XX													TWO MSB FROM VERTICAL COUNTER
ADDRESS				NAME											ALL 8 BITS OF NAME
												XXX			COLOR TABLE BYTE/LINE
3) PATTERN															PATTERN GENERATOR TABLE BASE BIT 5 (VDP REG4)
GENERATOR		XX													TWO MSB FROM VERTICAL COUNTER
ADDRESS				NAME											ALL 8 BITS OF NAME
												XXX			PATTERN GENERATOR BYTE/LINE NUMBER

TEXT MODE ADDRESS LOCATION

ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS
TEXT MODE PATTERN	NTB														PATTERN NAME TABLE BASE (VDP REG2)
NAME ADDRESS					TEXT POSITION										EQUAL (TEXT POSITION ROW # TIMES 40) PLUS (TEXT POSITION COLUMN NUMBER)
TEXT MODE PATTERN	PGB														PATTERN GENERATOR TABLE BASE (VDP REG4)
GENERATOR				NAME											NAME
ADDRESS												XXX			BYTE/ROW NUMBER

SPRITE ADDRESS LOCATION

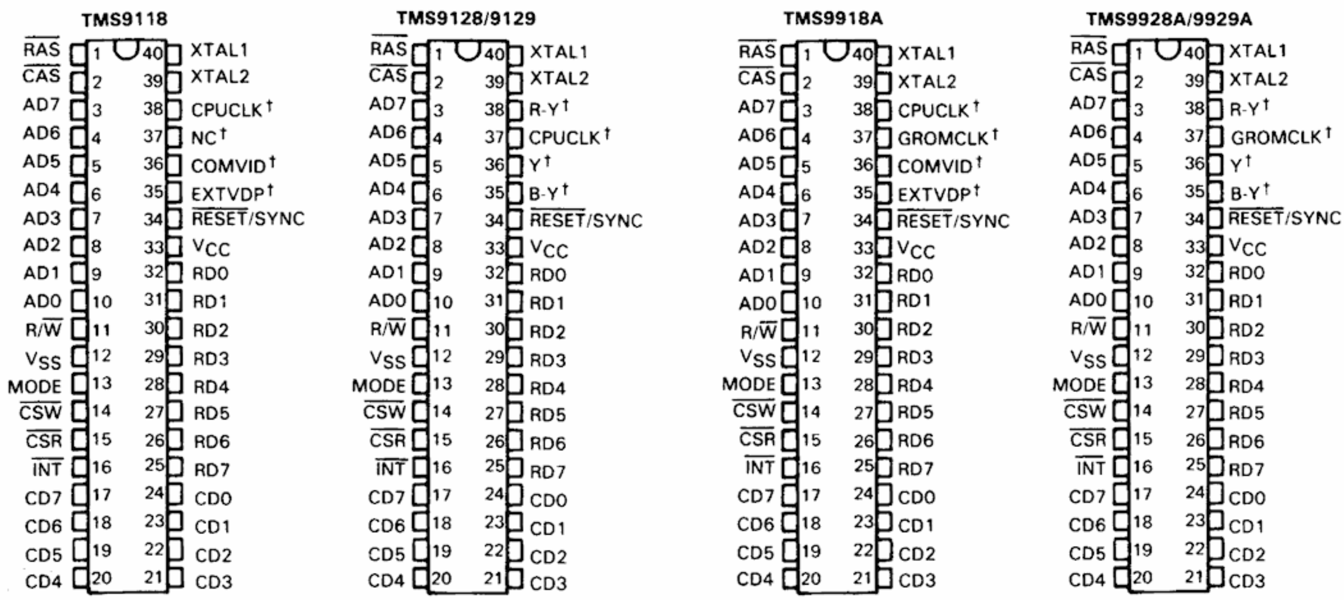
ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS
SPRITE ATTRIBUTE ADDRESS	SAB														SPRITE ATTRIBUTE TABLE BASE (VDP REG5)
								SPRITE							SPRITE NUMBER
													XX		ATTRIBUTE NUMBER: 00 FOR VERTICAL POSITION 01 FOR HORIZONTAL POSITION 10 FOR NAME 11 FOR TAG (EARLY CLOCK AND COLOR)
SIZE = 0 SPRITE PATTERN GENERATOR	SPGB														SPRITE PATTERN GENERATOR BASE (VDP REG4)
				NAME											NAME ATTRIBUTE OF SPRITE
												XXX			THREE LSB'S GIVE BYTE/ROW NUMBER
SIZE = 1 SPRITE PATTERN GENERATOR	SPGB														SPRITE PATTERN GENERATOR BASE (VDP REG4)
				NAME (0-5)											SIX MSB OF NAME
										X					0 FOR LSB OF PATTERN, 1 FOR MSB OF PATTERN
											XXXXX				SIZE = 1 SPRITE BYTE NUMBER

MULTICOLOR ADDRESS LOCATION

ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS
4) MULTICOLOR PATTERN NAME ADDRESS	NTB														PATTERN NAME TABLE BASE (VDP REG2)
					ROW										PATTERN POSITION ROW
											COLUMN				PATTERN POSITION COLUMN
5) MULTICOLOR PATTERN GENERATOR ADDRESS	PGB														PATTERN GENERATOR TABLE BASE (VDP REG4)
				NAME											NAME FROM NAME FETCH
											XXX				THREE LSB'S FORM BYTE/SQUARE ROW

APPENDIX D

D. IC PINOUTS FOR TMS9918A/28A/29A AND TMS9118/28/29



†Pins 35 to 38 are the only pins which vary for each device.

APPENDIX E

E. DEMO ASSEMBLY LANGUAGE PROGRAMS

1) 6502 Assembly Language
2) 8088 Assembly Language
3) TMS7000 Assembly Language
4) TMS9995 Assembly Language
5) TMS9995 Assembly Language Subroutines
   a) Draw Font To Bit-Map Screen
   b) Draw A Line
   c) Plot Pixel In VRAM
   d) Initialize Name Table
   e) Clear/Fill VDP RAM
   f) Block Move Memory: System To VDP

```
*-----------------------------------------------------*
*               DEMONSTRATION SOFTWARE                *
*                                                     *
*                  WRITTEN IN 6502                    *
*                ASSEMBLY LANGUAGE                    *
*-----------------------------------------------------*
*
        ORG $4000
*
*-----------------------------------------------------*
*                                                     *
*   THE FOLLOWING PROGRAM IS AN OUTLINE FOR CREATING  *
*   A GRAPHICS I MODE PICTURE. IT INITIALIZES THE     *
*   VDP REGISTERS, CLEARS ALL OF VIDEO RAM, AND THEN  *
*   LOADS THE NAME TABLE, COLOR TABLE, AND PATTERN    *
*   TABLE WHICH FORM THE DISPLAY. FILL IN THE TABLES  *
*   TO CREATE YOUR OWN DISPLAY.                       *
*                                                     *
*-----------------------------------------------------*
*
*-----------------------------------------------------*
*                      EQUATES                        *
*-----------------------------------------------------*
*
MODE0     EQU $C000       MODE LOW SELECTS A READ/WRITE VIDEO RAM
MODE1     EQU $C002       MODE HIGH SELECTS A READ/WRITE VIDEO REGISER
*
VDPCT     EQU $6000       VRAM COLOR TABLE ADDRESS
VDPPT     EQU $4800       VRAM PATTERN TABLE ADDRESS
VDPNT     EQU $4400       VRAM NAME TABLE ADDRESS
*
*-----------------------------------------------------*
*                  ZERO PAGE USAGE                    *
*-----------------------------------------------------*
*
PATTPT    EQU  $00        AND $01 (MEMORY POINTER)
NAMPT     EQU  $02        AND $03 (MEMORY POINTER)
*
*-----------------------------------------------------*
*                  INITIALIZATION                     *
*-----------------------------------------------------*
*
*     SETUP ZERO PAGE ADDRESSES
*
          LDA  #<PATTERNS
          STA  PATTPT
          LDA  #>PATTERNS
          STA  PATTPT+1
*
          LDA  #<NAMES
          STA  NAMPT
          LDA  #>NAMES
          STA  NAMPT+1
*
*-----------------------------------------------------*
*                   MAIN PROGRAM                      *
*-----------------------------------------------------*
*
```

```
         JSR   INITREG         INITIALIZE VDP REGISTERS
         JSR   ZAPRAM          CLEAR ALL VIDEO RAM
         JSR   LDPATT          LOAD THE PATTERN TABLE
         JSR   LDCLR           LOAD THE COLOR TABLE
         JSR   LDNAM           LOAD THE NAME TABLE
         RTS                   DONE. RETURN TO CALLER
*
*-----------------------------------------------------*
*                     SUBROUTINES                     *
*-----------------------------------------------------*
*
*     INITIALIZE VDP REGISTER
*
INITREG  LDY   #$80            FIRST REGISTER
         LDX   #0              LOOP COUNTER
INIT1    LDA   ITAB,X          GET REGISTER DATA
         STA   MODE1           SEND DATA TO VDP
         STY   MODE1           TELL VDP REGISTER NUMBER
         INY
         INX
         CPX   #$08            DONE ALL 8 REGISTERS?
         BNE   INIT1           NO, CONTINUE..
         RTS                   DONE  RETURN TO CALLER
*
*     CLEAR ALL VIDEO RAM ($0000-$3FFF)
*
ZAPRAM   LDY   #$40            SETUP VRAM ADDRESS
         LDA   #0
         STA   MODE1
         STY   MODE1
         LDX   #$C0            COUNT HIGH
NEXF     LDY   #0              COUNT LOW
FILL     STA   MODE0           WRITE A ZERO TO VRAM
         INY                   NEXT LOC.
         BNE   FILL            KEEP GOING TILL PAGE DONE
         INX                   DONE ALL 40 PAGES?
         BNE   NEXF            NO. KEEP GOING
         RTS
*
*     MOVE PATTERN TABLE TO VRAM
*
LDPATT   LDX   #8              PAGE COUNTER
         LDY   #>VDPPT         ADDRESS SETUP
         LDA   #<VDPPT
         STA   MODE1
         STY   MODE1
         LDY   #0              INIT COUNTER
NEXA     LDA   (PATTPT),Y      GET A BYTE
         STA   MODE0           WRITE TO VDP
         INY                   PAGE DONE?
         BNE   NEXA            NOT YET..
         INC   PATTPT+1        INCREMENT TABLE POINTER
         DEX                   DONE ALL 8 PAGES?
         BNE   NEXA            NOT YET..
         RTS
*
*     MOVE COLOR TABLE TO VRAM
*
```

```
LDCLR     LDY  #>VDPCT        ADDRESS SETUP
          LDA  #<VDPCT
          STA  MODE1
          STY  MODE1
          LDY  #0             INIT COUNTER
NEXC      LDA  COLORS,Y       GET A BYTE
          STA  MODE0          WRITE TO VDP
          INY
          CPY  #32            DONE?
          BNE  NEXC           NOT YET..
          RTS
*
*     MOVE NAME TABLE TO VRAM
*
LDNAM     LDX  #3             PAGE COUNTER
          LDY  #>VDPNT        ADDRESS SETUP
          LDA  #<VDPNT
          STA  MODE1
          STY  MODE1
          LDY  #0             INIT COUNTER
NEXN      LDA  (NAMPT),Y      GET A BYTE
          STA  MODE0          WRITE TO VDP
          INY                 PAGE DONE?
          BNE  NEXN           NOT YET..
          INC  NAMPT+1        INCREMENT TABLE POINTER
          DEX                 DONE ALL 3 PAGES?
          BNE  NEXN           NOT YET..
          RTS
*----------------------------------------------------*
*                      DATA TABLES                   *
*----------------------------------------------------*
*
*     INITIALIZATION TABLE
*
ITBL      BYTE >00       R0   EXTERNAL VIDEO OFF
          BYTE >C2       R1   4416,DISABLE INT,GRAPH1,SIZ1,MAG OFF
          BYTE >01       R2   NAME TABLE=>0400
          BYTE >80       R3   COLOR TABLE=>2000
          BYTE >01       R4   PATTERN GENERATOR=>0800
          BYTE >0E       R5   SPRITE NAME=>0700
          BYTE >00       R6   SPRITE PATTERN=>0000
          BYTE >01       R7   BACKGROUND COLOR= BLACK
*
*     NAME TABLE
*
NAMES     HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
          HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
```

```
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
        HEX  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
*
*     COLOR TABLE
*
COLORS  HEX  00     COLOR FOR PATTERNS 00-07
        HEX  00                        08-0F
        HEX  00                        10-17
        HEX  00                        18-1F
        HEX  00                        20-27
        HEX  00                        28-2F
        HEX  00                        30-37
        HEX  00                        38-3F
        HEX  00                        40-47
        HEX  00                        48-4F
        HEX  00                        50-57
        HEX  00                        58-5F
        HEX  00                        60-67
        HEX  00                        68-6F
        HEX  00                        70-77
        HEX  00                        78-7F
        HEX  00                        80-87
        HEX  00                        88-8F
        HEX  00                        90-97
        HEX  00                        98-9F
```

```
          HEX  00                               A0-A7
          HEX  00                               A8-AF
          HEX  00                               B0-B7
          HEX  00                               B8-BF
          HEX  00                               C0-C7
          HEX  00                               C8-CF
          HEX  00                               D0-D7
          HEX  00                               D8-DF
          HEX  00                               E0-E7
          HEX  00                               E8-EF
          HEX  00                               F0-F7
          HEX  00                               F7-FF
*
*    PATTERN TABLE
*
PATTERNS  HEX 00,00,00,00,00,00,00,00   PATTERN 0
          HEX 00,00,00,00,00,00,00,00   PATTERN 1
          HEX 00,00,00,00,00,00,00,00   PATTERN 2
          HEX 00,00,00,00,00,00,00,00   PATTERN 3
          HEX 00,00,00,00,00,00,00,00   PATTERN 4
          HEX 00,00,00,00,00,00,00,00   PATTERN 5
          HEX 00,00,00,00,00,00,00,00   PATTERN 6
          HEX 00,00,00,00,00,00,00,00   PATTERN 7
          HEX 00,00,00,00,00,00,00,00   PATTERN 8
          HEX 00,00,00,00,00,00,00,00   PATTERN 9
          HEX 00,00,00,00,00,00,00,00   PATTERN A
          HEX 00,00,00,00,00,00,00,00   PATTERN B
          HEX 00,00,00,00,00,00,00,00   PATTERN C
          HEX 00,00,00,00,00,00,00,00   PATTERN D
          HEX 00,00,00,00,00,00,00,00   PATTERN E
          HEX 00,00,00,00,00,00,00,00   PATTERN F
          HEX 00,00,00,00,00,00,00,00   PATTERN 10
*
          END
```

```
;*************************************************;
;              DEMONSTRATION SOFTWARE              ;
;                                                  ;
;                 WRITTEN IN 8088                  ;
;                ASSEMBLY LANGUAGE                 ;
;*************************************************;
;
CSEG    SEGMENT
        ASSUME   CS:CSEG
DEMO    PROC     NEAR
;
;*************************************************;
;                                                  ;
;   THE FOLLOWING PROGRAM IS AN OUTLINE FOR CREATING  ;
;   A GRAPHICS I MODE PICTURE. IT INITIALIZES THE     ;
;   VDP REGISTERS, CLEARS ALL OF VIDEO RAM, AND THEN  ;
;   LOADS THE NAME TABLE, COLOR TABLE, AND PATTERN    ;
;   TABLE WHICH FORM THE DISPLAY. FILL IN THE TABLES  ;
;   TO CREATE YOUR OWN DISPLAY.                       ;
;                                                  ;
;*************************************************;
;
;       MEMORY EQUATES
;
MODE0   LABEL    DWORD              Write to VDP Ram.
        DW       0C000H             VDP registers are DWORD+1(MODE1)
        DW       0
;
;       INITIALIZE VDP REGISTERS
;
        LES      DI,MODE0           Get VDP address
        MOV      SI,ITBL            Get address Init table
        MOV      CX,8               Counter (all 8 registers)
        MOV      BH,80H             Starting Register number
INIT:   MOV      AH,SI!             Get byte of data
        MOV      ES:DI+1!,AH        Send data to VDP
        MOV      ES:DI+1!,BH        Send register number to VDP
        INC      SI                 Next table value
        INC      BH                 Next register value
        LOOP     INIT               Loop until all 8 registers done
;
;       CLEAR VDP RAM
;
        MOV      AX,4000H           VDP starting address
        MOV      ES:DI+1!,AL        Set up VDP address register
        MOV      ES:DI+1!,AH
        MOV      CX,AX              Clear all 4000 Hex bytes of Vram
        MOV      AH,0               Data to write
CLRLP:  MOV      ES:DI!,AH          Write a zero to Vram
        LOOP     CLRLP              Loop until done
;
;       LOAD PATTERN TABLE
;
        MOV      SI,PATTB           Address Pattern table
        MOV      CX,800H            Number of bytes to write
        MOV      DX,4800H           Address of pattern table in Vram
        MOV      ES:DI+1!,DL        Setup Vram address
        MOV      ES:DI+1!,DH
```

```
LDPTLP: MOV     AH, SI!             Get data from table
        MOV     ES: DI!,AH          Write to Vram
        INC     SI                  Inc pointer to pattern table data
        LOOP    LDPTLP              Loop until done
;
;       LOAD COLOR TABLE
;
        MOV     SI,CLRTB            Address Color table
        MOV     CX,20H              Length color table
        MOV     DX,6000H            Vram color table address
        MOV     ES: DI+1!,DL        Setup VDP address register
        MOV     ES: DI+1!,DH
LDCLLP: MOV     AH, SI!             Get byte of data
        MOV     ES: DI!,AH          Send to VDP
        INC     SI                  Next
        LOOP    LDCLLP              Loop until done
;
;       LOAD NAME TABLE
;
        MOV     SI,NAMETB           Address Name table
        MOV     CX,300H             Length Name table
        MOV     DX,4400H            Address of Name table in Vram
        MOV     ES: DI+1!,DL        Setup VDP address
        MOV     ES: DI+1!,DH
LDNMLP: MOV     AH, SI!             Get byte of data
        MOV     ES: DI!,AH          Write to Vram
        INC     SI                  Next
        LOOP    LDNMLP              Loop until done
        RET                         Return to caller
;
;*************************************************;
;                                                 ;
;                  VDP DATA TABLES                ;
;                                                 ;
;*************************************************;
;
;  REGISTER INITIALIZATION TABLE
;
ITBL:   DB      000H        R0=GR2 mode
        DB      0C2H        R1=Enable display, size1 sprites, no mag.
        DB      001H        R2=Name table
        DB      080H        R3=Color table
        DB      001H        R4=Pattern table
        DB      006H        R5=Sprite attribute
        DB      000H        R6=Sprite Pattern table
        DB      00FH        R7=Backdrop color= white
;
;  NAME TABLE
;
NAMETB: DB      000H,000H,000H,000H,000H,000H,000H,000H
        DB      000H,000H,000H,000H,000H,000H,000H,000H
        DB      000H,000H,000H,000H,000H,000H,000H,000H
        DB      000H,000H,000H,000H,000H,000H,000H,000H
        DB      000H,000H,000H,000H,000H,000H,000H,000H
        DB      000H,000H,000H,000H,000H,000H,000H,000H
        DB      000H,000H,000H,000H,000H,000H,000H,000H
        DB      000H,000H,000H,000H,000H,000H,000H,000H
        DB      000H,000H,000H,000H,000H,000H,000H,000H
```

```
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
        DB      000H ,000H ,000H ,000H ,000H ,000H ,000H ,000H
;
;       COLOR TABLE
;
CLRTB:  DB      000H            Color for patterns 0-7
        DB      000H                               8-F
        DB      000H                                "
        DB      000H                                "
        DB      000H                                "
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
        DB      000H
```

```
        DB      000H
;
;       PATTERN TABLE
;
PATTB:  DB      000H,000H,000H,000H,000H,000H,000H,000H     Pattern0
        DB      000H,000H,000H,000H,000H,000H,000H,000H            1
        DB      000H,000H,000H,000H,000H,000H,000H,000H            2
        DB      000H,000H,000H,000H,000H,000H,000H,000H            3
        DB      000H,000H,000H,000H,000H,000H,000H,000H            4
        DB      000H,000H,000H,000H,000H,000H,000H,000H            5
        DB      000H,000H,000H,000H,000H,000H,000H,000H            6
        DB      000H,000H,000H,000H,000H,000H,000H,000H            7
        DB      000H,000H,000H,000H,000H,000H,000H,000H            8
        DB      000H,000H,000H,000H,000H,000H,000H,000H            9
        DB      000H,000H,000H,000H,000H,000H,000H,000H            A
        DB      000H,000H,000H,000H,000H,000H,000H,000H            B
        DB      000H,000H,000H,000H,000H,000H,000H,000H            C
        DB      000H,000H,000H,000H,000H,000H,000H,000H            D
DEMO    ENDP
CSEG    ENDS
        END
```

```
*-----------------------------------------------------*
*                DEMONSTRATION SOFTWARE               *
*                                                     *
*                  WRITTEN IN TMS7000                 *
*                  ASSEMBLY LANGUAGE                  *
*-----------------------------------------------------*
*
        IDT  'DEMO'
        TITL '7000-VDP DEMONSTRATION PROGRAM'
*
        OPTION BUNLST,TUNLST,DUNLST
*
*     MAIN PROGRAM WILL START AT >F080
*
        AORG >F080
*
*-----------------------------------------------------*
*                       EQUATES                       *
*-----------------------------------------------------*
*
MODE0     EQU  P128  >0180    MODE LOW SELECTS A READ/WRITE VIDEO RAM
MODE1     EQU  P129  >0181    MODE HIGH SELECTS A READ/WRITE VIDEO REISTER
*
*-----------------------------------------------------*
*                                                     *
*   THE FOLLOWING PROGRAM IS AN OUTLINE FOR CREATING  *
*   A GRAPHICS I MODE PICTURE. IT INITIALIZES THE     *
*   VDP REGISTERS, CLEARS ALL OF VIDEO RAM, AND THEN  *
*   LOADS THE NAME TABLE, COLOR TABLE, AND PATTERN    *
*   TABLE WHICH FORM THE DISPLAY. FILL IN THE TABLES  *
*   TO CREATE YOUR OWN DISPLAY.                       *
*-----------------------------------------------------*
*
*     INITIALIZE VDP REGISTERS
*
START   DINT                   DISABLE INTERRUPTS
        MOV    %7,B            NUMBER OF REGISTERS TO INITIALIZE
INIT    LDA    @ITBL(B)        GET REGISTER VALUE
        MOVP   A,MODE1         SEND REGISTER VALUE TO VDP
        MOV    B,A
        OR     %>80,A          SET MSB OF REGISTER NUMBER
        MOVP   A,MODE1         SEND REGISTER NUMBER TO VPD
        DEC    B               DECREMENT TO GET NEXT REGISTER
        JC     INIT            LOOP UNTIL ALL 8 ARE DONE
*
        CALL   @ZAPRAM         CLEAR ALL VDP RAM
*
*     LOAD GRAPHICS I MODE PICTURE
*
        MOVD  %PATTB,R5       ADDRESS OF PATTERN TABLE
        MOVD  %>800,R3        NO. ENTRIES
        MOVD  %>4800,B        VRAM DESTINATION ADDRESS
        CALL  @MOVE           PERFORM MOVE
*
        MOVD  %CLRTB,R5       ADDRESS OF PATTERN TABLE
        MOVD  %32,R3          NO. ENTRIES
        MOVD  %>6000,B        VRAM DESTINATION ADDRESS
        CALL  @MOVE           PERFORM MOVE
```

```
*
        MOVD  %NAMETB,R5     ADDRESS OF PATTERN TABLE
        MOVD  %768,R3        NO. ENTRIES
        MOVD  %>4400,B       VRAM DESTINATION ADDRESS
        CALL  @MOVE          PERFORM MOVE
*
        RETS
*
*-------------------------------------------------*
*                  SUBROUTINES                    *
*-------------------------------------------------*
*
*     MOVE ROM MEMORY TO VDP RAM
*
*
MOVE    MOVP   B,MODE1      SEND LOW  ADDRESS TO VDP
        MOVP   A,MODE1      SEND HIGH ADDRESS TO VDP
MEMOVE  LDA    *R5          GET DATA POINTED TO BY R5,R4
        MOVP   A,MODE0      SEND TO VDP
        INC    R5           POINT TO NEXT TABLE ADDRESS
        ADC    %0,R4
        DECD   R3           DECREMENT BYTE COUNT
        JNZ    MEMOVE       IF NOT DONE THEN SEND ANOTHER
        RETS                BACK TO CALLING ROUTINE
*
*     CLEAR ALL VDP RAM
*
ZAPRAM  MOVD %>4000,R3      NUMBER OF LOCATIONS TO CLEAR
        MOVP %>00,MODE1     SETUP LOW  ADDRESS BYTE
        MOVP %>40,MODE1     SETUP HIGH ADDRESS BYTE
        CLR  A              VALUE TO SEND TO VDP = 0
ZAP     MOVP A,MODE0        SENT VALUE TO VDP
        DECD R3             DECREMENT COUNTER
        JNZ  ZAP            IF NOT DONE THEN DO AGAIN
        RETS                RETURN TO CALLING ROUTINE
*
*-------------------------------------------------*
*                  DATA TABLES                    *
*-------------------------------------------------*
*
*     INITIALIZATION TABLE
*
ITBL    BYTE >0F       R7   BACKGROUND COLOR= WHITE
        BYTE >00       R6   SPRITE PATTERN=>0000
        BYTE >06       R5   SPRITE NAME=>0300
        BYTE >01       R4   PATTERN GENERATOR=>0800
        BYTE >80       R3   COLOR TABLE=>2000
        BYTE >01       R2   NAME TABLE=>0400
        BYTE >C2       R1   4416,DISABLE INT,GRAPH1,SIZ1,MAG OFF
        BYTE >00       R0   EXTERNAL VIDEO OFF
*
*     NAME TABLE
*
NAMETB  DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
        DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
        DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
        DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
        DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
```

```
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
*
*     COLOR TABLE
*
CLRTB  BYTE >00    COLOR FOR PATTERNS 00-07
       BYTE >00                       08-0F
       BYTE >00                       10-17
       BYTE >00                       18-1F
       BYTE >00                       20-27
       BYTE >00                       28-2F
       BYTE >00                       30-37
       BYTE >00                       38-3F
       BYTE >00                       40-47
       BYTE >00                       48-4F
       BYTE >00                       50-57
       BYTE >00                       58-5F
```

```
        BYTE >00                           60-67
        BYTE >00                           68-6F
        BYTE >00                           70-77
        BYTE >00                           78-7F
        BYTE >00                           80-87
        BYTE >00                           88-8F
        BYTE >00                           90-97
        BYTE >00                           98-9F
        BYTE >00                           A0-A7
        BYTE >00                           A8-AF
        BYTE >00                           B0-B7
        BYTE >00                           B8-BF
        BYTE >00                           C0-C7
        BYTE >00                           C8-CF
        BYTE >00                           D0-D7
        BYTE >00                           D8-DF
        BYTE >00                           E0-E7
        BYTE >00                           E8-EF
        BYTE >00                           F0-F7
        BYTE >00                           F7-FF
*
*      PATTERN TABLE
*
PATTB   DATA >0000,>0000,>0000,>0000       PATTERN 0
        DATA >0000,>0000,>0000,>0000       PATTERN 1
        DATA >0000,>0000,>0000,>0000       PATTERN 2
        DATA >0000,>0000,>0000,>0000       PATTERN 3
        DATA >0000,>0000,>0000,>0000       PATTERN 4
        DATA >0000,>0000,>0000,>0000       PATTERN 5
        DATA >0000,>0000,>0000,>0000       PATTERN 6
        DATA >0000,>0000,>0000,>0000       PATTERN 7
        DATA >0000,>0000,>0000,>0000       PATTERN 8
        DATA >0000,>0000,>0000,>0000       PATTERN 9
        DATA >0000,>0000,>0000,>0000       PATTERN A
        DATA >0000,>0000,>0000,>0000       PATTERN B
        DATA >0000,>0000,>0000,>0000       PATTERN C
        DATA >0000,>0000,>0000,>0000       PATTERN D
        DATA >0000,>0000,>0000,>0000       PATTERN E
        DATA >0000,>0000,>0000,>0000       PATTERN F
        DATA >0000,>0000,>0000,>0000       PATTERN 10
END
```

```
*-------------------------------------------------------*
*              DEMONSTRATION SOFTWARE                   *
*                                                       *
*                WRITTEN IN TMS9995                     *
*                ASSEMBLY LANGUAGE                      *
*-------------------------------------------------------*
*
         IDT   'DEMO'
         OPTION BUNLST,TUNLST,DUNLST
*
*     MAIN PROGRAM WILL START AT >0080
*
         AORG >0080
*
*-------------------------------------------------------*
*                       EQUATES                         *
*-------------------------------------------------------*
*
MODE0     EQU >C000       MODE LOW SELECTS A READ/WRITE VIDEO RAM
MODE1     EQU >C002       MODE HIGH SELECTS A READ/WRITE VIDEO REGITER
*
*-------------------------------------------------------*
*                                                       *
*   THE FOLLOWING PROGRAM IS AN OUTLINE FOR CREATING    *
*   A GRAPHICS I MODE PICTURE. IT INITIALIZES THE       *
*   VDP REGISTERS, CLEARS ALL OF VIDEO RAM, AND THEN    *
*   LOADS THE NAME TABLE, COLOR TABLE, AND PATTERN      *
*   TABLE WHICH FORM THE DISPLAY. FILL IN THE TABLES    *
*   TO CREATE YOUR OWN DISPLAY.                         *
*                                                       *
*-------------------------------------------------------*
*
*     INITIALIZE VDP REGISTERS
*
START    LIMI 0            DISABLE INTERRUPTS
         LWPI >F000        SETUP WORKSPACE AS 9995 INTERNAL RAM
         LI   R1,ITBL      GET INITIALIZATION TABLE START ADDRESS
         LI   R2,8         REGISTER COUNTER
         LI   R3,>8000     1ST REGISTER ADDRESS
*
INIT     MOVB *R1+,@MODE1   SEND DATA TO THE TMS 9118
         MOVB R3,@MODE1     TELL IT WHICH REGISTER TO PUT DATA IN.
         AI   R3,>0100     NEXT REGISTER
         DEC  R2           FINISHED WITH ALL REGISTERS?
         JNE  INIT         NO, THEN CONTINUE
*
         BL   @ZAPRAM      CLEAR ALL VDP RAM TO ZEROS
*
*     LOAD GRAPHICS I MODE PICTURE
*
         LI   R1,PATTB     ADDRESS OF PATTERN TABLE
         LI   R2,>800      NO. ENTRIES
         LI   R3,>4800     VRAM DESTINATION ADDRESS
         BL   @MOVE        PERFORM MOVE
*
         LI   R1,CLRTB     ADDRESS OF COLOR TABLE
         LI   R2,32        NUMBER OF TABLE ENTRIES
         LI   R3,>6000     VRAM DESTINATION ADDRESS
```

```
         BL    @MOVE           PERFORM MOVE
*
         LI    R1,NAMETB       ADDRESS OF NAME TABLE
         LI    R2,768          NUMBER OF NAME TABLE ENTRIES
         LI    R3,>4400        VRAM DESTINATION ADDRESS
         BL    @MOVE           PERFORM MOVE
*
         RT                    RETURN TO CALLER
*
*-----------------------------------------------------*
*                      SUBROUTINES                    *
*-----------------------------------------------------*
*
*      MOVE ROM MEMORY TO VDP RAM
*
MOVE     SWPB R3               SETUP VDP ADDRESS
         MOVB R3,@MODE1
         SWPB R3
         MOVB R3,@MODE1
MEMOVE   MOVB *R1+,@MODE0       GET PATTERN BYTE AND SEND TO VRAM
         DEC  R2               FINISHED?
         JNE  MEMOVE           NO, CONTINUE
         RT                    RETURN TO CALLER
*
*      CLEAR ALL VDP RAM
*
ZAPRAM   LI   R1,>0000         DATA TO WRITE
         LI   R2,>4000         CLEAR ALL 16K
         LI   R3,>4000         VRAM START ADDRESS=0
         SWPB R3               SWAP ADDRESS BYTES
         MOVB R3,@MODE1         SETUP LOW BYTE OF ADDRESS
         SWPB R3               SWAP ADDRESS BYTES
         MOVB R3,@MODE1         SETUP HIGH BYTE OF ADDRESS
ZAP      MOVB R1,@MODE0         MOVE ZEROS INTO VDP RAM
         DEC  R2               HAVE WE DONE ALL 16K BYTES YET?
         JNE  ZAP              NO, THEN CONTINUE
         RT                    RETURN TO CALLER
*
*-----------------------------------------------------*
*                      DATA TABLES                    *
*-----------------------------------------------------*
*
*      INITIALIZATION TABLE
*
ITBL     BYTE >00        R0    EXTERNAL VIDEO OFF
         BYTE >C2        R1    4416,DISABLE INT,GRAPH1,SIZ1,MAG OFF
         BYTE >01        R2    NAME TABLE=>0400
         BYTE >80        R3    COLOR TABLE=>2000
         BYTE >01        R4    PATTERN GENERATOR=>0800
         BYTE >06        R5    SPRITE NAME=>0300
         BYTE >00        R6    SPRITE PATTERN=>0000
         BYTE >0F        R7    BACKGROUND COLOR= WHITE
*
*      NAME TABLE
*
NAMETB   DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
         DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
         DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
```

```
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
       DATA >0000,>0000,>0000,>0000,>0000,>0000,>0000,>0000
*
*      COLOR TABLE
*
CLRTB  BYTE >00    COLOR FOR PATTERNS 00-07
       BYTE >00                       08-0F
       BYTE >00                       10-17
       BYTE >00                       18-1F
       BYTE >00                       20-27
       BYTE >00                       28-2F
       BYTE >00                       30-37
       BYTE >00                       38-3F
       BYTE >00                       40-47
       BYTE >00                       48-4F
```

```
         BYTE >00                              50-57
         BYTE >00                              58-5F
         BYTE >00                              60-67
         BYTE >00                              68-6F
         BYTE >00                              70-77
         BYTE >00                              78-7F
         BYTE >00                              80-87
         BYTE >00                              88-8F
         BYTE >00                              90-97
         BYTE >00                              98-9F
         BYTE >00                              A0-A7
         BYTE >00                              A8-AF
         BYTE >00                              B0-B7
         BYTE >00                              B8-BF
         BYTE >00                              C0-C7
         BYTE >00                              C8-CF
         BYTE >00                              D0-D7
         BYTE >00                              D8-DF
         BYTE >00                              E0-E7
         BYTE >00                              E8-EF
         BYTE >00                              F0-F7
         BYTE >00                              F7-FF
*
*     PATTERN TABLE
*
PATTB    DATA >0000,>0000,>0000,>0000     PATTERN 0
         DATA >0000,>0000,>0000,>0000     PATTERN 1
         DATA >0000,>0000,>0000,>0000     PATTERN 2
         DATA >0000,>0000,>0000,>0000     PATTERN 3
         DATA >0000,>0000,>0000,>0000     PATTERN 4
         DATA >0000,>0000,>0000,>0000     PATTERN 5
         DATA >0000,>0000,>0000,>0000     PATTERN 6
         DATA >0000,>0000,>0000,>0000     PATTERN 7
         DATA >0000,>0000,>0000,>0000     PATTERN 8
         DATA >0000,>0000,>0000,>0000     PATTERN 9
         DATA >0000,>0000,>0000,>0000     PATTERN A
         DATA >0000,>0000,>0000,>0000     PATTERN B
         DATA >0000,>0000,>0000,>0000     PATTERN C
         DATA >0000,>0000,>0000,>0000     PATTERN D
         DATA >0000,>0000,>0000,>0000     PATTERN E
         DATA >0000,>0000,>0000,>0000     PATTERN F
         DATA >0000,>0000,>0000,>0000     PATTERN 10
END
```

```
*-----------------------------------------------------------*
*       (* CALCULATE PIXEL ADDRESS AND DATA *)              *
*        - Written in 9900 assembly language -              *
*                                                           *
*   Inputs: R8 - LSB is X coordinate                        *
*           R9 - LSB is Y Coordinate                        *
*                                                           *
*  Outputs: R1 - MSByte contains pixel data                 *
*           R3 - Vram destination address                   *
*-----------------------------------------------------------*
*
        MOV    R9,R1          Get Y coordinate
        ANDI   R1,>00F8       Mask within 8 bytes
        SLA    R1,5           Multiply by >100
*
        MOV    R8,R3          Get X coordinate
        ANDI   R3,>00F8       Mask off lower bits
        A      R1,R3          Address of 8x8 pattern
*
        MOV    R9,R1          Get Y coordinate
        ANDI   R1,>0007       Get Y offset within 8x8 pattern
        A      R1,R3          Add to pattern address
*
        MOV    R8,R1          Get X coordinate
        ANDI   R1,>0007       Get pixel data
*
        MOVB @BITOFF(R1),R2   Lookup pixel data in table
        RT
*
*     Pixel Data Lookup Table
*
BITOFF  BYTE  >80,>40,>20,>10,>08,>04,>02,>01
        END
```

```
        TITLE 'FONTS'
*
*-----------------------------------------------------*
*           (* DRAW FONT TO BITMAP SCREEN *)          *
*                                                     *
*    This routine takes an 8x8 graphic or character   *
*    font located in system memory and maps it onto   *
*    the bit-mapped display with the upper lefthand   *
*    corner starting at an X,Y origin.                *
*                                                     *
*    Inputs:  R8=LSB is X Coordinate                  *
*             R9=MSB is Y Coordinate                  *
*            R10=Font Address                         *
*   Outputs: None                                     *
*     Kills: R0,R1,R2,R3,R4,R5,R10                    *
*-----------------------------------------------------*
*
*     Find Byte address in Vram
*
DFONT   MOV  R9,R1          Get Y coordinate
        ANDI R1,>00F8       Mask within 8 bytes
        SLA  R1,5           Multiply by >100
        MOV  R8,R3          Get X coordinate
        ANDI R3,>00F8       Mask off lower bits
        A    R1,R3          Address of 8x8 pattern
        MOV  R9,R2          Get Y coordinate
        ANDI R2,>0007       Get Y offset within 8x8 pattern
        A    R2,R3          Add to pattern address
        A    @PTBASE,R3     Add in Vram base address
        LI   R5,>0008
        S    R2,R5          Distance from 8x8 boundry
*
*     Get Byte of font data and justify if neccesary
*
        LI   R4,>0008       Vertical font size counter
GETFNT  CLR  R2
        MOVB *R10+,R2       Get byte of font data
*
        MOV  R8,R1          Get X coordinate
        ANDI R1,>0007       Get pixel start offset
        MOV  R1,R0          Save as Shift data
        JEQ  NOSFT          If 0 then skip Justify
        SRL  R2,R0          Justify it
NOSFT   EQU  $
*
*     Read Modify write Hbyte1
*
        ANDI R3,>3FFF       Setup for read
        SWPB R3             Setup VDP address register
        MOVB R3,@MODE1
        SWPB R3
        MOVB R3,@MODE1
        NOP
        NOP
        NOP
        MOVB @MODE0,R1       Read Vram
```

```
        ORI   R3,>4000        Set VDP write bit
        SWPB  R3              Setup VDP address register
        MOVB  R3,@MODE1
        SWPB  R3
        MOVB  R3,@MODE1
        SOCB  R1,R2           Or in new data
        NOP
        MOVB  R2,@MODE0        Write back to Vram
*
*    Read Modify Write Hbyte2
*
        SWPB  R2
        AI    R3,>0008
        ANDI  R3,>3FFF        Setup for read
        SWPB  R3              Setup VDP address register
        MOVB  R3,@MODE1
        SWPB  R3
        MOVB  R3,@MODE1
        NOP
        NOP
        NOP
        MOVB  @MODE0,R1         Read Vram byte
*
        ORI   R3,>4000        Setup for write operation
        SWPB  R3              Setup VDP address register
        MOVB  R3,@MODE1
        SWPB  R3
        MOVB  R3,@MODE1
        SOCB  R1,R2
        NOP
        MOVB  R2,@MODE0
*
        AI    R3,>FFF8        Restore Horizontal byte origin
        DEC   R5              Did we cross a vertical boundry
        JNE   SKIPL
        AI    R3,>0100        Skip to next pattern block
        ANDI  R3,>FFF8        Mask Offset
        JMP   ENDCHK
*
SKIPL   INC   R3
ENDCHK  DEC   R4
        JNE   GETFNT
        RT
        END
```

```
        TITLE 'LINES'
*
*-----------------------------------------------------*
*                 (* DRAW A LINE *)                   *
*                                                     *
*                                                     *
*     Inputs: X1,Y1 = Start point                     *
*             X2,Y2 = End point                       *
*    Outputs: None                                    *
*      Kills: R0,R1,R2,R6,R7,R8,R9                    *
*      Notes: -BLWP Entry                             *
*             -Calls HPLOT(Plots pixel)               *
*             -This routine expects Graphics II mode  *
*              Name table to be arranged for bit-     *
*              mapped graphics (See INAME routine)    *
*             -Six variable locations must be set up  *
*              these being XDIR,YDIR,XCNT,YCNT,XFRAC  *
*              YFRAC                                  *
*-----------------------------------------------------*
*
DLINE   DATA >F020,DRLNE
DRLNE   MOV  @X2,R1         Get X end
        MOV  @Y2,R2         Get Y end
*
*     Set Horizontal Pos/Neg Direction
*
        S    @X1,R1         Get Horiz. Distance
        JGT  HPOS
        LI   R0,>FFFF       Move negative
        MOV  R0,@XDIR
        JMP  HIDONE
HPOS    CLR  R0             Move positive
        MOV  R0,@XDIR
HIDONE  EQU  $
*
*     Set Vertical Pos/Neg Direction
*
        S    @Y1,R2         Get Vertical Distance
        JGT  VPOS
        LI   R0,>FFFF       Move negative
        MOV  R0,@YDIR
        JMP  VIDONE
VPOS    CLR  R0             Move positive
        MOV  R0,@YDIR
VIDONE  EQU  $
*
*     Set the Horiz/Vertical Counters and Inc Timers
*
STFRAC  ABS  R1
        MOV  R1,@XCNT
        SWPB R1
        MOV  R1,@XFRAC
        ABS  R2
        MOV  R2,@YCNT
        SWPB R2
        MOV  R2,@YFRAC
*
```

```
*       Draw the line
*
LSTART  MOV   @X1,R8        Set Initial X,Y coordinate
        MOV   @Y1,R9
*
        CLR   R6            Init X,Y counters
        CLR   R7
*
LINELP  BL    @HPLOT        Plot first point
*
        A     @XFRAC,R6     Did X overflow?
        JNC   VCHK          No.
        DEC   @XCNT         Are we done?
        JEQ   LNDONE        Yes
        MOV   @XDIR,R0      No, Check direction flag
        JEQ   HMPOS         Move positive
        DEC   R8
        JMP   HMDONE
HMPOS   INC   R8
HMDONE  EQU   $
*
VCHK    A     @YFRAC,R7     Did Y overflow?
        JNC   LINELP        No.
        DEC   @YCNT
        JEQ   LNDONE
        MOV   @YDIR,R0      Check direction flag
        JEQ   VMPOS
        DEC   R9
        JMP   LINELP
VMPOS   INC   R9
        JMP   LINELP
*
LNDONE  RTWP
        END
```

```
        TITLE  'PIXELS'
*
*----------------------------------------------------------*
*                 (* PLOT PIXEL IN VRAM *)                  *
*                                                           *
*    Inputs: R8 - LSB is X coordinate                       *
*            R9 - LSB is Y Coordinate                       *
*   Outputs: R1 - Pixel offset within byte                  *
*            R3 - Vram byte address                         *
*     Kills: R1,R2,R3,R4                                    *
*     Notes: Expects variable PTBASE to hold the            *
*            pattern table Vram starting address            *
*----------------------------------------------------------*
*
HPLOT   EQU   $
        MOV   R9,R1          Get Y coordinate
        ANDI  R1,>00F8       Mask within 8 bytes
        SLA   R1,5           Multiply by >100
*
        MOV   R8,R3          Get X coordinate
        ANDI  R3,>00F8       Mask off lower bits
        A     R1,R3          Address of 8x8 pattern
*
        MOV   R9,R1          Get Y coordinate
        ANDI  R1,>0007       Get Y offset within 8x8 pattern
        A     R1,R3          Add to pattern address
*
        MOV   R8,R1          Get X coordinate
        ANDI  R1,>0007       Get pixel data
*
        A     @PTBASE,R3     Add in Vram base address
        SWPB  R3             Setup VDP address register
        MOVB  R3,@VREG
        SWPB  R3
        MOVB  R3,@VREG
        NOP                  Don't write too fast.....
        NOP
        NOP
*
        MOVB  @VRAM,R4       Read Vram byte
*
        ORI   R3,>4000       Set VDP write bit
        SWPB  R3             Setup VDP address register
        MOVB  R3,@VREG
        SWPB  R3
        MOVB  R3,@VREG
*
        MOVB  @BITOFF(R1),R2  Lookup pixel data in table
*
        SOCB  R4,R2          Or it in with what's already there
        MOVB  R2,@VRAM
        RT
BITOFF  BYTE  >80,>40,>20,>10,>08,>04,>02,>01
        END
```

```
*-----------------------------------------------*
*                                               *
*          (* VDP SUBROUTINES *)                *
*                                               *
*-----------------------------------------------*
*
       TITLE  'VDPSUBS'
*
*-----------------------------------------------*
*          (* INITIALIZE NAME TABLE *)          *
*                                               *
*    This routine initializes the Name Table for *
*    bit-map graphics. Each entry points to a   *
*    unique pattern.                            *
*                                               *
*    Inputs: None                               *
*   Outputs: None                               *
*     Kills: R1,R2,R3,R4,R5                     *
*     Notes: The Name table start address and length *
*            must be set up in NTBASE, and NTLEN *
*-----------------------------------------------*
*
INAME   CLR  R3
        MOV  @NTLEN,R4      Get Name table length
        MOV  @NTBASE,R5     Get Name table Vram address
        ORI  R5,>4000       Set VDP write bit
        SWPB R5
        MOVB R5,@VREG       Set up Vram address
        SWPB R5
        MOVB R5,@VREG
        SWPB R5
LOOP2   MOVB R3,@VRAM       Write data to Vram
        AI   R3,>0100       Inc data
        DEC  R4
        JNE  LOOP2
        RT
*
*-----------------------------------------------*
*          (* CLEAR/FILL VDP RAM *)             *
*                                               *
*    Inputs: R1= fill data                      *
*            R3= start address                  *
*            R2= bytes to fill                  *
*-----------------------------------------------*
*
CLVRAM  LI   R3,>0000       Vram start address
        LI   R2,>4000       Clear all 64K
        LI   R1,>0000       Fill data
FILRAM  ORI  R3,>4000       Set write bit
        SWPB R3             Setup Vram address
        MOVB R3,@VREG
        SWPB R3
        MOVB R3,@VREG
        SWPB R3
ZAPRAM  MOVB R1,@VRAM       Fill Vram
        DEC  R2
        JNE  ZAPRAM
        RT
```

```
*
*-----------------------------------------------------*
*       (* BLOCK MOVE MEMORY: SYSTEM TO VDP *)        *
*                                                     *
*    Inputs: R1=Source address                        *
*            R3=Target address                        *
*            R2=Number bytes                          *
*-----------------------------------------------------*
*
SVMOVE   ORI  R3,>4000      Set write bit
         SWPB R3            Setup Vram address
         MOVB R3,@VREG
         SWPB R3
         MOVB R3,@VREG
MEMOVE   MOVB *R1+,@VRAM    Get byte and send to Vram
         DEC  R2
         JNE  MEMOVE
         RT
*
         END
```

TEXAS INSTRUMENTS

August 1984
Revision*
1602232-9701
Printed in U.S.A.

SPPU004