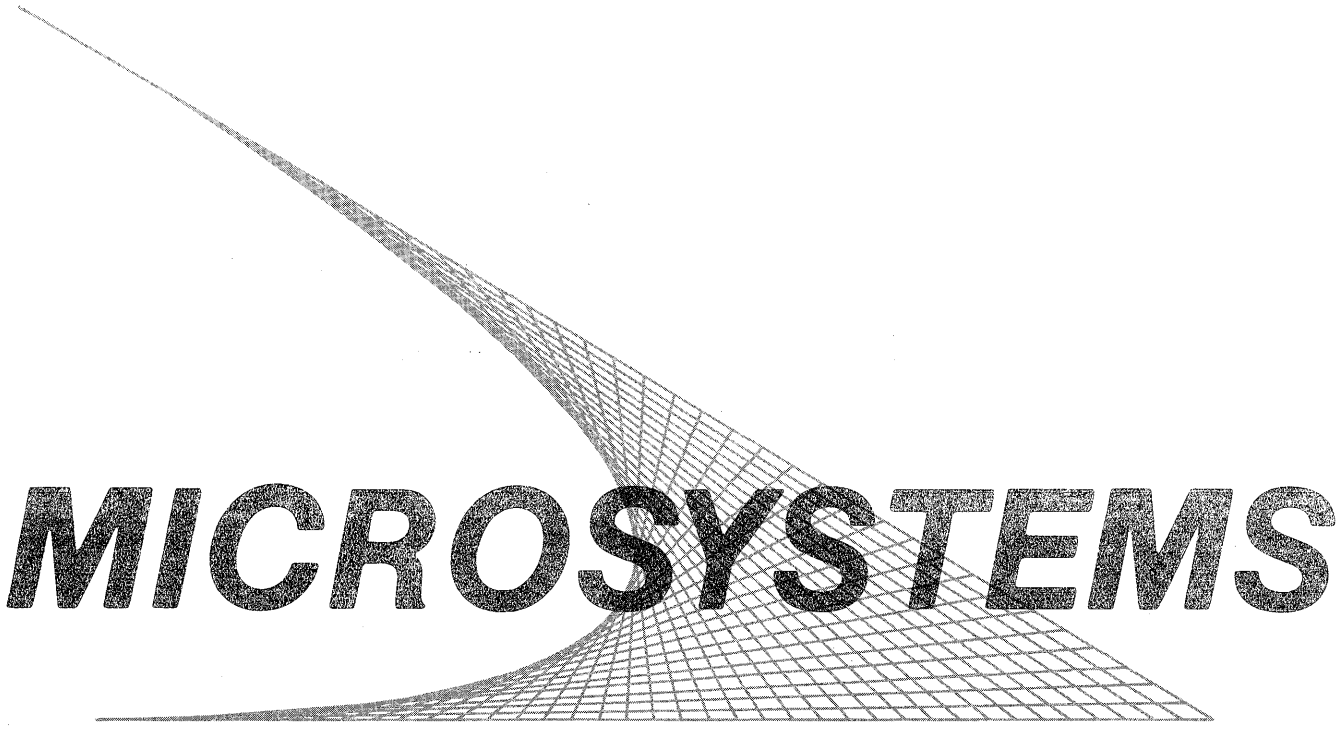
MOTOROLA

M68KVSREF/D1

VME/10
Microcomputer System
Reference Manual



QUALITY • PEOPLE • PERFORMANCE

M68KVSREF/D1

FEBRUARY 1984

VME/10

MICROCOMPUTER SYSTEM

REFERENCE MANUAL

First Edition

PREFACE

An asterisk (*) following the signal name for signals which are level significant denotes that the signal is true or valid when the signal is low.

An asterisk (*) following the signal name for signals which are edge significant denotes that the actions initiated by that signal occur on a high to low transition.

"Set" terminology referenced throughout this manual denotes placing (writing) a logical one (high state) into a device.

"Clear" terminology referenced throughout this manual denotes placing (writing) a logical zero (low state) into a device.

All hexadecimal references throughout this manual are preceeded by a dollar sign ($).

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

REFERENCE MANUAL PHILOSOPHY

1.1 INTRODUCTION

This reference manual provides both hardware and software information for the VME/10 Microcomputer System (hereafter referred to as VME/10). Information in this manual will permit the user to implement software to reconfigure (customize) the VME/10 operation to a specific application or to perform the VME/10 graphic capabilities.

The VME/10 contains a System Control Module (SCM) which is installed in the control unit chassis. The SCM provides the central intelligence for the VME/10. To understand the VME/10 operating environment, material in this manual is organized as follows:

a. System information

b. Graphics generation

b. Character display generation

CHAPTER 2

SYSTEM INFORMATION

2.1 INTRODUCTION

This chapter provides system information that permits the user to implement software to reconfigure (customize) the VME/10 operation to a specific application or to perform the VME/10 graphic capabilities. Information provided in this chapter is as follows:

a. System memory maps

b. Control and status registers

c. VMEbus interrupts

d. SCM MPU interrupts.

2.2 SYSTEM MEMORY MAPS

The system memory maps (Figures 2-1 through 2-4) identify all areas of memory that are reserved for system use, as well as areas of memory that are available for use by the user.

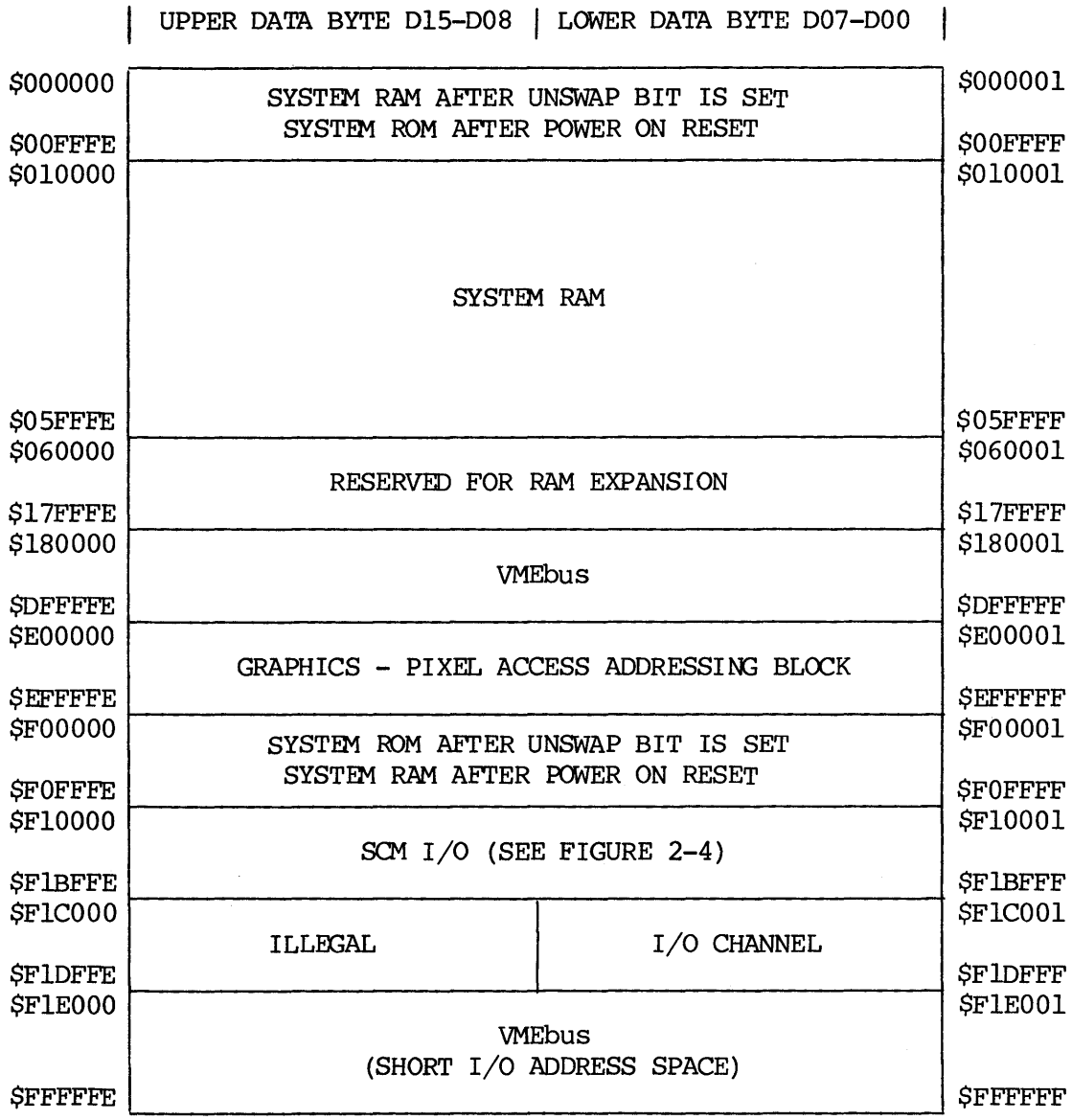


FIGURE 2-1. System Memory Map, No Graphics

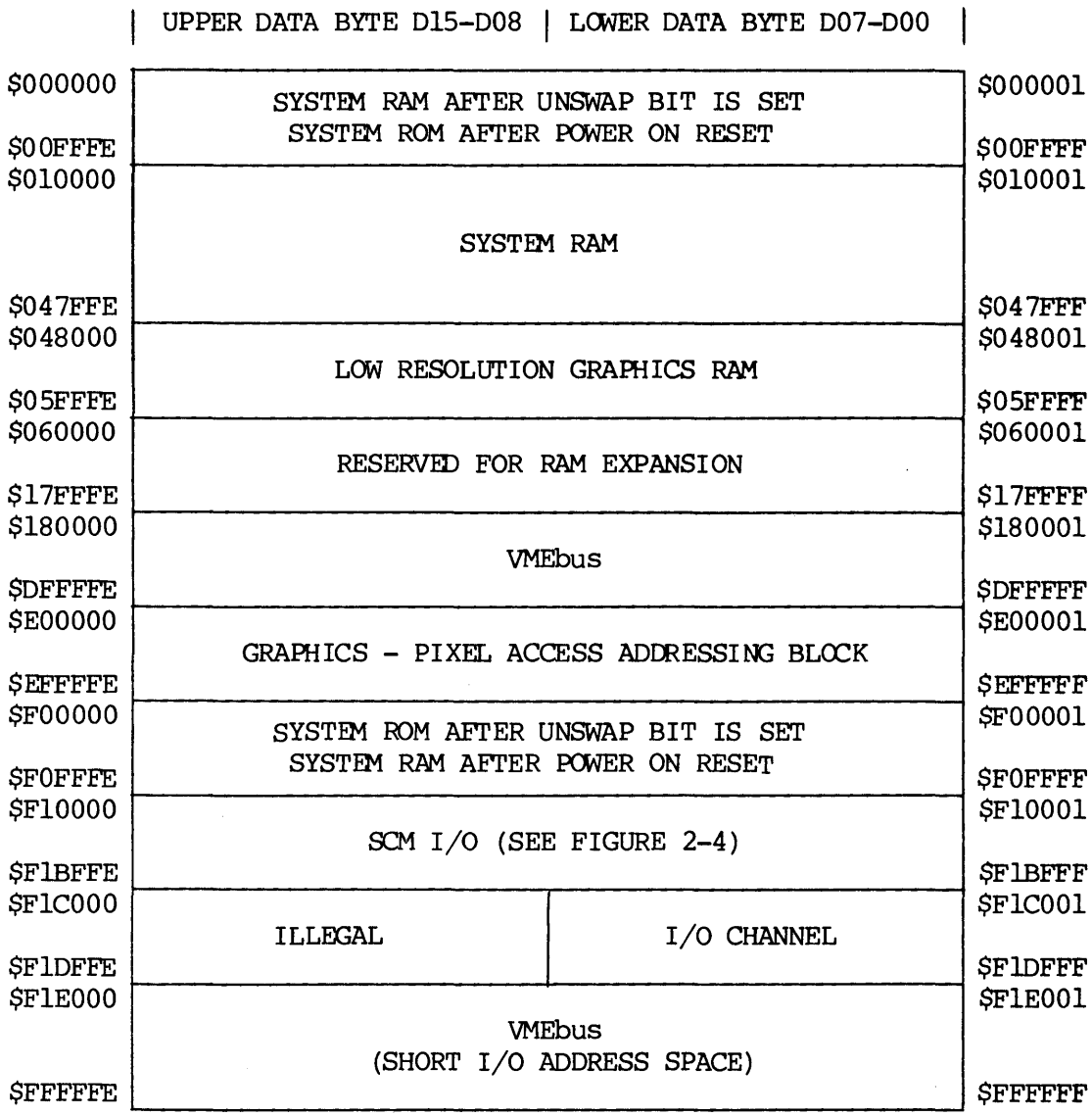


FIGURE 2-2. System Memory Map, Low Resolution Graphics

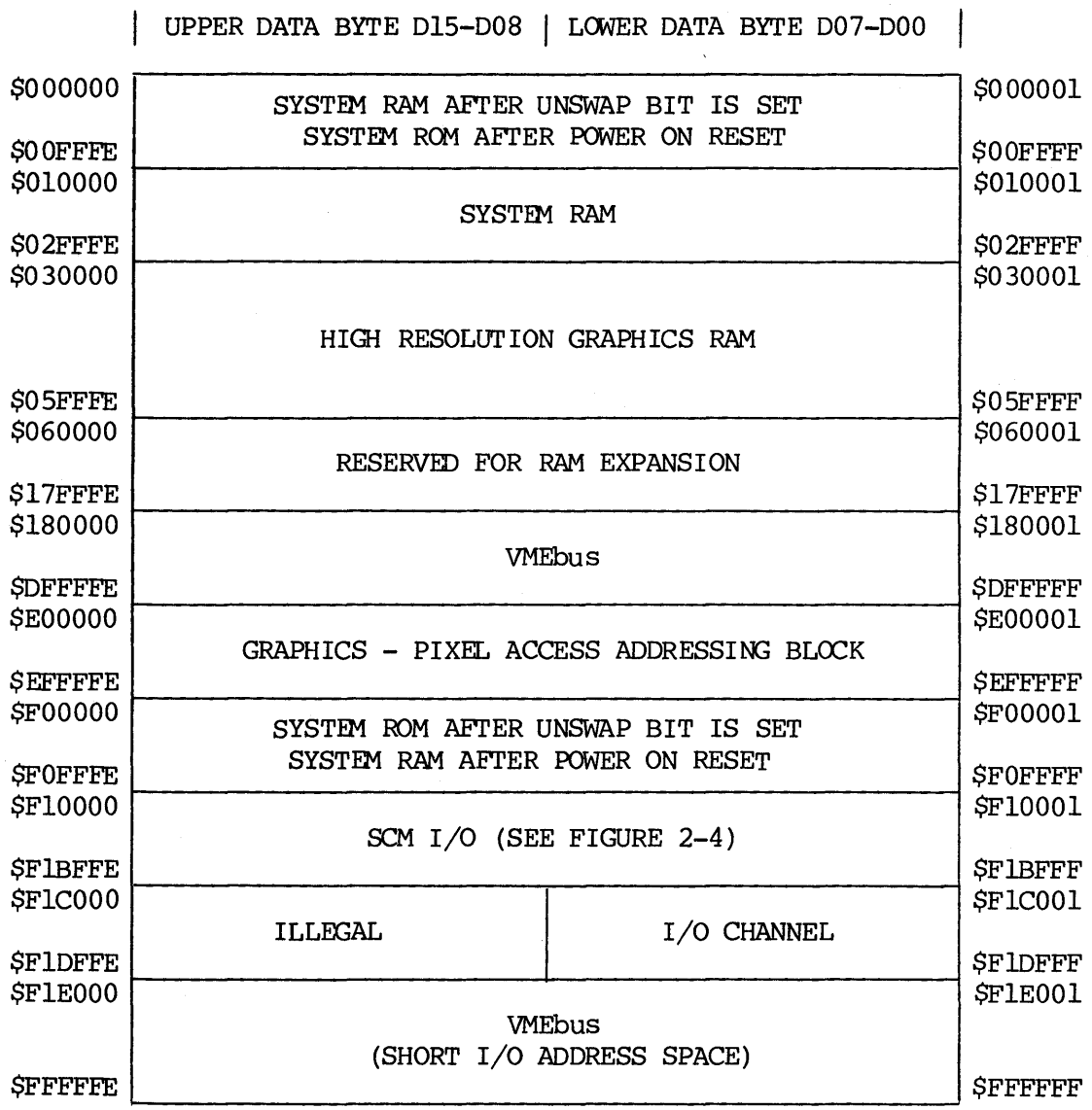


FIGURE 2-3. System Memory Map, High Resolution Graphics

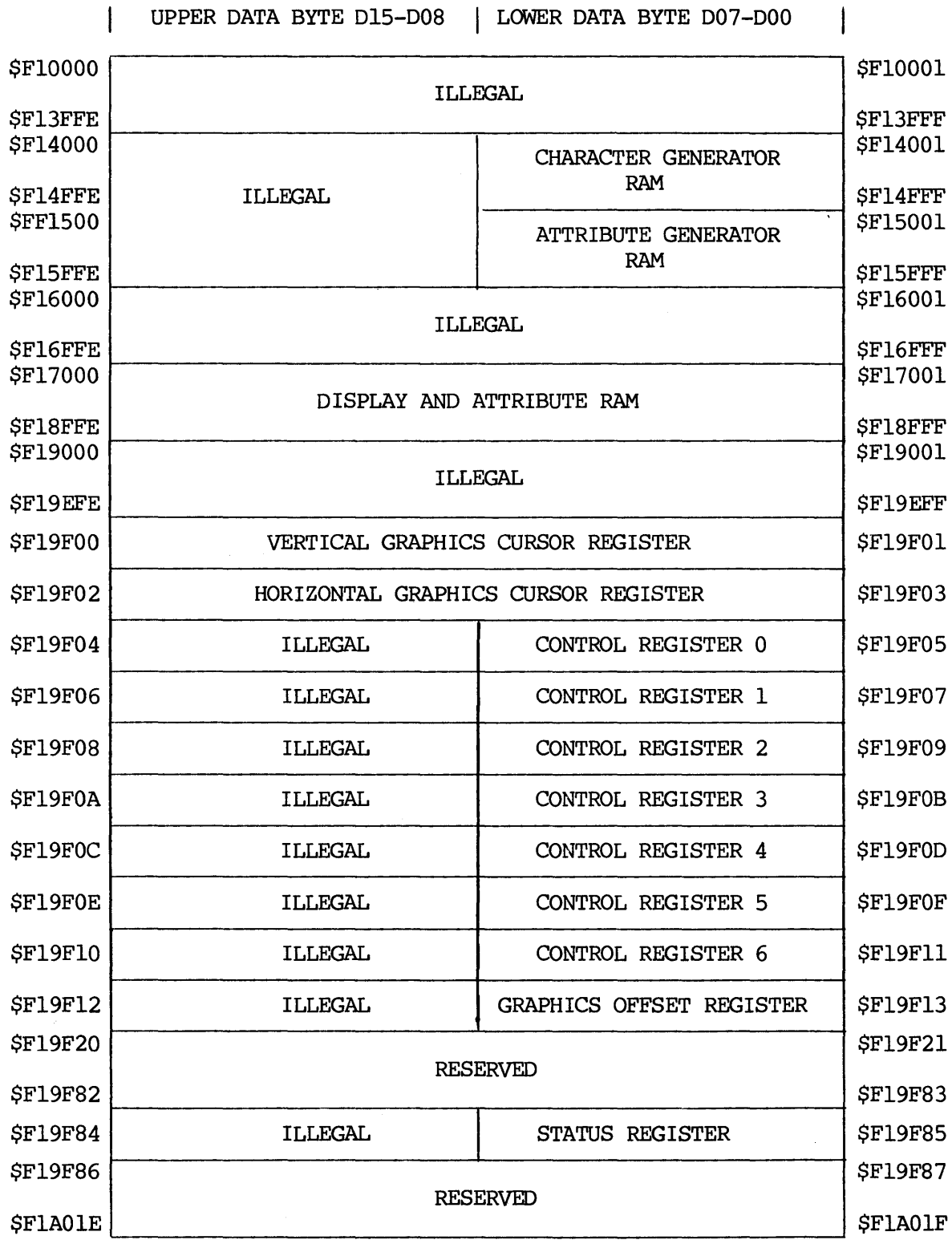


FIGURE 2-4. SCM I/O Memory Map (Sheet 1 of 2)

	UPPER DATA BYTE D15-D08	LOWER DATA BYTE D07-D00	
$F1A020	ILLEGAL	MC68A45 ADDRESS REGISTER	$F1A021
$F1A022		MC68A45 INTERNAL REGISTER FILE	$F1A023
$F1A024	ILLEGAL		$F1A025
$F1A02E			$F1A02F
$F1A030	ILLEGAL	MC2661 TX/RX DATA REGISTERS	$F1A031
$F1A032		MC2661 STATUS REGISTER	$F1A033
$F1A034		MC2661 MODE 1 AND MODE 2 REG.	$F1A035
$F1A036		MC2661 COMMAND REGISTER	$F1A037
$F1A038	ILLEGAL		$F1A039
$F1A07E			$F1A07F
$F1A080		MC146818 SECONDS REGISTER	$F1A081
$F1A082		MC146818 SECONDS ALARM REG.	$F1A083
$F1A084		MC146818 MINUTES REGISTER	$F1A085
$F1A086		MC146818 MINUTES ALARM REG.	$F1A087
$F1A088		MC146818 HOURS REGISTER	$F1A089
$F1A08A		MC146818 HOURS ALARM REGISTER	$F1A08B
$F1A08C		MC146818 DAY OF THE WEEK REG.	$F1A08D
$F1A08E		MC146818 DAY OF THE MONTH REG.	$F1A08F
$F1A090		MC146818 MONTH REGISTER	$F1A091
$F1A092		MC146818 YEAR REGISTER	$F1A093
$F1A094		MC146818 REGISTER A	$F1A095
$F1A096		MC146818 REGISTER B	$F1A097
$F1A098		MC146818 REGISTER C	$F1A099
$F1A09A		MC146818 REGISTER D	$F1A09B
$F1A09C	ILLEGAL	BATTERY BACKED UP RAM	$F1A09D
$F1A0FE		TIME-OF-DAY CLOCK (MC146818)	$F1A0FF
$F1A100	ILLEGAL		$F1A101
$F1A7FE			$F1A7FF
$F1A800	DMA/MMU		$F1A801
$F1AFFE			$F1AFFF
$F1B000	ILLEGAL		$F1B001
$F1BFFE			$F1BFFF

FIGURE 2-4. SCM I/O Memory Map (Sheet 2 of 2)

2.3 CONTROL AND STATUS REGISTERS

The SCM has seven control registers and one status register. Individual address locations of these registers are listed in the memory maps. Control registers 0 and 2 through 6 are cleared by any of the reset conditions occurring. Control register 1 is cleared only by the power-on-reset condition occurring. All control registers are writable by the MPU in both supervisory and user states.

NOTE

In VME/10's manufactured prior to 2/15/84 - all control registers are writable by the MPU in the supervisory state; only control registers 0 and 1 are also writable in the user state. Writing to control register 1 through 6 in the user state will cause the MPU readable image to change, but not the actual control register.

All control registers are readable by the MPU in any state. However, the data read is not reliable unless each control register has been written to by the MPU at least once since the last reset condition occurred. Bit definitions of the control registers are as follows:

2.3.1 Control Register 0 (Location Address $F19F05)

7	6	5	4	3	2	1	0
CDIS3	CDIS2	CDIS1	CURBK	DUTYCYCLE	IVS	TIMIMSK*	DMAIMSK*

CDIS3-CDIS1 — Character Disable - Used to disable a color bank from being displayed to the monitor (this affects character mode only). When set, CDIS1 through CDIS3 disables colors one through three, respectively. When cleared, CDIS1 through CDIS3 enables colors one through three, respectively.

CURBK — Cursor Blink - When set, causes character cursor to blink on and off. When cleared, CURBK has no effect on character cursor.

DUTYCYCLE — Duty Cycle - When set, corrects BX syndrome by not displaying every other dot on each line. This prevents horizontal lines, such as those in the uppercase letter B, from standing out more than nonhorizontal lines such as those in the letter X. When cleared, DUTYCYCLE has no effect on display.

IVS — Invert Video Screen - When set, video inversion is performed. When IVS is cleared, all characters are normal.

TIMIMSK* — Timer Interrupt Mask - When cleared, inhibits interrupts caused by the real-time clock (MC146818) low IRQ* signal. When set, TIMIMSK* performs no masking function.

DMAIMSK* — Direct Memory Access Interrupt Mask - When cleared, inhibits interrupts caused by the low DMAIRQ* signal. When set, DMAIMSK* performs no masking function.

2.3.2 Control Register 1 (Location Address $F19F07)

7	6	5	4	3	2	1	0
R	S1	S0	HIGH RES	GRE3	GRE2	GRE1	UNSWAP

R — Reserved for future enhancements. Must be kept cleared at all times.

S1,S0 — Select - Selects one of four optional character cursors which are user-definable.

HIGH RES — High Resolution - Affects SCM system RAM mapping.

GRE3,GRE2,GRE1 — Graphic Enable - Enables and disables the display of individual graphics memory banks. When set, enables a bank; when cleared, disables a bank. GRE1 controls bank 1 (red/low intensity), GRE2 controls bank 2 (blue/medium intensity), and GRE3 controls bank 3 (green/high intensity). When all three bits are cleared, no graphics are displayed; when all three bits are set, graphics of all colors/intensities are displayed. It should be noted that these bits do not affect the user's ability to read/write to the graphic banks.

UNSWAP — Unswap - When a power-on-reset (or chassis reset and abort reset) condition occurs, SCM memory map is swapped so that ROM appears at locations $000000-$007FFF. The system RAM which would normally appear at those locations ($000000-$007FFF) appears where ROM would normally appear (locations $F00000-$F0FFFF).

These sections of RAM and ROM may be restored to normal positions by setting the UNSWAP bit. After this action, UNSWAP bit has no affect on the memory map. Clearing the UNSWAP bit again does not cause RAM and ROM to swap normal positions in the memory map. The only conditions that swap RAM and ROM out of the normal positions are the reset conditions described above.

2.3.3 Control Register 2 (Location Address $F19F09)

7	6	5	4	3	2	1	0
RXRDYMSK*	SYSFMSK*	WPTCT*	KBDRST*	VMEAVMSK*	BCLRMSK*	TXRDYMSK*	MMUIMSK*

RXRDYMSK* Receiver Ready Mask - When cleared, inhibits interrupts caused by the EPCI low RXRDY* signal. When set, RXRDYMSK* performs no masking function.

SYSFMSK* System Fail Mask - When cleared, inhibits interrupts caused by the VMEbus low SYSFAIL signal. When set, SYSFMSK* performs no masking function.

WPTCT* Write Protect - When cleared, disallows all write operations to SCM RAM by other VMEbus devices. When set, WPTCT* allows these write operations.

KBDRST* Keyboard Reset - When cleared, sends a reset signal from the keyboard interface, and continually resets the MC2661. When set, KBDRST* performs no function.

VMEAVMSK* VMEbus Available Mask - When cleared, inhibits interrupts caused by the VMEbus becoming available.

BCLRMSK* Bus Clear Mask - When cleared, inhibits interrupts caused by the VMEbus low BCLR* signal when the SCM requester is holding the VMEbus in the release never mode. When set, BCLRMSK* performs no masking function.

TXRDYMSK* Transmit Ready Mask - When cleared, inhibits interrupts caused by the EPCI low TXRDY* signal. When set, TXRDYMSK* performs no masking function.

MMUIMSK* Memory Management Unit Interrupt Mask - When cleared, inhibits interrupts caused by the low MMUIRQ* signal. When set, MMUIMSK* performs no masking function.

2.3.4 Control Register 3 (Location Address $F19F0B)

7	6	5	4	3	2	1	0
IRQ7MSK*	IRQ6MSK*	IRQ5MSK*	IRQ4MSK*	IRQ3MSK*	IRQ2MSK*	IRQ1MSK*	VBIAMSK*

IRQ7MSK* Interrupt Request 7 Mask - When cleared, inhibits SCM MPU interrupts caused by VMEbus low IRQ7* signal. When set, IRQ7MSK* performs no masking function.

IRQ6MSK* Interrupt Request 6 Mask - When cleared, inhibits SCM MPU interrupts caused by VMEbus low IRQ6* signal. When set, IRQ6MSK* performs no masking function.

IRQ5MSK* Interrupt Request 5 Mask - When cleared, inhibits SCM MPU interrupts caused by VMEbus low IRQ5* signal. When set, IRQ5MSK* performs no masking function.

IRQ4MSK* Interrupt Request 4 Mask - When cleared, inhibits SCM MPU interrupts caused by VMEbus low IRQ4* signal. When set, IRQ4MSK* performs no masking function.

IRQ3MSK* Interrupt Request 3 Mask - When cleared, inhibits SCM MPU interrupts caused by VMEbus low IRQ3* signal. When set, IRQ3MSK* performs no masking function.

IRQ2MSK* Interrupt Request 2 Mask - When cleared, inhibits SCM MPU interrupts caused by VMEbus low IRQ2* signal. When set, IRQ2MSK* performs no masking function.

IRQ1MSK* Interrupt Request 1 Mask - When cleared, inhibits SCM MPU interrupts caused by VMEbus low IRQ1* signal. When set, IRQ1MSK* performs no masking function.

VBIAMSK* VMEbus Interrupt Acknowledge Mask - When cleared, inhibits SCM MPU interrupts caused by an interrupt acknowledge cycle having occurred for the interrupt request initiated by the SCM interrupter.

2.3.5 Control Register 4 (Location Address $F19F0D)

7	6	5	4	3	2	1	0
IDC7	IDC6	IDC5	IDC4	IDC3	IDC2	IDC1	IDC0

This register is the vector register. During a VMEbus interrupt acknowledge cycle, if the SCM initiates the interrupt request that is acknowledged, contents of this register (Identification Codes (IDC) bits 0 through 7) are placed on the VMEbus data lines as follows:

IDC7 - D07

IDC6 - D06

IDC5 - D05

IDC4 - D04

IDC3 - D03

IDC2 - D02

IDC1 - D01

IDC0 - D00

2.3.6 Control Register 5 (Location Address $F19F0F)

7	6	5	4	3	2	1	0
BRDFAIL*	AMA	VMETOEN	LTOEN	BRC1	BRC0	BRL1*	BRL0*

BRDFAIL* — Board Fail - When cleared, causes the VMEbus low SYSFAIL* signal, which indicates a board failure. When set, BRDFAIL* does not drive the SYSFAIL* signal line low.

AMA — Address Modifier A - Alters the way address modifier lines are driven by the SCM during VMEbus access. The AMA effect on the address lines is programmable in PROM.

VMETOEN — VME Time-out Enable - When set, enables VMEbus time-out circuitry to operate (causes low BERR* if DS0* or DS1* is low for 64 microseconds or longer). When cleared, VMETOEN disables VMEbus time-out circuitry.

LTOEN — Local Time-out Enable - When set, enables local resource time-out circuitry to operate. (If [UDS*] or [LDS*] is low for 64 microseconds or longer, LTOEN causes low MPU [BERR*] signal. When cleared, LTOEN disables VMEbus time-out circuitry.

BRC1,BRC0 — Bus Request Clear - Control the requester operating mode. Bit to mode correspondence is as follows:

BRC1	BRC0	MODE
0	0	Release on request
0	1	Release on bus clear
1	0	Release when done
1	1	Release never

BRL1*,BRL0* — Bus Request Level - Control the level at which the requester operates. This level should be set one time only, immediately after a reset condition.

2.3.7 Control Register 6 (Location Address $F19F11)

7	6	5	4	3	2	1	0
IMASK*	INT4MSK*	INT3MSK*	INT2MSK*	INT1MSK*	IL2	IL1	IL0

IMASK* Interrupt Mask - When cleared, inhibits all SCM MPU interrupts under all conditions. When set, IMASK* masks no interrupts.

INT4MSK* Interrupt 4 Mask - When cleared, inhibits SCM MPU interrupts caused by the I/O Channel low INT4* signal. When set, INT4MSK* provides no masking function.

INT3MSK* Interrupt 3 Mask - When cleared, inhibits SCM MPU interrupts caused by the I/O Channel low INT3* signal. When set, INT3MSK* provides no masking function.

INT2MSK* Interrupt 2 Mask - When cleared, inhibits SCM MPU interrupts caused by the I/O Channel low INT2* signal. When set, INT2MSK* provides no masking function.

INT1MSK* Interrupt 1 Mask - When cleared, inhibits SCM MPU interrupts caused by the I/O Channel low INT1* signal. When set, INT1MSK* provides no masking function.

IL2,IL1,IL0 Interrupt Level - Generate VMEbus interrupts. For further details, see the interrupter section.

2.3.8 Status Register (Location Address $F19F85)

The status register monitors several signal lines on the SCM. Bit definitions of the status register are as follows:

7	6	5	4	3	2	1	0
SWITCH2	SWITCH1	SWITCH0	KYBDLOCK*	IOCHEN	SYSFAIL	VBIACK*	VMEAV

SWITCH2 — Switch 2 - Factory-configured to a set state.

SWITCH1 — Switch 1 - Factory-configured to a set state.

SWITCH0 — Switch 0 - Factory-configured to a set state.

KYBDLOCK* — Keyboard Lock - When cleared, KYBDLOCK* switch is in the lock position. Software should respond accordingly to this condition. When set, KYBDLOCK* switch is in the unlock position.

IOCHEN — I/O Channel Enable - Factory-configured to a set state.

SYSFAIL — System Fail - When set, VMEbus SYSFAIL* signal line is driven low. When cleared, SYSFAIL* signal line is not driven low.

VBIACK* — VMEbus Interrupt Acknowledge - When cleared, indicates that the interrupt generated by the SCM interrupter has been acknowledged. When set, indicates that either the SCM interrupter is not generating a VMEbus interrupt or that the generated VMEbus interrupt has not been acknowledged.

VMEAV — VMEbus Available - When cleared, indicates that the SCM does not have VMEbus mastership; when set, indicates that SCM does have VMEbus mastership.

2.4 VMEbus INTERRUPTS

The SCM has an interrupter circuit which is capable of generating VMEbus interrupts. The interrupt VMEbus level and the status ID byte during the interrupt acknowledge cycle are both software programmable. To use the interrupter circuit to interrupt the VMEbus, the following sequence is required:

a. Ensure that control register 6 interrupt bits (bits 0-2) are cleared.

b. Initialize status ID byte (control register 4) to the desired value. The VMEbus interrupt handler normally shifts the status ID byte left twice and uses the result as the address in its exception table for handling the VMEbus interrupt.

c. Set interrupt bits (bits 0-2) to the desired interrupt level. This causes the appropriate IRQ to be generated on the VMEbus. The bit level to interrupt level correspondence is as follows:

BIT 2	BIT 1	BIT 0	IRQ
0	0	0	NONE
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

d. Wait for the VMEbus interrupt acknowledged bit (status register bit 1) to be cleared, indicating that the interrupt has been acknowledged.

e. Clear control register 6 interrupt bits (bits 0-2).

NOTE

It is possible to set up the interrupt acknowledged condition to cause a level 1 interrupt to the MPU. If this option is used, it is important to account for the fact that the VMEAV* interrupt has the same level and shares the same exception table location as does [VBIACK*].

2.5 SCM MPU INTERRUPTS

There are 22 sources of interrupts on the SCM. Each one is capable of interrupting the MPU on one of seven levels (1-7). All of the interrupt sources have an assigned distinct priority. For example, if three interrupt sources occur on the same level at the same time, they are serviced in the order of priority. The interrupt sources, levels, and priorities are as follows:

	PRIORITY WITHIN LEVEL		
LEVEL	LOWEST	MEDIUM	HIGHEST
7	IRQ7* (from VMEbus)	ACFAIL*	Software abort
6	IRQ6* (from VMEbus)	I/O Channel INT4*	System fail
5	IRQ5* (from VMEbus)	I/O Channel INT3*	Time-of-day interrupt
4	IRQ4* (from VMEbus)	I/O Channel INT2*	MMU interrupt
3	IRQ3* (from VMEbus)	RXRDY* interrupt	TXRDY* interrupt
2	IRQ2* (from VMEbus)	I/O Channel INT1*	DMA interrupt
1	IRQ1* (from VMEbus)	Bus clear interrupt	VMEbus available (or VMEbus interrupt acknowledged)

Each interrupt source (except for those from the VMEbus) is serviced through a different vector in the MC68010 MPU exception table. The interrupt source to exception table correspondence is shown in Table 2-1.

There are three methods by which interrupt sources may be masked. The first method is via control register 6 bit 7 (IMASK*). When this bit is cleared, it masks all interrupts; when set, it does not mask any interrupt. From a reset condition, this bit comes up cleared, masking all interrupts. Another method of masking interrupts is that of using the mask bit associated with each interrupt. The interrupts and corresponding mask bits are also listed in Table 2-1. Each of these bit masks its corresponding interrupt when it is cleared but does not when it is set. All of these mask bits come up masking at reset time. The third method of masking interrupts is that of using the MPU status register internal mask bits (see MC68010 data sheet for details).

TABLE 2-1. Interrupt Sources

INTERRUPT SOURCE	CORRESPONDING MASK BIT: CONTROL REGISTER #	CORRESPONDING MASK BIT: BIT NUMBER	EXCEPTION TABLE ADDRESS
IRQ1*	3	1	Vector passed by interrupting board shifted left twice
Bus clear interrupt	2	2	$100
VMEbus available	2	3	$120
Interrupt acknowledged	3	0	$120
IRQ2*	3	2	Same as IRQ1*
I/O Channel INT1*	6	3	$104
DMA interrupt	0	0	$124
IRQ3*	3	3	Same as IRQ1*
RXRDY* interrupt	2	7	$108
TXRDY* interrupt	2	1	$128
IRQ4*	3	4	Same as IRQ1*
I/O Channel INT2*	6	4	$10C
MMU interrupt	2	0	$12C
IRQ5*	3	5	Same as IRQ1*
I/O Channel INT3*	6	5	$110
Time-of-day interrupt	0	1	$130
IRQ6*	3	6	Same as IRQ1*
I/O Channel INT4*	6	6	$114
System fail	2	6	$134
IRQ7*	3	7	Same as IRQ1*
ACFAIL*	No mask exists for this interrupt		$118
Software abort	No mask exists for this interrupt		$138

CHAPTER 3

GRAPHICS GENERATION

3.1 INTRODUCTION

This chapter describes the VME/10 graphic capabilities. SCM graphics hardware description is first presented, followed by a software description required to drive the graphics hardware. Software application programs are also provided.

3.2 HARDWARE DESCRIPTION

This section describes the applicable hardware circuits that control the graphics generation capabilities of the VME/10. These circuits are as follows:

a. Graphics memory

b. Graphics control registers

c. Graphics cursor register

d. CRT Controller (CRTC)

e. Control register 0

f. Control register 1

g. Graphics offset register

3.2.1 Graphics Memory

The VME/10 implements bit-mapped raster graphics using three bit planes. This means that the display monitor is organized as a matrix of dots called pixels. The VME/10 supports a low-resolution mode (800 horizontal pixels x 300 vertical pixels) and a high-resolution mode (800 horizontal pixels x 600 vertical pixels). Graphical images that appear on the monitor are the result of directly mapping bits in system RAM to pixels on the dislay. For this purpose, there are three bit planes (banks) of memory, each of which contains one bit for each pixel. Each pixel is the result of combining three bits -- one from each of the three memory banks -- which allows for a total of eight values for each pixel on the display.

In color systems, each bank represents one of the primary colors -- red, green, or blue. Therefore, a pixel with corresponding bits set in the red and blue banks appears as magenta, while a pixel with corresponding bits set in all three color banks appears as white.

In monochrome systems, each bank represents an intensity level, which provides an 8-level gray scale from black (no banks enabled) to brightest (all banks enabled).

Table 3-1 lists the colors/intensities for each color/intensity bank and the results of combining banks.

TABLE 3-1. Color/Intensity

	COLOR MONITOR		MONOCHROME MONITOR (SEE NOTE)	
BANK(S)	PRIMARY COLOR(S)	RESULTING COLOR	GRAY SCALE LEVEL(S)	RESULTING GRAY SCALE LEVEL
None	None	Black	0	0
1	Red	Red	1	1
2	Blue	Blue	2	2
1,2	Red, Blue	Magenta	1,2	3
3	Green	Green	4	4
1,3	Red, Green	Yellow	1,4	5
2,3	Blue, Green	Cyan	2,4	6
1,2,3	Red, Blue, Green	White	1,2,4	7

NOTE: Gray scale levels are expressed as an integer from 0 (black) to 7 (brightest), inclusive.

Each color/intensity bank is arranged such that the first byte in a bank corresponds to the left-most eight pixels on the top row of pixels on the display, and the last byte in a bank corresponds to the right-most eight pixels on the bottom row of pixels on the display. Within a byte, the high order bit (bit 7) corresponds to the left-most pixel, and the low order bit (bit 0) corresponds to the right-most pixel.

All bytes within a bank are used. Thus, in low-resolution mode, each bank consists of 30,000 bytes ((800 x 300 pixels)/eight pixels per byte), and in high-resolution mode, each bank consists of 60,000 bytes ((800 x 600 pixels)/eight pixels per byte).

The three graphics memory banks may be accessed in any of the ways in which standard RAM may be accessed. (In fact, graphics memory is just standard RAM.) This means that one may write to or read from 1, 8, 16, or 32 consecutive pixels (in a given bit plane) at a time by using bit, byte, word, or long word operations. This provides for a rapid way of setting a large number of consecutive pixels (e.g., for drawing horizontal lines, for filling figures, or for filling the entire screen with a given color).

The VME/10 has special hardware which allows the user to write to all three color/intensity banks for a given pixel using a single instruction. This is performed using the pixel access area of memory. This memory is arranged in words, with one word per pixel. The first word in the pixel access area corresponds to the pixel in the upper left-hand corner of the display, while the last word corresponds to the pixel in the lower right-hand corner of the display. Within the area, the words are arranged in row-major order -- that is, the second word corresponds to the second pixel on the top line of the screen, the third word corresponds to the third pixel on the top line, etc.

Each pixel access word contains three bits that map directly to the corresponding bits in the three graphics memory banks -- bit 0 maps to bank 1, bit 1 maps to bank 2, and bit 2 maps to bank 3. Thus, setting bits 0 and 2 in a pixel access word results in setting the corresponding bits in graphics memory banks 1 and 3. Also, each pixel access word contains three mask bits, again one for each of the graphics banks -- bit 8 corresponds to bank 1, bit 9 corresponds to bank 2, and bit 10 corresponds to bank 3. These bits are used to enable and disable the writing to particular banks. This is accomplished by setting or clearing bit 0, 1, or 2 in a pixel access word, which causes the corresponding bit in bank 1, 2, or 3 to be set or cleared only if the corresponding mask bit in the pixel access word is set. Otherwise, the bank bit remains unchanged. For example, if the value $0306 is written to a pixel access word, the corresponding bit in bank 1 is cleared, the corresponding bit in bank 2 is set, and the corresponding bit in bank 3 remains unchanged. See Figure 3-1.

The pixel access area does not consist of real memory, but is special hardware that occupies a space in the memory map. Pixel access memory should be written to and read from only a word at a time. Each time a pixel access word is written, the mask must be included as well as the actual pixel data. Pixel access words may be read, but only the low-order byte of each word will be meaningful and will contain the current values of the corresponding bits in the three graphics memory banks.

The locations and lengths of the graphics memory banks and the pixel access area in the VME/10 memory map differ, depending on the graphics resolution mode. Figure 3-2 illustrates the graphics memory map when in low-resolution mode, while Figure 3-3 illustrates high-resolution mode.

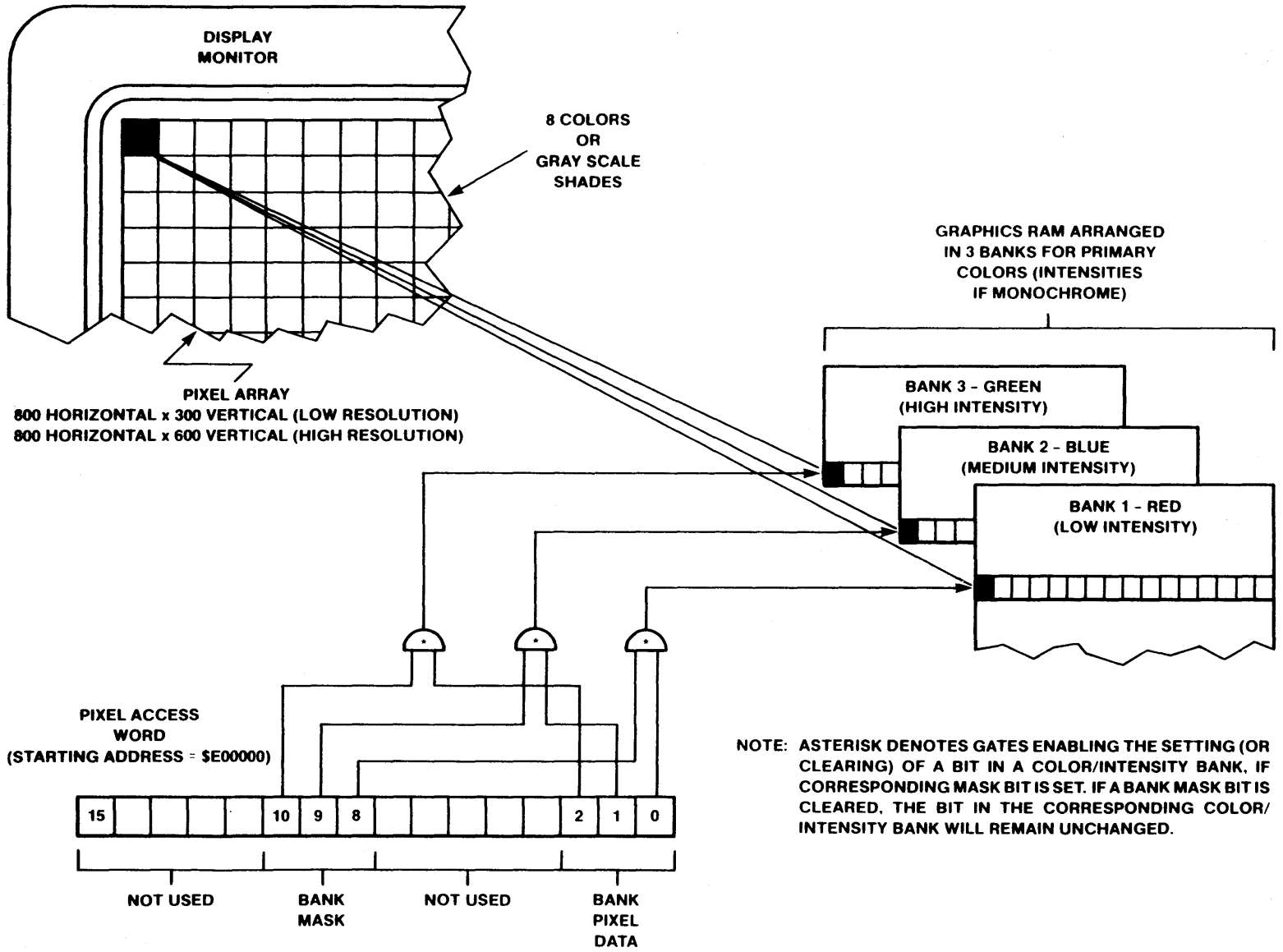
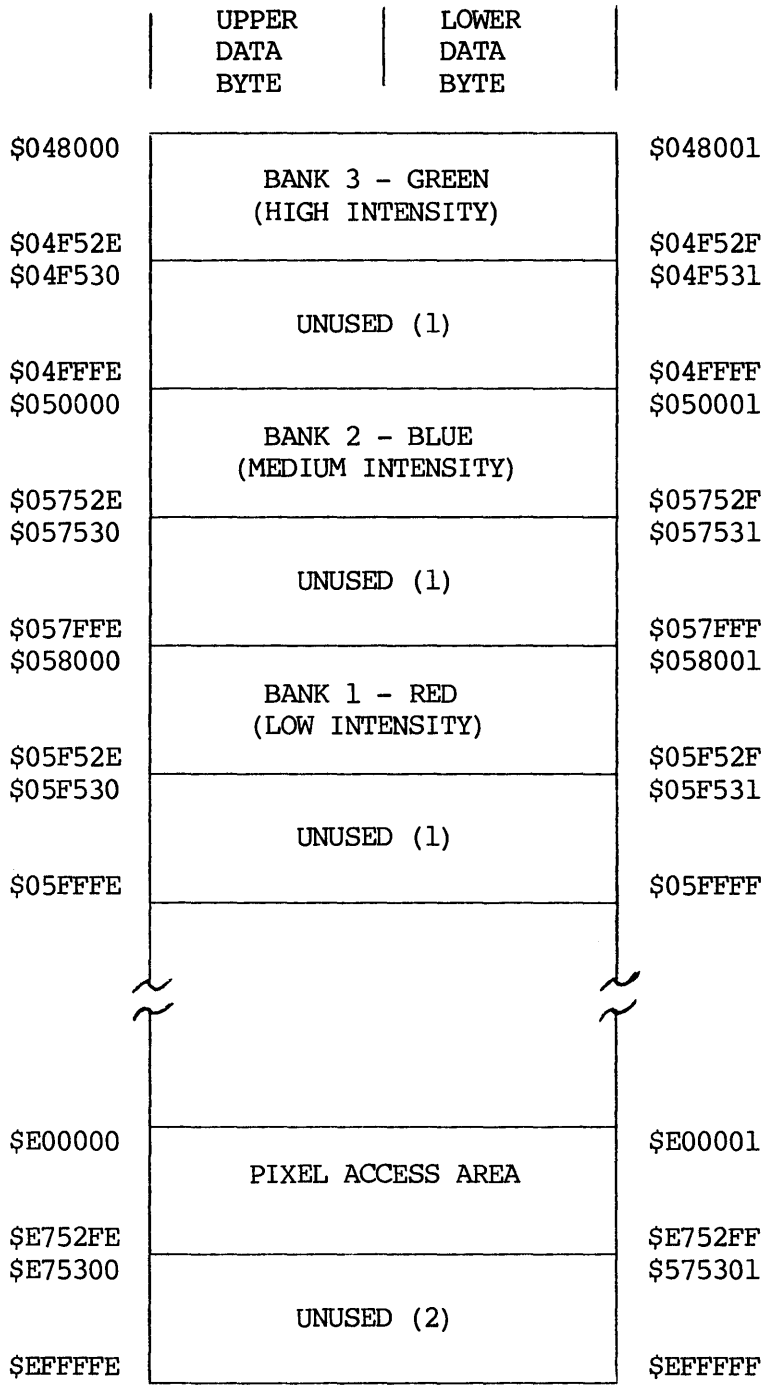


FIGURE 3-1. Pixel Access (Low and High Resolution)

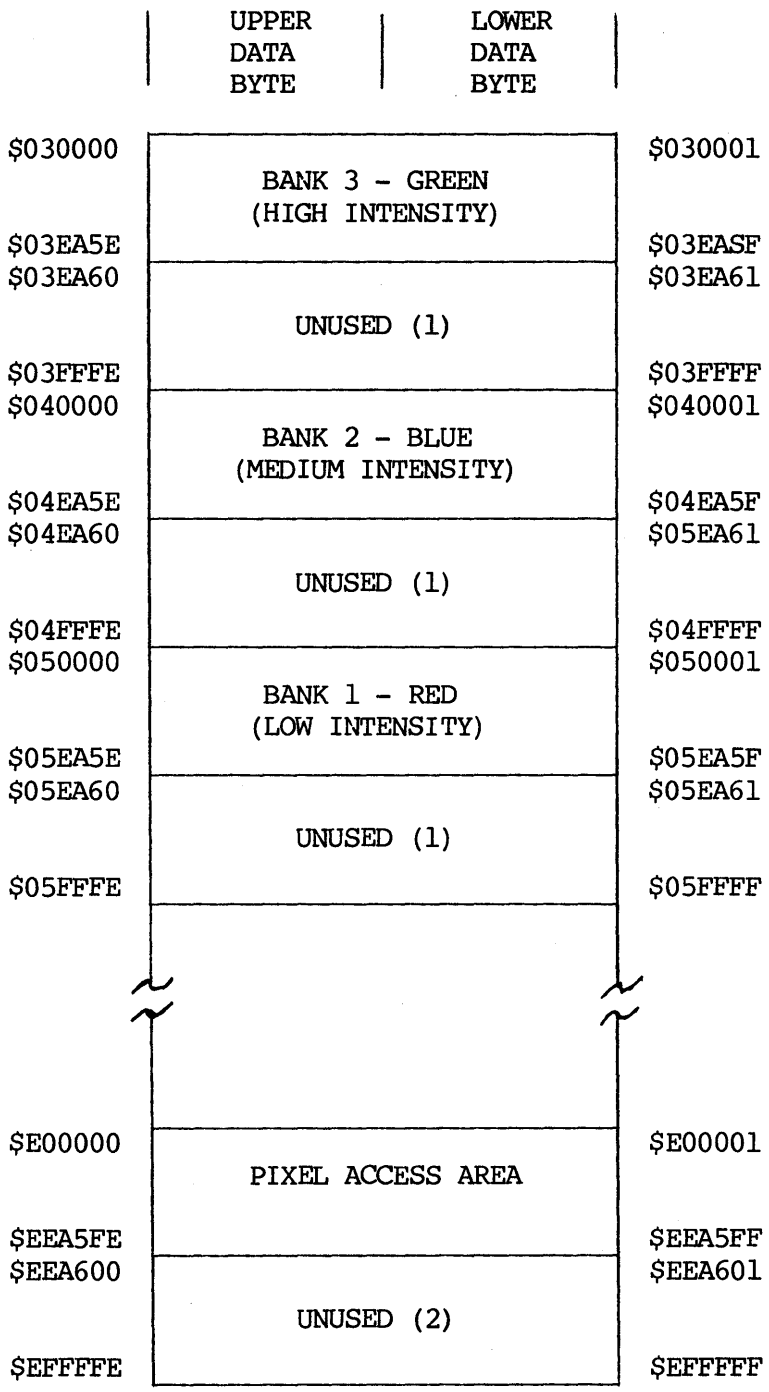


NOTES:

(1) Areas of unused memory are regular RAM and are available for use by the user.

(2) Area of unused memory is special RAM and is not available for use by the user.

FIGURE 3-2. Low Resolution Graphics Memory Map



NOTES:

(1) Areas of unused memory are regular RAM and are available for use by the user.

(2) Area of unused memory is special RAM and is not available for use by the user.

FIGURE 3-3. High Resolution Graphics Memory Map

3.2.2 Graphics Control Register

This section describes registers in the VME/10 that affect the operations of VME/10 graphics. Several registers deal with switching between low-resolution and high-resolution modes. The required register settings for each mode are summarized in Table 3-2.

TABLE 3-2. Required Settings for Graphics

CRTC CONTROLLER CHIP (1) (MC6845)	LOW RESOLUTION	HIGH RESOLUTION	MONOCHROME	COLOR
Register 0	$62	$62		
Register 1	$50	$50		
Register 2	$56	$56		
Register 3	$11	$11		
Register 4	$19	$19		
Register 5	$03	$02		
Register 6	$19	$19		
Register 7	$19	$19		
Register 8	$00	$03		
Register 9	$0B	$16		
Control register 0 ($F19F05) Bit 3 - Dutycycle			1	0
Control register 1 ($F19F07) Bit 4 - High resolution (2)	0	1		
Bits 3-1 - Graphics enable	7	7		
Graphics offset register ($F19F13)	$00	$32		

NOTES:

(1) Write register number into location $F1A021, then write corresponding register value into location $F1A023.

(2) Changing the value of this bit totally remaps all of dynamic RAM in the address space $000000 to $05FFFF. Thus, any program or data in this address space will probably be lost. It is recommended that the VM command in TENbug be used to toggle between low resolution and high resolution modes.

3.2.3 Graphics Cursor Registers

The VME/10 display supports a graphics cursor consisting of two cross hairs (a vertical line and a horizontal line). The cross hairs appear inverse to the color that is present on the display screen (e.g., no color - white cursor, green screen - magenta cursor). Each cross hair is controlled separately by one of the two graphics cursor registers, each of which is 12-bits in length.

The vertical graphics cursor (vertical cross hair) is controlled by the vertical graphics cursor register at address $F19F00. By loading the vertical graphics cursor register with any of the values $FCE0-$FFFF, the vertical graphics cursor may be placed at any one of 800 positions on the display. The value $FCE0 puts the vertical cursor at the far right-hand side of the display, while the value $FFFF puts the cursor at the far left-hand side of the display. The value $FE70 puts the vertical cursor at the center of the display. Storing the value $0 in the vertical graphics cursor register disables the display of the vertical cursor.

The horizontal graphics cursor (horizontal cross hair) is controlled by the horizontal graphics cursor register at address $F19F02. This cursor may be placed at any one of 300 positions on the display by loading the horizontal graphics cursor register with any of the values $FED4-$FFFF. The value $FED4 puts the horizontal cursor at the far bottom of the display, while the value $FFFF puts the cursor at the far top of the display. The value $FF6A puts the horizontal cursor at the center of the display. Storing the value $0 in the horizontal graphics cursor register disables the display of the horizontal cursor.

NOTE

Both of the cursor registers are write-only. Reading from either register will obtain meaningless values.

3.2.4 CRT Controller (CRTC)

To switch from low-resolution graphics display mode to high-resolution graphics display mode (or vice-versa), the CRTC Controller (CRTC) chip MC6845 must be reprogrammed. The CRTC has several byte-length internal registers, each of which can be programmed separately. To change the value of an CRTC register, first write the register number into location $F1A021 and then write the new register value into location $F1A023. Both of these writes must be byte operations. These registers are write-only; their contents cannot be read. Table 3-3 lists the required CRTC values for low- and high-resolution modes (the low-resolution values are also used for standard text display):

TABLE 3-3. Resolution Values

REGISTER	LOW RESOLUTION	HIGH RESOLUTION
0	$62	$62
1	$50	$50
2	$56	$56
3	$11	$11
4	$19	$19
5	$03	$02
6	$19	$19
7	$19	$19
8	$00	$03
9	$0B	$16

3.2.5 Control Register 0 (Location Address $F19F05)

Bit 3 of control register 0 controls the display dutycycle. This bit should be set when a monochrome monitor is being used. When a color monitor is being used, this bit should be cleared for adequate display brightness.

3.2.6 Control Register 1 (Location Address $F19F07)

Bit 4 of control register 1 selects the proper memory mapping for low-resolution and high-resolution modes. This bit must be cleared for low-resolution graphics, and must be set for high-resolution graphics. Changing the value of this bit totally remaps all of dynamic RAM in the address space $000000-$05FFFF. This would probably destroy any programs and/or data in this address space. Therefore, it is recommended that the VM command in TENbug be used to toggle this bit. By default, this bit is set (high-resolution) whenever the VME/10 is turned on.

NOTE

It is possible to switch between low- and high-resolution under program control. To do so requires that bit 4 of control register 1 already be set and remain set. To select the resolution, set the graphics offset register to the proper value (see section 3.2.7) and program the CRTC accordingly (see table 3-2). When using this method, banks 3, 2, and 1 will always start at $30000, $40000, and $50000, respectively. It is not possible to switch between low- and high-resolution when bit 4 of control register 1 is clear; only low-resolution is allowed.

Bits 1 through 3 of control register 1 are used to enable and disable the display of individual graphics memory banks. A set bit enables a bank while a clear bit disables a bank. Bit 1 controls bank 1 (red/low intensity), bit 2 controls bank 2 (blue/medium intensity), and bit 3 controls bank 3 (green/high intensity). Thus, when all three bits are cleared, no graphics are displayed; when all three bits are set, graphics of all colors/intensities are displayed.

NOTE

These bits do not affect the user's ability to write to the individual graphics memory banks; they affect only the displaying of those banks.

3.2.7 Graphics Offset Register (Location Address $F19F13)

The graphics offset register must also be altered when changing between low-resolution and high-resolution modes. For low-resolution graphics, this register must contain $00. For high-resolution graphics, this register must contain $32.

NOTE

By default, this register contains $00 (when the VME/10 is powered up). Also, use of the TENBug VM command does not change the value of this register.

3.3 SOFTWARE APPLICATION

This section presents several examples of programs that use the VME/10 graphics. All of the programs presented assume the presence of VERSAdos.

One of the first problems encountered when attempting to use VME/10 graphics under VERSAdos is having the application program loaded into graphics memory ($48000-$5FFFF in low-resolution; $30000-$5FFFF in high-resolution). It is almost impossible to use the same memory for graphics and for program execution at the same time. One way to solve this problem is by always writing position-independent application programs that relocate themselves if they get loaded into graphics RAM. An easier method is to write just one position-independent utility program that can relocate itself, if necessary, and reserves graphics RAM for use by subsequent application progams. The graphics RAM can be reserved by allocating a locally-shareable segment that encompasses the graphics memory. Then the various application programs can simply attach to that segment to gain access to graphics RAM. Also, there is no need for the application programs to be position-independent and self-relocating because there is no way they will be loaded into the already allocated graphics RAM.

Listing 1 is an example of such a utility program, called GRAF. When GRAF is executed, it first relocates itself to ensure that it is not occupying any graphics RAM. It then allocates the graphics memory in a locally-shareable segment called GRAF. Furthermore, GRAF enables the display of graphics by setting bits 1 through 3 in control register 1 (location address $F19F07) and by reprogramming the CRTC for the proper resolution.

When GRAF is finished, the program terminates. However, the locally-shareable segment GRAF remains, as does the enabling of the graphics display. The segment remains active until the session is terminated (i.e., the user logs off) or it is deallocated by a call to RMS68K. The graphics display remains enabled until bits 1 through 3 of control register 1 are cleared and the CRTC is reprogrammed. Listing 2 is an example of a program that deallocates the graphics RAM segment and disables the display of graphics. The program is called NOGRAF.

By using GRAF and NOGRAF, the user can develop application programs without worrying about having to self-relocate, how to enable the graphics display, or how to return the display to normal. All the user needs to do is run GRAF before an application program and run NOGRAF afterwards. If there are several application programs, invoke GRAF once at the beginning and NOGRAF once at the end.

Finally, listing 3 is an example of a simple application program called BARS. BARS draws a color/intensity chart consisting of eight horizontal bars and eight vertical bars. Each bar in a given axis is of a different color/intensity. Where two bars intersect, the intersecting area is the exclusive-OR of the two colors/intensities. Thus, where two bars of the same color/intensity intersect, the result is black. The bars are displayed against backgrounds of each of the eight possible colors/intensities. (The bars are also exclusive-OR'd with the background.)

BARS creates its graphics both by writing directly to the three color/intensity banks in graphics RAM and by writing to the pixel access area. The former is used to draw the background, while the latter is used to draw the bars. To access the graphics RAM, BARS attaches the shared segment created by GRAF. To use the pixel access area, BARS has to allocate a segment at the proper spot in the memory map. BARS determines if low-resolution or high-resolution graphics are in use and adjusts to work equally well in both modes.

```
                        *
                        *       GRAF
                        *
                        *       7 December 1983
                        *
                        *
                        *       This program creates a locally shareable segment called GRAF.
                        *       If the VME/10 is in low-resolution graphics mode, this segment
                        *       begins at $48000 and extends thru $5FFFF.  If the VME/10 is in
                        *       high-resolution graphics mode, this segment begins at $30000
                        *       and extends thru $5FFFF.
                        *
                        *       This program also "turns graphics on" by enabling the display
                        *       of graphics and reprogramming the CRTC controller chip.
                        *
                        *       Tasks within the same session can attach to the segment for the
                        *       purpose of accessing the graphics ram.
                        *
                        *       The segment can be de-allocated by either terminating the session,
                        *       or by invoking the program NOGRAF.  Invoking the program NOGRAF
                        *       will also disable the graphics display and return the CRT display
                        *       to normal.
                        *
                        *       The basic attack is as follows:
                        *
                        *       This task has probably been loaded into memory within the graphics
                        *       ram.  The task will move itself out of there and de-allocate it's
                        *       original code segment, thus freeing up the graphics ram.
                        *
                        *       Then, it allocates the physical ram from $48000 thru $5FFFF (for
                        *       low-resolution) or from $30000 thru $5FFFF (for high-resolution)
                        *       and establishes that segment as locally shareable.
                        *
                        *       Then, it clears all graphics memory and enables the the display
                        *       of graphics.
                        *
                        *       Finally, it terminates, leaving the GRAF segment available for use,
                        *       as well as preventing further allocations in that area.
                        *
                        *       The beginning of the program is here:
                        *
0         00000000      START   EQU     *
                        *
                        *       First, acquire a segment to gain access to the hardware
                        *       registers so we can find out what resolution mode we are
                        *       in.
                        *
0 00000000 41FA0134             LEA.L   PB3(PC),A0          Point to parameter block
0 00000004 7001                 MOVE.L  #1,D0               GTSEG directive number
0 00000006 4E41                 TRAP    #1                  Call RMS68K
0 00000008 6704                 BEQ.S   GOTREGS             Successful call
                        *
                        *       What follows is the error handler.  It simply terminates
                        *       ourself.
                        *
0         0000000A      ERROR   EQU     *
0 0000000A 700E                 MOVE.L  #14,D0              Terminate self directive number
0 0000000C 4E41                 TRAP    #1                  Call RMS68K
```

```
                        *
                        *       Get starting address of graphics memory in A5.
                        *
0          0000000E     GOTREGS EQU     *
0 0000000E 2A7C00048000         MOVE.L  #$48000,A5          Assume low-res
0 00000014 0839000400F1         BTST.B  #4,$F19F07          Really low-res?
           9F07
0 0000001C 6706                 BEQ.S   GOTRES1             Yes
0 0000001E 2A7C00030000         MOVE.L  #$30000,A5          No - set up for high-res
                        *
                        *       Acquire a new segment to receive a copy of the code.
                        *
0          00000024     GOTRES1 EQU     *
0 00000024 41FA00E0             LEA     PB1(PC),A0          Point to the GTSEG parameter block
0 00000028 214D0010             MOVE.L  A5,PB1LOC-PB1(A0)   Set up first address ...
0 0000002C 04A800001000         SUB.L   #$1000,PB1LOC-PB1(A0) to try.
           0010
0 00000034 7001                 MOVE.L  #1,D0               GTSEG directive number
0 00000036 4E41                 TRAP    #1                  Call RMS68K
0 00000038 670A                 BEQ.S   GOTCODE             Branch if it worked
0 0000003A 04A800000100         SUB.L   #$100,PB1LOC-PB1(A0) Else try a little lower
           0010
0 00000042 60E0                 BRA.S   GOTRES1             Give it another shot
0 00000044                 GOTCODE
                        *
                        *       Move my code to the new segment
                        *
0 00000044 2248                 MOVE.L  A0,A1               Point to the beginning of the new me
0 00000046 41FAFFB8             LEA     START(PC),A0        Point to the beginning of the old me
0 0000004A 303C015C             MOVE.W  #END-START,D0       My approximate length
0 0000004E 12D8            MOVE MOVE.B  (A0)+,(A1)+         Move a byte to the new place
0 00000050 51C8FFFC             DBRA    D0,MOVE             Until I'm all there
0 00000054 227A00C0             MOVE.L  PB1LOC(PC),A1       Point to the new START
0 00000058 4EE9005C             JMP     NEW-START(A1)       Jump to the new NEW
                        *
                        *       Here begins the code executed in the new code segment
                        *
0 0000005C                 NEW
                        *
                        *       De-allocate the old code segment
                        *
0 0000005C 41FA00A8             LEA     PB1(PC),A0          Point to the parameter block
0 00000060 217C53454730         MOVE.L  #'SEG0',PB1NAME-PB1(A0) Old segment name
           000C
0 00000068 7002                 MOVE.L  #2,D0               DESEG directive number
0 0000006A 4E41                 TRAP    #1                  Call RMS68K
0 0000006C 669C                 BNE     ERROR               Crash if it didn't work
                        *
                        *       Acquire the graphics ram
                        *
0 0000006E 41FA00AE             LEA     PB2(PC),A0          Point to parameter block
0 00000072 214D0010             MOVE.L  A5,PB2LOC-PB2(A0)   Physical address desired
0 00000076 227C00060000         MOVE.L  #$60000,A1          Calculate...
0 0000007C 93CD                 SUB.L   A5,A1               segment...
0 0000007E 21490014             MOVE.L  A1,PB2LEN-PB2(A0)   length.
0 00000082 7001                 MOVE.L  #1,D0               GTSEG directive number
0 00000084 4E41                 TRAP    #1                  Call RMS68K
0 00000086 6682                 BNE     ERROR               Crash if it didn't work
```

```
113                         *
114                         *       Make GRAF locally shareable
115                         *
116 0 00000088 41FA0094             LEA     PB2(PC),A0          Point to parameter block
117 0 0000008C 317C90000008         MOVE.W  #$9000,PB2OPT-PB2(A0) Options
118 0 00000092 317C2000000A         MOVE.W  #$2000,PB2ATTR-PB2(A0) Attributes
119 0 00000098 7007                 MOVE.L  #7,D0               DCLSHR directive number
120 0 0000009A 4E41                 TRAP    #1                  Call RMS68K
121 0 0000009C 6600FF6C             BNE     ERROR               Crash if it didn't work
122                         *
123                         *       Clear out the graphics memory and disable the graphics cursor.
124                         *
125 0 000000A0 224D                 MOVE.L  A5,A1               A1.L = address of graphics RAM
126 0 000000A2 203C00060000         MOVE.L  #$60000,D0          Ending address of graphics RAM
127 0 000000A8 9089                 SUB.L   A1,D0               D0.L = # of bytes in graphics RAM
128 0 000000AA E488                 LSR.L   #2,D0               D0.W = # of long words in graphics RAM
129 0 000000AC 5340                 SUB.W   #1,D0               Adjust for loop
130 0 000000AE 4299         CLRLOOP CLR.L   (A1)+               Clear graphics RAM, a...
131 0 000000B0 51C8FFFC             DBRA    D0,CLRLOOP          long word at a time.
132 0 000000B4 427900F19F00         CLR.W   $F19F00             Disable vertical graphics cursor
133 0 000000BA 427900F19F02         CLR.W   $F19F02             Disable horizontal graphics cursor
134                         *
135                         *       Enable graphics and set up resolution mode
136                         *
137 0 000000C0 0239007F00F1         AND.B   #$7F,$F19F07        Disable fast access to system RAM
                9F07
138 0 000000C8 0039000E00F1         OR.B    #$0E,$F19F07        Enable graphics display
                9F07
139 0 000000D0 423900F19F13         CLR.B   $F19F13             Assume...
140 0 000000D6 43FA0076             LEA.L   LOWRES(PC),A1       low-res.
141 0 000000DA 0839000400F1         BTST.B  #4,$F19F07          Really low-res?
                9F07
142 0 000000E2 670C                 BEQ.S   GOTRES2            Yes
143 0 000000E4 13FC003200F1         MOVE.B  #$32,$F19F13        No - set up...
                9F13
144 0 000000EC 43FA0067             LEA.L   HIRES(PC),A1        for hi-res.
145 0 000000F0 45F900F1A021 GOTRES2 LEA     $F1A021,A2          Load address of CRTC controller regs
146 0 000000F6 1019         CRTCLOOP MOVE.B (A1)+,D0            Reprogram the...
147 0 000000F8 6B08                 BMI.S   CRTCDONE           CRTC controller...
148 0 000000FA 1480                 MOVE.B  D0,(A2)             for the proper...
149 0 000000FC 15590002             MOVE.B  (A1)+,2(A2)         resolution...
150 0 00000100 60F4                 BRA     CRTCLOOP            mode.
151                         *
152                         *       Good.  This massive task is now finished.  So I will go away.
153                         *
154 0          00000102     CRTCDONE EQU    *
155 0 00000102 700F                 MOVE.L  #15,D0              TERM directive number
156 0 00000104 4E41                 TRAP    #1                  Call RMS68K
157                         *
158                         *       Parameter block to get new code segment.
159                         *       Also used to delete old code segment.
160                         *
161 0 00000106 000000000000 PB1     DC.L    0,0                 Acquire new code segment
162 0 0000010E 01000000     PB1OPT  DC.W    %0000000100000000,0 Physical address
163 0 00000112 434F4445     PB1NAME DC.L    'CODE'              Name
164 0 00000116 00000000     PB1LOC  DC.L    0                   Address
165 0 0000011A 0000015D     PB1LEN  DC.L    END-START+1         Length
166                         *
```

```
167                         *       Parameter block used to get segment at graphics RAM.
168                         *
169 0 0000011E 000000000000 PB2     DC.L    0,0                 Taskname and session
170 0 00000126 0000         PB2OPT  DC.W    0                   Options
171 0 00000128 0000         PB2ATTR DC.W    0                   Segment attributes
172 0 0000012A 47524146     PB2NAME DC.L    'GRAF'              Segment name
173 0 0000012E 00000000     PB2LOC  DC.L    0                   Segment address
174 0 00000132 00000000     PB2LEN  DC.L    0                   Segment length
175                         *
176                         *       Parameter block to get segment at hardware registers.
177                         *
178 0 00000136 000000000000 PB3     DC.L    0,0                 Taskname and session
179 0 0000013E 0000         PB3OPT  DC.W    0                   Options
180 0 00000140 0800         PB3ATTR DC.W    $800                Attributes (memory mapped I/O)
181 0 00000142 52454753     PB3NAME DC.L    'REGS'              Segment name
182 0 00000146 00F19F00     PB3LOC  DC.L    $F19F00             Segment address
183 0 0000014A 00000200     PB3LEN  DC.L    $F1A100-$F19F00     Segment length
184                         *
185                         *       The following are CRTC controller register values for
186                         *       both low- and high- resolution modes.
187                         *
188 0 0000014E 050308000908 LOWRES  DC.B    5,$03,8,$00,9,$08,-1
189 0 00000155 050208030916 HIRES   DC.B    5,$02,8,$03,9,$16,-1
190                         *
191                         *       Uh-duh-dee, uh-duh-dee, uh-duh-dee, that's all folks!
192                         *
193 0           0000015C    END     EQU     *
194                                 END

****** TOTAL ERRORS      0--
****** TOTAL WARNINGS    0--
```

```
                                *
                                *       NOGRAF
                                *
                                *       7 December 1983
                                *
                                *
                                *       This program de-allocates segment GRAF.
                                *       It also disables graphics and returns the CRT to normal.
                                *
                                *       The beginning of the program is here:
                                *
0          00000000     START   EQU     *
                                *
                                *       First, acquire a segment to gain access to the hardware
                                *       registers so we can find out what resolution mode we are
                                *       in.
                                *
0 00000000 41FA0096             LEA.L   PB3(PC),A0          Point to parameter block
0 00000004 7001                 MOVE.L  #1,D0               GTSEG directive number
0 00000006 4E41                 TRAP    #1                  Call RMS68K
0 00000008 6704                 BEQ.S   GOTREGS             Successful call
                                *
                                *       What follows is the error handler.  It simply terminates
                                *       ourself.
                                *
0          0000000A     ERROR   EQU     *
0 0000000A 700E                 MOVE.L  #14,D0              Terminate self directive number
0 0000000C 4E41                 TRAP    #1                  Call RMS68K
                                *
                                *       Get starting address of graphics memory in A1.
                                *
0          0000000E     GOTREGS EQU     *
0 0000000E 227C00048000         MOVE.L  #$48000,A1          Assume low-res
0 00000014 0839000400F1         BTST.B  #4,$F19F07          Really low-res?
           9F07
0 0000001C 6706                 BEQ.S   GOTRES1             Yes
0 0000001E 227C00030000         MOVE.L  #$30000,A1          No - set up for high-res
                                *
                                *       Attach to the segment GRAF, so's it's mine to delete
                                *
0          00000024     GOTRES1 EQU     *
0 00000024 41FA005A             LEA     PB1(PC),A0          Point to the parameter block
0 00000028 7004                 MOVE.L  #4,D0               ATTSEG directive number
0 0000002A 4E41                 TRAP    #1                  Call RMS68K
0 0000002C 66DC                 BNE     ERROR               Crash if didn't work
                                *
                                *       Clear out the graphics memory and disable the graphics cursor.
                                *
0 0000002E 203C00060000         MOVE.L  #$60000,D0          Ending address of graphics RAM
0 00000034 9089                 SUB.L   A1,D0               D0.L = # of bytes in graphics RAM
0 00000036 E488                 LSR.L   #2,D0               D0.W = # of long words in graphics RAM
0 00000038 5340                 SUB.W   #1,D0               Adjust for loop
0 0000003A 4299         CLRLOOP CLR.L   (A1)+               Clear graphics RAM, a...
0 0000003C 51C8FFFC             DBRA    D0,CLRLOOP          long word at a time.
0 00000040 427900F19F00         CLR.W   $F19F00             Disable vertical graphics cursor
0 00000046 427900F19F02         CLR.W   $F19F02             Disable horizontal graphics cursor
                                *
                                *       Disable graphics and return CRT to normal.
```

```
 58                          *
 59 0 0000004C 023900F100F1          AND.B   #$F1,$F19F07        Disable graphics display
               9F07
 60 0 00000054 0039008000F1          OR.B    #$80,$F19F07        Enable fast access to system RAM
               9F07
 61 0 0000005C 43FA0052              LEA.L   LOWRES(PC),A1       Load address of new CRTC reg values
 62 0 00000060 45F900F1A021          LEA     $F1A021,A2          Load address of CRTC controller regs
 63 0 00000066 1019          CRTCLOOP MOVE.B  (A1)+,D0            Reprogram the...
 64 0 00000068 6B08                  BMI.S   CRTCDONE            CRTC controller...
 65 0 0000006A 1480                  MOVE.B  D0,(A2)             for the proper...
 66 0 0000006C 15590002              MOVE.B  (A1)+,2(A2)         resolution...
 67 0 00000070 60F4                  BRA     CRTCLOOP            mode.
 68                          *
 69                          *       Now de-allocate segment GRAF.
 70                          *
 71 0          00000072      CRTCDONE EQU    *
 72 0 00000072 41FA000C              LEA     PB1(PC),A0          Point to parameter block
 73 0 00000076 7002                  MOVE.L  #2,D0               DESEG directive number
 74 0 00000078 4E41                  TRAP    #1                  Call RMS68K
 75 0 0000007A 668E                  BNE     ERROR               Crash if didn't work
 76                          *
 77                          *       All done, so off I go
 78                          *
 79 0 0000007C 700F                  MOVE.L  #15,D0              TERM directive number
 80 0 0000007E 4E41                  TRAP    #1                  Call RMS68K
 81                          *
 82                          *       Attach and de-allocate parameter block
 83                          *
 84 0 00000080 000000000000 PB1      DC.L    0,0                 Acquire new code segment
 85 0 00000088 28002000     PB1OPT   DC.W    %0010100000000000,$2000 Remove permanance when DESEGing
 86 0 0000008C 47524146     PB1NAME  DC.L    'GRAF'              Segment name
 87 0 00000090 00000000     PB1LOC   DC.L    $00000              Address (n/a)
 88 0 00000094 00000000     PB1LEN   DC.L    0                   Length  (n/a)
 89                          *
 90                          *       Parameter block to get segment at hardware registers.
 91                          *
 92 0 00000098 000000000000 PB3      DC.L    0,0                 Taskname and session
 93 0 000000A0 0000         PB3OPT   DC.W    0                   Options
 94 0 000000A2 0800         PB3ATTR  DC.W    $800                Attributes (memory mapped I/O)
 95 0 000000A4 52454753     PB3NAME  DC.L    'REGS'              Segment name
 96 0 000000A8 00F19F00     PB3LOC   DC.L    $F19F00             Segment address
 97 0 000000AC 00000200     PB3LEN   DC.L    $F1A100-$F19F00     Segment length
 98                          *
 99                          *       The following are CRTC controller register values for
100                          *       low-resolution mode (also used for standard text display).
101                          *
102 0 000000B0 05030800090B LOWRES   DC.B    5,$03,8,$00,9,$0B,-1
103                          *
104                          *       That's all she wrote.
105                          *
106 0          00000000             END     START

****** TOTAL ERRORS      0--
****** TOTAL WARNINGS    0--
```

```
                                *
                                *       BARS
                                *
                                *       7 December 1983
                                *
                                *
                                *       This program is an example of using the graphics
                                *       hardware of the VME/10 directly.  It draws
                                *       a color (grey scale) chart consisting of eight
                                *       horizontal and eight vertical bars.  Each bar in
                                *       a given axis is of a different color (grey scale).
                                *       Where a horizontal bar intersects a vertical bar,
                                *       the result is the Exclusive-OR of the two colors.
                                *
                                *       The color (grey scale) chart is drawn against each
                                *       of the eight possible background colors, with a slight
                                *       delay between each.
                                *
                                *       Before running this program, one should first run the
                                *       program GRAF to reserve the graphics RAM and to
                                *       enable the graphics display.  After running BARS,
                                *       the program NOGRAF should be executed to return the
                                *       display to normal.
                                *
                                *
                                *       The program starts here:
                                *
0          00000000     START   EQU     *
                                *
0 00000000 4FFA0366             LEA.L   STACK(PC),A7        First give ourselves a stack
                                *
                                *       Attach the segment that contains the graphics RAM.
                                *
0 00000004 41FA013E             LEA.L   PB1(PC),A0          Get address of parameter block
0 00000008 7004                 MOVE.L  #4,D0               Attach segment directive number
0 0000000A 4E41                 TRAP    #1                  Call RMS68K
0 0000000C 6704                 BEQ.S   GOTSEG              A successful call
                                *
                                *       The following is our error handler.
                                *       It simply causes the program to abort itself.
                                *
0          0000000E     ERROR   EQU     *
0 0000000E 700E                 MOVE.L  #14,D0              Abort self directive number
0 00000010 4E41                 TRAP    #1                  Call RMS68K -- never to return
                                *
0          00000012     GOTSEG  EQU     *
                                *
                                *       See if using low-res or high-res graphics.
                                *       Assume low-res and set up as such.
                                *
0 00000012 4205                 CLR.B   D5                  Indicate low-res
0 00000014 B1FC00048000         CMP.L   #$48000,A0          Really low-res?
0 0000001A 6702                 BEQ.S   GOTRES1            Yes
0 0000001C 4605                 NOT.B   D5                  Nope - indicate high-res
                                *
0          0000001E     GOTRES1 EQU     *
                                *
                                *       Create a segment to gain access to the pixel access area.
```

```
                        *
                        *       Get address of parameter block
0 0000001E 41FA013C             LEA.L   PB2(PC),A0      Get segment directive number
0 00000022 7001                 MOVE.L  #1,D0           Call RMS68K
0 00000024 4E41                 TRAP    #1              Crash if didn't work
0 00000026 66E6                 BNE     ERROR
                        *
                        *       Initialize the background color.
                        *
0 00000028 4207                 CLR.B   D7              First background color is 0
                        *
                        *       Loop here for each new background color.
                        *
0          0000002A     BACKLOOP EQU    *
                        *
0 0000002A 6170                 BSR.S   FILLSCR         Fill screen with desired background color
                        *
                        *       Draw the horizontal color (grey scale) bars
                        *
0 0000002C 4240                 CLR.W   D0              Starting row = 0
0 0000002E 4241                 CLR.W   D1              Starting col = 0
0 00000030 343C001E             MOVE.W  #30,D2          # of filled rows per bar = 30 (low-res)
0 00000034 363C0320             MOVE.W  #800,D3         # of filled cols per bar = 800
0 00000038 4204                 CLR.B   D4              Starting color = 0
0 0000003A 3C3C0025             MOVE.W  #37,D6          Total # of rows per bar = 37 (low-res)
0 0000003E 4A05                 TST.B   D5              Low-res assumption safe?
0 00000040 6708                 BEQ.S   GOTRES2         Yes
0 00000042 343C003C             MOVE.W  #60,D2          Nope - 60 filled rows per bar in high-res
0 00000046 3C3C004B             MOVE.W  #75,D6          Total of 75 rows per bar in high-res
                        *
0          0000004A     GOTRES2 EQU     *
                        *
                        *       Draw eight bars, one at a time.
                        *
0          0000004A     HORZLOOP EQU    *
0 0000004A 610000C0             BSR     FILLRECT        Draw a filled horizontal bar
0 0000004E 0C040007             CMP.B   #7,D4           Last bar?
0 00000052 6706                 BEQ.S   HORZDONE        Yep - on to vertical bars
0 00000054 5204                 ADDQ.B  #1,D4           Nope - change to next color
0 00000056 D046                 ADD.W   D6,D0           Change to start of next bar
0 00000058 60F0                 BRA.S   HORZLOOP        Loop for next bar
                        *
0          0000005A     HORZDONE EQU    *
                        *
                        *       Now draw the vertical bars
                        *
0 0000005A 4240                 CLR.W   D0              Starting row = 0
0 0000005C 4241                 CLR.W   D1              Starting column = 0
0 0000005E 343C012C             MOVE.W  #300,D2         # of filled rows per bar = 300 (low-res)
0 00000062 363C0050             MOVE.W  #80,D3          # of filled cols per bar = 80
0 00000066 4204                 CLR.B   D4              Starting color = 0
0 00000068 3C3C0064             MOVE.W  #100,D6         Total # of cols per bar = 100
0 0000006C 4A05                 TST.B   D5              Low-res assumption safe?
0 0000006E 6704                 BEQ.S   GOTRES3         Yep
0 00000070 343C0258             MOVE.W  #600,D2         Nope - 600 filled rows per bar in high-res
                        *
0          00000074     GOTRES3 EQU     *
                        *
                        *       Draw eight vertical bars, one at a time.
```

```
117                             *
118 0           00000074        VERTLOOP EQU     *
119 0 00000074 61000096                  BSR     FILLRECT            Draw a filled vertical bar
120 0 00000078 0C040007                  CMP.B   #7,D4               Last bar?
121 0 0000007C 6706                      BEQ.S   VERTDONE            Yep - time to finish
122 0 0000007E 5204                      ADDQ.B  #1,D4               Nope - change to next color
123 0 00000080 D246                      ADD.W   D6,D1               Change to start of next bar
124 0 00000082 60F0                      BRA.S   VERTLOOP            Loop for next bar
125                             *
126 0           00000084        VERTDONE EQU     *
127                             *
128                             *        See if have used all background colors.
129                             *        If not change to next color and do it again.
130                             *
131 0 00000084 0C070007                  CMP.B   #7,D7               Last background color?
132 0 00000088 670E                      BEQ.S   ALLDONE             Yep - that's it
133 0 0000008A 5207                      ADDQ.B  #1,D7               Nope - change to next one
134 0 0000008C 207C000003E8              MOVE.L  #1000,A0            Delay...
135 0 00000092 7015                      MOVE.L  #21,D0              ourselves for...
136 0 00000094 4E41                      TRAP    #1                  one second...
137 0 00000096 6092                      BRA     BACKLOOP            Loop back
138                             *
139 0           00000098        ALLDONE  EQU     *
140                             *
141                             *        Time to end to program - terminate ourself.
142                             *
143 0 00000098 700F                      MOVE.L  #15,D0              Terminate self directive number
144 0 0000009A 4E41                      TRAP    #1                  Call RMS68K - never to return
```

```
146                         *
147                         *       The routine FILLSCR fills the screen with a specified
148                         *       color by writing directly to the three individual color
149                         *       banks.
150                         *
151                         *       The registers must be passed to this routine as follows:
152                         *             D5.B  -   0 => low-resolution mode
153                         *                      <>0 => high-resolution mode
154                         *             D7.B  -  color with which to fill screen (0-7)
155                         *
156                         *       This routine preserves all the registers (data and address).
157                         *
158 0          0000009C     FILLSCR  EQU     *
159 0 0000009C 48E7E070             MOVEM.L  D0-D2/A1-A3,-(A7)     Save registers
160                         *
161                         *       See if using low-res or high-res graphics.
162                         *       Assume low-res and set up as such.
163                         *
164 0 000000A0 227C00058000         MOVE.L   #$58000,A1            Address of color bank 1
165 0 000000A6 247C00050000         MOVE.L   #$50000,A2            Address of color bank 2
166 0 000000AC 267C00048000         MOVE.L   #$48000,A3            Address of color bank 3
167 0 000000B2 343C1D4B             MOVE.W   #7499,D2              # of long words per bank - 1
168 0 000000B6 4A05                 TST.B    D5                    Really low-res?
169 0 000000B8 6716                 BEQ.S    GOTRES4               Yes
170 0 000000BA 227C00050000         MOVE.L   #$50000,A1            Address of color bank 1
171 0 000000C0 247C00040000         MOVE.L   #$40000,A2            Address of color bank 2
172 0 000000C6 267C00030000         MOVE.L   #$30000,A3            Address of color bank 3
173 0 000000CC 343C3A97             MOVE.W   #14999,D2             # of long words per bank - 1
174                         *
175 0          000000D0     GOTRES4  EQU     *
176                         *
177                         *       Take care of bank 1
178                         *
179 0 000000D0 4280                 CLR.L    D0                    Assume going to clear bank
180 0 000000D2 3202                 MOVE.W   D2,D1                 Get long word count
181 0 000000D4 08070000             BTST     #0,D7                 Clear or set bank?
182 0 000000D8 6702                 BEQ.S    B1LOOP                Clear
183 0 000000DA 4680                 NOT.L    D0                    Set
184 0 000000DC 22C0         B1LOOP  MOVE.L   D0,(A1)+              Change bank 32...
185 0 000000DE 51C9FFFC             DBRA     D1,B1LOOP             bits at a time.
186                         *
187                         *       Take care of bank 2
188                         *
189 0 000000E2 4280                 CLR.L    D0                    Assume going to clear bank
190 0 000000E4 3202                 MOVE.W   D2,D1                 Get long word count
191 0 000000E6 08070001             BTST     #1,D7                 Clear or set bank?
192 0 000000EA 6702                 BEQ.S    B2LOOP                Clear
193 0 000000EC 4680                 NOT.L    D0                    Set
194 0 000000EE 24C0         B2LOOP  MOVE.L   D0,(A2)+              Change bank 32...
195 0 000000F0 51C9FFFC             DBRA     D1,B2LOOP             bits at a time.
196                         *
197                         *       Take care of bank 3
198                         *
199 0 000000F4 4280                 CLR.L    D0                    Assume going to clear bank
200 0 000000F6 3202                 MOVE.W   D2,D1                 Get long word count
201 0 000000F8 08070002             BTST     #2,D7                 Clear or set bank?
202 0 000000FC 6702                 BEQ.S    B3LOOP                Clear
203 0 000000FE 4680                 NOT.L    D0                    Set
```

```
204 0 00000100 26C0        B3LOOP   MOVE.L   D0,(A3)+             Change bank 32...
205 0 00000102 51C9FFFC             DBRA     D1,B3LOOP            bits at a time.
206                        *
207                        *        That's all there is -- restore the registers and return.
208                        *
209 0 00000106 4CDF0E07             MOVEM.L  (A7)+,D0-D2/A1-A3    Restore the saved registers
210                        *
211 0 0000010A 4E75                 RTS                           All done
```

```
                        *
                        *       The routine FILLRECT draws a filled rectangle using the pixel
                        *       access area.  This "memory" allows the programmer to change
                        *       one pixel in all three banks in one shot.
                        *
                        *       The rectangle is filled using Exclusive-OR.  That is, the
                        *       color in which the rectangle is being drawn will be
                        *       Exclusive-OR'ed (on a pixel basis) with any graphics
                        *       already on the display.
                        *
                        *       The registers must be passed to this routine as follows:
                        *             D0.W  -  Starting row of rectangle
                        *             D1.W  -  Starting column of rectangle
                        *             D2.W  -  # of rows in rectangle (height)
                        *             D3.W  -  # of columns in rectangle (width)
                        *             D4.B  -  Color of rectangle (0-7)
                        *
                        *       The display is organized such that row-0, column-0 is at
                        *       the upper left-hand corner of the display.
                        *
                        *       All registers (data and address) are preserved by this routine.
                        *
235 0          0000010C FILLRECT EQU     *
                        *
237 0 0000010C 48E7FFFE           MOVEM.L  D0-D7/A0-A6,-(A7)     Save the registers
                        *
                        *       Compute address of pixel access word for upper
                        *       left-hand corner of rectangle.
                        *
242 0 00000110 41F900E00000       LEA.L    $E00000,A0            Base address of pixel access area
243 0 00000116 3A00               MOVE.W   D0,D5                 Calculate...
244 0 00000118 5345               SUBQ.W   #1,D5                 and...
245 0 0000011A C8FC0640           MULS     #1600,D5              add...
246 0 0000011E D1C5               ADD.L    D5,A0                 row offset.
247 0 00000120 D0C1               ADD.W    D1,A0                 Add column...
248 0 00000122 D0C1               ADD.W    D1,A0                 offset.
                        *
                        *       Set up for nested loops to draw rectangle.
                        *
252 0 00000124 5342               SUBQ.W   #1,D2                 D2.W = # of rows - 1
253 0 00000126 5343               SUBQ.W   #1,D3                 D3.W = # of cols - 1
254 0 00000128 4884               EXT.W    D4                    Make color into a word
                        *
                        *       Draw rectangle using doubly nested loops.
                        *
258 0 0000012A D1FC00000640 FRLOOP1  ADD.L    #1600,A0           A0 = address of start of next row
259 0 00000130 2248               MOVE.L   A0,A1                 Need a copy that can be destroyed
260 0 00000132 3A03               MOVE.W   D3,D5                 D5.W = # of columns - 1
261 0 00000134 B959         FRLOOP2  EOR.W    D4,(A1)+           Write a pixel
262 0 00000136 51CDFFFC           DBRA     D5,FRLOOP2            Loop for next column in a row
263 0 0000013A 51CAFFEE           DBRA     D2,FRLOOP1            Loop for next row
                        *
                        *       All done - restore registers and return.
                        *
267 0 0000013E 4CDF7FFF           MOVEM.L  (A7)+,D0-D7/A0-A6     Restore regs
                        *
269 0 00000142 4E75               RTS                            That's all folks
```

```
271                             *
272                             *       Parameter block to atach segment at graphics RAM.
273                             *
274 0 00000144 000000000000 PB1     DC.L    0,0                 Taskname and session (n/a)
275 0 0000014C 2000                 DC.W    $2000               Options (log addr = phys addr)
276 0 0000014E 2000                 DC.W    $2000               Attributes (locally shareable)
277 0 00000150 47524146             DC.L    'GRAF'              Segment name
278 0 00000154 000000000000         DC.L    0,0                 Segment address and length (n/a)
279                             *
280                             *       Parameter block to get segment at pixel access area.
281                             *
282 0 0000015C 000000000000 PB2     DC.L    0,0                 Taskname and session (n/a)
283 0 00000164 0000                 DC.W    0                   Options
284 0 00000166 0800                 DC.W    $800                Attributes (memory mapped I/O)
285 0 00000168 5049584C             DC.L    'PIXL'              Segment name
286 0 0000016C 00E00000             DC.L    $E00000             Segment address
287 0 00000170 00100000             DC.L    $100000             Segment length
288                             *
289                             *       The following defines our stack area.
290                             *
291 0 00000174 000001F4             DS.B    500                 500 bytes of stack
292 0          00000368     STACK   EQU     *
293 0 00000368 00000002             DS.B    2
294                             *
295                             *       There ain't no more.
296                             *
297 0          00000000             END     START

****** TOTAL ERRORS      0--
****** TOTAL WARNINGS    0--
```

CHAPTER 4

CHARACTER DISPLAY GENERATION

4.1 INTRODUCTION

This chapter describes the VME/10 character display generation functions. This information will permit the user to control the character display by the use of the SCM control registers, and to reconfigure the character set to a specific application.

Included in this chapter are initialization routines that are shipped with the VME/10 software package, and which configure the system in a specific way. This configuration is referred to as the "shipped software package".

4.2 HARDWARE DESCRIPTION

This section describes the applicable SCM hardware circuits that control the VME/10 character display. These circuits are as follows:

a. Display RAM
b. Control registers
c. Character generator RAM
d. CRT Controller (CRTC)

4.2.1 Display RAM

The display RAM is an array of characters and associated attributes which contains information that is to be displayed on the CRT monitor. The base address of the display RAM is $F17000; the top address is $F18FFE. The as-shipped configuration of the VME/10 includes only half of possible display RAM, ending at $F17FFE. The user has the option of installing the other half of this memory in the proper socket if the display application requires it.

The display RAM contains up to 4000 words (as shipped 2000 words), each of which contains the data required to display one character. The display logic in the VME/10 dedicates an attribute to each character rather than to fields. The code for the character to be displayed is defined in one-half of the word, while the attributes for this character are defined in the other half. This implementation does not require any CRT space to contain the attribute.

The display RAM character word is defined as follows:

bits 0-6 — Code for one of 128 possible characters. The shipped software package uses the 7-bit ASCII code to define the requested character.

bit 7 — A user optional display control bit. The shipped software package uses this bit as a TAB flag.

bits 8-10 — Control the color or the intensity of the character defined by bits 0-6. When a monochrome monitor is used, a value of 0 in these bits sets the display to the lowest intensity, while a 7 sets the display to the highest intensity. When a color monitor is used, the value of these bits select the colors as defined in Table 4-1.

TABLE 4-1. Color Control

COLOR	Bit 10	Bit 9	Bit 8
Black	0	0	0
Red	0	0	1
Blue	0	1	0
Magenta	0	1	1
Green	1	0	0
Yellow	1	0	1
Cyan	1	1	0
White	1	1	1

bit 11 When set, associated character video is inverted.

bit 12 When set, associated character is underlined.

bit 13 When set, associated character blinks.

bit 14 When set, associated character is displayed on the CRT.

bit 15 User optional display control bit. The shipped software package assigns this bit to be a character protect flag when set.

For example, to display the character A, to make it green with a black background, to underline it, and to make it blink, set the character word to be:

bit 15 = 0	No protect
bit 14 = 1	Set to display
bit 13 = 1	Set the blink function
bit 12 = 1	Underline the character
bit 11 = 0	Normal video
bit 10 = 1	Green on
bit 09 = 0	Blue off
bit 08 = 0	Red off
bit 07 = 0	No tab
bits 06-00 = 1000001	ASCII code for the uppercase A

The equivalent hexadecimal value for the word is $7441. Storing this word in the display RAM results in the display of an A formatted as defined above, assuming that the character generator RAM is initialized to the ASCII character set.

4.2.2 Control Registers

The VME/10 provides user control over the general format of the CRT character display through SCM control registers. These control bits will set character display attributes for the entire screen, rather than for individual characters. Table 4-2 lists the various control bits and functions. Note that bit 0 is the Least Significant Bit (LSB), and bit 7 is the Most Significant Bit (MSB).

TABLE 4-2. Character Display Control

CONTROL REGISTER	ADDRESS	BIT NUMBER BIT NAME	FUNCTION
CR0	$F19F05	bit 2 IVS	Setting this bit performs video inversion
		bit 3 DUTYCYCLE	When set, this bit corrects the BX syndrome by not displaying every other dot on each line. This prevents horizontal lines (such as those in the uppercase letter B) from standing out more than the nonhorizontal lines (such as those in the letter X). For brighter colors in the color monitor, this bit function should be turned off by clearing the bit.
		bit 4 CURBK	Setting this bit causes the cursor to blink.
		bits 5-7 CDIS1-3	These three bits provide a mask control over the three colors for character display. When cleared, all three colors are enabled to be displayed on the monitor. When set, CDIS1 masks the red, CDIS2 masks the blue, and CDIS3 masks the green. To clarify this function, consider a CRT that displays one red character, one blue, and one magenta. If CDIS1 is set masking the red, the red character is invisible, the blue remains blue, and the magenta also becomes blue (magenta-red). These bits have the same effect on a monochrome display where a certain intensity is masked.
CR1	$F19F07	bit 5, bit 6 S0,S1	The VME/10 provides three optional cursors that may be selected by setting these two bits. Full block cursor - bit 6=0, bit 5=0 Underline cursor - bit 6=0, bit 5=1 Frame cursor - bit 6=1, bit 5=0

4.2.3 Character Generator RAM

The character font is stored in the character generator RAM which starts at address $F14001 and ends at address $F14FFF and in which only the odd bytes of each memory word are active.

Each character is assigned a block of 16 bytes which will be stored in the character generator RAM in only the odd part of the word. Therefore, the offset between character blocks in the character generator RAM is 32, or $20.

As shipped, each character is defined as an array of up to twelve bytes; most characters use only nine bytes, leaving three bytes for descenders and ascenders. To design a character, lay out the desired array of pixels (dots), assign a logical 1 to each dot, a logical 0 to each blank, and then calculate the value of each horizontal line. Add four bytes with the value 0 to complete the character block (16 bytes).

Each character is assigned a code which is used as an offset into the character generator RAM. The character generator uses this offset to obtain the pixel matrix from the character generator RAM. When using the ASCII code, the base address of a character in the character generator RAM is calculated by multiplying the ASCII code of the character by the number of bytes assigned to each character in the memory map -- 32 or $20 (allowing for the unused even bytes in the memory map) -- and then adding the character generator RAM base address, F14001.

To change the dollar sign (ASCII $24) to the English pound sign, for example, proceed as follows:

a. Draw the character in an array of twelve rows, each containing eight squares:

R0								
R1						X	X	
R2					X			X
R3					X			
R4					X			
R5			X	X	X	X	X	
R6					X			
R7		X	X		X			
R8	X			X				
R9		X	X		X	X	X	X
R10								
R11								

b. Assign a binary value to each row:

```
R0  = 00000000 = $00
R1  = 00000110 = $06
R2  = 00001001 = $09
R3  = 00001000 = $08
R4  = 00001000 = $08
R5  = 00111110 = $3E
R6  = 00001000 = $08
R7  = 01101000 = $68
R8  = 10010000 = $90
R9  = 01101111 = $6F
R10 = 00000000 = $00
R11 = 00000000 = $00
```

c. Tag four more bytes to complete the character block.

```
R12 = 00000000 = $00
R13 = 00000000 = $00
R14 = 00000000 = $00
R15 = 00000000 = $00
```

c. Calculate the offset into the character generator RAM (in hexadecimal):

$24 x $20 + $F14001 = $F14481

d. Store the 16 bytes in the odd locations of the character generator RAM starting at address $F14481 and ending at address $F1449F.

4.2.4 CRT Controller (CRTC)

As shipped, the VME/10 has a display of 25 rows by 80 columns. It is possible to configure the MC6845 CRTC to produce other displays. The CRTC is configured by writing data into its control registers, residing at address $F1A023 of the memory map. Writing the requested control register number into location $F1A021 will select it.

4.3 SOFTWARE APPLICATION

This section presents two initialization program examples:

a. Listing 1 initializes the character generator RAM to the ASCII character set.

b. Listing 2 initializes the CRTC to control either a 25 x 80 or a 50 x 80 display.

```
                        ASCII    IDNT    1,00                ASCII character set          09/29/83

                        ************************************************************
                        *                                                          *
                        * Routine name:             ASCII                          *
                        *----------------------------------------------------------*
                        *              COPYRIGHTED 1983 BY MOTOROLA INC.           *
                        *----------------------------------------------------------*
                        * Current revision.....................1.00                *
                        * Date written.........................12-02-82            *
                        * Written by...........................S.Pri-Tal            *
                        * Date changed.........................                    *
                        * Changed by...........................                    *
                        *                                                          *
                        * Description of change:                                   *
                        *                                                          *
                        *                                                          *
                        ************************************************************
                        * Function: Define a character set ASCII $20-$FE for the   *
                        *           character generator, and move it into the RAM. *
                        *----------------------------------------------------------*
                        * Input parameters: none                                   *
                        *----------------------------------------------------------*
                        * Registers affected: none                                 *
                        *----------------------------------------------------------*
                        * External routines used: none                             *
                        ************************************************************
                                XDEF    ASCII

            00F14000    CRAM    EQU     $F14000             Base address of character generator RAM

            0000000B            SECTION 11
B 00000000 00000000     ASCII   DS.W    0
B 00000000 48E7E0C0             MOVEM.L D0-D2/A0-A1,-(A7)   Save user's values
B 00000004 227C00F14000         MOVE.L  #CRAM,A1            Base of character generator RAM
                        *
                        * Clear the character generator RAM up to the 33 character block.
                        * The 32 character is ascii $20 which is a space.
                        *
B 0000000A 323C0083             MOVE.W  #33*16/4-1,D1       Number of long words to move
B 0000000E 4280                 CLR.L   D0
B 00000010              SPACES
B 00000010 01C90001             MOVEP.L D0,1(A1)            All undefined characters will be
B 00000014 43E90008             LEA     8(A1),A1            initialized tp spaces
B 00000018 51C9FFF6             DBRA    D1,SPACES

B 0000001C 41FA0028             LEA     ASCII21(PC),A0      Base address of character table
B 00000020 323C005D             MOVE.W  #TABLEND-ASCII21/12-1,D1 Number of characters to move
                        *
                        * Write the character table for ascii $21 through ascii $7E.
                        * The 4 required 0 bytes are tagged in the code, thus saving 4 bytes per
                        * character in the table.
                        *
B 00000024 343C0002             MOVE.W  #2,D2
B 00000028              MOVE
B 00000028 2018                 MOVE.L  (A0)+,D0            Get 4 bytes from the table
B 0000002A 01C90001     MOVE1   MOVEP.L D0,1(A1)            Move only to odd addresses
B 0000002E 43E90008             LEA     8(A1),A1
```

```
 59 B 00000032 51CAFFF4            DBRA     D2,MOVE              After 3 long words (12 bytes)
 60 B 00000036 4280                CLR.L    D0                   Tag 4 bytes of 0
 61 B 00000038 343C0003            MOVE.W   #3,D2                Init the counter
 62 B 0000003C 51C9FFEC            DBRA     D1,MOVE1             Repeat till all bytes are moved
 63 B 00000040 4CDF0307            MOVEM.L  (A7)+,D0-D2/A0-A1    Restore use's values
 64 B 00000044 4E75                RTS                           Return to caller
 65                          *
 66                          * Character table for the ascii character set
 67                          *
 68 B 00000046 001010101010 ASCII21  DC.B     $00,$10,$10,$10,$10,$10,$10,$10,$00,$10,$00,$00 !
 69 B 00000052 002424240000 ASCII22  DC.B     $00,$24,$24,$24,$00,$00,$00,$00,$00,$00,$00,$00 "
 70 B 0000005E 00242424FF24 ASCII23  DC.B     $00,$24,$24,$24,$FF,$24,$FF,$24,$24,$24,$00,$00 #
 71 B 0000006A 00187F98987E ASCII24  DC.B     $00,$18,$7F,$98,$98,$7E,$19,$19,$FE,$18,$00,$00 $
 72 B 00000076 0041A3460C18 ASCII25  DC.B     $00,$41,$A3,$46,$0C,$18,$30,$62,$C5,$82,$00,$00 %
 73 B 00000082 007884844830 ASCII26  DC.B     $00,$78,$84,$84,$48,$30,$49,$86,$86,$79,$00,$00 &
 74 B 0000008E 001010200000 ASCII27  DC.B     $00,$10,$10,$20,$00,$00,$00,$00,$00,$00,$00,$00 '
 75 B 0000009A 000810202020 ASCII28  DC.B     $00,$08,$10,$20,$20,$20,$20,$20,$10,$08,$00,$00 (
 76 B 000000A6 000804020202 ASCII29  DC.B     $00,$08,$04,$02,$02,$02,$02,$02,$04,$08,$00,$00 )
 77 B 000000B2 0010925438FE ASCII2A  DC.B     $00,$10,$92,$54,$38,$FE,$38,$54,$92,$10,$00,$00 *
 78 B 000000BE 0000101010FE ASCII2B  DC.B     $00,$00,$10,$10,$10,$FE,$10,$10,$10,$00,$00,$00 +
 79 B 000000CA 000000000000 ASCII2C  DC.B     $00,$00,$00,$00,$00,$00,$00,$00,$60,$60,$20,$40 ,
 80 B 000000D6 0000000000FF ASCII2D  DC.B     $00,$00,$00,$00,$00,$FF,$00,$00,$00,$00,$00,$00 -
 81 B 000000E2 000000000000 ASCII2E  DC.B     $00,$00,$00,$00,$00,$00,$00,$00,$60,$60,$00,$00 .
 82 B 000000EE 000001020408 ASCII2F  DC.B     $00,$00,$01,$02,$04,$08,$10,$20,$40,$80,$00,$00 /
 83 B 000000FA 007E83858999 ASCII30  DC.B     $00,$7E,$83,$85,$89,$99,$91,$A1,$C1,$7E,$00,$00 0
 84 B 00000106 000818280808 ASCII31  DC.B     $00,$08,$18,$28,$08,$08,$08,$08,$08,$7F,$00,$00 1
 85 B 00000112 007C82820408 ASCII32  DC.B     $00,$7C,$82,$82,$04,$08,$10,$20,$40,$FE,$00,$00 2
 86 B 0000011E 007E8101013E ASCII33  DC.B     $00,$7E,$81,$01,$01,$3E,$01,$01,$81,$7E,$00,$00 3
 87 B 0000012A 00040C142444 ASCII34  DC.B     $00,$04,$0C,$14,$24,$44,$FF,$04,$04,$04,$00,$00 4
 88 B 00000136 00FF8080FE01 ASCII35  DC.B     $00,$FF,$80,$80,$FE,$01,$01,$01,$81,$7E,$00,$00 5
 89 B 00000142 007E818080FE ASCII36  DC.B     $00,$7E,$81,$80,$80,$FE,$81,$81,$81,$7E,$00,$00 6
 90 B 0000014E 00FF81020408 ASCII37  DC.B     $00,$FF,$81,$02,$04,$08,$10,$10,$10,$10,$00,$00 7
 91 B 0000015A 007E8181817E ASCII38  DC.B     $00,$7E,$81,$81,$81,$7E,$81,$81,$81,$7E,$00,$00 8
 92 B 00000166 007E8181817F ASCII39  DC.B     $00,$7E,$81,$81,$81,$7F,$01,$01,$81,$7E,$00,$00 9
 93 B 00000172 000000606000 ASCII3A  DC.B     $00,$00,$00,$60,$60,$00,$00,$00,$60,$60,$00,$00 :
 94 B 0000017E 000000606000 ASCII3B  DC.B     $00,$00,$00,$60,$60,$00,$00,$00,$60,$60,$40,$80 ;
 95 B 0000018A 000408102040 ASCII3C  DC.B     $00,$04,$08,$10,$20,$40,$20,$10,$08,$04,$00,$00 <
 96 B 00000196 00000000FF00 ASCII3D  DC.B     $00,$00,$00,$00,$FF,$00,$FF,$00,$00,$00,$00,$00 =
 97 B 000001A2 002010080402 ASCII3E  DC.B     $00,$20,$10,$08,$04,$02,$04,$08,$10,$20,$00,$00 >
 98 B 000001AE 007E81810204 ASCII3F  DC.B     $00,$7E,$81,$81,$02,$04,$08,$08,$00,$08,$00,$00 ?
 99 B 000001BA 007E8199A5A5 ASCII40  DC.B     $00,$7E,$81,$99,$A5,$A5,$BE,$80,$80,$7E,$00,$00 @
100 B 000001C6 003C428181FF ASCII41  DC.B     $00,$3C,$42,$81,$81,$FF,$81,$81,$81,$81,$00,$00 A
101 B 000001D2 00FE4141417E ASCII42  DC.B     $00,$FE,$41,$41,$41,$7E,$41,$41,$41,$FE,$00,$00 B
102 B 000001DE 003E41808080 ASCII43  DC.B     $00,$3E,$41,$80,$80,$80,$80,$80,$41,$3E,$00,$00 C
103 B 000001EA 00FC42414141 ASCII44  DC.B     $00,$FC,$42,$41,$41,$41,$41,$41,$42,$FC,$00,$00 D
104 B 000001F6 00FF808080F8 ASCII45  DC.B     $00,$FF,$80,$80,$80,$F8,$80,$80,$80,$FF,$00,$00 E
105 B 00000202 00FF808080F8 ASCII46  DC.B     $00,$FF,$80,$80,$80,$F8,$80,$80,$80,$80,$00,$00 F
106 B 0000020E 003E41808088 ASCII47  DC.B     $00,$3E,$41,$80,$80,$80,$8F,$81,$41,$3E,$00,$00 G
107 B 0000021A 0081818181FF ASCII48  DC.B     $00,$81,$81,$81,$81,$FF,$81,$81,$81,$81,$00,$00 H
108 B 00000226 007C10101010 ASCII49  DC.B     $00,$7C,$10,$10,$10,$10,$10,$10,$10,$7C,$00,$00 I
109 B 00000232 003E08080808 ASCII4A  DC.B     $00,$3E,$08,$08,$08,$08,$08,$08,$08,$70,$00,$00 J
110 B 0000023E 0082848890E0 ASCII4B  DC.B     $00,$82,$84,$88,$90,$E0,$90,$88,$84,$82,$00,$00 K
111 B 0000024A 008080808080 ASCII4C  DC.B     $00,$80,$80,$80,$80,$80,$80,$80,$80,$FF,$00,$00 L
112 B 00000256 0081C3A59999 ASCII4D  DC.B     $00,$81,$C3,$A5,$99,$99,$81,$81,$81,$81,$00,$00 M
113 B 00000262 0081C1A19189 ASCII4E  DC.B     $00,$81,$C1,$A1,$91,$89,$85,$83,$81,$81,$00,$00 N
114 B 0000026E 003C42818181 ASCII4F  DC.B     $00,$3C,$42,$81,$81,$81,$81,$81,$42,$3C,$00,$00 O
115 B 0000027A 00FE818181FE ASCII50  DC.B     $00,$FE,$81,$81,$81,$FE,$80,$80,$80,$80,$00,$00 P
116 B 00000286 003C42818181 ASCII51  DC.B     $00,$3C,$42,$81,$81,$81,$81,$85,$42,$3D,$00,$00 Q
```

```
117 B 00000292 00FE818181FE ASCII52  DC.B    $00,$FE,$81,$81,$81,$FE,$80,$84,$82,$81,$00,$00 R
118 B 0000029E 007E8180807E ASCII53  DC.B    $00,$7E,$81,$80,$80,$7E,$01,$01,$81,$7E,$00,$00 S
119 B 000002AA 00FE10101010 ASCII54  DC.B    $00,$FE,$10,$10,$10,$10,$10,$10,$10,$10,$00,$00 T
120 B 000002B6 008181818181 ASCII55  DC.B    $00,$81,$81,$81,$81,$81,$81,$81,$81,$7E,$00,$00 U
121 B 000002C2 008181818181 ASCII56  DC.B    $00,$81,$81,$81,$81,$81,$81,$42,$24,$18,$00,$00 V
122 B 000002CE 008181818199 ASCII57  DC.B    $00,$81,$81,$81,$81,$99,$99,$A5,$C3,$81,$00,$00 W
123 B 000002DA 008181422418 ASCII58  DC.B    $00,$81,$81,$42,$24,$18,$24,$42,$81,$81,$00,$00 X
124 B 000002E6 008282824428 ASCII59  DC.B    $00,$82,$82,$82,$44,$28,$10,$10,$10,$10,$00,$00 Y
125 B 000002F2 00FF02040810 ASCII5A  DC.B    $00,$FF,$02,$04,$08,$10,$20,$40,$80,$FF,$00,$00 Z
126 B 000002FE 001E10101010 ASCII5B  DC.B    $00,$1E,$10,$10,$10,$10,$10,$10,$10,$1E,$00,$00 [
127 B 0000030A 000080402010 ASCII5C  DC.B    $00,$00,$80,$40,$20,$10,$08,$04,$02,$01,$00,$00 \
128 B 00000316 007808080808 ASCII5D  DC.B    $00,$78,$08,$08,$08,$08,$08,$08,$08,$78,$00,$00 ]
129 B 00000322 001824428100 ASCII5E  DC.B    $00,$18,$24,$42,$81,$00,$00,$00,$00,$00,$00,$00 ^
130 B 0000032E 000000000000 ASCII5F  DC.B    $00,$00,$00,$00,$00,$00,$00,$00,$00,$FF,$00,$00 _
131 B 0000033A 000808040000 ASCII60  DC.B    $00,$08,$08,$04,$00,$00,$00,$00,$00,$00,$00,$00
132 B 00000346 000000003C02 ASCII61  DC.B    $00,$00,$00,$00,$3C,$02,$3E,$42,$42,$3D,$00,$00 a
133 B 00000352 004040405C62 ASCII62  DC.B    $00,$40,$40,$40,$5C,$62,$42,$42,$62,$5C,$00,$00 b
134 B 0000035E 000000003C42 ASCII63  DC.B    $00,$00,$00,$00,$3C,$42,$40,$40,$42,$3C,$00,$00 c
135 B 0000036A 000202023A46 ASCII64  DC.B    $00,$02,$02,$02,$3A,$46,$42,$42,$46,$3A,$00,$00 d
136 B 00000376 000000003C42 ASCII65  DC.B    $00,$00,$00,$00,$3C,$42,$7E,$40,$40,$3E,$00,$00 e
137 B 00000382 000C1210107C ASCII66  DC.B    $00,$0C,$12,$10,$10,$7C,$10,$10,$10,$10,$00,$00 f
138 B 0000038E 000000003A46 ASCII67  DC.B    $00,$00,$00,$00,$3A,$46,$42,$46,$3A,$02,$42,$3C g
139 B 0000039A 004040405C62 ASCII68  DC.B    $00,$40,$40,$40,$5C,$62,$42,$42,$42,$42,$00,$00 h
140 B 000003A6 000800001808 ASCII69  DC.B    $00,$08,$00,$00,$18,$08,$08,$08,$08,$1C,$00,$00 i
141 B 000003B2 000400000404 ASCII6A  DC.B    $00,$04,$00,$00,$04,$04,$04,$04,$04,$04,$44,$38 j
142 B 000003BE 004040404448 ASCII6B  DC.B    $00,$40,$40,$40,$44,$48,$70,$48,$44,$42,$00,$00 k
143 B 000003CA 001808080808 ASCII6C  DC.B    $00,$18,$08,$08,$08,$08,$08,$08,$08,$1C,$00,$00 l
144 B 000003D6 000000007649 ASCII6D  DC.B    $00,$00,$00,$00,$76,$49,$49,$49,$49,$49,$00,$00 m
145 B 000003E2 000000005C62 ASCII6E  DC.B    $00,$00,$00,$00,$5C,$62,$42,$42,$42,$42,$00,$00 n
146 B 000003EE 000000003C42 ASCII6F  DC.B    $00,$00,$00,$00,$3C,$42,$42,$42,$42,$3C,$00,$00 o
147 B 000003FA 000000005C62 ASCII70  DC.B    $00,$00,$00,$00,$5C,$62,$42,$42,$62,$5C,$40,$40 p
148 B 00000406 000000003A46 ASCII71  DC.B    $00,$00,$00,$00,$3A,$46,$42,$42,$46,$3A,$02,$02 q
149 B 00000412 000000005C62 ASCII72  DC.B    $00,$00,$00,$00,$5C,$62,$40,$40,$40,$40,$00,$00 r
150 B 0000041E 000000003C42 ASCII73  DC.B    $00,$00,$00,$00,$3C,$42,$30,$0C,$42,$3C,$00,$00 s
151 B 0000042A 000010107C10 ASCII74  DC.B    $00,$00,$10,$10,$7C,$10,$10,$10,$12,$0C,$00,$00 t
152 B 00000436 000000004242 ASCII75  DC.B    $00,$00,$00,$00,$42,$42,$42,$42,$46,$3A,$00,$00 u
153 B 00000442 000000004444 ASCII76  DC.B    $00,$00,$00,$00,$44,$44,$44,$44,$28,$10,$00,$00 v
154 B 0000044E 000000004141 ASCII77  DC.B    $00,$00,$00,$00,$41,$41,$49,$49,$49,$36,$00,$00 w
155 B 0000045A 000000004224 ASCII78  DC.B    $00,$00,$00,$00,$42,$24,$18,$18,$24,$42,$00,$00 x
156 B 00000466 000000004242 ASCII79  DC.B    $00,$00,$00,$00,$42,$42,$42,$46,$3A,$02,$42,$3C y
157 B 00000472 000000007E04 ASCII7A  DC.B    $00,$00,$00,$00,$7E,$04,$08,$10,$20,$7E,$00,$00 z
158 B 0000047E 000E10101020 ASCII7B  DC.B    $00,$0E,$10,$10,$10,$20,$10,$10,$10,$0E,$00,$00 {
159 B 0000048A 001010100000 ASCII7C  DC.B    $00,$10,$10,$10,$00,$00,$00,$10,$10,$10,$00,$00 |
160 B 00000496 007008080804 ASCII7D  DC.B    $00,$70,$08,$08,$08,$04,$08,$08,$08,$70,$00,$00 }
161 B 000004A2 000030490600 ASCII7E  DC.B    $00,$00,$30,$49,$06,$00,$00,$00,$00,$00,$00,$00 ~
162 B          000004AE     TABLEND EQU     *
163                                 END

****** TOTAL ERRORS      0--
****** TOTAL WARNINGS    0--
```

```
1                       CRTCINIT IDNT    1,00                CRT Controller initialization routine

3                       ************************************************************
4                       *                                                          *
5                       * Routine name:             CRTCINIT                       *
6                       *----------------------------------------------------------*
7                       *              COPYRIGHTED 1983 BY MOTOROLA INC.           *
8                       *----------------------------------------------------------*
9                       * Current revision.....................1.00                *
10                      * Written by...........................S. Pri-Tal          *
11                      * Date written.........................12-30-82            *
12                      * Date changed.........................09-14-83            *
13                      * Changed by...........................                    *
14                      *                                                          *
15                      * Description of change:                                   *
16                      *                                                          *
17                      *                                                          *
18                      *                                                          *
19                      ************************************************************
20                      * Function: Initialize the CRTC to control a 25 or a 50    *
21                      *          line by 80 chcracter CRT.                       *
22                      *                                                          *
23                      *----------------------------------------------------------*
24                      * Input parameters:                                        *
25                      *               if D0.B = 0  25 by 80                      *
26                      *               if D0.B = 1  50 by 80                      *
27                      *----------------------------------------------------------*
28                      * Registers affected: none                                 *
29                      *----------------------------------------------------------*
30                      * External routines used: none                             *
31                      *----------------------------------------------------------*
32                      * Additional XDEF's: none                                  *
33                      ************************************************************

35                              XDEF    CRTCINIT

37          00F1A021    CRTCADD EQU     $F1A021
38          00F1A023    CRTCREG EQU     $F1A023

40          00000008            SECTION 11
41 B 00000000 00000000  CRTCINIT DS.W    0
42 B 00000000 48E74080          MOVEM.L A0/D1,-(A7)         Save caller's values
43 B 00000004 223C0000000F      MOVE.L  #TABLEND-TABLE/2,D1 Set D1 to number of words to move
44 B 0000000A 4A00              TST.B   D0                  If = 0, init to 25 by 80
45 B 0000000C 6706              BEQ.S   SMALL
46 B 0000000E 41FA003E          LEA     TABLE2(PC),A0       Get base address of 50 by 80 table
47 B 00000012 6004              BRA.S   LOOP
48 B 00000014               SMALL
49 B 00000014 41FA0018          LEA     TABLE(PC),A0        Get base address of 25 by 80 table
50 B 00000018               LOOP
51 B 00000018 13D800F1A021      MOVE.B  (A0)+,CRTCADD       Select the register
52 B 0000001E 13D800F1A023      MOVE.B  (A0)+,CRTCREG       Initialize it
53 B 00000024 51C9FFF2          DBRA    D1,LOOP             Repeat till all registers are inited.
54 B 00000028 4CDF0102          MOVEM.L (A7)+,A0/D1         Restore caller's values
55 B 0000002C 4E75              RTS
```

```
                        *
                        * Initialization table for 25 lines by 80 characters
                        *
0000002E                TABLE
0000002E 0062                   DC.W    $0062           Total characters per line   =  98 ($62)
00000030 0150                   DC.W    $0150           Characters displayed        =  80 ($50)
00000032 0256                   DC.W    $0256           Blank characters to start   =   8
00000034 0311                   DC.W    $0311           Characters per sync         =  17 ($11)
00000036 0419                   DC.W    $0419           Lines per screen            =  25 ($19)
00000038 0503                   DC.W    $0503           Fraction of above           =  03
0000003A 0619                   DC.W    $0619           Lines per screen            =  25 ($19)
0000003C 0719                   DC.W    $0719
0000003E 0800                   DC.W    $0800
00000040 090B                   DC.W    $090B           Rows per character-1        =  11 ($0B)
00000042 0A00                   DC.W    $0A00           Cursor start register
00000044 0B0F                   DC.W    $0B0F           Cursor end register
00000046 0C00                   DC.W    $0C00
00000048 0D00                   DC.W    $0D00
0000004A 0E00                   DC.W    $0E00           Cursor address H
0000004C 0F00                   DC.W    $0F00           Cursor address L
         0000004D       TABLEND  EQU     *-1
                        *
                        * Initialization table for 50 lines by 80 characters
                        *
0000004E                TABLE2
0000004E 0062                   DC.W    $0062           Total characters per line   =  98 ($62)
00000050 0150                   DC.W    $0150           Characters displayed        =  80 ($50)
00000052 0256                   DC.W    $0256           Blank characters to start   =   8
00000054 0311                   DC.W    $0311           Characters per sync         =  17 ($11)
00000056 0432                   DC.W    $0432           Lines per screen            =  50 ($32)
00000058 0502                   DC.W    $0502           Fraction of above           =  02
0000005A 0631                   DC.W    $0631           Lines per screen - 1        =  49 ($31)
0000005C 0731                   DC.W    $0731
0000005E 0803                   DC.W    $0803
00000060 090B                   DC.W    $090B           Rows per character-1        =  11 ($0B)
00000062 0A00                   DC.W    $0A00           Cursor start register
00000064 0B0F                   DC.W    $0B0F           Cursor end register
00000066 0C00                   DC.W    $0C00
00000068 0D00                   DC.W    $0D00
0000006A 0E00                   DC.W    $0E00           Cursor address H
0000006C 0F00                   DC.W    $0F00           Cursor address L
                                END

****** TOTAL ERRORS      0--
****** TOTAL WARNINGS    0--
```

SUGGESTION/PROBLEM REPORT

MICROSYSTEMS

QUALITY • PEOPLE • PERFORMANCE

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Inc.
Microsystems
2900 S. Diablo Way
Tempe, Arizona 85282
Attention: Publications Manager
Maildrop DW164

Product: _____ Manual: _____

COMMENTS: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please Print

Name _____ Title _____

Company _____ Division _____

Street _____ Mail Drop _____ Phone _____

City _____ State _____ Zip _____

For Additional Motorola Publications
Literature Distribution Center
616 West 24th Street
Tempe, AZ 85282
(602) 994-6561

Microsystems Field Service Support
(800) 528-1908
(602) 829-3100

MOTOROLA

MOTOROLA ***Semiconductor Products Inc.***

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

16302 PRINTED IN USA (3/84) MPS 4000