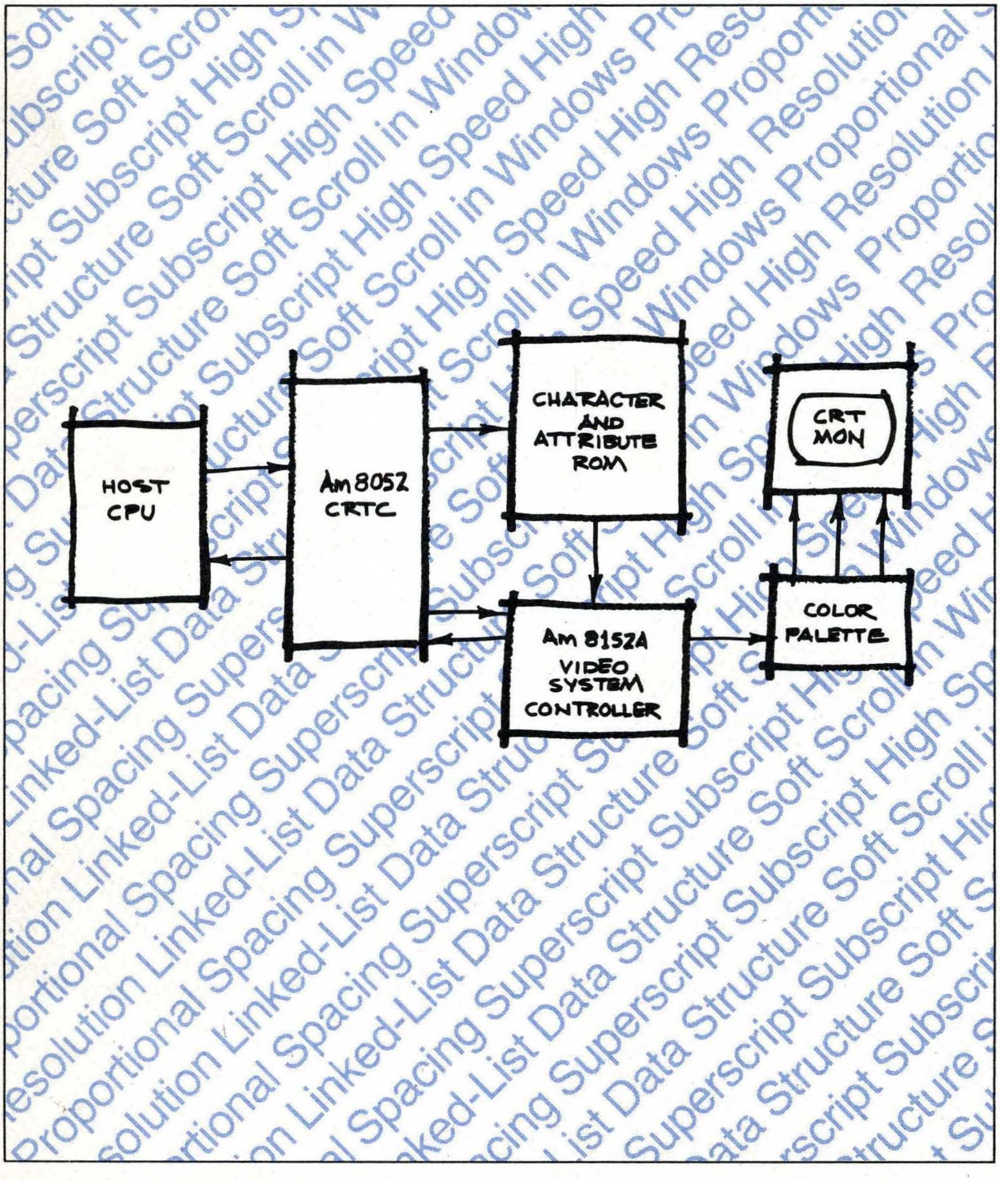
The Am8052 CRT Controller

Technical Manual



ADVANCED MICRO DEVICES

Advanced Micro Devices

Am8052
Alphanumeric
CRT
Controller

Technical Manual

Advanced Micro Devices reserves the right to make changes in its products without notice in order to improve design or performance characteristics. The company assumes no responsibility for the use of any circuits described herein.

901 Thompson Place, P.O. Box 3453, Sunnyvale, California 94088
(408) 732-2400 TWX: 910-339-9280 TELEX: 34-6306

Author	:	Jürgen Stelbrink	
Contributors	:	Ka Wai Leung	Ch. 3
		Olivier Garbe	Ch. 5.1; 5.2
		Robert Earley	Ch. 5.3
		Mark Young	Ch. 6.3
		Joe Brcich	Ch. 6.5
		Hans Joachim Rühl	Ch. 6.6
Copyeditor	:	Harry Lau	

Table of Contents

Appendices

CHAPTER 1

CRT DISPLAY PRODUCTS

1.0 INTRODUCTION

Raster-scan CRT (Cathode Ray Tube) displays form the principle communication link between computers and users in business, science and educational applications. The trend toward using high-resolution displays to enhance information transfers between man and machine is accelerating.

As CRT terminals become increasingly sophisticated, the designer is faced with many new problems in areas of data manipulation and display. The high-resolution screen necessary to display a full-size typewriter page results in pixel rates exceeding 50 MHz. Additionally, the use of microprocessor technology in modern terminal designs has transferred the editing tasks from the host system to the terminal itself.

CRT terminal designs can be divided into two categories. Alphanumeric terminals are used in office workstations. They incorporate features such as flexible attribute handling, proportional spacing of characters, split-screens or multiple window display, smooth-scrolling of windows, and variable character width and height in full-page, 132x60 screen formats. The video subsystem of a CRT terminal with these sophisticated features can be implemented with as few as three devices. This significantly reduces IC and system development cost and board space without sacrificing performance. The three devices consist of the Am8052 Alphanumeric CRT Controller (CRTC), the Am8152A Video System Controller (VSC), and a character font generator. This subsystem talks to the system bus on one side and generates a high-speed pixel stream on the other. This chip set is subject of this handbook.

Terminals of the second category employ a bit-mapped graphic display. The main application area for these terminals are engineering workstations in CAD/CAM systems. In bit-mapped displays, each pixel can be set or reset independently. A graphic controller with a high processing power is needed to update a high-resolution screen containing more than one million pixels in a reasonable time. The Am815x family supports this kind of application.

New designs of high-end alphanumeric CRT systems tend to use bit- mapped displays because of the flexibility. However, because of the high processing power needed to generate the display and the large display memory storing the bit-map, an alphanumeric terminal based on bit-mapped graphic is more expensive and takes up more board space than a dedicated, alphanumeric CRT subsystem based on the CRT Controller chip set. On the other side, a CRTC-based system can handle limited bit-mapped graphics to display pie charts or bar graphs in business-type applications.

1.1 ALPHANUMERIC DISPLAY PRODUCTS

Figure 1.1 shows a typical proportional-spacing application based on the CRT Controller chip set. The distinctive characteristics of this subsystem are as following:

- Up to 80 MHz video dot rate for high-resolution, flicker-free displays.
- Linked-list display data structure in system memory simplifying text-editing tasks.
- Background or window smooth-scroll capability without external MSI or software overhead.
- User-friendly, 16-bit CPU interface. Compatible with 8086, Z8000, and 68000 CPUs. 16-Mbyte memory addressing capability.

The chip set capabilities are contributed to the CRTC and VSC as described below:

Am8052. The CRT Controller (CRTC), is a general-purpose interface device for raster scan CRT displays. The on-chip DMA controller interprets a linked-list data structure in system memory defining the text displayed on the screen. This simplifies text-editing tasks. It supports attributes such as subscript, superscript, underline, multiple cursors and blinking. User-definable attributes provide flexibility. Windows and background can be smooth- scrolled at user-definable rates.

The CRTC is register-oriented and fully user-programmable. The frame timing and operating mode are initialized by the host CPU.

Am8152A. The Video System Controller is basically a programmable (2- to 17-bit) shift register.

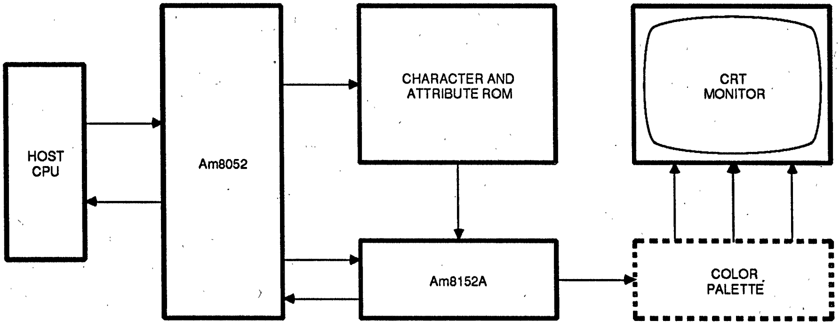


Figure 1-1 Typical Application

05098B 1-1

It serializes the character slices supplied by the character font generator. Attributes such as highlight and reverse video are incorporated in the serial pixel stream put out. The VSC provides two video outputs: a two-bit digital output and a four-level analog (composite) video output. An on-chip, crystal-driven oscillator provides the pixel shift clock (dot clock), the character clock, and the system clock.

1.2 ADVANCED DISPLAY FEATURES

State-of-the-art, letter-quality printers support fancy text display features such as proportional spacing with block justification and double print. Workstations for word processing should be able to display edited text on the screen that looks like the print-out of these letter-quality printers, in order to make the word processing task more ergonomical for the operator. For example, it is intolerable that some workstations display the beginning and end of an underline with a special character sequence instead of simply underlining the string. Additionally, it should support features like highlighting, which is equivalent to double print in case of a printer, blinking of characters and multiple cursors to emphasize parts of the text.

Vertical smooth-scroll will become a standard feature of future designs. Smooth-scrolling is much more ergonomical for the user. Also helpful are windows (overlaid on the displayed page) to provide temporary information about issued commands.

Additionally, a CRT controller should supply a display data structure organized as a linked-list in system memory. However, the editing response time is shorter compared to a system using linear data structures.

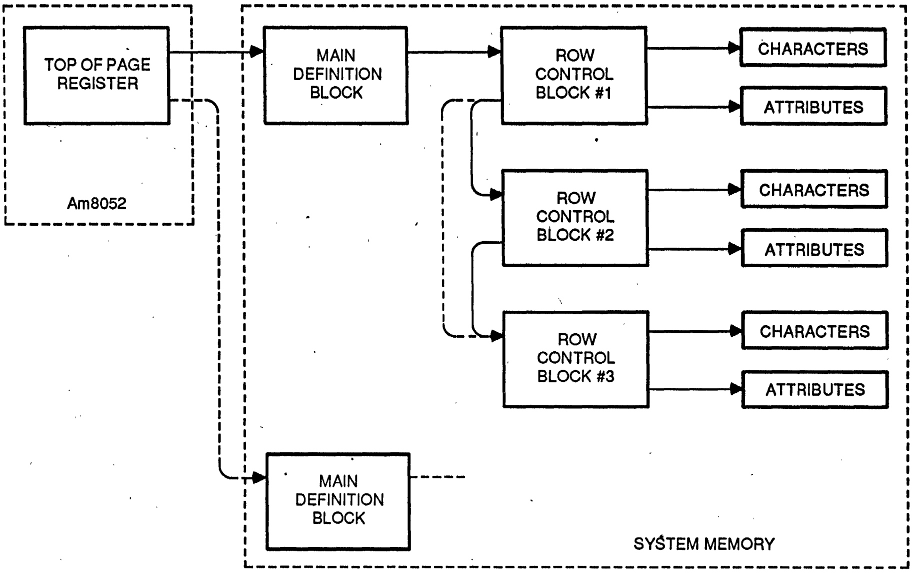
The features expected of a state-of-the-art CRT controller will now be discussed in more detail. The CRT controller chip set implements all these features in silicon.

Linked-List Data Structure

In standard CRT subsystems the display data is organized as contiguous memory blocks. These blocks are associated with video frames and stored in special memory called video refresh memory. When editing tasks like character or line insertion or deletion are to be executed, the CPU has to move blocks of the display data. This time-consuming operation slows down the editing process.

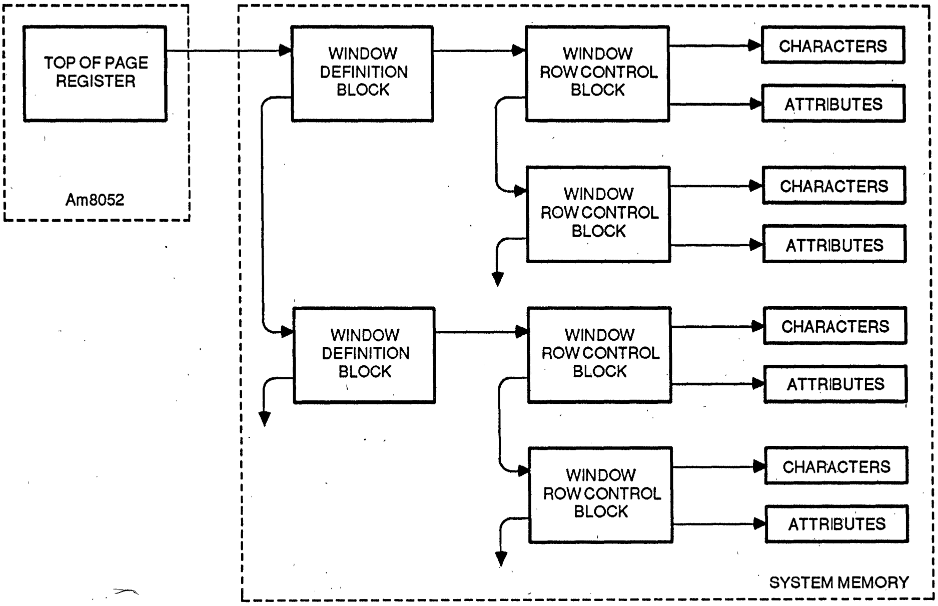
Text editing becomes much more elegant and faster when operating on a linked-list data structure where the display data is organized in small strings, usually rows, glued together by pointers (Figures 1.2, 1.3). The advantage of the linked-list data structure becomes obvious when looking at the execution speed of editing tasks. A line can be inserted or deleted by modifying one pointer instead of moving half the screen down, thereby increasing the execution speed significantly. Pages can be swapped simply by altering one pointer.

The linked-list data structure has a second advantage: If the display data is stored in the main system memory the CRT controller can directly fetch the data from the list the word processor is



05098B 1-2

Figure 1-2 Linked-List Display Data Management: Background



05098B 1-3

Figure 1-3 Linked-List Display Data Management: Windows

operating on. This eliminates the need of setting up a special list of display data.

In an Am8052-based video system, the display data is stored in system memory and is easily accessible by the host CPU when executing display-editing tasks. The display data consisting of characters and their attributes is grouped into strings called segments. One or more segments build up a row. These segments are tied together by a linear list of pointers containing in Row Control Blocks. Each Row Control Block holds all information relevant to describe an entire character row on the screen. Row Control Blocks again are chained via pointers; each block points to its successor.

One block located at the top of the linked-list defines screen attributes such as cursor type, blink rate, and positioning. This Main Definition Block is pointed to by a pointer stored inside the CRTC.

The CRTC interprets the linked-list and transfers the character code strings and attributes sequentially to the character font generator. The character slice output of the character font generator is then serialized by the companion part of the Am8052, the Video System Controller (VSC), and sent to the monitor.

Windows

Windows are text blocks overlaying the background to provide temporary information for the viewer. Windows can be displayed or removed without corrupting the background. Windows are defined by a linked-list data structure similar to the background data structure. The Am8052 can support any number of windows as long as they are vertically separated by at least two character rows. Any number of windows or the background may be scrolled.

The Top of Window register inside the Am8052 points to the beginning of the window linked-list, the Window Definition Block for the top-most window. The Window Definition logically is similar to the Main Definition Block of the background; it contains the general characteristics of this particular window (for example, size and positioning).

Each Window Definition Block links to the next Window Definition Block. Window Definition Blocks need to be arranged in the sequence the windows are supposed to appear on the display (the topmost window first, the bottom window last).

The Window Row Control Block pointer located in the Window Definition Block links to the first Window Row Control Block which is similar to the background Row Control Block. The row segmentation feature is also available for windows.

Virtual Windows or Split Screens

Although the rules of window positioning do not permit overlapping or adjacent windows, the background and window data structures can be used to implement virtual horizontally or vertically aligned windows. This can be best described using the illustration in Figure 1.4. This sample display consists of two rows with each two segments: “ONE” and “TWO,” “THREE” and “FOUR.” The user wishes to be able to scroll any of these segments at a given time. The window positioning rules do not permit assignment of all four segments as windows. However, any of these four segments can be dynamically assigned to be a window; anyone of these windows can be scrolled independently from the other three. This gives the viewer the illusion of aligned windows.

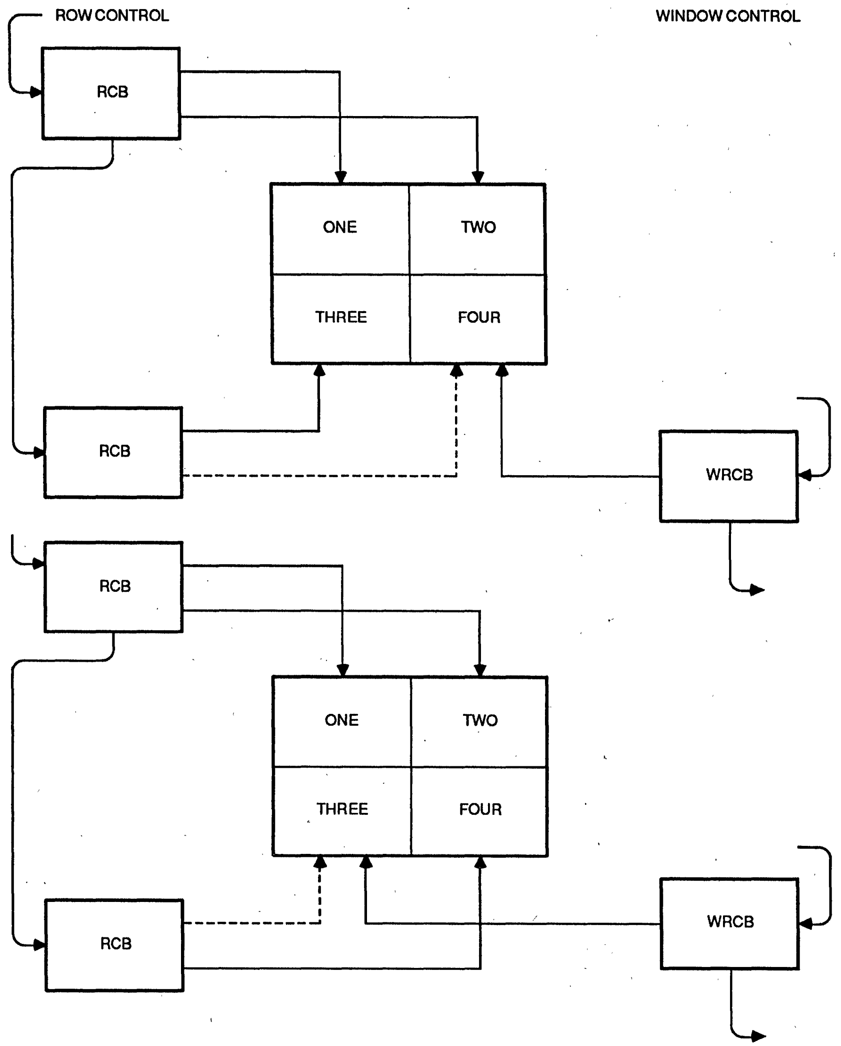
Smooth-Scrolling

Vertical smooth-scrolling is the gradual replacement of a character row on a scan line by scan line basis. The visual effect is more eye-pleasing to the viewer and will become an ergonomical requirement for future terminal designs. The smooth- scroll of the entire screen is a relatively easy task and can be accomplished with a minimum of hardware. However, smooth-scrolling an overlaid window or smooth-scrolling the background when displaying windows is a much more sophisticated task. If a window is smooth-scrolled, text seems to appear and disappear within the window while the background stays absolutely stable (Figure 1.5). If, on the other hand, the background is scrolled, then the background text will appear to pass under the window.

Vertical smooth-scrolling of the background or of windows is executed requiring very few interactions of the host CPU. Only when a row is totally scrolled in or out does the CRTC interrupt the CPU to relink the data structure. The scroll rate being programmable covers the range from very low-speed scrolling, where the eye can identify the scan line stepping, to high-speed scrolling, where the text moves too fast to be readable. The medium speed gives the smoothest effect.

Attributes

There are three kinds of attributes which are distinguished by the number of characters to which they correspond. The screen attributes, such as smooth-scroll rate, cursor style, and blink rate, effect the text display of the entire screen. Row



05098B 1-4

Figure 1-4 Virtual Window or Split Screens

attributes, such as scan line count and character positioning within the character cell, are valid for entire character rows. The third kind of attribute is directly associated with particular characters or character strings. Examples of character attributes are: highlight, underline, blinking; subscript and superscript.

Many CRT controllers treat characters and attributes in the same fashion; they fetch one attribute per character. This straightforward relation is also the easiest to handle by software. However, the price for this scheme is the increased bus occupancy of the CRT controller to fetch 24 bits per character compared to 8 bits per character in applications requiring no attribute fetches at all. Especially in high-end alphanumeric applications asking for maximum system performance, the system designer's goal is to keep bus occupancy as low as possible. This application asks for a more flexible and less bus time consuming attribute architecture.

Characters are typically uncorrelated along a character string. Attributes, on the other hand, are highly correlated; features such as reverse video affect a character string rather than individual characters. For this reason, a flexible correspondence between characters and attributes can save memory space and can reduce the bus occupancy.

In demand attribute mode, an attribute is loaded only if the attribute characteristics should be changed. A flag is inserted in the character list to instruct the CRT controller to fetch a new attribute word. This attribute word may apply either to the next character (unlatched attribute) or to all following characters not invoking attributes (latched attribute). This flag could either be a specific character which is not displayed on the screen or it could be any bit of the character code (usually the most significant bit). The first option allows a 255-character set with the trade-off that a flag character has to be inserted when the attribute characteristics are to be changed. The second option does not require this character string modification, but it halves the available character set (128 character codes).

The CRTC has been designed to allow a great versatility in attribute options. Ten attribute bits are predefined, four attribute bits are user-definable. If the number of user-definable attributes is not sufficient to satisfy the specific requirements of the application any predefined attributes may be redefined to increase the set of user-definable attributes. The predefined attributes are listed below:

Highlight. It causes the VSC to switch to the highest intensity level when displaying the characters.

Reverse. The color of the background and the foreground are exchanged. If the normal character appears white on a black background the reversed character will appear black on a white background.

Superscript. The character is shifted up a defined number of scan lines.

Subscript. The character is shifted down a defined number of scan lines.

Underline. The character is underlined, the position of the underline is programmable.

Strike Through. The affected character is struck through; sometimes this attribute is called shifted underline.

Blink. The affected character blinks at a programmable rate and duty cycle.

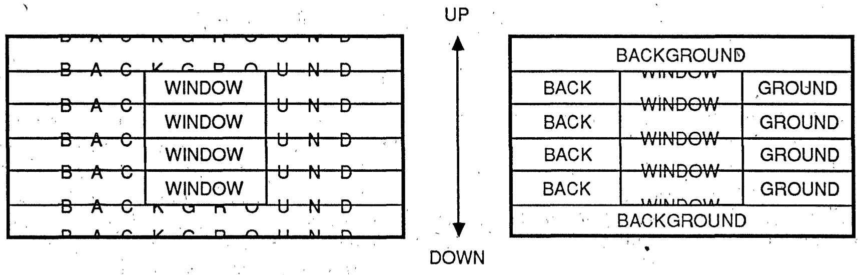
The internal processing of the attribute bits superscript and subscript may be disabled to access a special character font generator for displaying smaller subscript or superscript characters. The two attributes listed below cannot be redefined as user-definable attribute bits, since they do not correspond to an attribute port pin; they effect only the internal attribute processing.

Ignore. The character is not loaded into the line buffer; a character can be erased by setting this bit.

Latched. This attribute word applies to all following characters; it gets latched in the CRTC.

Proportional Spacing

Proportional spacing has become a standard feature of higher performance, letter-quality printers. In order to display a text on the screen similar to the printed text on paper, the CRT system should be



05098B 1-5

Figure 1-5 Smooth Scrolling

able to support proportional spacing.

Proportional spacing means that narrow characters such as "i" use less space in a character row than wider characters such as "W" (Figure 1.6). The screen is no longer divided into a raster of character fields. The number of characters which can be put into one line becomes a function of the characters itself. Summarized, it provides a typeset look of the text.

Text right-justification in proportional-spacing applications requires a user-definable number of blank pixels to be tailored to characters to get a straight right border of the text (Figure 1.7). Trailing blanks allow lines to be stretched smoothly and unnoticeably.

In proportional-spacing applications, the character font generator also stores, in parallel to the character font, the width of the individual character and passes this 4-bit value (2...17 pixels) to the Video System Controller which uses it to determine the divide ratio for the character clock. The character clock is modulated along the width of the characters in the string.

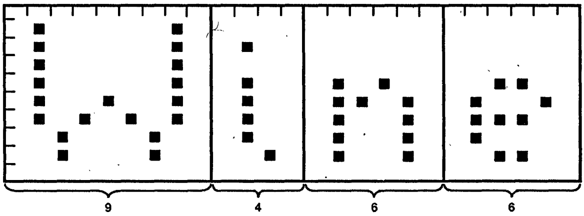
The system clock times the DMA transfers when the CRTC is bus master. In proportional-spacing applications, this clock is also used to determine the screen timing (screen blanking, horizontal and vertical sync timing), because the character clock rate no longer provides a constant clock for the counters.

Both the character and the system clock are divided from the dot clock. A crystal directly connected to the VSC controls the dot clock frequency. Internal PLL logic multiplies the crystal frequency by five to generate the dot clock. This allows the designer to use inexpensive crystals oscillating in fundamental mode even when generating dot clocks of 80 MHz.

Cursors

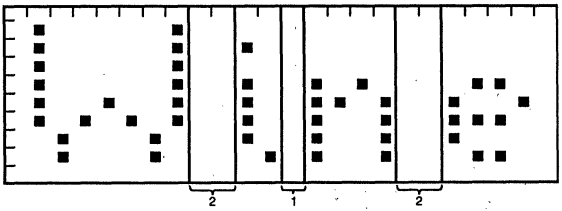
The Am8052 supports two kinds of cursors. The X-Y cursor appears on a programmable X-Y coordinate. This cursor is tied to this position on the screen. When a scroll occurs the cursor will still appear on the same location, but will apply to a new character. The second cursor type is specified via the character attribute word. The cursor is attached to a particular character and will move with the character when the text is scrolled. Due to the way the two cursors are specified, a screen may have only one X-Y cursor (the Main Definition Block can store only one pair of coordinates) and as many attribute cursors as there characters on the screen.

The cursor style is very flexible. Examples of cursor styles are as follows:



05098B 1-6

Figure 1-6 Proportional Spacing



05098B 1-7

Figure 1-7 Trailing Blanks

- Static or blinking underline
- Blinking by switching between normal display and blank
- Blinking by switching between normal display and reverse
- Reverse character

The X-Y cursor and the attribute cursor may have different styles to be able to distinguish them. For example, the X-Y cursor may be a blinking underline whereas the attribute cursor may reverse the character.

Host Bus Interface

The CRTC can easily be interfaced to most 16-bit system buses. In slave mode the CPU initializes the CRTC by programming the registers for the timing parameters. After being activated, the CRTC tries to gain the bus mastership to fill the line buffers and then starts displaying. The CRTC bus interface supports 24-bit linear address buses (68000, 8086) and 23-bit segmented address buses (Z8000).

CHAPTER 2

Am8052 ARCHITECTURE

2.1. OVERVIEW

The Am8052 can be used together with the Am8152A Video System Controller, which is specially designed to complement the Am8052 and enhance its displaying capabilities.

The Am8052, after initialization by the host processor, acts as a stand-alone device in the following manner:

- It fetches the data to be displayed from the main memory using its internal DMA controller.
- It manipulates the displayable character codes along with their attributes.
- It provides all the timing signals to synchronize beam-scanning with the character-pixel stream.
- It provides useful features such as size-programmable windows and vertical smooth-scroll.

The Am8052 is a real-time raster scan display controller that keeps track of updating the display screen on a character-row basis by toggling its internal row-buffers; one being displayed by the Display Control Unit while the other two are loaded through the DMA interface under control of the Row Management Unit.

All the above operations are synchronized by the Video Timing Control Unit and initialized by the host processor through bus interface logic. The Am8052 block diagram (Figure 2.1) shows the functional units and how they interface with each other.

Following reset, the Am8052 remains in Slave Mode and waits for the host processor to initialize the timing and control registers. It also waits for the host CPU to load a single register address, pointing to the start of the display data list in the host memory.

While in the idle state, the device holds both HSYNC and VSYNC signals inactive (LOW) to prevent undefined synchronization to the CRT which might damage high bandwidth tubes. It also holds the Blank signal active to inhibit the CRT beam.

Once the device has been initialized, and upon a command from the CPU, the DMA enters a bus request sequence to update the three internal row buffers whenever possible. A row buffer cannot be loaded at the same time that it is being displayed.

The Row Management Unit governs the loading of the characters to be displayed, as well as their attributes (whenever they are invoked), into the row buffers. This logic also updates the Display Control Registers (not accessible to the user), on a row by row basis, as specified by the Row Definition Blocks located in main memory.

With the beginning of Vertical Blank (VBLANK going High), the Am8052 terminates any processes/active from the current frame, and starts loading the information defining the next frame. It takes the Top Of Page Pointer stored in an internal register, and begins loading the Main Definition Block, the Window Definition Block (if present) and the first Row Control Block including character and attribute strings. By the end of vertical blank (VBLANK going Low) the Am8052 must have the first internal row buffer filled to ensure a flicker-free screen.

The Display Control Unit combines the character stream from one of the three row buffers with the row- or character-dependent display characteristics of these characters. As a result, the Display Control Unit provides, on R_0–R_4, the scan line address of the one currently being displayed, and outputs the sequence of character codes contained in this row, on CC_0–CC_7. These two values form the address sent to the Character Font Generator. The character code (most significant part of the address) points to the matrix of pixels synthesizing the character on the screen, while the scan line number (least significant part of the address) indicates which line of the matrix is to be displayed on the screen. The Character Font Generator provides the resultant line of pixels, which subsequently is serialized by the Video System Controller and processed according to the various attributes.

2.2. INTERFACE SIGNALS

With the exception of $CLK1_1$ and CLK_2 inputs, all inputs and outputs of the CRTC are TTL-compatible. Figure 2.2 shows the device pin-out.

V_{SS1}, V_{SS2} (Ground)
V_{CC1}, V_{CC2} (+5V Power Supply)

(For tolerance specification, refer to the DC characteristics)

CLK_1 (System Clock, Input)

The system clock controls the DMA and peripheral portion of the CRTC and times all memory accesses. It requires a timing duty cycle of about 50% at its highest frequency and is driven by an external timing source, usually the system/CPU clock. In proportional spacing applications, where the character clock (CLK_2) is variable, the system clock should be used to time the horizontal and vertical sync rates. CLK_1 is not TTL-compatible (for specifications refer to the DC characteristics). Figure 2.3 shows a CLK_1/CLK_2 driver generating a clock signal with the required High and Low levels.

CLK_2 (Character Clock, Input)

The character clock times the Character Code and Attribute outputs of the CRTC. In applications not using proportional spacing, CLK_2 is fixed in frequency and can, therefore, time horizontal and vertical sync (HSYNC and VSYNC). This allows CLK_1, the system clock, to be unrelated and asynchronous to the display timing. CLK_2 is not TTL-compatible.

AD_0–AD_{15} (Address/DataBus, Input/Output)

The Address/Data Bus is a time-multiplexed, bidirectional, active-High, three-state bus. The presence of addresses is indicated by Address Strobe ($\overline{AS}$); presence of data is indicated by Data Strobe ($\overline{DS}$). When the CRTC is in control of the system bus (Bus Master), it dominates the AD Bus. When the CRTC is idle (Bus Slave), the CPU or other external devices can control the AD Bus and may use it to access the internal registers of the CRTC. In upper address update cycles (Bus Master Write) the CRTC strobes out the new, most sig-

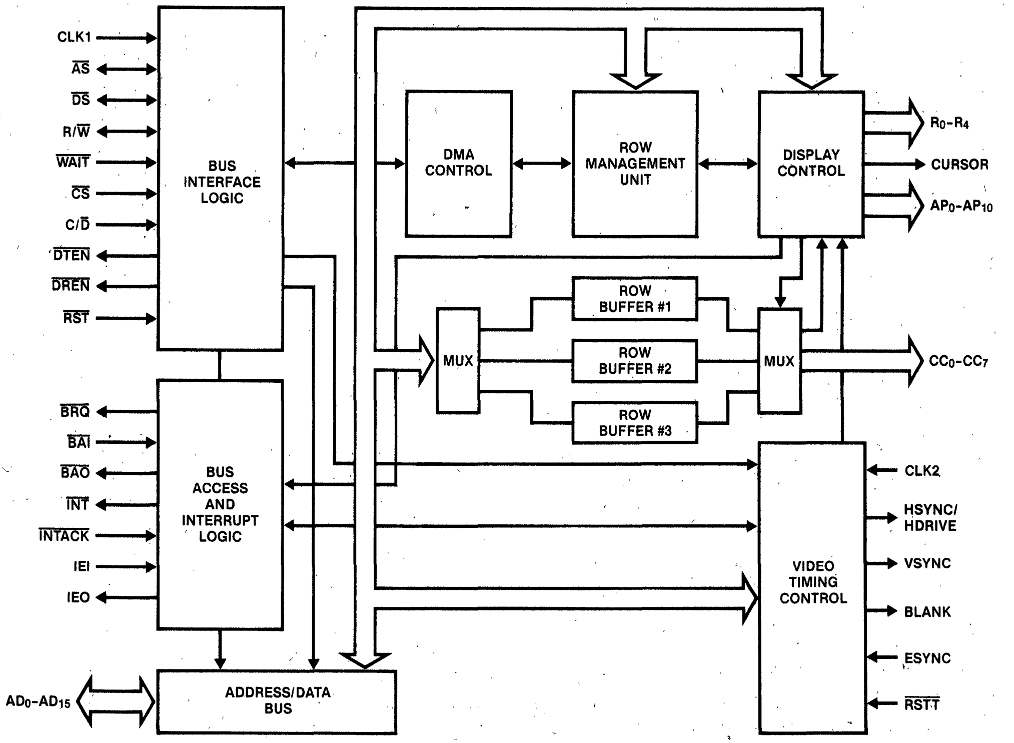


Figure 2-1 Am8052 Block Diagram

nificant part of the memory address (upper 7 or 8 bits). For both Linear and Segmented Addressing Mode, this address is output on AD$_0$-AD$_7$; the interrupt vector is also strobed out on AD$_0$–AD$_7$.

$\overline{\text{AS}}$ (Address Strobe, Input/Output, Active Low)

Address Strobe is a bidirectional, three-state signal. In Slave Mode, this input controls the internal transparent latches at the C/$\overline{\text{D}}$ and $\overline{\text{CS}}$ inputs. In multiplexed address/data bus systems, the rising edge of $\overline{\text{AS}}$ latches C/$\overline{\text{D}}$ and $\overline{\text{CS}}$. In demultiplexed address/data bus systems, $\overline{\text{AS}}$ may be held Low to make the above-mentioned latches transparent.

When the CRTC is the bus master, $\overline{\text{AS}}$ is an output indicating a valid address on the AD bus. The address may be latched with the rising edge of $\overline{\text{AS}}$. During Upper Address Update Cycles, $\overline{\text{AS}}$ and R/$\overline{\text{W}}$ are both driven Low. Refer to the Section 6 for application hints.

$\overline{\text{DS}}$ (Data Strobe, Input/Output, Active Low)

Data Strobe is a bidirectional, three-state signal. When the CRTC is in the Slave Mode and the host CPU is accessing internal registers of the CRTC, $\overline{\text{DS}}$ is the input timing the transfer. $\overline{\text{DS}}$ may be asynchronous to CLK1. When the CRTC is bus master, $\overline{\text{DS}}$ is an output, timing the Memory Read operation.

$\overline{\text{CS}}$ (Chip Select, Input, Active Low)

The $\overline{\text{CS}}$ input is used by the host CPU to access the CRTC's internal registers. $\overline{\text{CS}}$ may be latched internally by a transparent latch controlled by the $\overline{\text{AS}}$ input.

$\overline{\text{WAIT}}$ (Wait, Input, Active Low)

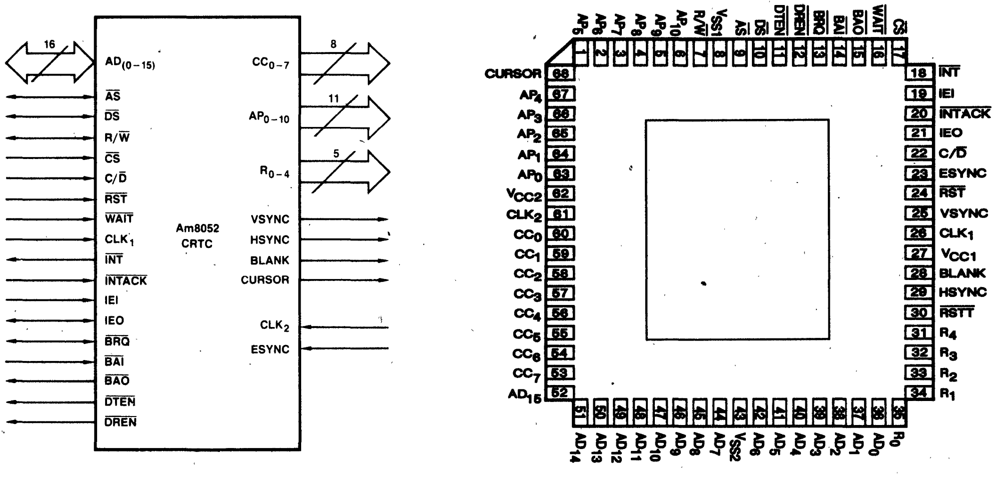
The $\overline{\text{WAIT}}$ input is used to stretch the $\overline{\text{DS}}$ strobe whenever the CRTC accesses slow system memory. The status of the $\overline{\text{WAIT}}$ signal is sampled only on the falling edge of CLK$_1$, in T2 of Bus Master Read Cycles. $\overline{\text{WAIT}}$ is ignored during Bus Master Writes or Slave Mode register accesses.

R/$\overline{\text{W}}$ (Read/Write, Input/Output)

Read/Write is a bidirectional, three-state signal. R/$\overline{\text{W}}$ indicates the data flow direction for the bus transaction under way, and in Master Mode remains stable for the length of the bus cycle. During Idle DMA Cycles, R/$\overline{\text{W}}$ is driven High.

C/$\overline{\text{D}}$ (Command/Data, Input)

In Slave Mode, C/$\overline{\text{D}}$ determines whether the host CPU transfers a pointer or data information. In Master Mode, C/$\overline{\text{D}}$ is disregarded; C/$\overline{\text{D}}$ flows through a transparent latch controlled by AS.



LS001211

CD005191

Figure 2-2 Am8052 Pinout

DTEN, DREN (Data Transmit Enable, Data Receive Enable, Open Drain Output)

Data Transmit Enable and Data Receive Enable control external address/data bus transceivers, when required. When DTEN is Low, the transceivers should be driven out from the CRTC onto the bus. When DREN is Low, the transceivers should be driven from the bus into the CRTC. DTEN and DREN are never Low simultaneously.

BRQ (Bus Request, Input/Open Drain Output)

When the CRTC asserts BRQ Low to gain bus mastership, it remains Low until the CRTC has released the bus. A bus release will occur, when the programmed DMA burst length is counted out (see Burst Register programming), when an entire Internal Row Buffer has been filled, or when DMA preemption is being requested (BAI High). This pin is also an input pin which allows the CRTC to sense the BRQ line.

BAI (Bus Acknowledge In, Input)

Bus Acknowledge In is an active-Low input. When the CRTC requires host bus access and has successfully pulled its BRQ pin Low, a BAI Low input flags the CRTC that it can obtain bus mastership. BAI is internally synchronized for two periods of CLK1 to alleviate metastable problems. When the CRTC does not require host bus access, the BAI input ripples to the BAO output.

DMA preemption may be implemented by removing BAI during a DMA burst, forcing the CRTC to finish the current DMA cycle and to release BRQ. If the DMA burst is not completed and no other device requests the bus (BRQ is High), the CRTC reasserts BRQ. The CRTC releases the bus for a minimum of three bus clock (CLK_1) cycles.

BAO (Bus Acknowledge Out, Output, Active Low)

BAO output is forced Inactive High when the CRTC has obtained bus mastership; otherwise, the BAI input ripples out of the CRTC via the BAO output.

INT (Interrupt Request, Output, Open Drain, Active Low)

This line is used to indicate an interrupt request to

Mode	Description	C/$\overline{\text{D}}$	R/$\overline{\text{W}}$	Data Bus
Slave Mode	Pointer Write	H	L	Pointer input
Slave Mode	(not defined)	H	H	(undefined)
Slave Mode	Data write	L	L	Data input
Slave Mode	Data Read	L	H	Data output
Master Mode	Memory Read	X	H	Data input
Master Mode	Upper addr.update	X	L	Address output

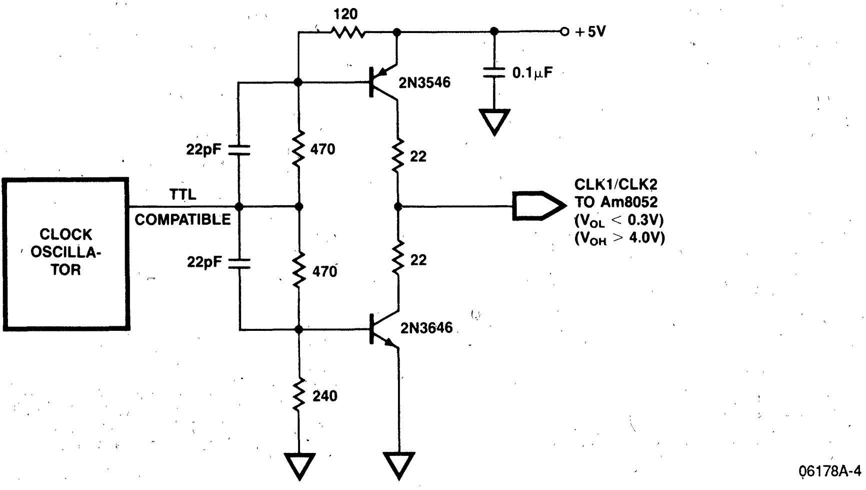


Figure 2-3 CLK_1/CLK_2 Driver

the host processor. It is driven Low by the CRTC until an Interrupt Acknowledge is received on the INTACK pin or until the host CPU acknowledges the interrupt by updating Mode Register 2.

$\overline{\text{INTACK}}$ (Interrupt Acknowledge, Input, Active Low)

When this line is driven Low, the CRTC examines its IEI line to determine if it has been granted an acknowledge by the CPU. $\overline{\text{INTACK}}$ must be High for normal operations. If INTACK is kept Low or floating, the CRTC will not respond to any slave accesses nor will it execute DMA transfers.

IEI (Interrupt Enable-In, Input, Active High)

A Low on IEI during Interrupt Acknowledge signifies that a higher priority interrupt on the daisy-chain is being acknowledged. IEI being High indicates that the CRTC has highest interrupt priority. If the CRTC is not requesting an interrupt, IEI ripples to IEO.

IEO (Interrupt Enable-Out, Output, Active High)

IEO follows IEI during Interrupt Acknowledge if the CRTC has not made an interrupt request. IEO Low disables lower priority devices from issuing interrupt requests. Refer to the Interrupt Section for a detailed description of the interrupt protocol.

HSYNC (Horizontal Sync, Output, Active High)

HSYNC is an active High output which controls the horizontal retrace of the CRT's electron beam. This output is held inactive (LOW) when the CRTC is reset to prevent unknown synchronization of the CRT which might cause damage to high bandwidth tubes.

VSYNC (Vertical Sync, Output, Active High)

VSYNC is an active High output which controls the vertical retrace of the CRT's electron beam. This output is held Low when the CRTC is reset to prevent damage to the CRT.

BLANK (Blank Video, Output, Active High)

BLANK is an active High output. It serves to blank out inactive display areas of the CRT. It is a composite of horizontal and vertical blank. This output is held High when the CRTC is reset.

ESYNC (External Sync, Input, Active High)

This pin is the external synchronization input and should be used exclusively for power line synchronization. The ESYNC input cannot synchronize two video systems since HSYNC is not altered by this signal. This input is enabled by setting the External Sync Enable (ES) bit in Mode Register 1.

$\overline{\text{RSTT}}$ (Test Reset, Input, Active Low)

$\overline{\text{RSTT}}$ resets the horizontal and vertical internal counters, and therefore can be activated to synchronize multiple CRTCs. Whenever $\overline{\text{RSTT}}$ input goes Low, the following takes effect:

- HSYNC Low
- VSYNC Low
- BLANK High
- Mode Register 2: D_{0-8} reset to "0"
- Horizontal counter reset
- Vertical counter reset

For synchronizing two CRTCs, RSTT should be driven synchronously to the Video Timing Clock (CLK$_1$ or CLK$_2$).

$\overline{\text{RST}}$ (Reset, Input, Active Low)

A Low on this input for at least 5 clock cycles is interpreted by the CRTC as a Reset signal. The effect of Reset is to drive all CRTC bus signals into the high-impedance state and initialize Mode Registers 1 and 2. Any Bus Master transaction is terminated and the CRTC will switch to Slave Mode.

CC$_{0-7}$ (Character Code, Outputs, Active High)

This character port outputs 8 bits of character data stored in the Character Code Section of the row buffer currently being displayed. The character code output can be delayed by 1 or 2 clock periods (CLK$_2$) in order to allow the attribute bits associated with the particular character code to be masked and decoded and to generate suitable synchronized attribute control (refer to Character Period Skew Programming in Mode Register 1).

R_{0-4} (Scan Line Address, Outputs, Active High)

These outputs provide the binary address of the character slice being displayed. Usually, R_{0-4} form the least significant address portion of a character font generator. All outputs are High ($1F_H$) for scan lines outside the range specified by Character Start and End (refer Row Redefinition Block programming).

AP_{0-10} (Attribute Port, Outputs)

These 11 lines output the attribute information associated with the characters. During HSYNC the Row Attribute Word contained in the Row Redefinition Block is output on AP_{0-4} and AP_{6-10}. This word can be stored externally by the falling edge of HSYNC.

CURSOR (Cursor, Output)

This pin is the cursor output indicator. Refer to the Cursor Section for further information.

2.3 REGISTER DESCRIPTIONS

This section provides a brief description of the Command, Status, and Display Timing registers in the CRTC. Each register description includes the register address, the operation of the individual register fields and the state of the register after a reset (hardware or software).

Table 1 is a summary of the CRTC's 22 registers. The registers are addressed by an internal pointer which is 5 bits wide. The pointer is loaded via AD_{0-4} on the external AD bus in Slave Mode write cycles with C/$\overline{D}$ being High.

After power-up, the registers should be initialized in the following sequence:

- Clear the DE bit of Mode Register 1 by hardware reset or by loading the registers
- Initialize all registers starting with Mode Register 2 (except Mode Register 1) with the appropriate values
- Load Mode Register 1, with the DE-bit set, to enable the display
- Load Mode Register 2

Addressing the CRTC with non-specified pointers ($0D$–$0F_H$, 19–$1F_H$) causes no problems. The registers can be loaded using a simple software loop, starting at 00_H and ending at $1F_H$.

Register Addressing

The registers can be accessed only when the CRTC is in the Slave Mode. They are addressed in a two-step sequence, to simplify slave accesses via a demultiplexed address/data bus:

- First load the internal pointer register by asserting $\overline{CS}$ Low and C/$\overline{D}$ High to indicate a command-type cycle. The subsequent Data Strobe latches the register address provided by the low part of the address/data bus (AD0–AD4). This latched register address remains valid until a subsequent slave write cycle with C/$\overline{D}$ High changes it.
- Reaccess the CRTC with $\overline{CS}$ Low and C/$\overline{D}$ Low to read or write the register pointed by the latched address. The data is strobed in or out by the $\overline{DS}$ signal.

The CRTC is in Slave Mode if it has not been granted control of the bus. After the CRTC has asserted $\overline{BRQ}$, it is remains in Slave Mode until it receives an bus acknowledge ($\overline{BAI}$ Low). The CPU can access the CRTC registers any time; the CRTC places no restrictions on slave accesses.

CRTC Slave Transfers

All slave transfers with the CRTC can be carried out asynchronously with respect to the CRTC CLK_1 input. Only $\overline{AS}$ and $\overline{DS}$ are used to transfer the information.

The slave transaction typically starts with a pointer write, although repetitive accesses to the same CRTC register can be made without any intervening pointer modification. The transaction is timed off the $\overline{DS}$ signal, since $\overline{AS}$ may not be present in certain systems. The read transaction commences from the low going edge of $\overline{DS}$. The write transaction takes place on the rising edge of $\overline{DS}$.

The $\overline{AS}$ input is used to drive a transparent latch on the CRTC, which is used to capture C/$\overline{D}$ and $\overline{CS}$ in a multiplexed address/data system. If the system is demultiplexed, then $\overline{AS}$ should be driven Low when the CRTC is in the Slave Mode. This drives the latch permanently transparent, allowing the

Table 1 Am8052 Registers

Pointer Address (AD4–AD0)

HEX	TYPE	ACTIVE BITS	REGISTER NAME
00	R/W	16	Mode 1
01	R/W	16	Mode 2
02	R/W	12	Attribute Enable
03	W	5	Attribute Redifinition
04	R/W	8	Top of page soft (High Order)
05	R/W	16	Top of page soft (Low Order)
06	R/W	8	Top of window soft (High Order)
07	R/W	16	Top of window soft (Low Order)
08	W	16	Attribute Flag
09	R/W	8	Top of page hard (High)
0A	R/W	16	Top of page hard (Low)
0B	R/W	8	Top of window hard (High)
0C	R/W	16	Top of window hard (Low)
10	W	16	DMA Burst
11	W	12	*VSYNC Width/Scan Delay
12	W	12	*Vertical Active Lines
13	W	12	*Vertical Total Lines
14	W	16	*HSYNC/VERTINT
15	W	9	*HDRIVE
16	W	9	*H Scan Delay
17	W	10	*H Total Count
18	W	10	*H Total Display

*These registers should be only accessed when display enable ("DE" bit in mode1) is reset, since they control the video timing signals

demultiplexed CS and C/D to pass into the CRTC. When the $\overline{DS}$ goes Low and a read transaction is in progress, the CRTC drives the read data onto its AD_0–AD_{15} lines and also drives $\overline{DTEN}$ Low. This enables any off-chip bus transceivers, allowing the data to be transmitted to the bus master. When the bus master captures the data, it drives the $\overline{DS}$ signal High. This causes the CRTC to cease driving its AD_0–AD_{15} lines and also causes $\overline{DTEN}$ to return High, switching off the bus transceivers.

Register Test

When designing register test routine the software designer must consider the following points:

- The Attribute Enable, the Attribute Redefinition, the DMA Burst, and all video timing registers are write only.
- All reserved fields in the registers should be set to zero, however, the state of these fields when reading the programmed value back is undefined. For verification purposes these fields must be masked out (logical AND) before comparing the value read back with the value programmed.
- The TOP hard register and the TOP soft register use the same internal register. Therefore, writing to one register also changes the value of the other register. (The CRTC uses internal flags to differentiate between write accesses to either register).
- If the CRTC is programmed for segmented mode, all upper address registers are loaded via the upper half of the 16-bit address/data bus (for linear mode via the lower half of the address/data bus). However, the value read back appears on the lower half of the address/data bus (for both segmented and linear mode).

Mode Register 1

Mode Register 1 contains display and DMA control bits (Figure 2.4). On reset, all Mode Register 1 bits set to "0".

Video Timing Clock—CLK1/2 (D_{15})

This bit indicates whether CLK_1 or CLK_2 drives the video timing logic to time the HSYNC (or HDRIVE), VSYNC and BLANK outputs. In non-proportional spacing applications CLK_1 is selected, whereas in proportional spacing applications CLK_2 usually times the sync signal, since the frequency of CLK_2 is modulated by the character width.

$CLK_{1/2}$ = 0: Selects CLK_2 for clocking the sync counters

$CLK_{1/2}$ = 1: Selects CLK_1 for clocking the sync counters

Character Shift—CSHIFT (D_{14})

This bit affects the relative order assigned to the two bytes (character codes) fetched from memory in a word access (Figure 2.5).

CSHIFT=0. The LOW byte is displayed first. This mode is compatible with iAPX microprocessors.

CSHIFT=1 The HIGH byte is displayed first. This mode is compatible with 68000 microprocessors.

CSHIFT does not affect 16-bit word data, such as addresses, pointers, control information, and attributes.

Invisible Attribute Flag—IAF (D_{13})

IAF=0: The character that invoked an attribute is loaded into the row buffer, and subsequently displayed. The character is affected by the attribute word (see option 1 or 2 in Figure 2.39).

IAF=1: The characters that invoked an attribute are not loaded into the row buffer. The invoked attribute applies to the next character. One character word (two characters) should contain only one Attribute Flag. The second Attribute flag within one character word will be disregarded. If two Attribute Flags are separated by a word boundary (within two character words), both will be processed.

Screen Width Limit—SLIM (D_{12})

The SLIM bit controls the number of characters loaded in each row buffer to either 132 or 96. This can reduce bus overhead when the CRTC row length is 96 characters or less. If the CRTC reaches the limit of the row buffer (132 characters), and more characters are requested, the last, 132nd, character is repeated. In the 96-character mode, the CRTC continues with the random data of the row buffers.

SLIM=0: The row buffer size is set to 132 characters.

SLIM=1: The row buffer size is set to 96 characters.

Linear/Segmented Mode—L/S (D_{11})

This bit indicates whether the system/display memory access is accomplished by addressing it in a linear or segmented mode.

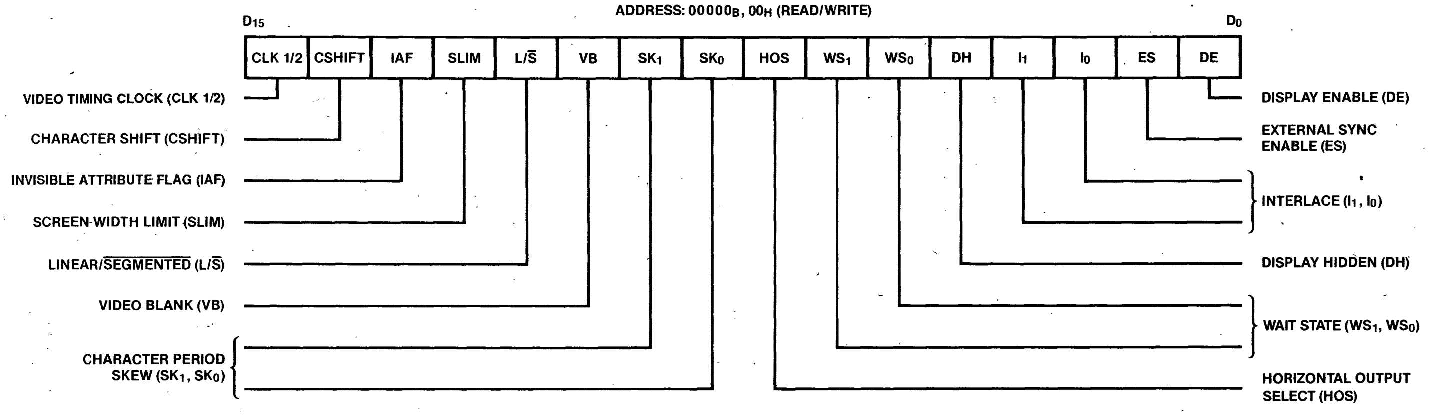
L/S=0: The CRTC is set for segmented addressing. The linked-list address pointers are two words long. Seven bits ($D_{8\text{-}14}$) of the first word define the segment address. The second 16-bit word is the offset address within the segment. Any overflow of the 16-bit offset address does not carry into the upper 7-bit segment address.

L/S=1: The CRTC is set up for a linear addressing scheme. The most significant byte of the 24-bit linear address is stored in the lower half of the first word ($D_{0\text{-}7}$). The second word holds the remaining 16 bits. Any overflow of the 16-bit offset increments the 8-bit upper address.

During page update cycles the CRTC puts out the upper part of the 23/24-bit address on AD_0-AD_7. The user may latch the 7/8-bit address (refer to Section 6).

Video Blank—VB (D_{10})

This bit allows the user to blank the screen while making changes in the displayed text or when switching the context. The linked-list must, however, be valid before VB is reset.


ADDRESS: 00000B, 00H (READ/WRITE)
D15
D0
CLK 1/2
CSHIFT
IAF
SLIM
L/S̄
VB
SK1
SK0
HOS
WS1
WS0
DH
I1
I0
ES
DE
VIDEO TIMING CLOCK (CLK 1/2)
CHARACTER SHIFT (CSHIFT)
INVISIBLE ATTRIBUTE FLAG (IAF)
SCREEN WIDTH LIMIT (SLIM)
LINEAR/SEGMENTED (L/S̄)
VIDEO BLANK (VB)
CHARACTER PERIOD SKEW (SK1, SK0)
DISPLAY ENABLE (DE)
EXTERNAL SYNC ENABLE (ES)
INTERLACE (I1, I0)
DISPLAY HIDDEN (DH)
WAIT STATE (WS1, WS0)
HORIZONTAL OUTPUT SELECT (HOS)

03901A-05

Figure 2-4 Mode Register 1

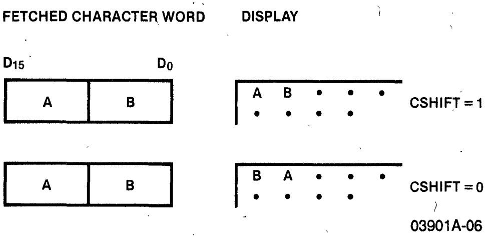


Figure 2-5 Character Shift

VB=0: Normal Operation

VB=1: The horizontal and vertical sync circuitry and outputs operate normally and the BLANK output is forced High. DMA operation is suspended--normal operation resumes when VB=0 and the next vertical blanking period occurs.

Do not use Video Blank (VB-bit in Mode Register 1) to blank the display while the linked-list is being modified. Instead, synchronize the CPU to the Am8052 linked-list scanning via Vertical Interrupts ("working on a busy railroad"), or use double-buffered linked-lists (the Am8052 interprets one while the CPU updates the other).

If Video Blank is used, first switch to a linked-list defining a blank screen, wait until the Am8052 has completely loaded the three top-most rows (all three internal row buffers are filled with blanks), and then set the Video Blank bit in Mode Register 1. This procedure ensures that, when the VB-bit is reset, no random characters are displayed from VB being reset to the beginning of the next frame. During this time interval, the Am8052 will display the contents of the internal row buffers which were preloaded with Blanks. No DMA activity will occur until the beginning of the next frame, when normal operation is resumed.

Character Period Skew—SK_1, SK_2 (D_9, D_8)

The skew bits compensate externally introduced clock skew between, character code, attribute word, and/or video control signals, e.g. pipelined character code path to the Video System Controller (Am8152A) to relax the required access time of the character font generator (see Section 4). The skew bits program various delays in number of character clock cycles applied to the VSYNC, HSYNC, and BLANK signals with respect to character code output. The attributes and cursor outputs can also be selectively delayed by SK_0 and SK_1. The following combinations are programmable:

Bit Settings		Signal Skew (# of CLK2 Cycles)		
SK1	SK0	HSYNC,VSYNC & BLANK	AP0-AP10 & CURSOR	CC0-CC7 & R0-R4
0	0	0	0	0
0	1	1	0	0
1	0	2	1	0
1	1	1	1	0

Horizontal output Select–HOS (D_7)

HOS=0: The HSYNC/HDRIVE output pin outputs the horizontal sync timing as programmed in the HSYNC Register (8-bit counter).

HOS=1: The HSYNC/HDRIVE output pin outputs the horizontal drive timing as programmed in the HDRIVE Register (9-bit counter).

Wait State—WS_2, WS_0 (D_6, D_5)

These bits indicate the number of Wait states inserted for each DMA cycle. These Wait states are in addition to any externally applied Wait states. When the CRTC is in Slave Mode, these bits are ignored.

WS1	WS0	WAIT STATE
0	0	No Wait State
0	1	DS stretched by one clock
1	0	DS stretched by two clocks
1	1	Reserved

Display Hidden—DH (D_4)

Applies only to characters which have the Ignore attribute bit set ("1") in the attribute word associated with this character.

DH=0: The Ignore attribute is active; characters with the Ignore attribute set ("1") are not loaded into the row- buffer.

DH=1: Those characters are treated as displayable information (see Section 2.6).

Interlace—I_1, I_0 (D_3, D_2)

Control the timing of non-interlaced, interlaced, repeat field interface video to support different

CRTs (see Section 2.10).

I_1	I_0	MODE OF OPERATION
0	0	Non-Interlaced Video
0	1	Reserved
1	0	Repeat Field Interlace (RFI)
1	1	Interlaced Video

External Sync Enable—ES (D_1)

Enables the ESYNC input for power line synchronization.

ES=0: ESYNC input is ignored.

ES=1: A rising edge at the ESYNC input during a vertical-retrace active period (even frame only in interlaced mode) causes the HSYNC output to go (or remain) active for a full horizontal retrace period. The VSYNC active period is stretched, even when register timing signifies an end to vertical retrace, until an ESYNC falling edge occurs.

Display Enable—DE (D_0)

DE=0: VSYNC, HSYNC outputs are inactive (LOW) and the BLANK output is held active (HIGH). DMA operation is disabled. The DE bit is reset by a hardware reset (RST=Low) or may be reset by the host processor (software reset). DE=0 resets the scroll logic to the non-scrolling state.

DE=1: The CRTC display operation is enabled. DE can be set only by a host processor access of Mode Register 1. Setting the DE=1 causes the VSYNC, HSYNC, and BLANK outputs to become active and the DMA controller on board the CRTC eventually requests access to the system bus.

Mode Register 2:

Mode Register 2 contains the primary control bits for the interrupt control logic and cursor definition (Figure 2.6).

Upon reset, all Mode Register 2 bits are reset to zero.

Cursor Enable—CUE (D_{15})

CUE=0: The CRTC does not output any XY cursor information.

CUE=1: The XY Cursor Register is enabled. CRTC outputs cursor at the character position defined by the XY Cursor Register (see Main Definition Block).

Attribute Cursor Mask—ACM_1, ACM_0 (D_{13}, D_{12}) Cursor Mask—$XYCM_1$, $XYCM_2$ (D_{10}, D_9)

The cursor mask field (D_{13}, D_{12}, D_{10}, D_9) defines the type of cursor that is generated when a cursor is required. This field is divided into two parts:

D_{13}	D_{12}	CURSOR ATTRIBUTE DEFINITION
0	0	Cursor Pin Whole
0	1	Cursor Pin Part
1	0	Underline
1	1	Reverse
D_{10}	D_9	XY CURSOR DEFINITION
0	0	Cursor Pin Whole
0	1	Cursor Pin Part
1	0	Underline
1	1	Reverse

"Cursor Pin Whole" means that the cursor signal will appear on the cursor pin for every scan line of that character position (TSLC). CURS and CURE of the Row Redefinition Block are ignored.

"Cursor Pin Part" means that the cursor signal will appear on the cursor pin for those scan lines specified in the Row Redefinition Block (CURS and CURE).

"Underline" (BLOB) means that the cursor signal will appear on the underline pin (AP1) for the scan lines specified in the Row Redefinition Block (CURS and CURE).

"Reverse" (part) means that the cursor signal will appear on the reverse pin (AP5) for the scan lines specified in the Row Redefinition Block (CURS and CURE).

Scroll In Progress—SIP (D_8)

SIP is a status bit that is set/reset by the CRTC smooth scroll control logic.

SIP=0: The CRTC is not currently scrolling.

SIP=1: The CRTC is scrolling either window or background.

Disable Lower Chain—DLC (D_7)

DLC=0: IEO operates normally.

DLC=1: The Interrupt Enable Out (IEO) output of the device is forced Low, disabling interrupts from all lower priority devices on the daisy-chain.

No Vector—NV (D_6)

NV=0: The CRTC outputs the interrupt vector programmed in the Main Definition Block. (See the section on Main Definition Block and Interrupt.)

NV=1: During an Interrupt Acknowledge cycle, the interrupt vector is inhibited. The vector can, therefore, be provided by external hardware if necessary. It has no effect on the setting of the Interrupt Under Service bits.

Interrupt Under Service Vertical Event—IUSV (D_5)

This status bit is automatically set if IPV (Interrupt Pending Vertical Event) is the highest priority interrupt request pending when an Interrupt Acknowledge sequence takes place. It can also be set or cleared directly by CPU command. While the IUSV is set, internal and external daisy-chains prevent the same and lower priority sources of interrupt from requesting interrupts. The IUSV can be cleared to "0" only by CPU command. For details of Interrupt Operation, see Section 2.7.

Interrupt Enable Vertical Event—IEV (D_4)

This bit enables or disables the vertical event interrupt logic.

IEV = 0: The Vertical Interrupt is disabled. The CRTC does not request an interrupt at vertical event nor respond to an interrupt acknowledge.

IEV = 1: The Vertical Interrupt is enabled.

Interrupt Enable (IEV) does not affect the normal operation of Interrupt Pending (IPV) and Interrupt Under Service (IUSV). If IEV disables the interrupt (IEV=0), then setting the Interrupt Pending Bit (IPV) does not activate the Interrupt Request Line. If IEV=0, then a "1" in IUSV affects the interrupt daisy-chain; all lower priority devices are disabled.

Interrupt Pending Vertical Event—IPV (D_3)

IPV is a status bit which, when set to "1," indicates that a vertical event has occurred and CPU service is required. A vertical event occurs when the CRTC internal load row counter matches the VERTINT value loaded in the HSYNC/VERTINT Register. This interrupt provides real-time positional information. This is the lowest priority IP bit in the CRTC. The IPV can be cleared only by a CPU command.

Interrupt Under Service Smooth-Scroll—IUSS (D_2)

Same as vertical event but applies for smooth-scroll event.

Interrupt Enable Smooth-Scroll—IES (D_1)

This bit enables or disables the smooth-scroll's interrupt logic. Same as vertical event.

Interrupt Pending Smooth-Scroll—IPS (D_0)

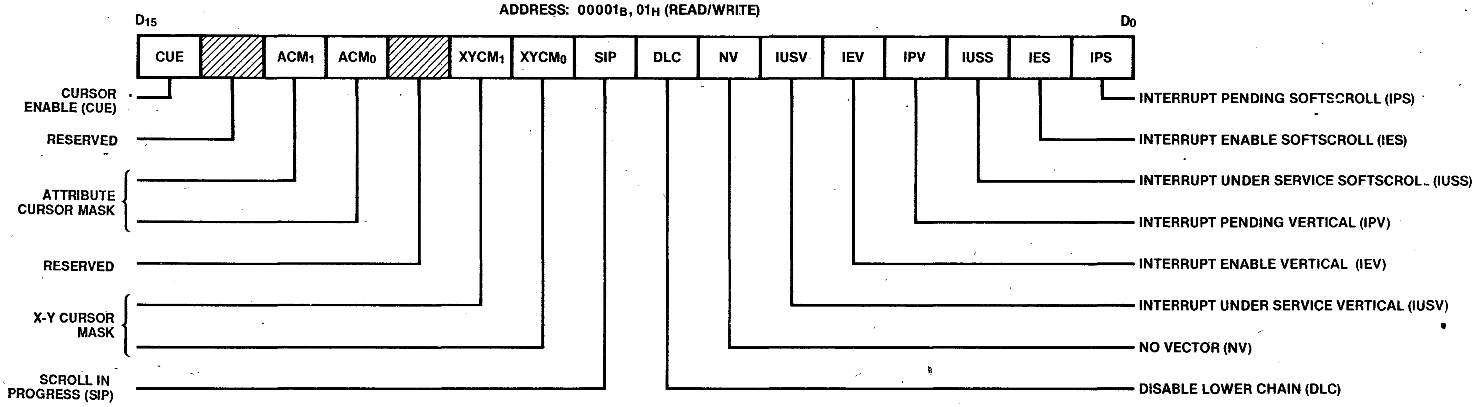
IPS is a status bit which, when set, indicates that a smooth-scroll event requires CPU intervention. This is the highest priority IP bit.

Attribute Port Enable Register

Bits D_0 through D_{10} in the Attribute Port Enable Register allow the corresponding attribute information to be output on the matching attribute pin (Figure 2.7). When reset ("0"), the corresponding attribute pin is driven Low. When set, the corresponding pin outputs attribute information. Bits D_3 and D_4 of this word affect the subscript and superscript attribute pin operation. If these bits are enabled for subscript or superscript, the corresponding pins will be active. These attributes are independent of the R_0-R_4 outputs. The user can thus address a separate character font generator for subscript or superscript display, e.g. a smaller font. The CURSOR PIN ENABLE (CPE, D_{13}) bit of this register enables/disables only the cursor pin. When disabled, neither the X-Y cursor nor the attribute cursor is output through the cursor pin (CURSOR=Low).

Attribute Cursor Enable—ACE (D_{14})

The Attribute Cursor Enable Register enables/disables the path between attribute cursor and



ADDRESS: 00001B, 01H (READ/WRITE)

D15															D0
CUE	▨	ACM_1	ACM_0	▨	$XYCM_1$	$XYCM_0$	SIP	DLC	NV	IUSV	IEV	IPV	IUSS	IES	IPS

CURSOR ENABLE (CUE)
RESERVED
ATTRIBUTE CURSOR MASK
RESERVED
X-Y CURSOR MASK
SCROLL IN PROGRESS (SIP)
INTERRUPT PENDING SOFTSCROLL (IPS)
INTERRUPT ENABLE SOFTSCROLL (IES)
INTERRUPT UNDER SERVICE SOFTSCROLL (IUSS)
INTERRUPT PENDING VERTICAL (IPV)
INTERRUPT ENABLE VERTICAL (IEV)
INTERRUPT UNDER SERVICE VERTICAL (IUSV)
NO VECTOR (NV)
DISABLE LOWER CHAIN (DLC)

03901A-07

Figure 2-6 Mode Register 2

▨ WRITE: 0
READ: X

cursor output pin.

Attribute Redefinition Register

The Attribute Redefinition Register allows the user to redefine some of the internally processed attributes, which can, therefore, be treated as user-definables (Figure 2.8). A "0" keeps normal attribute operation; a "1" directly outputs the attribute state to its corresponding pin without any internal processing of the attributes.

Top of Page/Top of Window Registers

Figures 2.9 and 2.10 show the format of these registers.

The Top Of Page and Top Of Window Registers point to the Main Definition Block and Window Definition Block respectively; these blocks contain the primary information concerning the background display and the window display.

Two different forms of Top of Page/Window Register writes are available: hard and soft. "Top of Page/Window Soft" is used to trigger the smooth-scroll and to interact with the smooth-scroll controller (see section on smooth-scroll). "Top of Page/Window Hard" has no effect on the smooth-scroll procedure and should be used for link manipulations that do not involve smooth-scroll. If the Top of Window Register contains "0," no window is displayed on the screen.

Top Of Page/Window Hard and Top Of Page/Window Soft access the same internal register. When loading Top Of Page/Window Hard the information the value gets strobed into the visible register and, in addition, gets immediately transferred to the DMA unit. When loading the Top Of Page/Window Soft register the value gets only loaded into this visible register. The transfer to the DMA unit is delayed until the CRTC re-loads the hard register with the value stored in the soft register (only for smooth scrolling being activated). This means, that loading the hard register overwrites the contents of the soft register, but loading the soft register does not over-write the contents of the hard register.

Attribute Flag Register

The Attribute Flag Register defines the bit pattern that will invoke an attribute word from the attribute segment (Figure 2.11).

This 16-bit register is divided into two sections, Mask and Value. Each 8-bit character code loaded from memory, is analyzed, to determine whether this character is an attribute invoking character. Any binary group of character can be defined as attribute invoking characters. The analysis is based on a mask operation (using Mask) and a comparison of the remaining pattern with Value. If the remaining pattern and the Value are equal, this character is an attribute word invoking character. In this manner, it is possible to define a group of 1, 2, 4, 8, ..., 256 character codes as attribute invoking character codes.

The attribute fetch mechanism can be completely turned off (0 attribute invoking character codes) by setting the least significant Mask-bit (D8) to "0", and the corresponding value-bit (D_0 to "1", e.g. loading 0001H into the Attribute Flag Register. (This feature is only available on devices with copyright date of 1985 or later).

Mask (7–0) (D_{15}–D_8)

The Mask Register defines which bits of the 8-bit character field will be compared against the Value Register to determine if the character invokes an attribute word. A "0" in bit position N of the mask indicates that character bit N is a "don't care" in the value comparison. A "1" in bit position N of the Mask Register indicates that character bit N should be compared against value bit N.

Page And Window Registers

	# Of Active Bits		Address		
Register	LINEAR	SEG.	BINARY	HEX	TYPE
Top Of Page Soft (HI)	8	7	00100	04	R/W
Top Of Page Soft (LO)	16	16	00101	05	R/W
Top Of Window Soft (HI)	8	7	00110	06	R/W
Top Of Window Soft (LO)	16	16	00111	07	R/W
Top Of Page Hard (HI)	8	7	01001	09	R/W
Top Of Page Hard (LO)	16	16	01010	0A	R/W
Top Of Window Hard (HI)	8	7	01011	0B	R/W
Top Of Window Hard (LO)	16	16	01100	0C	R/W

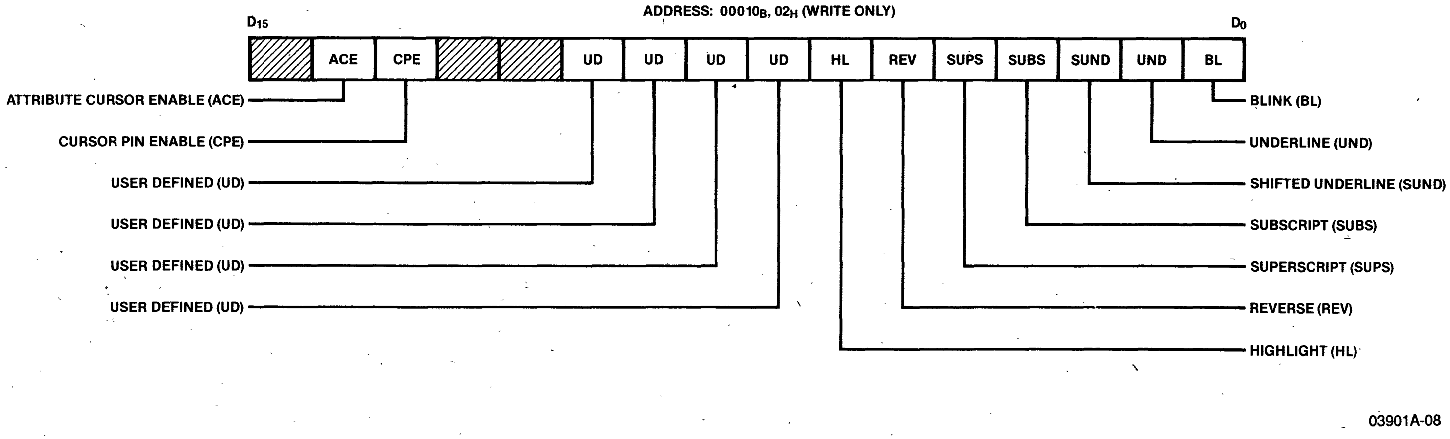
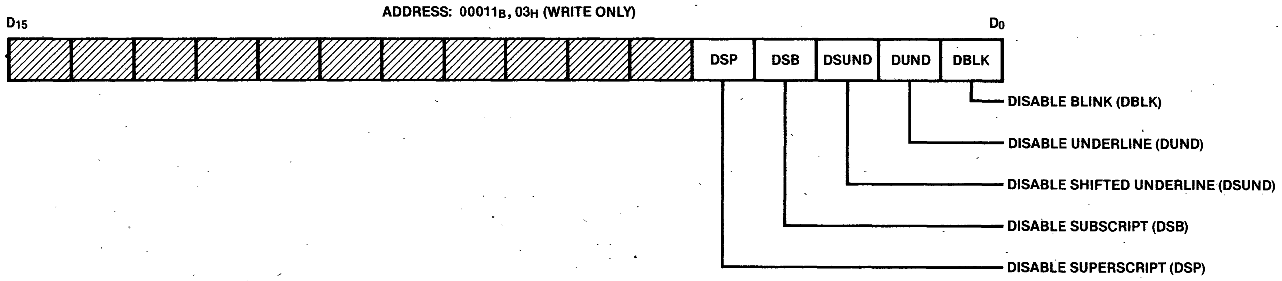


Figure 2-7 Attribute Port Enable Register

▨ WRITE: 0
READ: X


ADDRESS: 00011B, 03H (WRITE ONLY)
D15
D0
DSP
DSB
DSUND
DUND
DBLK
DISABLE BLINK (DBLK)
DISABLE UNDERLINE (DUND)
DISABLE SHIFTED UNDERLINE (DSUND)
DISABLE SUBSCRIPT (DSB)
DISABLE SUPERSCRIPT (DSP)

03901A-09

Figure 2-8 Attribute Redefinition Register

▨ WRITE: 0
READ: X

Value (7–0) (D_7–D_0)

The Value Register holds up to eight bits of information for comparison with the fetched character, to determine if an attribute should be invoked. Note that only those bits of the Value Register which have the corresponding bits of the Mask Register set to "1" are compared against the character code. Value bits with corresponding Mask bits set to "0" should be set also to "0," unless the attribute fetch mechanism is disabled.

Example 1:

All control characters (character code within 00_H and $1F_H$) invoke an attribute. To display these control characters IAF=0; not to display these characters IAF=1 (see Mode Register 1). All control characters are of the form:

Control Characters:	0	0	0	X	X	X	X	X
So the mask is:	1	1	1	0	0	0	0	0
and the value is:	0	0	0	0	0	0	0	0

(X is "Don't Care")

So the Attribute Flag Register contents are:

1110000000000000 ($E000_H$).

Example 2

One specific flag ($7F_H$) invokes an attribute. In this case, all bits of the character code are compared to the Value.

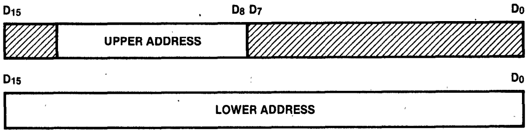
Flag character:	0	1	1	1	1	1	1	1	($7F_H$)
So the mask is:	1	1	1	1	1	1	1	1	
and the value is:	0	1	1	1	1	1	1	1	($7F_H$)

Hence the Attribute Flag Register contains:

1111111101111111 ($FF7F_H$).

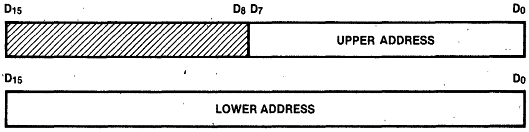
Burst Register

The Burst Register (Figure 2.12) specifies the bus occupancy of the CRTC DMA unit. Burst Count determines the maximum burst length in Number



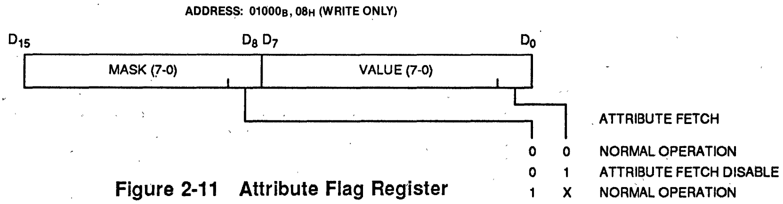
03901A-9

Figure 2-9 Top of Page and Top of Window Pointer Formats with L/S = 0



03901A-10

Figure 2-10 Top of Page and Top of Window Pointer Formats with L/S = 1



03901A-11

Figure 2-11 Attribute Flag Register

of DMA transfer cycles. Burst Space determines the minimum release time between two bursts. This guarantees real-time responses of the CPU to other peripherals. Burst Count and Burst Space must be programmed with reasonable values that allow the CRTC to fetch all data needed for a flicker-free screen.

Burst Space—BS_{7-0} (D_{15}–D_8)

This 8-bit value specifies the number of 15 system clock cycle (CLK_1) periods before another bus request will be issued, after the CRTC has released the bus due to burst count out. If this value is set to "0" the CRTC occupies the bus as long as necessary to accomplish its DMA activity, e.g. fetching all information related to a particular character row. If a DMA burst is interrupted due to DMA preemption or "end of row", the next burst completes the remaining burst count. This means, that the first DMA burst loading a row usually is shorter than programmed.

Burst Count—BC_{0-7} (D_7–D_0)

The CRTC executes Burst Count-1 DMA transfer cycles per burst. If BC_{0-7} is set to "0," no DMA activity will occur. If BC_{0-7} is set to "1," the CRTC only requests the bus and after granting the bus, immediately releases the bus, because the first cycle is an Idle DMA Cycle (no bus activity for three clocks). So, the minimum value for normal operation is "2."

Video Timing Registers:

These registers are initialized before setting the DE-bit in Mode Register 1. They hold the parameters needed to generate vertical and horizontal sync and blank (VSYNC, HSYNC, and BLANK). These signals are put out on the like-named pins of the CRTC and are used by the Am8152A. BLANK combines horizontal and vertical blank (HBLANK and VBLANK).

Horizontal timing parameters are expressed in number of bus or character clock cycles ($CLK_{1/2}$ bit of Mode Register 1). Vertical timing parameters are expressed in number of scan lines (HSYNC cycles).

HSYNC (8-bit counter) and HDRIVE (9-bit counter) represent two ways of specifying the signal waveform on the HSYNC output pin. With the exception of the width, these two counters are functionally identical.

In the following discussion a frame consists of one field in non-interlaced mode and two fields (even and odd) in RFI and Video Interlace modes. Figures 2.13 and 2.14 show the vertical timing.

Vertical Sync Width/Vertical Scan Delay Register

Figure 2.15 shows the register format.

D_{15}–D_{12}	NOT USED
D_{11}–D_6	VERTICAL SCAN DELAY (VSD)
D_5–D_0	VERTICAL SYNC WIDTH (VSW)

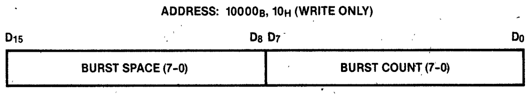
Vertical Scan Delay—VSD (D_{11}–D_6)

The Vertical Scan Delay field specifies the vertical blank time after the falling edge of VSYNC, thus defining the top border width, or vertical back porch, of the screen. VSD is expressed in scan-line units. When in non-interlaced mode, the actual vertical scan delay is equal to VSD + 1 scan lines. When in video interlace mode or Repeat Field Interlace (RFI) mode, the actual vertical scan delay is equal to [(VSD + 1) / 2 lines]. In this case, VSD must be odd.

Vertical Sync Width—VSW (D_5–D_0)

The Vertical Sync Width determines the width of the active-High pulse signal which is sent through VSYNC output to the CRT monitor in order to synchronize it vertically.

VSW is expressed in scan line units. In non-interlaced mode, the actual vertical sync width is


ADDRESS: 10000_B, 10_H (WRITE ONLY)

D_{15} – D_8	D_7 – D_0
BURST SPACE (7-0)	BURST COUNT (7-0)

03901A-13

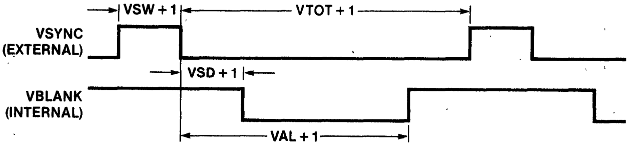
Figure 2-12 Burst Register

equal to VSW + 1 scan lines.

In interlaced and RFI mode, the actual vertical sync is equal to [(VSW+1)/2 lines]. In this case, VSW must be odd.

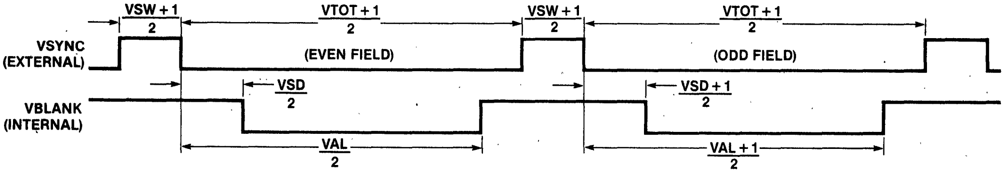
Vertical Active Lines Register

D_{15}–D_{12}	NOT USED
D_{11}–D_0	VERTICAL ACTIVE LINES (VAL)



03901A-17

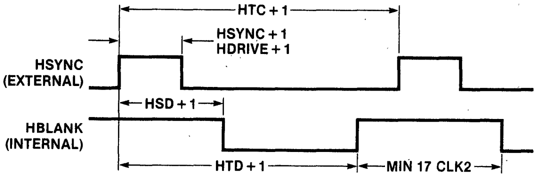
Figure 2-13 Non-Interlaced Video Vertical Sync Timing



VSD, VSW, VAL MUST BE ODD
VTOT MUST BE EVEN

03901A-18

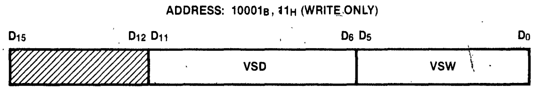
Figure 2-14 RFI and Video Interlace Vertical Sync Timing



HSD≥9
INTERLACED VIDEO: HTC MUST BE EVEN

03901A-19

Figure 2-14a Horizontal Sync Timing



03901A-14

Figure 2-15 Vertical Sync Width/Vertical Scan Delay Register

This 12-bit field defines the number of scan lines between the end of a vertical sync pulse and the start of vertical blanking (Figure 2.16).

When in non-interlaced mode, the actual scan-line number between the falling edge of VSYNC and the rising edge of VBLANK is equal to VAL+1. The active video area height on the screen is then (VAL+1) – (VSD+1) = VAL – VSD scan lines.

When in video interlace or RFI mode, the actual scan-line number between VSYNC and VBLANK is equal to [(VAL + 1) / 2]. In this case VAL must be odd. The active video area height on the screen is then given by [(VAL + 1) / 2] – [(VSD + 1) / 2] = [(VAL – VSD) / 2 scan lines]. This is true for the odd and even field.

Vertical Total Lines Register

D_{15}–D_{12}	NOT USED
D_{11}–D_0	VERTICAL TOTAL LINES (VTOT)

The Vertical Total Lines Register defines the total number of scan lines per field minus the vertical sync width (Figure 2.17).

In non-interlaced mode, the actual scan line number between VSYNC and next VSYNC is (VTOT + 1).

In interlaced or RFI mode, this timing is [(VTOT+ 1)/2], and VTOT must be even (half scan line between even and odd fields).

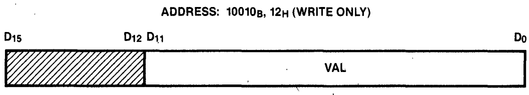
Horizontal Sync and Vertical Interrupt Row Register

Figure 2.18 shows the register format.

D_{15}–D_8	VERTICAL INTERRUPT ROW (VERTINT)
D_7–D_0	HORIZONTAL SYNC WIDTH (HSYNC)

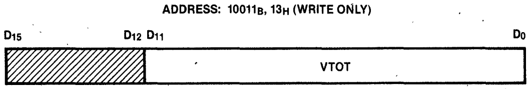
Vertical Interrupt Row—VERTINT (D_8–D_{15})

This field determines the row number which, after being completely loaded by DMA, causes an interrupt. If VERTINT is set to "0," the vertical interrupt occurs after the rising edge of VBLANK, before the CRTC starts loading the Main Definition Block. If VERTINT is set to "1" ("n"), the vertical interrupt is generated right after the first (nth) row has been loaded.



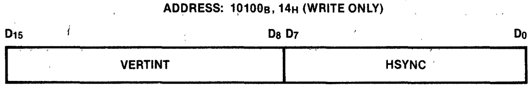
03901A-15

Figure 2-16 Vertical Active Lines Register



03901A-16

Figure 2-17 Vertical Total Lines Register



03901A-20

Figure 2-18 Horizontal Sync Width/Vertical Interrupt Row Register

Horizontal Sync Width—HSYNC (D_0–D_7)

This field determines the width of the horizontal sync (active High) pulse in video clock units (CLK_1 or CLK_2 depending upon $CLK_{1/2}$ bit in Mode Register 1), provided that HSYNC is selected (HOS=0 in Mode Register 1). These pulses are output on the HSYNC pin. The actual width of the signal is HSYNC + 1 clock periods.

Horizontal Drive Register

D_{15}–D_9	Reserved
D_8–D_0	HORIZONTAL DRIVE (HDRV)

This register determines the width of HSYNC if horizontal drive is selected (HOS=1 in Mode Register 1). The actual width of HSYNC is HDRV + 1 clock periods. This is also an output on the HSYNC pin. (See Figure 2.19.)

Horizontal Scan Delay Register

D_{15}–D_9	Reserved
D_8–D_0	HORIZONTAL SCAN DELAY (HSD)

The Horizontal Scan Delay Register determines the interval from rising edge of HSYNC to the falling edge of HBLANK, which defines the left border (back porch) on the screen. The actual interval value is HSD + 1 clock periods. (See Figure 2.20.)

Horizontal Total Count Register

D_{15}–D_{10}	Reserved
D_9–D_0	HORIZONTAL TOTAL COUNT (HTC)

This register determines the period of the HSYNC waveform. The period is HTC + 1 clock periods. In Interlaced mode, HTC must be even. (See Figure 2.21.)

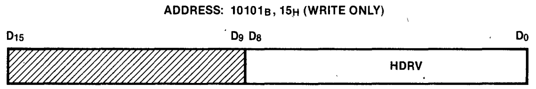
Horizontal Total Display Register

D_{15}–D_{10}	Reserved
D_9–$D0_0$	HORIZONTAL TOTAL DISPLAY (HTD)

This register determines the interval from the rising edge of HSYNC to the rising edge of HBLANK. HTD must be odd in interlaced mode. The actual interval value is HTD + 1 clock periods. (See Figure 2.22.)

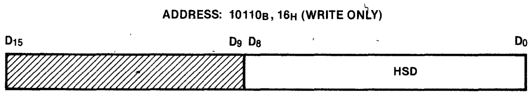
Video Timing Programming Example

The following example outlines the computation of the display timing parameters for a 30 row by 80 character display, each character embedded in a 8



03901A-21

Figure 2-19 Horizontal Drive Register



03901A-22

Figure 2-20 Horizontal Scan Delay Register

x 17 (H x V) matrix, with a refresh rate of 50 Hz in non-interlaced mode using a CRT monitor with the following characteristics:

Display Resolution:	720 pixels horizontal 512 lines vertical
Scanning frequency:	28–36 kHz horizontal 45–65 Hz vertical
Horizontal retrace time:	6 microseconds
Vertical retrace time:	600 microseconds
Horizontal SYNC width:	3 microseconds

Computation:

The appropriate character clock and the timing parameters for the video timing registers must be calculated.

The active display size is given by:

Horizontal:	80 characters • 8 pixels/char. = 640 pixels
Vertical:	30 rows • 17 scan lines/row = 510 scan lines

Assuming a 20% blank border vertically, the 510 scan lines occupy 80% of frame time. At a frame rate of 50 Hz, the horizontal frequency can be calculated as:

Total Scan Lines/frame:	510 scan lines / 0.80 = 637 scan lines
Horizontal Frequency:	637 • 50 Hz = 31.85 kHz

Assuming a 20% blank horizontally, the 80 characters occupy 80% of row time. Character clock is therefore 100 times the horizontal frequency (3.185 MHz). Each character occupies 1/100 of the row.

Let us use a more convenient frequency, 3.00 MHz, as character clock and re-calculate the parameters:

Character clock	3.00 MHz
Horizontal frequency	30 kHz
Scan line time	33.3 microseconds
Frame time	637 • 33.3 microseconds = 21.2 ms
Frame rate	47 Hz

Now the registers' contents can be calculated:

Mode Register 1

The character clock is 3 MHz; the $CLK_{1/2}$ bit is set to "0" to select CLK_2 for the frame timing generation.

With only 80 characters/row, we select "SLIM=1" which reduces the row buffer length to 96 characters.

The monitor accepts an HSYNC signal: "HOS=0"

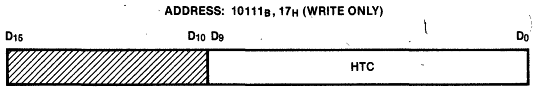
Non-interlaced made yields in: "I_1=0," "I_0=0."

External Sync Enable is set to "0," since we do not need to be synchronized to another signal.

Display Enable should be set to "1," once the other registers are set to the proper values.

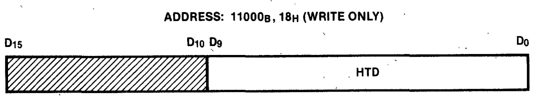
Vertical sync width: The vertical sync width is equal to the specified horizontal retrace time of the monitor.

VSW + 1 = 600 microseconds
VSW + 1 = 600/33.3 = 18 scan lines



03901A-23

Figure 2-21 Horizontal Total Count Register



03901A-24

Figure 2-22 Horizontal Total Display Register

VSW $= 17_{10} = 11_H$

Vertical Total Line Register (VTOT): The number of vertical total lines equals to the number of scan lines (637) minus the Vertical Sync Width (VSW). (see Figure 2.13)

VTOT+1 $= 637 - (\text{VSW} + 1) = 619$
VTOT $= 618_{10} = 26A_H$

Vertical Active Line Register

This value is the total scan line number of the screen minus the number of scan lines contained in the bottom border area (10% of the screen height):

VAL + 1 $= 0.9 \bullet (\text{VTOT} + 1)$
$= 0.9 \bullet 619 = 557$ scan lines
VAL $= 556_{10} = 22C_H$

Vertical Total Line Register:

VTOT + 1 $= 637-(\text{VSW} + 1) = 619$
VTOT $= 618_{10} = 26AH$

Vertical Sync Width/Vertical Scan Delay Register

Vertical Sync Width (VSW)=11_H (as computed above)

Vertical Scan Delay (VSD): (see Figure 2.13)

VSD + 1 $= (\text{VAL} + 1)-510$
VSD $= 46_{10} = 2E_H$
VSD shifted six bits left to fit the field in the register.
$\text{VSD}_{\text{shift}} = B80_H$

VSW/VSD Register $= \text{VSD}_{\text{shift}} + \text{VSW}$
$= 0B80_H + 11_H = 0B91_H$

Horizontal Sync and Vertical Interrupt Row Register

VERTINT is set to "0" in this example.
HSYNC + 1 = 3 microseconds $= 3 \bullet 3$
= 9 character clocks
HSYNC $= 8_{10} = 8_H$

Horizontal Drive Register

This is a "don't care" since HOS=0. (HSYNC selected)

Horizontal Scan Delay Register

HSD + 1 = (HSYNC + 1) + (HSYNC to HBLANK delay)
HSD + 1 = (HSYNC + 1) + [HTC + 1 – (HSYNC +1) – number of displayed characters]/2
HSD + 1 $= (100-9-80)/2+9 = 15$ character clocks
HSD $= 14_{10} = 0E_H$

Horizontal Total Count Register

HTC + 1 = 100 character clocks
HTC $= 99_{10} = 63_H$

Horizontal Total Display Register:

HTD + 1 = number of characters displayed + (HSD +1)
HTD + 1 $= 80 + 15$
HTD $= 94_{10} = 5E_H$

2.4 DMA OPERATIONS

Once the CRTC has been initialized and the various registers programmed to meet the application's needs, the CRTC is responsible for initiating System Bus Requests to fetch Control Data and Display Data from memory and to transfer them into its on-board registers and row buffers, respectively. The CRTC requests the bus after the DE-bit in Mode Register 1 has been set to a "1."

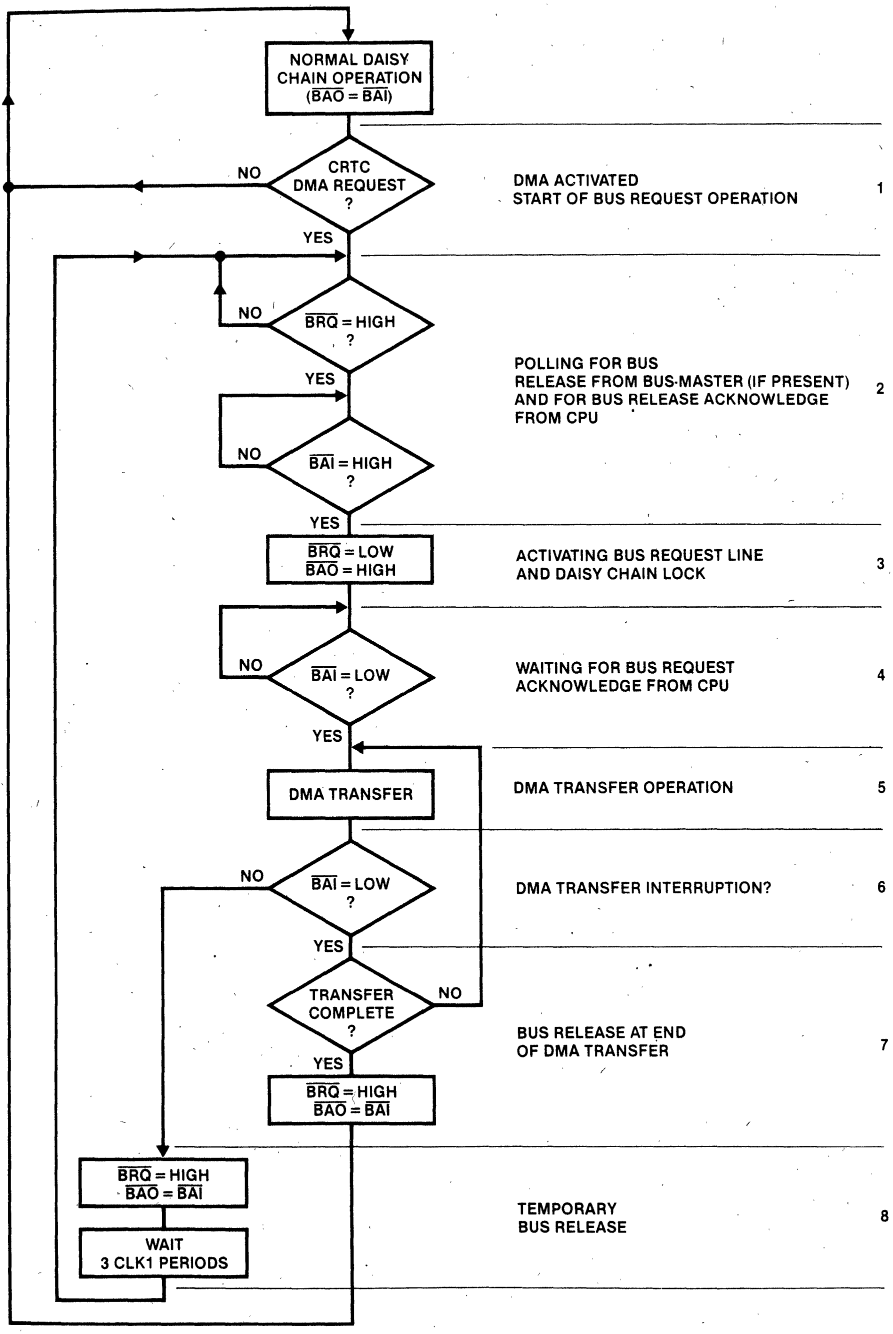
DMA Signals and Protocol

Before the CRTC can perform a DMA operation, it must gain control of the System Bus. The $\overline{\text{BRQ}}$, $\overline{\text{BAI}}$ and $\overline{\text{BAO}}$ interface pins constitute the basic interface between the CRTC and other devices capable of bus arbitration (e.g. microprocessors and other DMA devices). Whenever the CRTC requests bus control, the operation is executed according to the flowchart in Figure 2.23. The DMA sequence can described as the following:

1. If the CRTC needs to perform a DMA access, it triggers the bus request operation.

2. First, it checks whether the bus is being used by another peripheral device by polling the $\overline{\text{BRQ}}$ line until it is High. Then, it waits for the CPU to gain bus control. This is indicated through the daisy-chain ($\overline{\text{BAI}}$=High).

3. At that time the bus is under control of the


NORMAL DAISY CHAIN OPERATION ($\overline{BAO}$ = $\overline{BAI}$)
CRTC DMA REQUEST ?
NO
YES
$\overline{BRQ}$ = HIGH ?
NO
YES
$\overline{BAI}$ = HIGH ?
NO
YES
$\overline{BRQ}$ = LOW
$\overline{BAO}$ = HIGH
$\overline{BAI}$ = LOW ?
NO
YES
DMA TRANSFER
$\overline{BAI}$ = LOW ?
NO
YES
TRANSFER COMPLETE ?
NO
YES
$\overline{BRQ}$ = HIGH
$\overline{BAO}$ = $\overline{BAI}$
$\overline{BRQ}$ = HIGH
$\overline{BAO}$ = $\overline{BAI}$
WAIT 3 CLK1 PERIODS
DMA ACTIVATED START OF BUS REQUEST OPERATION 1
POLLING FOR BUS RELEASE FROM BUS-MASTER (IF PRESENT) AND FOR BUS RELEASE ACKNOWLEDGE FROM CPU 2
ACTIVATING BUS REQUEST LINE AND DAISY CHAIN LOCK 3
WAITING FOR BUS REQUEST ACKNOWLEDGE FROM CPU 4
DMA TRANSFER OPERATION 5
DMA TRANSFER INTERRUPTION? 6
BUS RELEASE AT END OF DMA TRANSFER 7
TEMPORARY BUS RELEASE 8

03901A-25

Figure 2-23 DMA Bus Request Flow Chart

CPU, and the CRTC can issue its request by pulling BRQ Low. It also inhibits Bus Acknowledge from propagating to lower priority devices (in the lower part of the daisy-chain) by pulling BAO High; this avoids granting the Bus to lower priority devices which may have issued BRQ at the same time as the CRTC.

4. Before initiating any DMA transfer, the CRTC waits for bus request acknowledge from the CPU by polling its BAI input.

5. The CRTC now acts as Bus Master and performs the required transfers.

6. The CRTC DMA transfer can be temporarily interrupted by removing Bus Acknowledge In (BAI=High)—external bus preemption. The CRTC requires that BAI is active for a minimum of four clocks. If the CRTC is preempted within the first four clocks, the CRTC might not detect the bus acknowledge causing the CRTC to keep waiting for BAI Low. The result is that the bus arbitration locks up. To overcome this lock condition either the minimum width of BAI must be guaranteed or the external arbiter must be able to recover from this lock condition (detect of lock, then temporary release the preempting signal).

7. The CRTC terminates the transfer when it has filled the internal row buffers or when the burst count reaches zero. The bus is released (BRQ=High) and bus acknowledge ripples through (BAO=BAI). Then either the CPU or a lower priority device on the daisy chain can gain control of the bus. The lower priority device might have pulled BRQ Low concurrently with the CRTC and is waiting for BAI=Low to start its activity.

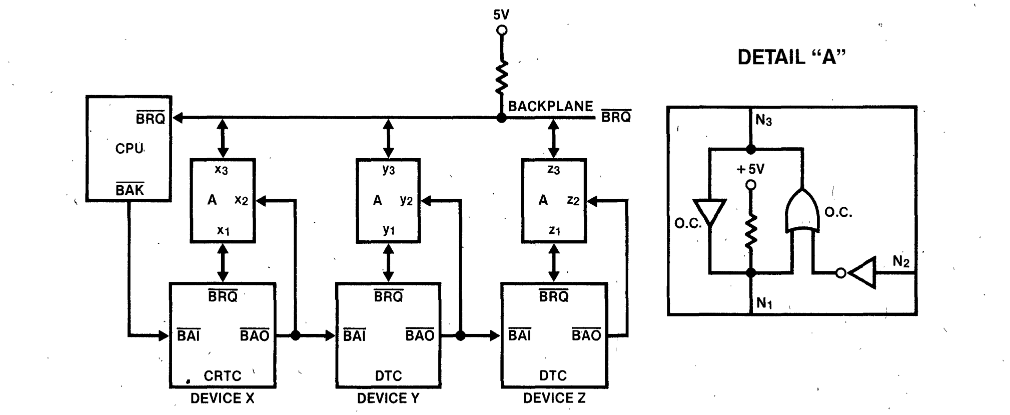
8. The CRTC DMA transfer is interrupted by removing BAI. The CRTC finishes the current bus cycle and releases the bus for three system clocks (BRQ=High, BAO=BAI). Then it tries to resume DMA activity and continues DMA operations and burst count from where it was interrupted.

Buffering BRQ

When BRQ needs to be buffered (for example, to drive a system backplane), a specific bidirectional interface buffer must be used. Such an interface and its implementation is described below:

Detail "A" in Figure 2.24 shows the BRQ buffer logic. Note that the "buffer" and the "OR gate" are both open collector (OC) devices. When the backplane BRQ is High, and no DMA device requested the bus, then all BAI's and BAO's are High, hence X3 and X2 are High and X1 is driven High.

If device X requests the bus, it locks BAO High and pulls X1 Low to initiate a bus request, which in turn pulls X3 Low since X2 is High (BAO=High). The detail "A" logic is then locked into this state through the open collector buffer, as the CPU and the other detail "A" interfaces on the bus. All these interfaces are locked the same way as the



03901A-26

Figure 2-24 System with Multiple DMA Devices

requesting one. A few cycles later, the CPU acknowledges the bus request by pulling BUSACK Low, the CRTC (device X) then executes its transfers. When the CRTC finishes its transfers, it releases BRQ and relinks its BAI input to BAO output, hence driving BAO Low. The Low propagates through the daisy-chain, and as long as one of the BAO is High, the backplane BRQ line and the devices BRQ signals will be held Low due to detail "A" logic structure.

Once all the BAO's have gone High, the backplane BRQ goes High, and the CPU gains control over the bus.

DMA Transfer Operation

The DMA transfer itself consists of data moves from memory into the CRTC, controlled by the CRTC's DMA unit.

If a control block is fetched, the words loaded are steered toward the internal control registers. If display data (characters or attributes) are fetched from memory, it is steered toward an internal row buffer.

In both cases the CRTC must:

1. Output the address of the data location.
2. Sample the WAIT input and stretch the read cycle if needed. WAIT is sampled only at the falling edge of the system clock in T2 of a Bus Master Read cycle.
3. Read the data and transfer it to the proper destination (buffer or internal register).

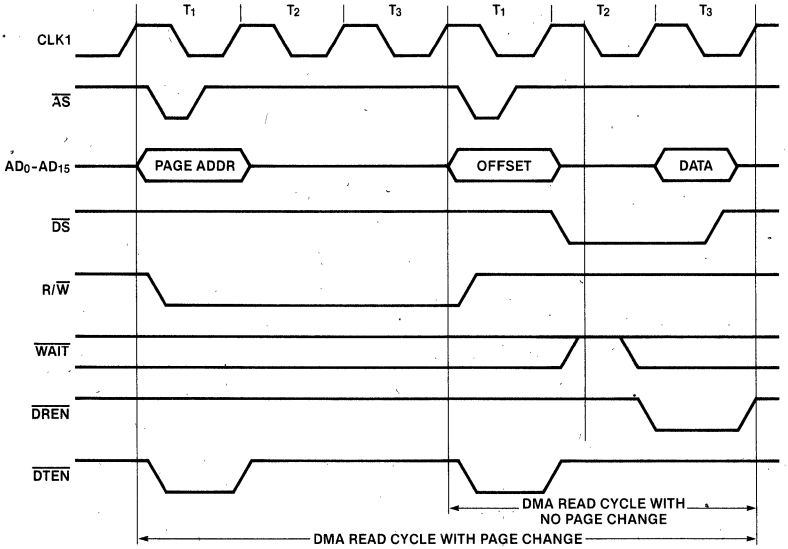
The Am8052 can address up to 16-Mbyte addresses as 256 pages of 64K bytes each. The upper address is updated on a demand basis, as outlined below:

There is a upper address change between the previous fetch cycle and the current one, or this is the first fetch of a new frame. In either case, succeeding read cycles are preceded by a single write cycle to latch the new upper address address. (See Figure 2.25)

There is no upper address change since the previous fetch cycle and it is not the first fetch of a new frame. In this case the succeeding fetches are not preceded by a upper address write cycle. A new burst does not necessarily begin with a page address update.

DMA Read and Write Operations

The start of a DMA cycle is initiated by AS being


T1
T2
T3
T1
T2
T3
CLK1
AS
AD0-AD15
PAGE ADDR
OFFSET
DATA
DS
R/W
WAIT
DREN
DTEN
DMA READ CYCLE WITH NO PAGE CHANGE
DMA READ CYCLE WITH PAGE CHANGE

03901A-27

Figure 2-25 DMA Transfer Operation

driven Low, which indicates a valid address on the AD_0-AD_{15} address/data lines. At that time DTEN is also driven Low and allows the valid address to be buffered on the system bus through external buffers. The valid address may be latched on the system bus on the rising edge of AS.

During the first portion of a DMA read cycle with a page change, R/W is pulled Low by the CRTC for three complete clock cycles, and the address present on the AD_0-AD_7 bus during T1 is the updated page address which should be latched externally on the rising edge of AS. Refer to section 6 on interfacing the upper address latch. The CRTC never outputs an active DS during a write cycle. The next three clock cycles represent a normal DMA read cycle.

During T2 the CRTC ceases driving the AD_0-AD_{15} bus with the address information, and DTEN goes inactive (HIGH). DS is driven Low as an indication to the memory system that it may drive the bus with the read data. Half of a clock cycle later, DREN is driven Low to enable the receiving buffers local to the CRTC.

Data is captured by the CRTC on the falling edge of the T3 clock cycle; then both DS and DREN return High. The system might turn off the data with either DS or DREN. In both cases the data hold time required by the CRTC is satisfied.

Wait Operation

During T2 of the read cycle, the WAIT signal is sampled by the falling edge of CLK_1. If Low, the cycle is stretched by one CLK_1 cycle. However, the WAIT input can be operated as a READY input, by taking Low as the default level. In both cases, the input signal must satisfy the setup and hold time requirements of the CRTC, to avoid metastable conditions (see Section 6).

The CRTC also has a software Wait state capability: zero, one or two wait states can be specified in Mode Register 1 and are automatically inserted in each Bus Master Read cycle independently of the WAIT input line.

When both hardware and software Wait states are requested, they occur consecutively and not concurrently: The hardware Wait States are honored first, immediately followed by software wait states if so programmed.

Idle DMA Cycles

An Idle DMA cycle is a bus cycle (three clocks) during which the CRTC executes internal operations (e.g., row linkage and window overlay). Since Idle DMA cycles are single bus cycles, the CRTC does not release the bus; otherwise, bus overhead would be increased. The CRTC releases the bus (burst of Idle DMA Cycles) only if a window or the background row needs to be filled with Fill Code characters.

Each DMA burst executes in the following sequence:

1. The CRTC asserts BRQ to arbitrate the bus.
2. The CRTC waits for BAI to be asserted by the external bus arbiter (usually a CPU).
3. BAI is sampled with the next rising edge of CLK_1. If the set-up time (parameter 75) is not satisfied, the CRTC may perhaps not catch BAI with that edge, but definitely catches it with the next edge (metastable conditions cannot occur).
4. Then BAI is internally synchronized to T2 of the running state machine. After synchronization the CRTC executes the first DMA cycle, which externally starts on the next T1 state. The time elapsed from receiving BAI is between six and eight clocks depending on when BAI comes relative to the free running internal state machine.

Table of Idle DMA Cycles:

The table below lists conditions were the CRTC inserts Idle DMA cycles (this list might not be complete).

Event	# of Idle DMA Cycles
Begining of DMA burst if previous burst was preempted or counted out	0
Begining of the first burst of a frame	1
Begining of first burst for a new row	2
Loading the Window Definition Block	1
Loading a Row Redefinition Block	1
Loading a Window Row Control Block	1
End of a row (background)	1
(window)	2
End of preempted burst	0
Fill Code segment (segment with character pointer equal zero)	1
Window segment filled with Fill Code	3clks/2char

DMA Burst Control

During DMA action, the CPU is denied access to the bus and therefore cannot execute programs. This situation can lead to problems in the interrupt response time of the CPU, since the CPU can only recognize and service an interrupt request while in control of the bus. Note that at the beginning of every frame, immediately after the vertical blanking interval, the CRTC tries to request the bus.

To allow the CPU control of the bus within certain limits, a Burst Register is provided inside the CRTC and is programmable by the CPU. This Burst Register specifies a time slot during which the CRTC is allowed to request the bus. Both the time slot duration and its cycle time are programmable. For further information, refer to Section 2.3.

2.5 ROW MANAGEMENT UNIT OPERATIONS

The Row Management Unit controls the system for fetching, interpreting, and steering the information contained in memory; loading the three row-buffers with displayable information; and updating internal registers to redefine some of the screen characteristics.

Listed below is the information that the Row Management Unit may steer for updating.

Steer into the row-buffers:

- characters
- attributes

Steer into the internal registers:

alterable on a *frame* basis:

- absolute cursor coordinates (CUX, CUY)
- fill character code
- blink control and parameters (for cursors and characters)
- scroll control and parameters
- interrupt vectors (for vertical event and smooth-scroll event

alterable on a *row* basis:

- total scan line count per row (TSLC)
- normal character start and end line numbers (NCS, NCE)
- superscript character start and end scan-line numbers (SBCS, SBCE)
- subscript character start and end scan-line numbers(SBCS, SBCE)
- cursor pattern start and end scan-line numbers (CURS, CURE)
- underline position (UND)
- shifted underline position (SUND)

The information to be fetched by the Row Management Unit is addressed by linked-list pointers, and the Row Management Unit keeps track of the addresses of the information present in memory. The Row Management Unit also interprets window information when it is present.

The final task performed by the Row Management Unit is the selection of displayable characters (which are the only ones loaded into the row buffers) depending upon the "ignore" and "invisible attribute flag" bits settings.

Windows

The CRTC is capable of controlling and displaying a text file on the screen (known as background) concurrently with other text files embedded in rectangles (known as windows) positioned anywhere inside the active display area of the screen. With conventional CRT controllers, this feature can only be implemented if the CPU is aware of the position and size of the window, with all the inconvenience and software complexity this implies. One of the important features of the CRTC is that it allows the CPU to process a background file and a window file independently without being continuously concerned with size and position of the window.

The CRTC holds two pointer registers; each containing the starting address of a linked-list residing in memory: one pointer corresponds to the background information, while the other corresponds to the first window's information. The first window is the first one encountered when scanning the screen from top to bottom. The user is able to define an arbitrary number of windows on the screen, as long as two background character rows (three for interlaced video) separate the windows vertically. Virtual windows, however, may occur side by side (horizontal split-screen).

Each window links to the following one (ranging from top to bottom of the screen) with a link pointer. There are no more windows when the link pointer of the last window contains zero.

Two main linked-lists reside in system memory holding the entire information defining a particular display:

The background list pointed to by Top of Page (TOP) Register, containing the parameters of the background display.

The window(s) list pointed to by Top of Window (TOW) Register, containing the parameters of the window(s) display.

Depending upon the memory addressing scheme, the user can choose either of two addressing modes: segmented mode or linear mode.

Segmented Mode

The segmented mode divides the memory into pages containing 64K bytes each. The CRTC can address 128 pages. In this case, the pointer is 23 bits wide arranged in two 16-bit words with the following configuration:

Seven bits pointing to one page among the 128 addressable pages. These seven bits are right justified in the most significant byte of the first 16-bit word.

16 bits pointing to the address within the selected page. These 16 bits constitute the second word.

When operating in the segmented mode, crossing a page boundary does not increment the page number. It results in wrap-around operation within the same page.

Linear Mode

In the linear mode the CRTC addresses memory as one 16-megabyte block, with a 24-bit-wide pointer arranged in two 16-bit words with the following configuration:

Eight bits representing the most significant part of the address embedded in the least significant byte of the first word.

16 bits representing the least significant part of the address in the second word.

In this mode, when the second word crosses a 64K boundary, the first word is incremented by one.

The selection between these two modes is accomplished through the L/S bit in Mode Register 1.

L/S=0 segmented mode enabled
L/S=1 linear mode enabled

Consistent with the byte addressing method used by all 16-bit microprocessors, AD$_0$ always outputs a "0" at address time. This means that the CRTC actually addresses 32K 16-bit words instead of 64K bytes. This applies for both linear and segmented addressing modes. This implies that all character strings must start at an even address — *they have to be word boundary aligned.*

Background Information Management

The TOP (Top Of Page) Register points to the first data word of a block called "Main Definition Block." This block is unique for each background list, and the information it contains is fetched on a frame basis and stored into the applicable internal registers of the CRTC. Simply by changing the pointer in the TOP register entire pages can be swapped at an instant without any flickering.

Main Definition Block (MDB) Overview

The Main Definition Block contains seven data words (MD$_0$–MD$_6$) defined as follows (Figures 2.26 and 2.27):

	D_{15}							D_8	D_7							D_0
MD_0	▨	ROW CONTROL BLOCK POINTER (PAGE)							▨							
MD_1	ROW CONTROL BLOCK POINTER (OFFSET)															
MD_2	x-CURSOR (CUX)								y-CURSOR (CUY)							
MD_3	▨							FAT	FILL CHARACTER CODE							
MD_4	CHD	CHB_1	CHB_0	CATBE	CXYBE	CUD	CUB_1	CUB_0	▨	SR_3	SR_2	SR_1	SR_0	SWB	SUD	SSE
MD_5	INT VECTOR (VERTICAL)								INT VECTOR (SOFT SCROLL)							
MD_6	▨	TSLC					▨									

03901A-28

Figure 2-26 Main Definition Block (L/S = 0)

MD_0,MD_1. Pointer to first Row Control Block
MD_2. Absolute cursor coordinates ("X" coordinate byte and "Y" coordinate byte)
MD_3. Fill character code (one flag bit + one byte code)
MD_4. Blink control/scroll control
MD_5. Interrupt vectors: vertical event/scroll event
MD_6. Total scan line count per row.

MDB Detailed Description:

MD_0,MD_1. The Row Control Block pointer points to the block defining the first row's control information.

MD_2. The absolute cursor coordinates indicate the row number and the character position within this row where the absolute cursor is displayed. The topmost row is row "0" the leftmost character position is "0".

MD_3. The fill character code is a user-defined 8-bit code. This is used as a filler in the row buffer if all the characters for that row have been loaded and did not fill the programmed buffer size. Segments with a character code pointer of "0" are also filled with the fill code. The number of visible characters (visible #) specifies the length of these segments. Windows, where the window segments do not fill up the window size, are filled by the fill code too. The flag bit (flag attribute), when set, causes the CRTC to load an extra attribute word from the attribute list and use it as a latched attribute (immediately active) for the fill character. The extra attribute word must invoke a latched attribute.

MD_4. The blink control/scroll control is composed of 15 bits.

Smooth-Scroll Enable (SSE) enables the smooth-scroll operation for either the background or a window.

0 Smooth-scroll disabled
1 Smooth-scroll enabled

Scroll Up/Down (SUD) indicates the direction of the scroll.

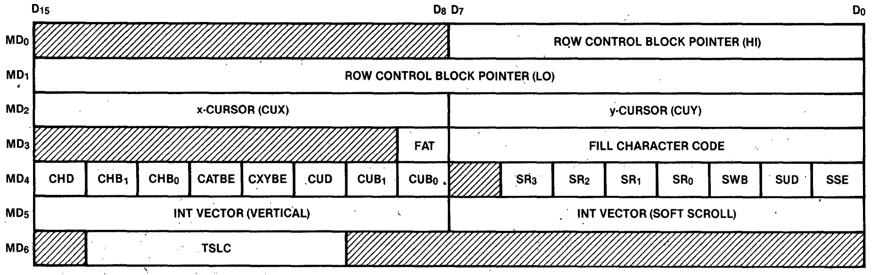
0 Smooth-scroll down
1 Smooth-scroll up

Scroll Window/Background (SWB) indicates whether the background or a window will be scrolled.

0 Smooth-scroll background
1 Smooth-scroll window

Scroll Rate (SR_3–SR_0) is a 4-bit word specifying the smooth-scroll rate according to the following table:

SR_3	SR_2	SR_1	SR_0	Scroll Rate
0	0	0	0	1 Scan Line/Frame
0	0	0	1	2 Scan Lines/Frame
0	0	1	0	3 ScanLines /Frame
0	0	1	1	4 Scan Lines/Frame
0	1	0	0	5 Scan Lines/Frame
0	1	0	1	6 Scan Lines/Frame
0	1	1	0	7 Scan Lines/Frame
0	1	1	1	8 Scan Lines/Frame (fastest)
1	0	0	0	1 Scan Line/Frame
1	0	0	1	1 Scan Line/2 Frames
1	0	1	0	1 Scan Line/3 Frames
1	0	1	1	1 Scan Line/4 Frames
1	1	0	0	1 Scan Line/5 Frames
1	1	0	1	1 Scan Line/6 Frames
1	1	1	0	1 Scan Line/7 Frames
1	1	1	1	1 Scan Line/8 Frames (slowest)



03901A-29

Figure 2-27 Main Definition Block (L/S = 1)

Cursor Blink Rate (CUB1,CUB0) defines the blinking rate for both attribute and absolute cursors:

CUB_1	CUB_0	Blink Period	Blink Frequency (at 60 Hz Frame Rate)
0	0	16 Frames	3.75 Hz
0	1	32 Frames	1.85 Hz
1	0	64 Frames	0.93 Hz
1	1	128 Frames	0.46 Hz

Cursor Blink Duty Cycle (CUD)

CUD	Cursor Blink Duty Cycle
0	Blink Output 75% Inactive, 25% Active
1	Blink Output 50% Inactive, 50% Active

Character Blink Duty Cycle (CHD)

CHD	Character Blink Duty Cycle
0	Blink Output 75% Inactive, 25% Active
1	Blink Output 50% Inactive, 50% Active

Absolute Cursor Blink Enable (CXYBE)

0	Cursor Blink Disable
1	Cursor Blink Enable

Attribute Cursor Blink Enable (CATBE)

0	Cursor Blink Disable
1	Cursor Blink Disable

Character Blink Rate (CHB_1, CHB_0)

CHB_1	CHB_0	Blink Period	Blink Frequency (at 60 Hz Frame Rate)
0	0	16 Frames	3.75 Hz
0	1	32 Frames	1.85 Hz
1	0	64 Frames	0.93 Hz
1	1	128 Frames	0.46 Hz

The character and the cursor can have different blink rates and different duty cycles.

MD_5. The Interrupt Vector Register contains the smooth-scroll and vertical event interrupt vectors. When one of these interrupts is activated, the corresponding 8-bit vector is output on AD_7–AD_0 at Interrupt Acknowledge time, if the NV-bit in Mode Register 2 is reset.

The vertical event interrupt vector is totally user-programmable.

The smooth-scroll interrupt vector is partially user-programmable: Bits 0 and 2 through 7 are user-definable, while Bit 1 reflects the state of the SIP (Scroll Interrupt Pending) bit. This feature allows the user to steer the smooth-scroll interrupts into two different routines.

SIP=1	The CRT is informing the CPU to execute a relink during scrolling operation.
SIP=0	The CRT does not need CPU intervention but signals the CPU that the scroll operation is completed.

MD_6. TSLC is a 5-bit value defining the number of total scan lines per row minus one. This value is reprogrammable on a row basis via the Row Definition Block.

This TSLC must be equal to the TSLC of the first row in the linked-list.

In video interlace or RFI mode, the TSLCs of all rows displayed must be even or the TSLCs of all rows must be odd. In non-interlaced video, rows with odd and even TSLCs may be mixed. However, this is restricted when displaying windows (refer to Section 2.5.4). Figure 2.28 shows the values of the total number of scan lines for all video modes.

Row Control Block (RCB)

Once the CRTC has loaded the Main Definition Block into its internal registers, it fetches the first Row Control Block (Figures 2.29 and 2.30). To ease text-editing procedures, the CRTC allows the user to split each row into segments. This partitioning is necessary when dealing with window positioning within the screen. The "window" section provides detailed information. Each segment may contain up to 255 visible characters and up to 255 hidden characters limited by the 8-bit counter.

Hidden characters are characters that the CRTC fetches from system memory but that are not loaded into the internal row buffers. They are identified by the Ignore Bit of the attribute word when DH in Mode Register 1 is reset. An attribute flag character is also a hidden character if the

Invisible Attribute Flag (IAF) of Mode Register 1 is set.

The CRTC pre-fetches two rows to keep all three internal row buffers filled. This results in fetching two redundant rows at the bottom of the screen. To minimize bus occupancy of the CRTC these last two rows can be "termination Row Control Blocks." This block consists of a Row Control Block Pointer pointing to itself, a Character Code Pointer set to "0," and C-flag=0 (a single, empty character segment).

RCB Overview

RA_0,RA_1. A two-word link pointer pointing to the next Row Control Block.

RA_{2-6}. The first segment's block composed of five data words:

> The numbers of visible and hidden characters in the segment constitute the first data word,
> The segment's character-list pointer (next two data words),
> The segment's attribute-list pointer (two words),
> Successive segments are identical to the first,
> An optional "Row Redefinition Block" pointer (two data words).

The user must set at least one Row Redefinition Block after power-up. A Row Redefinition Block contains characteristics applicable to a row. This information stays latched until another Row Redefinition Block is encountered. If no Row Redefinition Block is fetched after power up, information such as character start and end scan lines is undefined. If N segments are present in a Row Control Block, its length is either:

N • 5 + 2 if no Row Redefinition Block is present,
N • 5 + 4 if a Row Redefinition Block is present.

RCB Detailed Description:

RA_0,RA_1. The most significant bit in the first word indicates if a Row Definition Block has to be loaded for the current row. When this flag (LNK) is "1," the Row Definition Block is loaded. The remainder of the first two words contain the link pointer to next Row Control Block.

RA_2. The sum of hidden and visible characters must be at least "1". The number of hidden characters and the number of visible characters are interpreted by the CRTC in the following way:

No window within the current row

The DMA uses the sum of the hidden and visible character numbers to determine the number of characters to be fetched. In this case the CRTC does not distinguish between those two numbers; it uses only the sum. Note, that the segment length is not determined by # Visible. The segment length is only determined by the number of visible characters the CRTC extracts out of the characters loaded in by DMA.

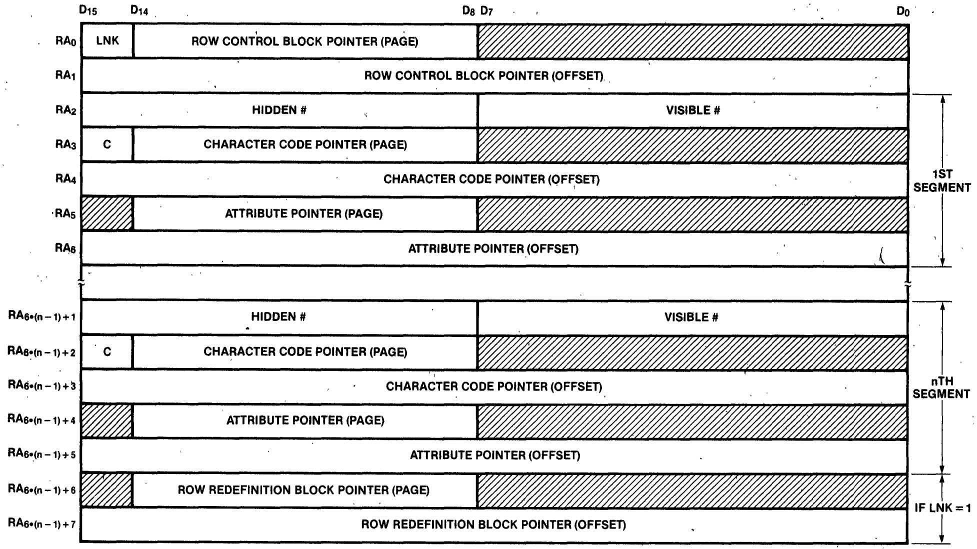
Window within the current row

In a window, both the number of hidden and visible characters in background, and the number of window segments have to be specified correctly. The total number of hidden and visible characters determines the number of characters fetched from memory. The CRTC takes the number of visible characters in the segment and the window coordinates of the Window Block in order to place the window. The specified number of visible characters for a particular segment has to match the number the CRTC extracts from the characters loaded by DMA.

TOTAL NUMBER OF SCAN LINES

TSLC	NON-INTERLACED OR RFI MODE	INTERLACED MODE
00000	1	1 + 1 = 2
00001	2	1 + 2 = 3
00010	3	2 + 2 = 4
00011	4	2 + 3 = 5
00100	5	3 + 3 = 6
00101	6	3 + 4 = 7
00110	7	4 + 4 = 8
00111	8	4 + 5 = 9
01000	9	5 + 5 = 10
01001	10	5 + 6 = 11
01010	11	6 + 6 = 12
01011	12	6 + 7 = 13
01100	13	7 + 7 = 14
01101	14	7 + 8 = 15
01110	15	8 + 8 = 16
01111	16	8 + 9 = 17
-		
-		
-		
-		
-		
11111	32	16 + 17 = 33

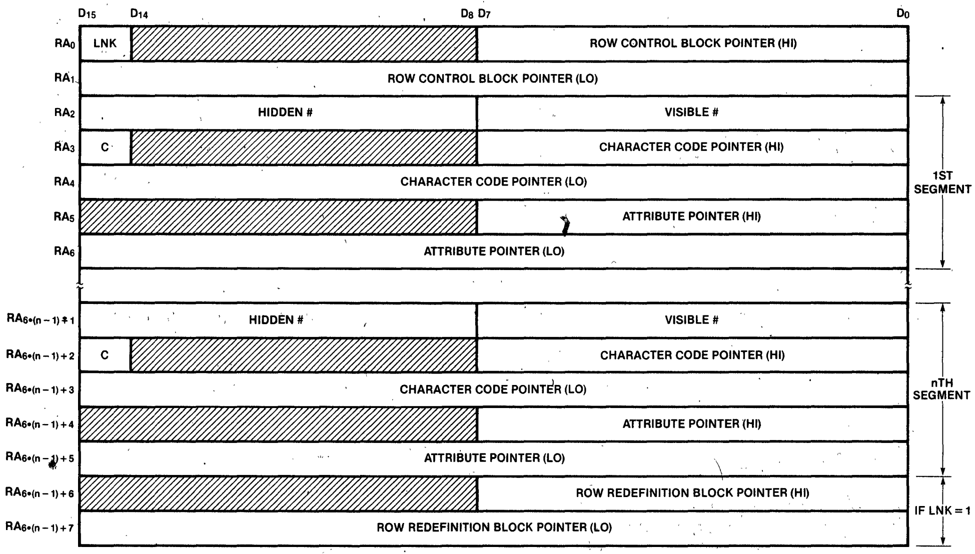
Figure 2-28 Total Number Of Scan Lines As A Function Of TSLC



	D15	D14 – D8	D7 – D0	
RA_0	LNK	ROW CONTROL BLOCK POINTER (PAGE)	▨	
RA_1	ROW CONTROL BLOCK POINTER (OFFSET)			
RA_2	HIDDEN #		VISIBLE #	1ST SEGMENT
RA_3	C	CHARACTER CODE POINTER (PAGE)	▨	
RA_4	CHARACTER CODE POINTER (OFFSET)			
RA_5	▨	ATTRIBUTE POINTER (PAGE)	▨	
RA_6	ATTRIBUTE POINTER (OFFSET)			
$RA_{6 \bullet (n-1)+1}$	HIDDEN #		VISIBLE #	nTH SEGMENT
$RA_{6 \bullet (n-1)+2}$	C	CHARACTER CODE POINTER (PAGE)	▨	
$RA_{6 \bullet (n-1)+3}$	CHARACTER CODE POINTER (OFFSET)			
$RA_{6 \bullet (n-1)+4}$	▨	ATTRIBUTE POINTER (PAGE)	▨	
$RA_{6 \bullet (n-1)+5}$	ATTRIBUTE POINTER (OFFSET)			
$RA_{6 \bullet (n-1)+6}$	▨	ROW REDEFINITION BLOCK POINTER (PAGE)	▨	IF LNK = 1
$RA_{6 \bullet (n-1)+7}$	ROW REDEFINITION BLOCK POINTER (OFFSET)			

03901A-30

Figure 2-29 Row Control Block (L/S = 0)



	D15	D14 – D8	D7 – D0	
RA_0	LNK	▨	ROW CONTROL BLOCK POINTER (HI)	
RA_1	ROW CONTROL BLOCK POINTER (LO)			
RA_2	HIDDEN #		VISIBLE #	1ST SEGMENT
RA_3	C	▨	CHARACTER CODE POINTER (HI)	
RA_4	CHARACTER CODE POINTER (LO)			
RA_5	▨		ATTRIBUTE POINTER (HI)	
RA_6	ATTRIBUTE POINTER (LO)			
$RA_{6 \bullet (n-1)+1}$	HIDDEN #		VISIBLE #	nTH SEGMENT
$RA_{6 \bullet (n-1)+2}$	C	▨	CHARACTER CODE POINTER (HI)	
$RA_{6 \bullet (n-1)+3}$	CHARACTER CODE POINTER (LO)			
$RA_{6 \bullet (n-1)+4}$	▨		ATTRIBUTE POINTER (HI)	
$RA_{6 \bullet (n-1)+5}$	ATTRIBUTE POINTER (LO)			
$RA_{6 \bullet (n-1)+6}$	▨		ROW REDEFINITION BLOCK POINTER (HI)	IF LNK = 1
$RA_{6 \bullet (n-1)+7}$	ROW REDEFINITION BLOCK POINTER (LO)			

03901A-31

Figure 2-30 Row Control Block (L/S = 1)

RA_3,RA_4. These two words contain the character code address pointing to the beginning of the character code string of this segment and the continue bit (C).

C=0 This is the last segment of this row.
C=1 The segment list continues.

If this pointer is "0," then the space specified by the visible number of characters for this segment is filled with the fill code.

RA_5,RA_6. The pointer links to the attribute string of this segment.

The segment header (RA_3–RA_6) must be repeated for each additional segment. If the LNK-bit in RA_0 is set, the two words following the last segment header must contain the pointer to the Row Redefinition Block.

Row Redefinition Block

The Row Redefinition Block is composed of five words. These words hold information relevant to the display characteristics of the row (Figure 2.31).

RR_0	Total Scan Line Count (TSLC)	5 Bits
	Normal Character Start (NCS)	5 Bits
	Normal Character End (NCE)	5 Bits
RR_1	Row Attributes	5 Bits
	Superscript Character Start (SPCS)	5 Bits
	Superscript Character End (SPCE)	5 Bits
RR_2	Row Attributes	5 Bits
	Subscript Character Start (SBCS)	5 Bits
	Subscript Character End (SBCE)	5 Bits
RR_3	Cursor Start	5 Bits
	Cursor End	5 Bits
RR_4	Double Row (DR)	2 Bits
	Underline(UND)	5 Bits
	Shifted Underline(SUND)	5 Bits

All this information is captured by the CRTC. It acts on the invoking character row and succeeding ones until a new Row Redefinition Block is invoked.

The Total Scan Line Count (TSLC) defines the total number of scan lines per row minus one.

Normal Character Start (NCS) and End (NCE) define the vertical position and height of normal characters within the row.

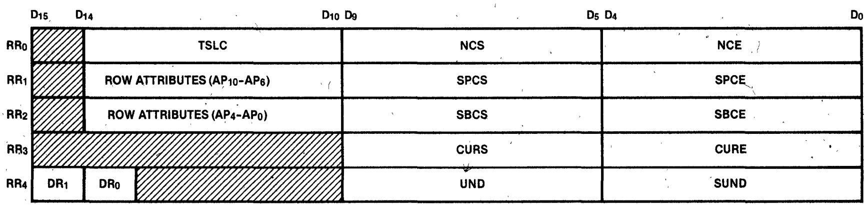
The same definition applies to superscript and subscript characters with SPCS, SPCE, SBCS, SBCE.

When the scan line count is less than the character start scan line value (NCS, SPCS, or SBCS) or larger than the character end scan line value (NCE, SPCE, or SBCE), R_0–R_4 puts out $1F_H$. Figure 2.32 shows an example. Normally the character slice with the address $1F_H$ is programmed to be blank.

More details concerning these parameters are included in Section 2.6, Attributes.

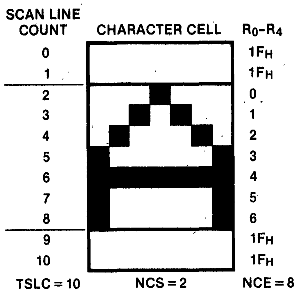
There are ten user-definable row attribute bits which are output on the AP_0–AP_4 and AP_6–AP_{10} pins during the horizontal retrace time. Bits D_{14} through D_{10} in RR_1 are output on AP_{10}–AP_6, while bits D_{14} through D_{10} in RR_2 are output on AP_4–AP_0. This row attribute can be registered externally to the CRTC with the falling edge of HSYNC. This feature can be used for a set of user-definable attributes or to implement functions which are not directly supported by the CRTC; for example, loadable character font generator or horizontal smooth-scroll. Cursor start and cursor end applies to partial, reverse and underline cursors, and defines the position and height of the corresponding cursor. (See Section 2.6.5, Cursor Display.)

The Double Row bits (DR_1,DR_0) allow the user to insert double row characters in the text on a row



03901A-32

Figure 2-31 Row Redefinition Block



03901A-33

Figure 2-32 Character Placement

basis. The code is interpreted as follows:

DR_1	DR_0	
0	0	Normal Character Row
0	1	Reserved
1	0	Top Half of a Double Row
1	1	Bottom Half of a Double Row

The linked-list for a double size row consists of two Row Control Blocks, one for the top half of the row and one for the bottom half. The data accessed by these row control blocks should be identical, apart from the DR bits in the Row Redefinition Blocks.

Underline specifies the scan line number on which the underline attribute acts.

Shifted underline acts the same way as underline except that it applies to the shifted underline attribute.

Row Redefinition Block Loading Process

If RCB_n initiates loading of a Row Redefinition Block (LNK=1), the CRTC will load the same Row Redefinition Block also for Row_{n+1} (LNK=0) and Row_{n+2} (LNK=0) to get the new parameters also for the two remaining row buffers. Note, that for these two rows the CRTC only loads the Row Redefinition Block (5 words) and not the Row Redefinition Block Pointer (last two words of the RCB). This means, that the Row Redefinition Block should not be modified, until the CRTC has fetched these two rows.

Window Information Management

The Top Of Window Register (TOW) points to the first word of a Window Definition Block (WDB), which specifies the window characteristics. There is one Window Definition Block per window, and they are linked together starting with the topmost window on the screen (whose WDB is pointed by TOW). If TOW=0, no window is displayed on the screen.

The Window Definition Block defines the following parameters (see Figures 2.33 and 2.34):

WD_0,WD_1. First Window Control Block link-pointer (two words)

WD_2,WD_3. Next Window Definition Block link-pointer (two words)

WD_4. The start and end window row numbers (one word)

WD_5. The start and end window character numbers (one word)

The Window Row Control Block Point points to the Window Row Control Block specifying the first row of the window. The most significant bit of WD_0 (Smooth-Scroll Window, SCW) indicates if this particular window should be scrolled:

SCW	Smooth-Scroll Window
0	Window Smooth-Scroll Disabled
1	Window Smooth-Scroll Enabled

Note, that smooth-scrolling does not occur until conditions specified in the Main Definition Block are satisfied.

When the pointer to the next Window Definition Block is equal to zero, there are no more windows on the screen. Otherwise, the pointer indicates the address of next Window Definition Block.

The start and end window row numbers are two bytes which indicate the vertical position of the first and last window rows on the screen expressed in row number.

<table>
<tr><th></th><th>D15</th><th>D14 … D8</th><th>D7 … D0</th></tr>
<tr><td>WD0</td><td>SCW</td><td>WINDOW ROW CONTROL BLOCK POINTER (PAGE)</td><td>▨</td></tr>
<tr><td>WD1</td><td colspan="3">WINDOW ROW CONTROL BLOCK POINTER (OFFSET)</td></tr>
<tr><td>WD2</td><td>0</td><td>WINDOW DEFINITION BLOCK POINTER (PAGE)</td><td>▨</td></tr>
<tr><td>WD3</td><td colspan="3">WINDOW DEFINITION BLOCK POINTER (OFFSET)</td></tr>
<tr><td>WD4</td><td colspan="2">START WINDOW ROW #</td><td>END WINDOW ROW #</td></tr>
<tr><td>WD5</td><td colspan="2">START WINDOW CHAR #</td><td>END WINDOW CHAR #</td></tr>
</table>

03901A-34

Figure 2-33 Window Definition Block (L/S = 0)

<table>
<tr><th></th><th>D15</th><th>D14 … D8</th><th>D7 … D0</th></tr>
<tr><td>WD0</td><td>SCW</td><td>▨</td><td>WINDOW ROW CONTROL BLOCK POINTER (HI)</td></tr>
<tr><td>WD1</td><td colspan="3">WINDOW ROW CONTROL BLOCK POINTER (LO)</td></tr>
<tr><td>WD2</td><td colspan="2">▨</td><td>WINDOW DEFINITION BLOCK POINTER (HI)</td></tr>
<tr><td>WD3</td><td colspan="3">WINDOW DEFINITION BLOCK POINTER (LO)</td></tr>
<tr><td>WD4</td><td colspan="2">START WINDOW ROW #</td><td>END WINDOW ROW #</td></tr>
<tr><td>WD5</td><td colspan="2">START WINDOW CHAR #</td><td>END WINDOW CHAR #</td></tr>
</table>

03901A-35

Figure 2-34 Window Definition Block (L/S = 1)

The most significant bit of WD2 must be "0" when L/S=0.

The start and end window character numbers are two bytes which indicate the horizontal position of the first and last window characters on the screen.

As mentioned above, the Window Control Block is identical to the Row Control Block (Figure 2.35 and 2.36). However, some restrictions should be observed when dealing with windows:

The number of visible characters of overwritten background segment is effectively interpreted by the row management unit whenever a window is present within the row. When no window is present, the CRTC needs only the sum of hidden and visible characters of the loading segment to know the length of the segment in memory.

The start and end positions of the window have to match segment boundaries in the background display. A window may span multiple segments (see Figure 2.37).

Only one window can exist between the row numbers specified by start window row # and end window row #.

When the contents of a window row's linked-list do not fill the window's row, the fill code is used to fill the remaining character positions of that window's row. During that time, the bus is not released and dummy DMA cycles are executed.

The Window Redefinition Block (Figure 2.38) is structured similar to the Row Redefinition Block. TSLC is left out, since a window row has to have the same number of scan lines as the background row it overlays.

2.6 ATTRIBUTES

This section focuses on the Character Attribute architecture and the various character display options handled by the CRTC. Since the user may have very specific display requirements that match his own design, the CRTC has been designed to provide great versatility in the attribute options.

In the character stream two pieces of information

<table>
<tr><th></th><th>D_{15}</th><th>D_{14} … D_8</th><th>D_7 … D_0</th><th></th></tr>
<tr><td>RA_0</td><td>LNK</td><td>WINDOW ROW CONTROL BLOCK POINTER (PAGE)</td><td>▨</td><td></td></tr>
<tr><td>RA_1</td><td colspan="3">WINDOW ROW CONTROL BLOCK POINTER (OFFSET)</td><td></td></tr>
<tr><td>RA_2</td><td colspan="2">HIDDEN #</td><td>VISIBLE #</td><td rowspan="5">1ST SEGMENT</td></tr>
<tr><td>RA_3</td><td>C</td><td>CHARACTER CODE POINTER (PAGE)</td><td>▨</td></tr>
<tr><td>RA_4</td><td colspan="3">CHARACTER CODE POINTER (OFFSET)</td></tr>
<tr><td>RA_5</td><td>0</td><td>ATTRIBUTE POINTER (PAGE)</td><td>▨</td></tr>
<tr><td>RA_6</td><td colspan="3">ATTRIBUTE POINTER (OFFSET)</td></tr>
<tr><td></td><td colspan="3"></td><td></td></tr>
<tr><td>$RA_{6\cdot(n-1)+1}$</td><td colspan="2">HIDDEN #</td><td>VISIBLE #</td><td rowspan="5">nTH SEGMENT</td></tr>
<tr><td>$RA_{6\cdot(n-1)+2}$</td><td>C</td><td>CHARACTER CODE POINTER (PAGE)</td><td>▨</td></tr>
<tr><td>$RA_{6\cdot(n-1)+3}$</td><td colspan="3">CHARACTER CODE POINTER (OFFSET)</td></tr>
<tr><td>$RA_{6\cdot(n-1)+4}$</td><td>0</td><td>ATTRIBUTE POINTER (PAGE)</td><td>▨</td></tr>
<tr><td>$RA_{6\cdot(n-1)+5}$</td><td colspan="3">ATTRIBUTE POINTER (OFFSET)</td></tr>
<tr><td>$RA_{6\cdot(n-1)+6}$</td><td>0</td><td>WINDOW REDEFINITION BLOCK POINTER (PAGE)</td><td>▨</td><td rowspan="2">IF LNK = 1</td></tr>
<tr><td>$RA_{6\cdot(n-1)+7}$</td><td colspan="3">WINDOW REDEFINITION BLOCK POINTER (OFFSET)</td></tr>
</table>

03901A-37

Figure 2-35 Window Row Control Block (L/S = 0)

<table>
<tr><th></th><th>D_{15}</th><th>D_{14} … D_8</th><th>D_7 … D_0</th><th></th></tr>
<tr><td>RA_0</td><td>LNK</td><td>▨</td><td>WINDOW ROW CONTROL BLOCK POINTER (HI)</td><td></td></tr>
<tr><td>RA_1</td><td colspan="3">WINDOW ROW CONTROL BLOCK POINTER (LO)</td><td></td></tr>
<tr><td>RA_2</td><td colspan="2">HIDDEN #</td><td>VISIBLE #</td><td rowspan="5">1ST SEGMENT</td></tr>
<tr><td>RA_3</td><td>C</td><td>▨</td><td>CHARACTER CODE POINTER (HI)</td></tr>
<tr><td>RA_4</td><td colspan="3">CHARACTER CODE POINTER (LO)</td></tr>
<tr><td>RA_5</td><td colspan="2">▨</td><td>ATTRIBUTE POINTER (HI)</td></tr>
<tr><td>RA_6</td><td colspan="3">ATTRIBUTE POINTER (LO)</td></tr>
<tr><td></td><td colspan="3"></td><td></td></tr>
<tr><td>$RA_{6\cdot(n-1)+1}$</td><td colspan="2">HIDDEN #</td><td>VISIBLE #</td><td rowspan="5">nTH SEGMENT</td></tr>
<tr><td>$RA_{6\cdot(n-1)+2}$</td><td>C</td><td>▨</td><td>CHARACTER CODE POINTER (HI)</td></tr>
<tr><td>$RA_{6\cdot(n-1)+3}$</td><td colspan="3">CHARACTER CODE POINTER (LO)</td></tr>
<tr><td>$RA_{6\cdot(n-1)+4}$</td><td colspan="2">▨</td><td>ATTRIBUTE POINTER (HI)</td></tr>
<tr><td>$RA_{6\cdot(n-1)+5}$</td><td colspan="3">ATTRIBUTE POINTER (LO)</td></tr>
<tr><td>$RA_{6\cdot(n-1)+6}$</td><td colspan="2">▨</td><td>WINDOW REDEFINITION BLOCK POINTER (HI)</td><td rowspan="2">IF LNK = 1</td></tr>
<tr><td>$RA_{6\cdot(n-1)+7}$</td><td colspan="3">WINDOW REDEFINITION BLOCK POINTER (LO)</td></tr>
</table>

03901A-38

Figure 2-36 Window Row Control Block (L/S = 1)

are present:

1. The actual character code.

2. An attribute invoking flag that may be part of the character code, or a specific code by itself. This option is programmed via that Attribute Flag Register internal to the CRTC. The function of this register is described in Section 2.3 (Register Description).

Once the choice of attribute-invoking flag(s) has been made it is possible to either display or inhibit the display of the flag by using the Invisible Attribute Flag (IAF) bit contained in Mode Register 1. If IAF=0, each code invoking an attribute is displayed, meaning that this specific code not only invokes an attribute, but is also output on CC_0–CC_7 to address the character font generator. This character is affected by the invoked attribute. If IAF=1, any code invoking an attribute is not loaded into the row-buffer and the invoked attribute then affects the following character. If two or more successive flags are present in the stream, only the last one (and the attribute it invokes) affects the first displayable character code encountered (see Figure 2.39). Figure 2.40 shows the Attribute Flag detect mechanism.

A character attribute is a code which affects the display characteristics of a character or set of characters on the screen.

The CRTC distinguishes four levels of attributes:

- Character attributes
- Field attributes
- Row attributes
- Frame attributes

2.6.1 Demand Attribute Fetch

The CRTC supports a flexible relationship between character code fetches and associated attribute fetches. Since attributes usually do not change on a character basis, the bus occupancy of the CRTC can be reduced (increasing system performance), by invoking attributes only at attribute transitions, i.e., demand attribute fetch.

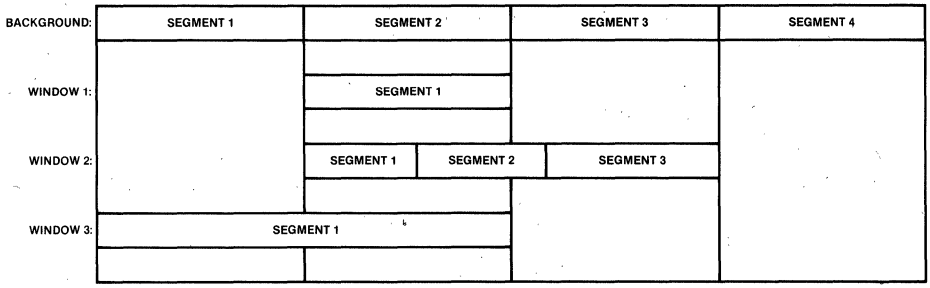
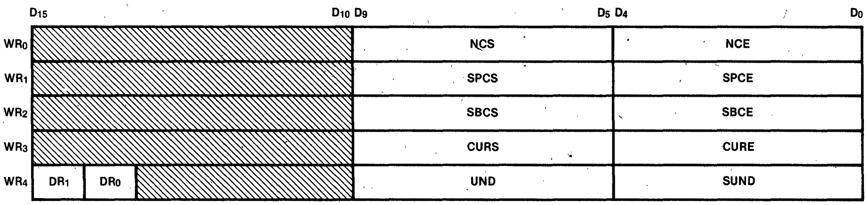


Figure 2-37 Window Overlay



	D15 – D10	D9 – D5	D4 – D0
WR_0	▨	NCS	NCE
WR_1	▨	SPCS	SPCE
WR_2	▨	SBCS	SBCE
WR_3	▨	CURS	CURE
WR_4	DR_1 DR_0 ▨	UND	SUND

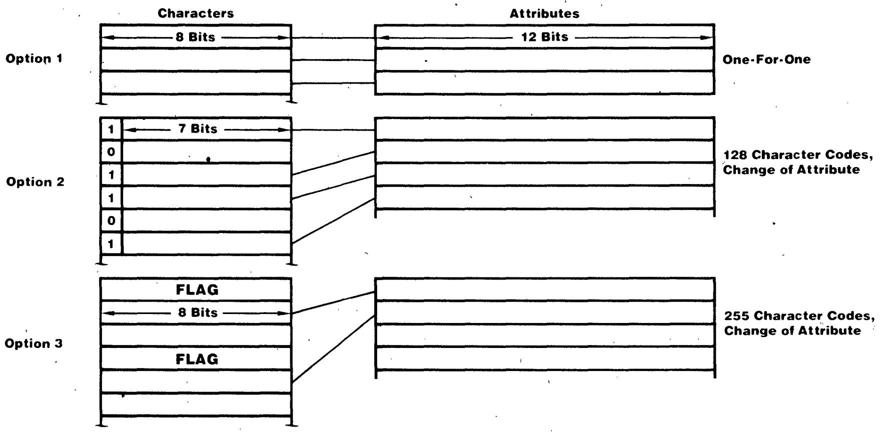
Figure 2-38 Window Redefintion Block

After power-up, at least one latched attribute must be specified to set (initialize) the default attribute word.

The CRTC supports various options; the three most common implementations are outlined below. All three options have similar implications on text editing. They differ, however, when analyzing bus utilization and attribute editing.

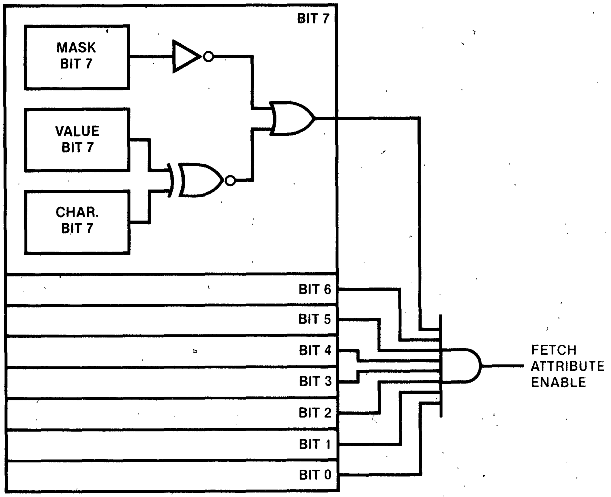
Option 1

Each character code invokes an attribute. This is the most straightforward implementation, and editing is very easy. However, it puts the highest burden on the bus (low performance system). For this mode IAF=0 and the Attribute Flag Register contains 0000_H.



03684B-6

Figure 2-39 Attribute Fetch Options



03901A-41

Figure 2-40 Attribute Flag Defect Mechanism

Option 2

A single bit within the character code specifies whether an attribute should be invoked. Adding or deleting attributes involves two actions:

Set or reset bit in character code

Update the attribute list (block move)

This options reduces the required bus bandwidth by about 50% (permanent savings), with the cost of a single data block move, to update the attribute list. Segmentation can reduce the editing overhead. However, it increases the required bus bandwidth (larger RCBs). The editing impact to the character list is relatively low, but the character set is reduced to 128 characters.

Option 3

This option implements a demand attribute scheme with a character set of 255 characters and a single attribute flag character. Adding and deleting attributes involves two actions.

Insertion or deletion of the flag character (block move)

Update of the attribute list (block move)

This option, similar to the previous option, reduces the required bus bandwidth by approximately 50%, but demands more CPU effort when editing the attribute list.

Character Attributes

Character attributes are word quantities which affect various CRTC output signals and other operations on a character-by-character basis. These words reside in memory and are accessed via the attribute-segment pointers associated with the character-segment pointers in the Row Control Blocks. The character attributes are stored in parallel with the corresponding character code in each row buffer. The bits in the attribute word are discussed below:

The Attribute Port Enable and Attribute Redefinition Register affect the attribute processing. Refer to Section 2.3 (Register Description).

Blink

When this bit is set in the attribute word, the AP_0 pin outputs a periodic signal whose rate and duty cycle are specified in the Main Definition Block. When this bit is reset, AP_0 outputs a Low level. Blink may be programmed to be a user-definable attribute. In this case, no internal blink attribute processing is done.

Underline

When this bit is set in the attribute word, the AP_1 pin outputs a High for one scan line in the character cell. The scan line on which the underline is active is specified in the Row Redefinition Block and can, therefore, be changed on a row-by-row basis. If this attribute is made user-definable (see Attribute Redefinition Register), the pin is active for all scan lines of the character cell. Underline is active for two scan lines when displaying double-height rows.

Shifted Underline

This bit acts like Underline except that the signal is output on AP_2 and the scan line number is specified by an independent 5-bit word also contained in the Row Redefinition Block. Shifted Underline also may be Overbar or Strike Through.

Subscript

When this bit is set, the affected character is displayed on a set of scan lines specified by subscript character start line number and subscript

Attribute Word Organization

Bit 15:	Latched/Unlatched	Bit 7:	User-Definable
Bit 14:	Cursor	Bit 6:	Highlight
Bit 13:	Ignore Character	Bit 5:	Reverse
Bit 12:	Reserved	Bit 4:	Superscript
Bit 11:	Reserved	Bit 3:	Subscript
Bit 10:	User-Definable	Bit 2:	Shifted Underline/Strike Through
Bit 9:	User-Definable	Bit 1:	Underline
Bit 8:	User-Definable	Bit 0:	Blink

character end line number in the Row Redefinition Block. This bit is generally used to display subscript characters. In addition to this internal process, a High level is output on AP_3 indicating a subscript character. This feature may be used to switch to a different character font generator. The subscript attribute pin is active for all scan lines between start line number and end line number. If it is programmed to be a user-definable attribute, the pin is active for all scan lines of the character cell.

Superscript

Similar to subscript. The set of scan lines is specified by superscript character start line # and superscript character end line # in the Row Redefinition Block. The attribute is output on AP_4. It can also be programmed to be a user-definable attribute.

Reverse

When this bit is set, a High level is output on AP_5. This bit may be used to reverse the invoking character on the screen. No internal attribute processing is done, so this attribute can be treated as a user-definable one. Reverse is exclusive ORed with the reverse cursor.

Highlight

When this bit is set, a High level is output on AP_6. This bit may be used to highlight the invoking character on the screen. No internal attribute processing is done, so it can be treated as user definable if desired.

User-Definable

These four bits have their state output on the matching pins (AP_7–AP_{10}) and can be used as desired to affect the invoking characters.

Ignore Character

When the Ignore Bit is set to "1," and the Display Hidden (DH) bit in Mode Register 1 is reset ("0"), neither the affected character nor its attribute code are loaded into the row buffer and thus are not displayed. When DH is set, the ignore characters (those having invoked the ignore attribute) are loaded along with their attribute code. The ignore bit is not put out on the attribute port.

Cursor

If this bit is set, an attribute cursor is displayed at the affected character position, dependent upon the mode of the cursor display logic. See section on cursor display.

Latched/Unlatched

When this bit of the attribute word is set ("latched") the attribute information applies to all characters following the character that invoked the attribute word. This is described in more detail in the section on field attributes. This bit is not put out on the attribute port.

Character Attribute Timing

The attribute information present on the attribute port is output coincident to, or one character clock after the invoking character, depending upon the skew-bits in Mode Register 1. This compensates skew between character codes and attributes, if external character code pipelining is required.

Attribute Port Enable Register

The function of this register is described in Section 2.3, Register Description. The superscript and subscript effect are not cancelled by resetting the corresponding bits in this register; in fact, this only drives the corresponding attribute port pins Low. The internal attribute processing still takes place. To disable subscript and superscript action, the Attribute Redefinition Register must be used.

The subscript and superscript, when enabled, may be used to choose between a standard character generator and a specific character generator for subscript and/or superscript. However, in most applications, one standard font generator can be used for all three.

Attribute Redefinition Register

Four user-definable attributes are provided for optional external attribute processing. If this number is not sufficient, then the highlight and reverse attributes may be used as user-definable without any modification.

If this is still not enough, the user can disable the normal effect of other attributes and turn them into user-definable attributes. These attributes are:

superscript
subscript
shifted underline
underline
blink

This yields 11 user-definable attributes. The function of the Attribute Redefinition Register is described in the Register Description Section.

If a user-definable attribute is directely mixed with the serial video signal put out by the Am8152A, the attribute must be delayed by one character clock plus one dot clock. This compensates for the internal delay in the Am8152A.

2.6.2 Field Attributes

A field attribute affects a set of successive characters. This feature reduces memory consumption and software complexity compared to character attributes when dealing with character strings. Field attributes are similar to character attributes and are implemented by setting the latched attribute bit.

When a character does not invoke an attribute, it implicitly invokes the default attribute. Therefore, every character appearing on the screen is associated with an attribute, in one of the following manners:

- The character invokes either a latched or unlatched attribute. This attribute affects that specific character (if it is a displayable character).

- The character does not invoke an attribute. The default attribute affects this character.

Additionally, invoking a latched attribute also reloads the default attribute, As specified earlier, when an Ignore attribute is invoked and Display Hidden is reset, the attribute word and the character are not loaded in the Row buffers. However, if the invoked attribute is a latched attribute, then the Ignore attribute is latched and succeeding characters are not loaded. On the other hand, if they invoke an attribute with Ignore reset, the ignore function is cancelled for all succeeding characters as soon as a latched attribute with ignore bit reset is invoked.

A latched attribute affects all subsequent characters not involving attributes, whether they are in windows or background, until a new latched attribute is encountered. As a result, a latched attribute wraps around the screen, ripples through rows, background-window and window-background, etc.

2.6.3 Row Attributes

The Row attributes are 10 bits that are output on AP_0–AP_4 and AP_6–AP_{10}, at horizontal retrace time. This is a CRTC feature that enables the user to modify display characteristics on a row-by-row basis.

The Row attributes are specified in the Row Redefinition Block and may be latched by external logic at HSYNC fall-time. Some examples in the applications of Row attributes will follow. The shape of the modified area(s) is always a horizontal screen slice(s):

reverse row(s)
highlight row(s)
blink row(s)
color palette addressing
row(s) underline
change character set
switch to semi-graphic generator
switch video output to a graphic display unit to mix graphic and text
blank row(s) (secret prompts)

The row attributes are internally latched and do not need to be rewritten on each row. Therefore, the internal Row Attribute Register is updated each time a Row Redefinition Block is invoked (see Figure 2-48 Row Attribute Timing).

The row attribute word is output seven clocks after BLANK goes High and is removed one clock before BLANK goes Low. However, a programmed skew between BLANK and the attribute output still applies. The horizontal timing parameters must be chosen in such a way that the edge of HSYNC falls in the interval where the attribute port provides valid data.

2.6.4 Frame Attributes

Frame attributes affect the character display characteristics of the entire screen. These attributes are stored in the Main Definition Block and define:

x-y cursor positioning
fill character code
x-y cursor blink rate and duty cycle
smooth-scroll of window or background
smooth-scroll rate and direction

2.6.5 Cursor Displays

Cursors are used to locate specific points in the text that need particular attention. Two types of cursors are supported by the CRTC:

single absolute cursor (x-y cursor)
multiple attribute cursors

The Absolute Cursor

This cursor is positioned on the screen according to its "X" (horizontal) and "Y" (vertical) coordinates specified in the Main Definition Block, and fetched by the CRTC during the vertical retrace time.

"X" is expressed in character units. "X=0" indicates the first character column. "Y" is expressed in row units. "Y=0" indicates the first row on the screen. This cursor is called absolute because it refers to the screen boundaries and is not dependent upon the text displayed on the screen. When the text is scrolled, the cursor position stays stationary relative to the screen. However, while the screen is smooth scrolling, this cursor stays with row "Y," until the topmost row is relinked. At that time, the absolute cursor jumps to the new row "Y." This behavior can cause the absolute cursor to move temporarily across background/window boundaries. Therefore, while smooth scrolling mixed screens, absolute cursor display should be disabled.

When the CRT monitor beam matches the cursor position, a CRTC internal cursor signal is activated to indicate the match. This signal may be steered internally to one of three output pins: cursor pin, reverse pin, and underline pin.

The choice of the output pin is made through the cursor mask contained in Mode Register 2. In the same register, a cursor enable bit, when reset, controls disabling the Absolute Cursor. Furthermore, it is possible to partially affect the character position on the screen by specifying the scan line boundaries in which the output signal will be active. These boundaries are specified in the Row Redefinition Blocks by CURS and CURE.

The Attribute Cursor

This cursor is positioned with the visible character that invoked an attribute with Cursor Bit=1. A display can therefore contain as many attribute cursors as there are character positions.

An attribute cursor is implicitly linked to the text in which it is contained. If the text scrolls up, the attribute cursor scrolls with the text, whereas the absolute cursor would remain steady.

When an attribute cursor is encountered, the same operation as with the absolute cursor occurs. However, a different set of bits in the cursor Mask Register steers the attribute cursor signal to one of the three outputs. This allows the user to distinguish the attribute cursor from the absolute cursor on the screen. The same scan line boundaries are used for both cursors.

Cursor Characteristics

One out of four shapes may be chosen for each of the two cursors described earlier:

Cursor Whole. The cursor signal is output on the cursor pin for each scan line of the character position.

Cursor Part. The cursor signal is output on the cursor pin for the specific scan lines contained between cursor start and cursor end boundaries specified in the Row Redefinition Block.

Reverse. Same operation as cursor part except that the signal is output on the reverse attribute pin after being exclusive ORed with the internal reverse attribute signal.

Underline. Same operation as cursor part except that the signal is output on the underline attribute pin.

2.6.6 Fill-Code Attributes

When the Row Management Unit reaches the end of the last segment of a row, and the row-buffer is not full (96 characters or 132 depending upon "slim" setting), the Row Management Unit fills the remaining space in the row buffer with a specific code specified by the user in the Main Definition Block. This code is the fill code, and needs special attention when it appears in text. Each time the row buffer is not filled by the contents of the linked-list, the fill code is loaded into the row buffer.

If the fill code is an attribute invoking code, the Row Management unit may not invoke an attribute, depending on the "FAT" bit in the Main Definition Block. If the user needs to display the fill code associated with an attribute, he should then set the "FAT" flag (Fill Code Attribute in the Main Definition Block) to one and add the desired attribute in the attribute list of the last segment invoked. Only one attribute word is fetched for the fill characters, so this attribute must be a latched attribute to affect all fill characters loaded into the row buffer.

The ignore attribute is discarded when associated with the fill code.

2.7 INTERRUPT OPERATIONS

An interrupt may occur whenever the CPU needs to be notified of various events internal to the CRTC or that an operation has just been completed. There are two sources of CRTC interrupts:

Vertical Interrupt

The vertical interrupt, if enabled, can be used as a real-time interrupt by the CPU or it can be used as an indication that certain CRT updates should take place. The vertical interrupt is issued when the "n-th" character row has been loaded by the CRTC into its internal row buffers. The value of "n" is determined by the 8-bit VERTINT field in the HSYNC Register. When "n" is set to "1," the CRTC issues a vertical interrupt after the last segment of the first row is completely loaded. (See also section on register programming.)

Smooth-scroll Interrupt

The smooth-scroll interrupt is used to inform the CPU when to update the display linked-lists during smooth-scrolling. See Section 2.8, smooth-scroll mechanism, for more details.

Interrupt Protocol

A complete interrupt cycle consists of an interrupt request by the CRTC followed by an Interrupt Acknowledge of the CPU (Figure 2.41). The request, which consists of $\overline{\text{INT}}$ being pulled Low by the CRTC, notifies the CPU that an interrupt is pending. The Interrupt Acknowledge cycle notifies the peripheral that its interrupt has been recognized. In return, the peripheral may provide an interrupt vector to the CPU to identify itself (see the section on Row Management Unit).

The CRTC has two sources of interrupt and each interrupt source has three bits that control the issuance of an interrupt. These bits are the Interrupt Pending bit (IP), the Interrupt Enable bit (IE), and the Interrupt Under Service bit (IUS). In addition to the control bits, two further bits control the interrupt behavior of the CRTC. These are the Disable Lower Chain bit (DLC) and the No Vector bit (NV) in Mode Register 2.

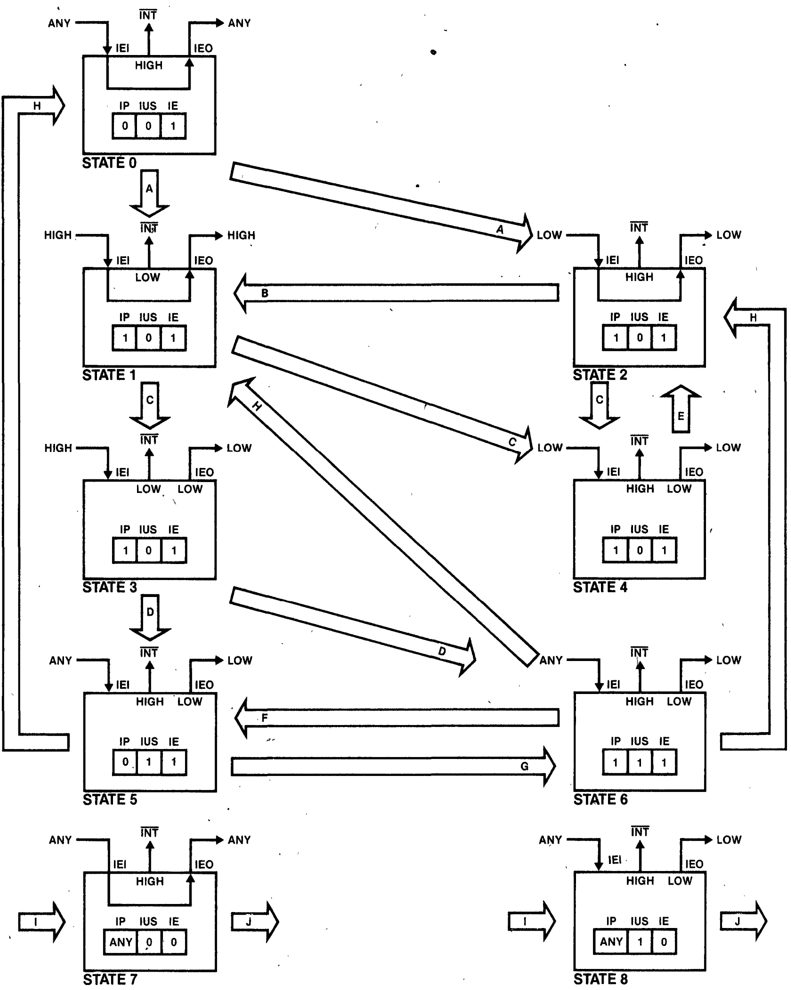
Peripherals are connected together via an interrupt daisy-chain formed with their IEI (Interrupt Enable In) and IEO (Interrupt Enable Out) pins. The daisy-chain resolves the interrupt priority.

For the purpose of this description, the CRTC may be considered as having two interrupt sources: Smooth-scroll, and Vertical Interrupt. The Smooth-scroll Interrupt has higher priority.

Figure 2.41 is a state diagram of interrupt processing for an interrupt source (assuming its IE bit is "1"). An interrupt source with an interrupt pending (IP=1) makes an interrupt request (by pulling $\overline{\text{INT}}$ Low) only if it does not have an interrupt under service (IUS=Low), no higher priority interrupt is being serviced (IEI=High), and no Interrupt Acknowledge transaction is in progress. IEO is not pulled down by the interrupt source at this time. IEO continues to follow IEI until an Interrupt Acknowledge occurs. Some time after $\overline{\text{INT}}$ has been pulled Low, the CPU initiates an Interrupt Acknowledge bus cycle. Between the falling edge of $\overline{\text{INTACK}}$ and the falling edge of $\overline{\text{DS}}$, the IEI/IEO daisy-chain settles. AS is optional. Any interrupt source with an interrupt pending (IP=1) holds its IEO line Low during Interrupt Acknowledge. All other interrupt sources make IEO follow IEI (transparent). When $\overline{\text{DS}}$ falls, only the highest priority interrupt source with a pending interrupt (IP=1) has its IEI input High and its IUS bit set at "0." This is the interrupt source being acknowledged, and at this point it sets its IUS bit to "1." If the peripheral's NV bit is "0," the interrupt source identifies itself by placing the interrupt vector on AD_0–AD_7. Each time $\overline{\text{DS}}$ is activated during Interrupt Acknowledge cycles, the vector is put out. The upper byte is driven Low. If the NV bit is "1," the peripheral's AD_0–AD_{15} pins remain floating, thus allowing external circuitry to supply the vector.

While an interrupt source has an Interrupt Under Service (IUS=1), it prevents all lower priority devices from requesting interrupts by forcing IEO Low. When interrupt servicing is complete, the CPU must reset the IUS and the IP bits.

A peripheral's Interrupt Enable bit (IE) modifies the peripheral's behavior in the following manner—if the IE bit is "0," the effect is as if all interrupts from the peripheral are disabled. However, the peripheral can still set its IP bit if an interrupt is required. If the IE bit is cleared while the source is driving $\overline{\text{INT}}$ Low, $\overline{\text{INT}}$ returns High until IE is set. To prevent race conditions, the CPU should mask out interrupts from the peripheral before clearing IE. Note that IE, when cleared, also prevents the CRTC from responding to an Interrupt Acknow-



Transition Legend

A The peripheral detects an interrupt condition and sets Interrupt Pending.

B All higher priority peripherals finish interrupt service, thus allowing IEI to go High.

C An interrupt-acknowledge transaction starts, and the IEI/IEO daisy chain settles.

D The interrupt-acknowledge transaction terminates with the peripheral selected. Interrupt Under Service (IUS) is set to 1, and Interrupt Pending (IP) may or may not be reset.

E The interrupt-acknowledge transaction terminates with a higher priority device having been selected.

F The Interrupt Pending bit in the peripheral is reset by an I/O operation.

G A new interrupt condition is detected by the peripheral, causing IP to be set again.

H Interrupt service is terminated for the peripheral by resetting IUS.

I[3] IE is reset to zero, causing interrupts to be disabled.

J[4] IE is set to one, re-enabling interrupts.

State Legend

0 No interrupts are pending or under service for this peripheral.

1 An interrupt is pending, and an interrupt request has been made by pulling $\overline{\text{INT}}$ Low.

2 An interrupt is pending, but no interrupt request has been made because a higher priority peripheral has an interrupt under service, and this has forced IEI Low.

3 An interrupt-acknowledge sequence is in progress, and no higher priority peripheral has a pending interrupt.

4 An interrupt-acknowledge sequence is in progress, but a higher priority peripheral has a pending interrupt, forcing IEI Low.

5 The peripheral has an interrupt under service. Service may be temporarily suspended (indicated by IEI going Low) if a higher priority device generates an interrupt.

6 This is the same as State 5 except that an interrupt is also pending in the peripheral.

7 Interrupts are disabled from this source because IE = 0.

8 Interrupts are disabled from this source and lower priority sources because IE = 0 and IUS = 1.

1. This diagram assumes MIE = 1. The effect of MIE = 0 is the same as that of setting IE = 0.
2. The DLC bit does not affect the states of individual interrupt sources. Its only effect is on the IEO output of a whole peripheral.
3. Transition I to state 6 or 7 can occur from any state except 3 or 4 (which only occur during interrupt acknowledge).
4. Transition J from state 6 or 7 can be to any state except 3 or 4, depending on the value of IEI, IP, and IUS.

03901A-40

Figure 2-41 State Diagram for an Interrupt Source

ledge. While IE is cleared, IEO follows IEI. The peripheral's IEO line can be forced unconditionally into the Low state by setting the DLC bit to "1."

2.8 SMOOTH-SCROLL MECHANISMS

The Am8052 provides very powerful smooth-scroll capability with minimum interaction by the CPU. Window(s) or background can be smooth-scrolled either up or down at a rate that is programmable via the scroll parameters field in the Main Definition Block. Since the CRTC is designed to work with a linked-list structure, some precautions should be taken when relinking the text after each scrolled row.

General Smooth-Scrolling Rules

Either windows or background can be scrolled at one time; they cannot be scrolled at the same time.

When a window splitting the screen vertically (sharing the row buffer with background characters) is intended to be smooth-scrolled, then all of its rows must have the same total scan line counts (TSLC).

Double Buffering Technique

Smooth-scrolling operation is achieved by moving the appropriate data up or down on a scan line basis. Therefore, the CRTC adds an offset to the internal row's scan line count and outputs the result on R_0–R_4. This results in a displacement of the data on the screen by the number of scan lines equal to the offset. As soon as the last scan line (top or bottom depending on the scroll direction) of the first row of text has reached the top extremity of the screen, a text relink has to be made. This relink serves to push the disappearing row off the screen or to link a new row onto the top of the screen.

In order to maintain a smooth relink transaction and allow for CPU time constraints, the Am8052 controls the relink timing through interrupts and double buffering of pointer register. As soon as the CRTC has begun smooth-scrolling a character row, it generates an interrupt. The CPU which maintains the linked-lists responds by writing to "Top of Page (Window) Soft" a pointer value that provides the correct linked-list for the display after it has completed the scroll of the current row. The CRTC uses this new value as the active "Top of Page (Window)" only after the row scroll in progress is completed. This double buffering of the "Top of Page (Window)" values allows maximum time (one character row scroll time) for the CPU to relink and respond to the interrupt.

According to the preceding, when the user wants to smooth-scroll a portion of the display (background or window), he should define two Main/Window Definition Blocks, and flip between those two blocks each time a smooth-scroll interrupt occurs. This technique allows the user to execute the link modifications on the unused definition block while the other is being processed by the CRTC.

Detailed Interlock Mechanism:

The Top of Page/Window Soft is the key interface between the CPU and the CRTC when dealing with smooth-scrolling.

When the CPU writes a pointer value into this register, it does not modify the actual Top of Page/Window Register (Hard Register) used by the CRTC to fetch the Main/Window Definition Block. In fact, the transfer between this temporary register to the actual register takes place according to the smooth-scroll algorithm internal to the CRTC. Therefore, if the smooth-scroll process has not been enabled, writing to Top of Page/Window Soft does not change anything in the link architecture and this register should be used only if smooth-scroll operation is (or will be) performed. If the user wants to change the link in a non-smooth-scroll condition he should use the "Top of Page/Window Hard" Register.

The smooth-scroll mechanism is enabled by setting the Smooth- Scroll Enable bit (SSE) in the Main Definition Block. Two other bits in the Main Definition Block are used to select Window/Background scrolling and Up/Down scrolling directions. Additionally, when scrolling windows, the Smooth-Scroll Window bit (SCW) in the corresponding Window Definition Blocks must be set. All windows which have SCW set are scrolled simultaneously. Windows which have SCW reset remain steady.

Smooth scrolling is stopped by resetting the enable bit (SSE-Bit) in the Main Definition Block.

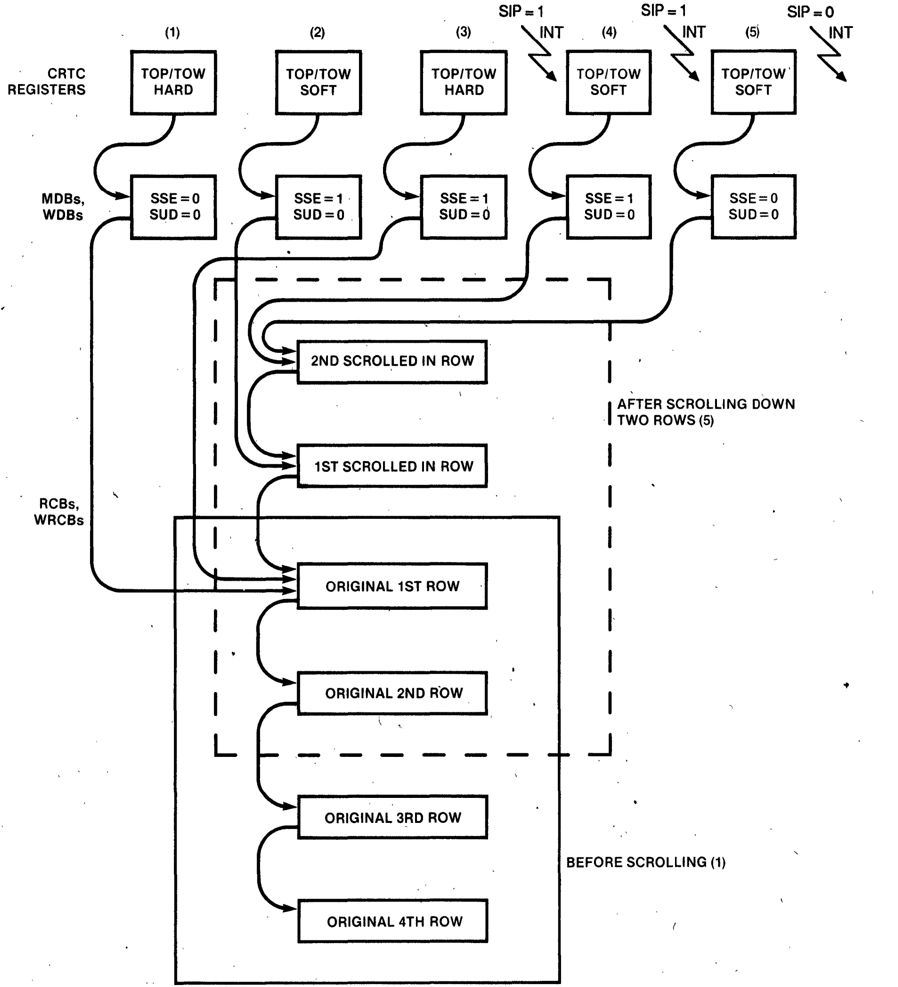
When the background is scrolled only Top Of Page Soft needs to be updated; loading Top of Window Soft has no effect. Similarly, when scrolling windows only Top Of Window Soft is relevant.

Scroll Down

The Top of Page/Window Hard Register links to the Main/Window Definition Block of the currently displayed text. When a down scroll is initiated, the current text is moved down a fraction of a row. The empty space at the top of the screen is filled with a fraction of the scrolled-in row. Therefore, the CRTC has to know the pointer to the new Main/Window Definition Block before it can start scrolling. The pointer is loaded into the Top of Page/Window Soft Register.

The programming sequence shown in Figure 2.42 refers to both scrolling background or windows. The example shows two rows scrolling in a background or window consisting of a total of four rows. When scrolling the background the TOP Soft Register is reloaded and two Main Definition Blocks are used to implement the "Double Buffer" technique. If a window is scrolled, the TOW Soft Register and two Window Definition Blocks are involved. The numbers in the programming sequence below correspond to Figure 2.42.

1. The CRT system displays a steady screen. The TOP/TOW Hard Register links to a MDB/WDB with smooth-scroll disabled. The smooth-scroll process is initiated from this steady state.



03901A-42

Figure 2-42 Scroll Down Sequence

2. The CPU prepares another MDB/WDB with smooth-scroll enabled. This MDB/WDB contains a pointer to the RDB/WRCB for the scrolled-in row which in turn points onto the first row currently displayed on the screen. The CPU loads the pointer to this MDB/WDB into TOP/TOW Soft Register.

3. The CPU then enables smooth-scrolling by setting the smooth- scroll bit in the MDB/WDB described in Step 1. The CRTC detects this change when it fetches this block during the next vertical retrace period. The first frame after this change still reflects the same unscrolled display. Scrolling begins with the following frame. If the TOP/TOW Soft Register was not initialized, the start of scrolling waits for the initialization. At this time the CRTC transfers the contents of the TOP/TOW Soft Register to the TOP/TOW Hard Register to allow scrolling to the new row. It issues an interrupt on smooth-scroll event to notify the CPU that the TOP/TOW Soft Register can be updated. The update can take place at any time until the new row is entirely scrolled-in. If the update was not performed at that time, the displayed text scrolls up (hard-scroll) one row and this same row is smooth-scrolled in again.

4. The TOP/TOW Soft Register is relinked to the MDB/WDB pointing to the RDB/*WRCB* of the next row to be scrolled-in. If only one row should be scrolled, Step 4 is left out. For scrolling "n" rows, Step 4 is repeated after each interrupt issued by the CRTC "n–1" times.

5. To stop the smooth-scroll process, the new pointer in the TOP/TOW Soft Register points to a copy of the previous MDB/WDB in which the SSE-bit is cleared. Scrolling of both background and windows is stopped by resetting SSE. The CRTC notifies the host CPU that smooth scrolling is completed by issuing a last smooth scroll interrupt with SIP (Smooth Scroll in Progress) being reset.

Scroll Up

The numbers in the programming sequence below correspond to Figure 2.43.

1. The TOP/TOW Hard Register links to the MDB/WDB of the currently displayed text. Smooth-scroll is disabled.

2. The scroll process is initiated by enabling smooth-scrolling in the MDB/WDB. The TOP/TOW Soft Register does not need to be loaded at that time. The last row displayed links to the row to be scrolled-in. The CRTC detects the change of the scroll enable bit when it fetches the block during the next vertical retrace period. After it has started smooth-scrolling it issues an interrupt on smooth-scroll event to make the CPU update the TOP/TOW Soft Register.

3. The TOP/TOW Soft Register links to the MDB/WDB pointing to the RCB/WRCB of the row following the scrolled-out row. If only one row should be scrolled, Step 3 is left out. For scrolling "n" rows, Step 3 is repeated "n–1" times.

4. To stop the smooth-scroll process, the TOP/TOW Soft Register points to a MDB/WDB with scroll disabled (SSE=0).

Smooth Scroll in Progress Bit (SIP-Bit)

The SIP-bit is a status bit in the Mode Register 2 indicating to the CPU that the CRTC is actually scrolling either window or background while the SSE bit (Smooth-Scroll Enable) is set. The SIP bit is set as soon as the CRTC has loaded the Main Definition Block with SSE=1. Nevertheless, once the CPU resets SSE to "0," the CRTC waits until the entire smooth-scroll is finished before resetting SIP to "0." Furthermore, when using vectored interrupt, the SIP bit appears in Bit 1 of the interrupt vector and, therefore, allows the user the ability to vector to two different programs depending on the status of smooth-scroll without polling the SIP bit.

The CRTC scans the SSE-bit in the Main Definition Block only at the top of the frame (not scrolling) and after transferring TOP/TOW soft register to TOP/TOW hard register (previous frame was smooth scrolled). After scanning the MDB, and a relink took place, and the previous frame was scrolled, then the CRTC sets the interrupt pending bit for smooth scroll. At that time the SIP-bit reflects exactly the state of the SSE-bit in the scanned MDB.

If at that time SSE=1 the CRTC issues an interrupt with SIP=1 asking the host CPU to load a new pointer into the soft register; a pointer required for the subsequent relink. In this case scrolling continues.

If at that time SSE=0 the CRTC issues an interrupt with SIP=0 notifying the host CPU that scrolling has been terminated.

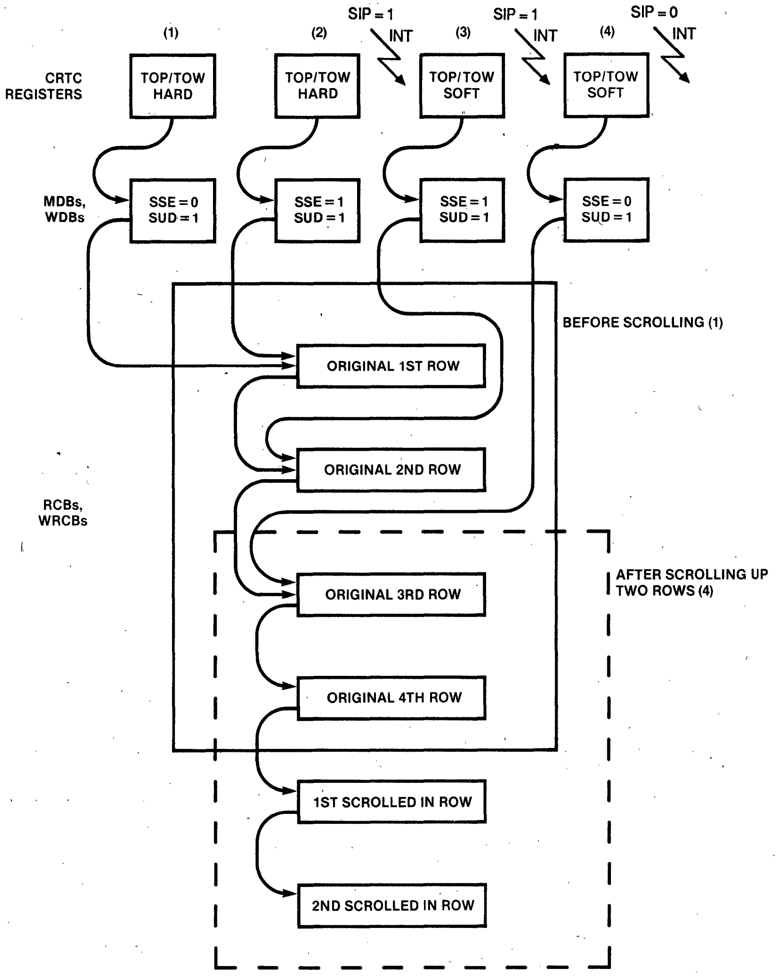
Smooth-Scroll Parameters

IUSS. Interrupt Under Service for Smooth-Scroll operation (Bit 2 in Mode Register 2) is set either by a hardware interrupt acknowledge ($\overline{\text{INTACK}}$ Low) or by a software interrupt acknowledge (host CPU sets IUSS).

IES. Interrupt Enable Smooth-Scroll Bit 1 in Mode Register 2 enables smooth scroll interrupts. Alternatively, the host CPU can poll the interrupt pending bit to perform the smooth scroll relinks. This bit can only be set and reset by the host CPU.

IPS. Interrupt Pending for Smooth-Scroll event. Bit 0 in Mode Register 2. This bit indicates that the smooth scroll logic requires service by the host CPU. This bit is set by the CRTC or the CPU, and reset only by the CPU. It it independent of the state of IES.

SIP. Scroll in Progress, Bit 8 in Mode Register 2. Set and reset by the CRTC.



03901A-43

Figure 2-43 Scroll Up Sequence

2.9 SYNCHRONIZATION

The CRTC has two built-in synchronization mechanisms: External SYNC (ESYNC) and Reset for Test ($\overline{\text{RSTT}}$). These mechanisms are activated by applying signals to the synchronization input pins (ESYNC and $\overline{\text{RSTT}}$). The ESYNC input synchronizes the CRTC to an external frame frequency. In most applications this input locks the vertical timing to the power-line frequency to avoid screen swimming. RSTT synchronizes multiple CRT controllers.

Multiple CRT Controller Synchronization

The Reset for Test ($\overline{\text{RSTT}}$) input synchronizes two or more CRTCs. This synchronization sequence is executed only upon system initialization. Figure 2.44 shows the timing diagram. $\overline{\text{RSTT}}$ can synchronize multiple CRTCs only once after power-on, because applying $\overline{\text{RSTT}}$ would corrupt the display. It cannot be used to synchronize multiple CRTCs on a frame basis. This means, that all CRTCs have to programmed in a way that they operate synchronously forever (e.g. same clock and same timing parameters). The sequence of operation for $\overline{\text{RSTT}}$ is:

Reset all CRTC's by pulling Reset ($\overline{\text{RST}}$) Low for at least five clock cycles (CLK$_1$ or CLK$_2$, whichever is slower).

After $\overline{\text{RST}}$ becomes inactive, initialize all CRTC registers including Mode Register 1 and 2 with DE=0.

Activate $\overline{\text{RSTT}}$ synchronous to CLK$_1$ or CLK$_2$ depending on the CLK1/2 bit in Mode Register 1. It must be synchronous to the clock determining the frame timing. It must meet the set-up time t_S to avoid metastable problems.

Reload Mode Register 1 and 2. Set DE=1 (Mode Register 1).

Deactivate $\overline{\text{RSTT}}$ synchronous to CLK$_1$ or CLK$_2$. $\overline{\text{RSTT}}$ must be active for a minimum of five clock cycles and its rising edge must meet the hold time requirement. The rising edge of $\overline{\text{RSTT}}$ triggers all CRTC's to start display synchronously. Detailed Reset for Test Timing is shown in Figure 2.44.

External Sync Operation

The ESYNC input allows synchronization of the CRT display vertical frame rate to the power line frequency to eliminate waviness and other effects. The ES bit in Mode Register 1 defines whether ESYNC controls the Vertical Sync rate.

ESYNC is recognized by the CRTC for every field or frame. It causes the VSYNC signal to become active at the occurrence of HSYNC. In non-interlaced mode, VSYNC becomes active at the first rising edge of HSYNC following ESYNC's rising edge (Figure 2.46). In interlaced mode, VSYNC comes active at the next HSYNC active when in the even frame, or in the middle between two HSYNC's in the odd frame (Figure 2.47).

The VSYNC and HSYNC are inactive (BLANK is active) before, during, and after reset. When the display is enabled via mode bit DE, HSYNC output becomes active, while VSYNC waits for ESYNC active. The display is delayed up to one ESYNC period.

ESYNC cannot be used to synchronize multiple CRTCs, since it synchronizes only VSYNC, but not

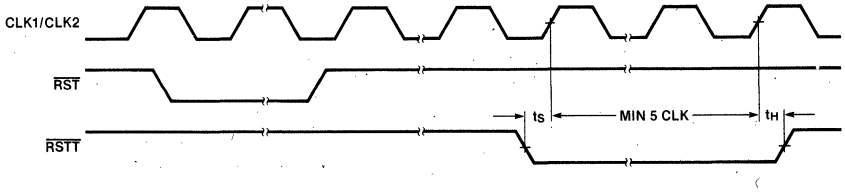


Figure 2-44 Reset for Test Timing

HSYNC. Only $\overline{\text{RSTT}}$ can synchronize multiple CRTCs.

2.10 RFI and INTERLACED VIDEO

There are two types of interlace, Repeat Field Interlace (RFI) and Interlaced Video (IV). Both types use the same vertical and horizontal timing as described in the Vertical and Horizontal Timing Section. Both schemes offset the vertical position of the scan lines of the odd numbered fields so that they are physically interleaved with the scan lines of the even fields. For RFI, the same video information is displayed on both odd and even fields. The slight offset of the odd field eliminates the horizontal stripes that sometimes occur between scan lines on non-interlaced displays. (See Figure 2.48)

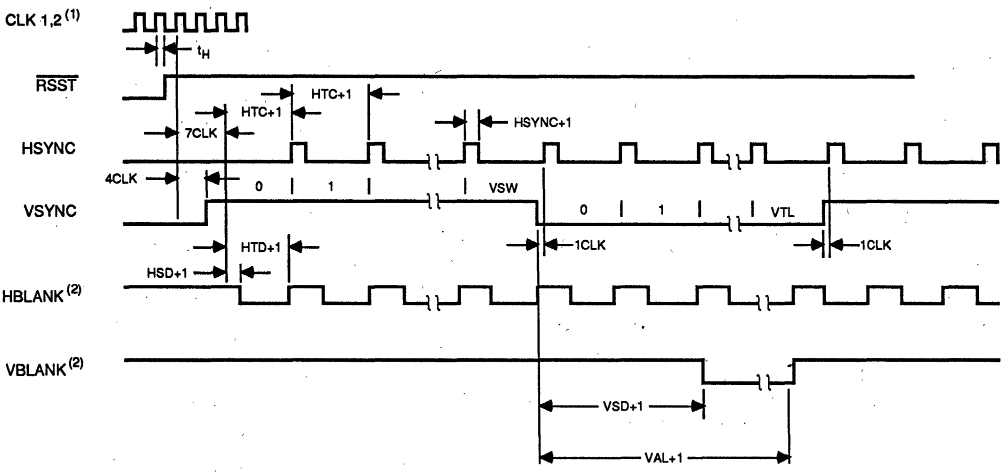
Interlaced Video is used to increase the amount of information displayed on a monitor .without increasing the horizontal or vertical scan rates. IV takes advantage of the odd field scan line offset by displaying half the video in the even field (alternating lines) and half in the odd field. The effect is to essentially double the vertical character density with respect to RFI or non-interlaced video. One problem with IV is the potential imbalance of CRT beam current between the odd and even fields and the resulting loss of perfect video interleave. This imbalance is greatest if the character rows consist of an even number of scan lines (adding up the scan lines in the even field and the odd field).

Restrictions for Interlace Video

The restrictions mentioned below apply only to Interlace Video. They do not apply to RFI or non-interlace video.

If smooth scrolling is disabled, any mixture of background and windows can be displayed, as long as windows are horizontally separated by three or more character rows (not scan lines). Windows should not overlap horizontally.

The Am8052 does not support split-screen smooth-scrolling in Video Interlace mode. Also, in Video Interlace mode, a screen containing only background and no windows can only be smooth-scrolled if all rows have an even scan count (TSLC even) and the number of scan lines scrolled per frame is also even (scroll rates: 2, 4, 6, 8 scan lines/frame. No scrolling restriction applies to non-interlace or RFI video.

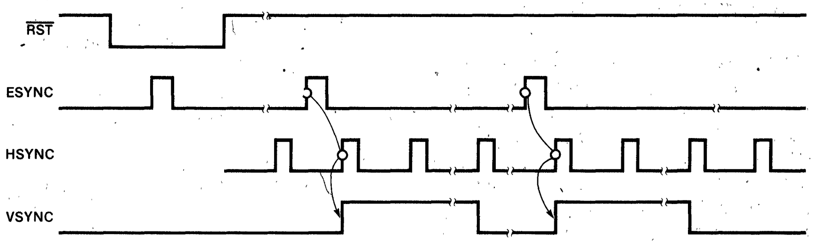


[1] CLK1 OR CLK2 DEPENDING ON CLK1/2 IN MODE REGISTER 1

[2] BLANK = HBLANK + VBLANK

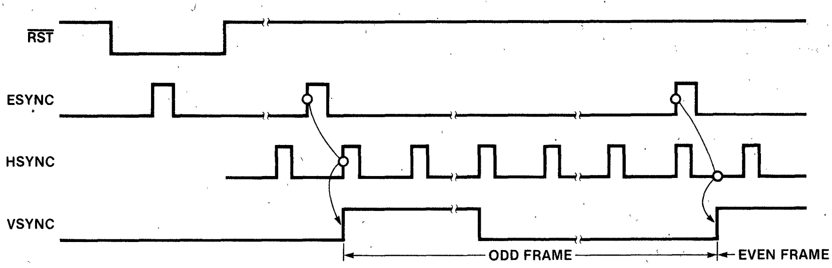
03901A-43

Figure 2-45 Detailed Reset for Test Timing Diagram



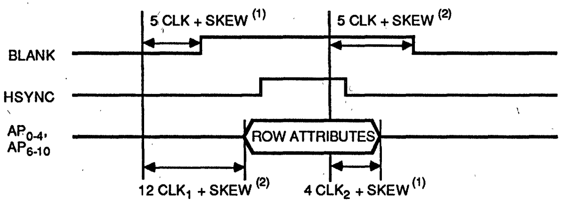
03901A-45

Figure 2-46 Non-Interlaced ESYNC Operation



03901A-46

Figure 2-47 Interlaced ESYNC Operation

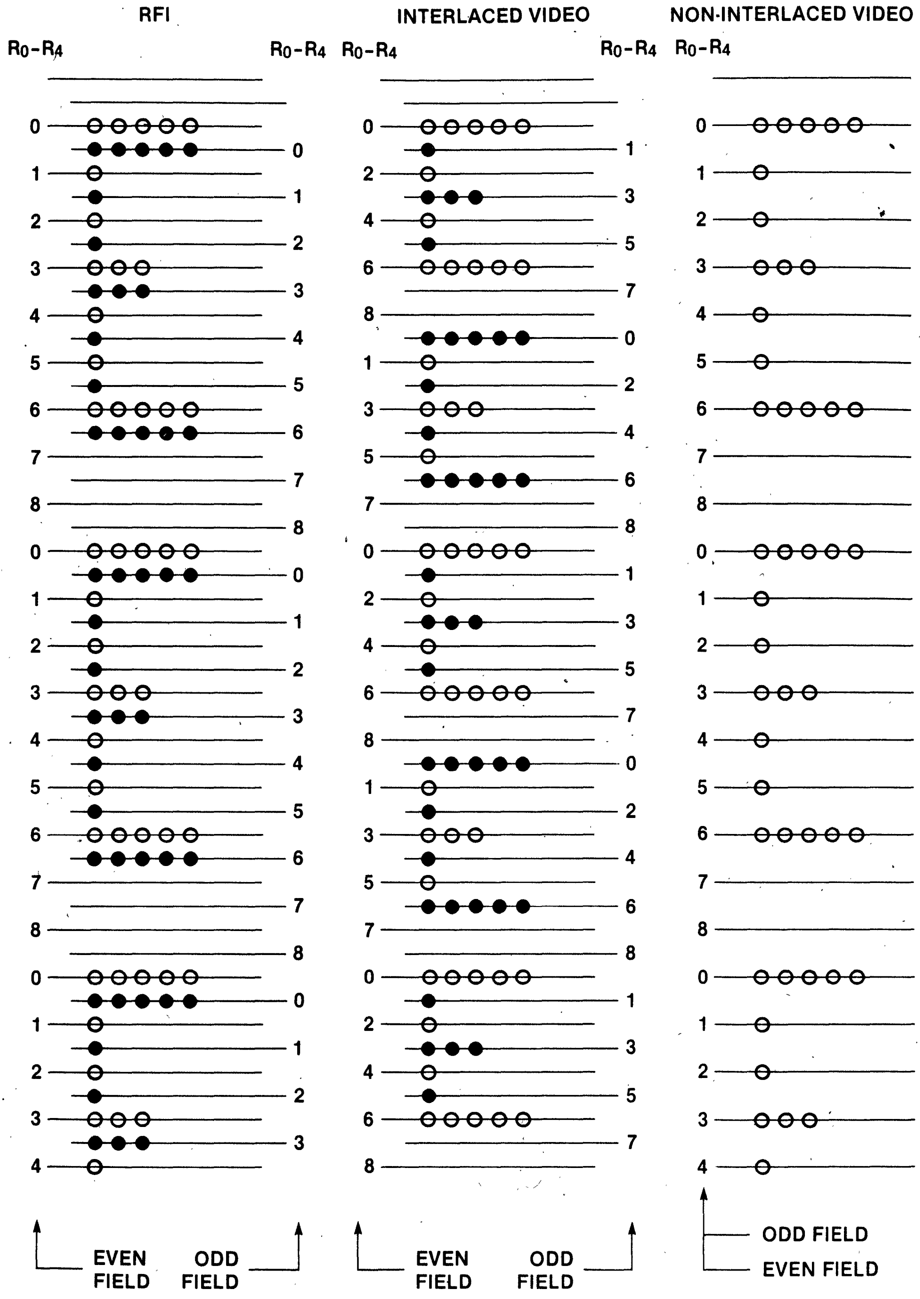


(1) CLK IS CLK_1 OR CLK_2, DEPENDING ON PROGRAMMING OF MODE REGISTER 1 (D_{15}). SKEW IS CLK_1 OR CLK_2 CYCLES; VALUE SPECIFIED IN MODE REGISTER 1 (D_9,D_8)

(2) SKEW IS CLK_2 CYCLES; VALUE SPECIFIED IN MODE REGISTER 1 (D_9,D_8)

05098B 2-48

Figure 2-48 Row Attribute Timing



03901A-47

Figure 2-49 Scan Line Addressing

CHAPTER 3
SOFTWARE COOKBOOK

3.1 INTRODUCTION

The previous chapter discussed the capabilities and features of the CRTC in detail. It addressed the hardware and software design engineer, supplying all the information about the Am8052 needed to design a CRTC based CRT subsystem.

This chapter addresses the software design engineer in particular. It accesses all the related topics, when programming the CRTC. The first section (3.2) describes how the CRTC internal control registers are to be programmed. For frame-timing-register programming, refer to Chapter 2.3.4. The second section (3.3) guides the reader in setting up the linked-list display data structure in memory. Section 3.4 covers window and background strategies and what happens when windows are not aligned correctly. The fourth lists hints on attribute incorporation. Smooth-scrolling is described in Section 3.6. Several diagrams and flowcharts aid the reader in understanding the appropriate programming sequence. Section 3.7 shows how easy text editing becomes when operating on a linked-list data structure. The last section contains three sample programs written in Z8002 assembly language.

The user must perform the six steps listed below to set up a display consisting of background and windows:

- Initialize the 22 control and timing registers of the CRTC.
- Prepare the character strings (segments) for the background and window text. These segments can be placed in any order in memory.
- Prepare matching attribute word strings (segments) for the background and window text. The rules for invoking attributes are described in Sections 2.6 and 3.5.
- Define a Main Definition Block for the background, and a Window Definition Block for each window present on the screen.
- Set up a Row Control Blocks linked-list for the background text and a Window Row Control Block linked-list for each of the windows present. Each Control Block defines one row by linking the appropriate character and attribute segments together.
- Define a set of Row Redefinition Blocks and Window Row Redefinition Blocks. The CRTC must encounter at least one Redefinition Block after power-up to initialize the internal registers storing the row attributes.

3.2 REGISTER INITIALIZATION

The CRTC contains 22 control and timing registers. To prevent damages to monitors all timing registers should be loaded with the desired values before the display is enabled by setting the DE-bit in Mode Register 1. Section 2.3.2 describes how the CRTC registers can be accessed in Slave Mode. The following paragraphs suggest values to be programmed in the control registers.

Mode Register 1. A hardware reset ($\overline{\text{RST}}$ input pulled Low) or a software reset (DE-bit in Mode Register 1) clears it initially. After the linked-list in memory is set up and after all other register are initialized, Mode Register 1 is reloaded with the DE-bit set to one. The Display Hidden feature (DH-bit in Mode Register 1) is intended as a debugging tool for the system programmer. If the DH-bit is set, characters with the invisible-attribute set are displayed. Also, when the DH-bit is set, the rows of displayed windows may not be aligned.

Mode Register 2. The CUE-bit enables the X-Y cursor. The two cursor mask fields (ACM0,1 and XYCM0,1) define the layout of the attribute and X-Y cursor. For example, to specify the attribute cursor as a blinking underline, the attribute cursor definition "Cursor Pin Part" is selected, the Attribute Cursor Blink Enable bit (CATBE) in the Main Definition Block is set, and Cursor Start and End scan line numbers in the Redefinition Block are equal. IES and IEV enable the interrupts on smooth scroll or vertical event (refer to Section 2.7).

Attribute Port Enable Register. Unless the user wants to disable any existing attribute features, a value of $67FF_H$ in the Attribute Port Disable Register is recommended (refer to Sections 2.6 and 4.5). Subscript and Superscript can only be disabled by programming the Attribute Redefinition Register below.

Attribute Redefinition Register. This register should be set to 0000_H unless the user wants to redefine the attribute bits for other purposes.

Top of Page Hard Register & Top of Window Hard Register. These four registers link to the Main Definition Block and the first Window Definition Block. In non-soft-scrolling applications the CPU reloads the "hard" register when altering pages or windows.

Top of Page Soft Register & Top of Window Soft Register. These four registers hold temporarily the updated pointers to the Main Definition Block and the first Window Definition Block. After soft-scrolling an entire row, the CRTC updates the "hard" pointer with the pointer stored in the "soft" register. This double-buffering technique keeps the CPU response time constrains as low as possible. If smooth-scroll is disabled, any write to the TOP Soft Register or the TOW Soft Register will be disregarded by the CRTC.

Attribute Flag Register. Refer to Section 3.5 for programming hints.

Burst Register. The values for the burst count and burst space specified in this register determine the ratio the CRTC is allowed to gain mastership of the system bus. The reader must keep in mind that bus bandwidth for the CRTC must be sufficient enough the fetch the display information. If the allocated bus bandwidth is too low, the screen may only show partial rows, repeated rows, or may be garbage. The burst count and burst space should be programmed to fulfill this requirement in worst case.

Vertical Interrupt Row Register. This register determines the row number which (after being completely loaded) causes the vertical interrupt. The vertical interrupt can be used either to drive a real time clock or to notify the CPU that a certain row just has been loaded. This guarantees that the CRTC does not scan this part of the linked list for about one frame time. The CPU can update this row.

Timing registers. Refer to Sections 2.3.3 and 2.3.4 for description.

3.3 BACKGROUND AND WINDOW TEXT

The background and the window text is stored in the system memory as character strings called character segments. The characters are byte quantities usually encoded in ASCII (American Standard Code for Information Interchange). However, there is no restriction to the ASCII code. The CRTC only compares the characters against the attribute flag mask to decide whether this character is an attribute invoking character. The character font is stored in the external character font generator.

The 16-bit attribute words are stored in attribute strings, called attribute segments, corresponding to the character segments. The character and attribute segments of each row are bound together by the Row Control Blocks (window or background). In the Main Definition Block are the headers of background linked list consisting of Row Control Blocks. The Window Definition Blocks are the headers of the window linked-lists consisting of Window Row Control Blocks. For details refer to Section 2.5.

Main Definition Block and Window Definition Blocks

The following paragraphs list some suggestions how to set up the Definition Blocks. X and Y are zero-origin.

Main Definition Block:

MD_0–MD_1. Contains the pointer to the first background Row Control Block.

MD_2. If an X-Y cursor is desired, the user must set the CUE-bit in Mode Register 2 and load MD_2 with the cursor's x and y coordinates. If an X-Y cursor is not desired, the user should reset the CUE-bit.

MD_3. The CRTC will put the fill character code into the portions of the line buffer not filled by visible characters. For example, if the fill character code is a blank character and the text segments occupy 100 of the 132 characters of the line buffer then the CRTC will assign blanks to the remaining 32 characters of the line buffer.

Setting the FAT-bit will cause the CRTC to load one attribute word for the first fill character of the fill character string. This attribute should be a latched attribute to effect the entire fill character string.

MD_4. The cursor or character blink rate can be programmed from 0.46–3.5 Hz assuming a 60 Hz frame rate. A 75% output inactive duty cycle will make the character visible 75% of the time while a 50% output inactive duty cycle will make it visible 50% of the time.

The slowest programmable smooth scroll rate is one scan line per eight frames and the fastest is eight scan lines per frame.

MD_5. When an interrupt is issued by the CRTC to the host processor, the CRTC returns a vector number stored in MD_5 (soft scroll or vertical interrupt) if the NV bit in Mode Register 1 is set to zero.

MD_6. The TSLC value in MD_6 is applicable only when the CRTC is scrolling rows with variable TSLCs (refer to Section 3.6). The TSLC in MD_6 is set equal to the TSLC of the first displayable row.

Window Definition Block:

WD_0–WD_1. Points to the first Window Row Control Block (the first displayable row in the window). The SCW bit should be set if the window is going to scroll.

WD_2–WD_3. If another window exists after this one, then WD_2 and WD_3 contain the pointer address of that window's Window Definition Block. If no further window exists then WD_2 and WD_3 contain zeros.

WD_4. Specifies the vertical positioning of the current window in terms of the position of the first row of the window ("0" for the topmost row) and the last row of the window.

WD_5. Specifies the horizontal positioning of the current window ("0" for the leftmost character).

Background Row Control Block and Window Row Control Block

A Row Control Block describing a row containing only one segment has a length of seven words (nine words including the pointer to the optional Row Redefinition Block if LNK is set). If the row is partitioned into segments, each segment adds five words to the standard length. Segmented rows are desirable because they simplify editing tasks. Segmentation is required when displaying windows (refer to Chapter 3.4).

Example of Row Control Block (one segment)

RA_0	8000_H	Link bit (LNK) is set to make the CRTC fetch the Row Redefinition Blockpointer. The upper address is set to zero assuming less than 64 kbytes of memory is used.
RA_1	$XXXX_H$	Address of next Row Control Block
RA_2	0010_H	No hidden characters and 16 displayable characters in this row.
RA_3	0000H	Upper address set to zero assuming less than 64 kbytes of memory.
RA_4	XXXXH	Address of character string
RA_5	0000H	Upper address set to zero
RA_6	XXXXH	Address of matching attribute string
RA_7	0000H	Upper address set to zero
RA_8	$XXXX_H$	Address of Row Redefinition Block

Example of a RCB with 3 segments

RA_0	0000H	Most significant bit is reset to specify that this RCB has no Row Redefinition Block
RA_1	$XXXX_H$	Address of the next RCB
RA_2	0010_H	No hidden characters and 16 displayable characters in segment 1
RA_3	8000_H	Most significant bit to signify that more segments follow
RA_4	$XXXX_H$	Address of character string of first segment
RA_5	0000_H	Upper address set to zero
RA_6	$XXXX_H$	Address of attribute string for first segment
RA_7	0020_H	No hidden characters and 32 displayable characters in segment 2
RA_8	8000_H	Signifies more segments to follow
RA_9	$XXXX_H$	Address of character string for second segment
RA_{10}	0000_H	
RA_{11}	$XXXX_H$	Address of attribute string for second segment
RA_{12}	0014_H	No hidden characters and 20 displayable characters in third segment
RA_{13}	0000_H	Most significant bit reset to signify that the following segment is the last one
RA_{14}	$XXXX_H$	Address of character string for third segment
RA_{15}	0000_H	
RA_{16}	$XXXX_H$	Address of attribute string for third segment

Background Row Redefinition Block and Window Row Redefinition Block:

After power-up the CRTC requires at least one Background Row Redefinition Block to initialize internal CRTC registers storing the character positioning. Additionally, when displaying windows, at least one Window Row Redefinition Block has to be provided after power-up. The CRTC does not reset these registers when displaying a new page; it overrides the contents only when it encounters a new Row Redefinition Block. However, it is a good practice to add a Row Redefinition Block to the first Row Control Block of both, window and background.

The maximum number of scan lines (TSLC + 1) is 32 since the CRTC provides a 5-bit scan line address. The minimum value for the Total Scan Line Count (TSLC) is determined by the height of the character font. In order not to truncate a part of the displayed character TSLC should be at least equal to NCE (Normal Character End). NCE minus NCS plus 1 (NCE – NCS + 1) equals the actual height of the character but it does not start on the first scan-line unless NCS = 0.

Example of a Row Redefinition Block

TSLC = $0D_H$	Row height is 14 scan lines
NCS = 02_H	Characters are displayed on the NCE
= $0A_H$	3rd through 11th scan lines
SPCS = 00_H	Superscripts are displayed on the
SPCE = 08_H	1st through 9th scan lines
SBCS = 04_H	Subscripts are displayed on the
SBCE = $0C_H$	5th through 13th scan lines
CURS = $0B_H$	Cursor is displayed on the 12th and 13th scan line
CURE = $0C_H$	
DR = 00_H	Normal character row
UND = $0C_H$	Underline is displayed on 13th scan line
SUND = 01_H	Shifted Underline on 2nd scan line (over bar)

The two Row Attributes (10 bits) are not processed internally; this word is output during horizontal retrace to extend the attribute capabilities of the CRTC.

Attribute Processing

If a row displayed does not contain any attributes then the CRTC will not examine the attribute addresses in that row's RCB. Otherwise, these attribute addresses contain the starting location of the attributes list for that row. The attribute codes accessed by the attribute address should appear in the order the attributes are referenced. For example, if the 1st character on a particular row is a superscripted, the 2nd character is a subscripted, and the 3rd character underlined then the attribute string should be 0010_H (superscript), 0008_H (subscript) and 0002_H (underline) respectively. Note that the attribute string might be shorter than the character string since attribute can be fetched on a demand basis. Refer to Chapter 3.5 for details.

3.4 BACKGROUND AND WINDOWS

There are two independent linked-list data structures that describe background and windows. Windows are rectangular blocks of text that overlay the background without altering the background data structure. The background remains intact when the overlaying window is removed. When compared to a software implementation of windows, this hardware approach eliminates the modification of the display linked-list when displaying or removing windows. Window boundaries can be defined as large as the entire display screen, or as small as one character in width. When displaying windows, the user must take into consideration that the window boundaries fall on segment boundaries of the background. Consequently, a heavily segmented background row increases the number of choices of window placements and sizes. If the sum of the number of visible characters for a row is less than the window size specified in the Window Definition Block, the window row will be filled by the fill character code.

The rule for placing multiple windows on the screen is:

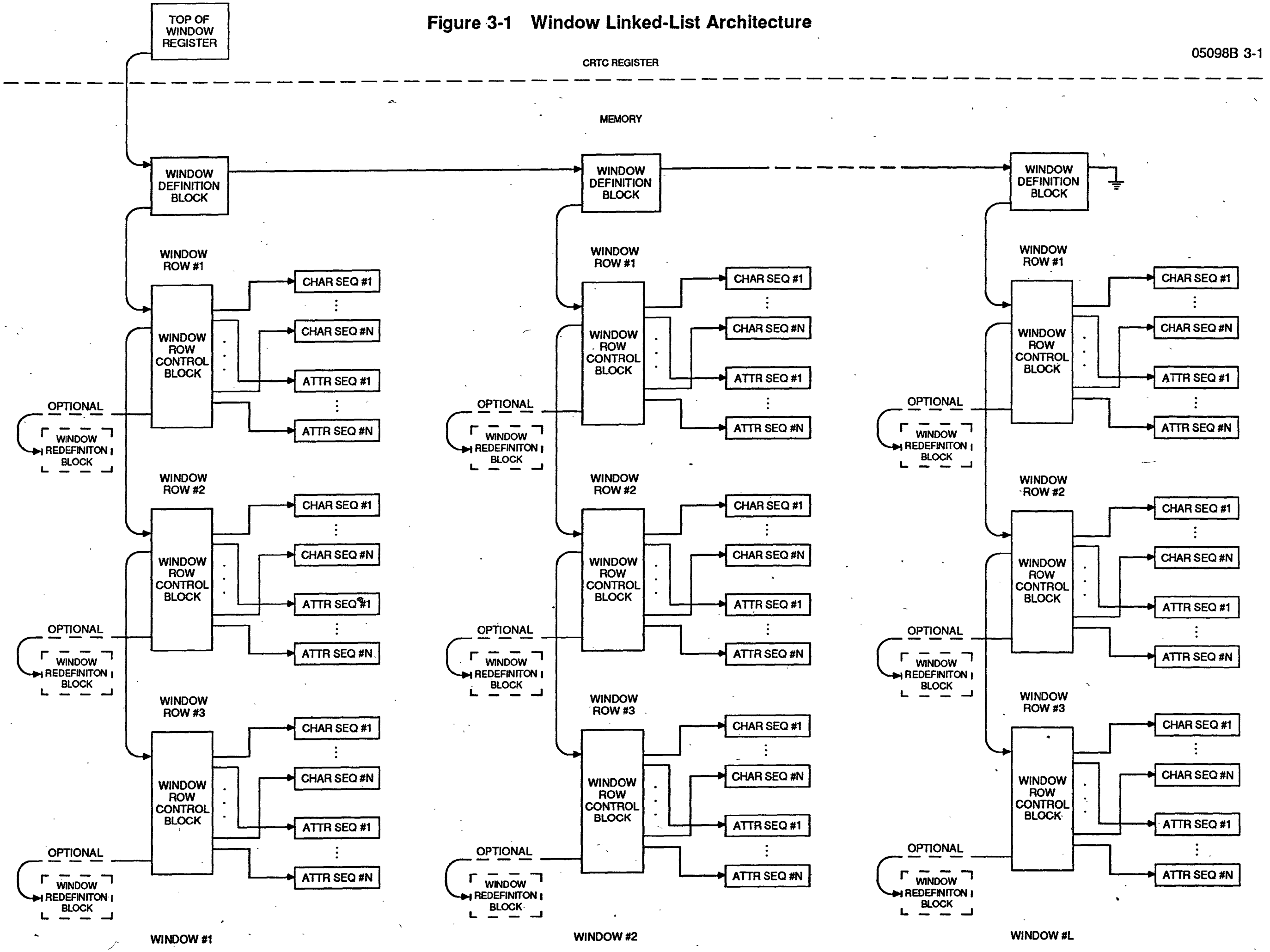
- Windows must be separated vertically by at least two background rows for non-interlaced mode, and three background rows for interlaced or RFI modes.

Figure 3.1 shows the linked-list structure for a multiwindow display. The Top Of Window Hard Register (TOWH) points to the Window Definition Block (WDB) of the first (topmost) window. Each WDB links to the WDB describing the window below. The WDB for the window on the bottom of the screen (here: the third WDB) contains a pointer set to zero, specifying that the current window is the last displayed window. If no window is to be displayed, TOWH is set to zero. Additionally, each WDB contains the pointer to the first Window Row Control Block (WRCB). A WRCB has a similar structure as a background Row Control Block (RCB). To add or delete a window, the user simply changes the next WDB pointer in the desired Window Definition Block.

Non-Aligned Windows

If a window is not aligned to the segment boundaries of the background, a forced alignment will occur after each re-link. This forced alignment affects the background segments overlayed by the window. Some example for forced alignment are illustrated in Figures 3.2 to 3.6.

Figure 3-1 Window Linked-List Architecture



Background/Window Strategies

The flexibility of the window linked-list structure allows the placement of a window anywhere on the screen, provided that the constraints mentioned earlier in the chapter are met. The user can use the flexibility of the window placements to implement a split screen format, or a display containing virtual side by side windows.

A split screen format can place two equal-size texts on the screen simultaneously, one in the window and one in the background. This feature is useful for character searching, comparing, and other text processing purposes. Figure 3-7 shows examples of split screens.

The window placement rules specify that two windows must be separated vertically by two or three background rows. However, virtual windows can be placed side by side. Figure 3-8 shows an example where the screen is divided into four quarters. Any one of these four windows can be scrolled independently. Virtual side-by-side windows give the illusion that windows can be adjacent to each other by redefining background and windows via the control block structure.

Examples of virtual side by side windows

The screen in Figure 3-8 is composed of two rows, consisting of a total of four strings: ONE, TWO, THREE and FOUR. These strings (segments) can be placed anywhere in the system memory. Two Row Control Blocks (RCBs) link the segments together.

Each segment is also pointed to by a Window Row Control Block (WRCB). To be able to scroll a particular segment, this segment must first be defined as a window. Figure 3-9 shows the linked list configuration for scrolling the segment ONE. Window display is enabled by changing the Top Of Window Register (TOW) from "0" to the address of the Window Definition Block (WDB). The WDB links to the segment to be scrolled.

To enable scrolling of the segment FOUR, the pointer in the WDB linking to the WRCB linked list needs to be modified (Figure 3-10).

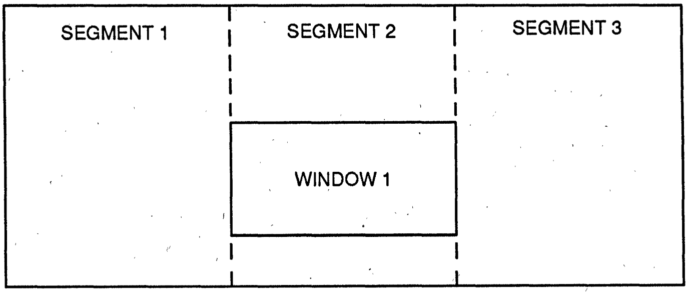
3.5 ATTRIBUTES

The CRTC supports nine character attributes such as: Cursor, Blink, Underline, Shifted Underline/Strike Through, Subscript, Superscript, Reverse, Highlight, and Ignore Character. Four additional attribute bits are user definable. One attribute bit specifies whether this attribute is latched or unlatched. The total number of fourteen attribute bits are stored in the sixteen-bit attribute word fetched on a character basis. The four user-definable attributes are predefined attributes; except for the Ignore Character and Cursor attribute (D_0–D_{10} of the attribute word) which may be put out on the Attribute Port lines AP_0–AP_{10} respectively.

To maximize the flexibility of attribute processing, the internal attribute processing of the CRTC can be disabled. This gives the user up to 11 user-definable attributes. The internal processing of the five attributes (Blink, Underline, Shifted Underline/Strike Through, Subscript, and Superscript) is controlled by the Attribute Redefinition Register. The Attribute Port lines themselves are controlled by the Attribute Enable Register. This register allows the disabling of the output of particular attributes; the line becomes Low.

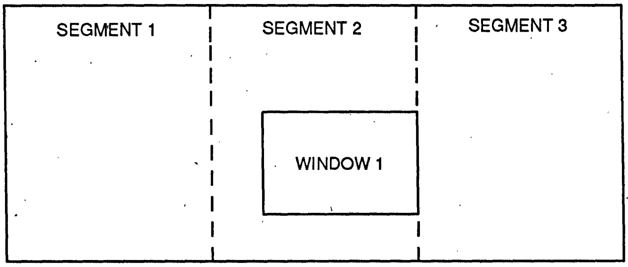
A character may have any combination of these attributes. The only exception is that one character cannot have both the superscript and subscript attribute.

The number of hidden characters (Hidden #) in the Row Control Block or Window Row Control Block



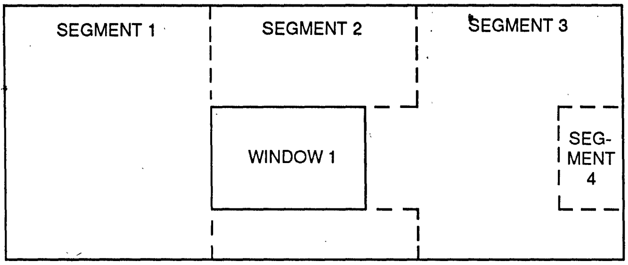
05098B 3-2

Figure 3-2 The Original Aligned Structure



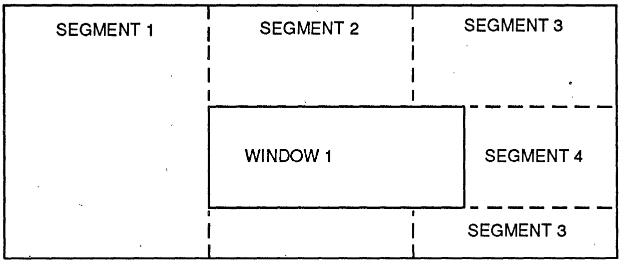
05098B 3-3

Figure 3-3 The left boundary of the window is drawn inward. The front portion of Segment 2's data will appear in the gap not covered by the window.



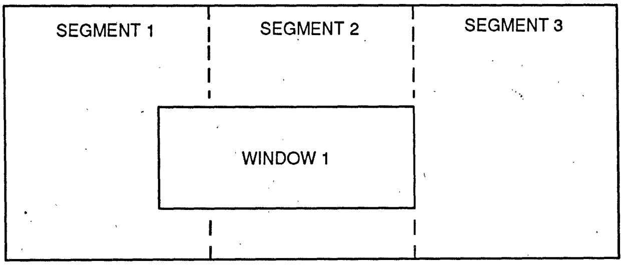
05098B 3-4

Figure 3-4 The right boundary of the window is drawn inward. The data from Segment 3 starts immediately after the window and part of the previously invisible Segment 4 becomes visible.



05098B 3-5

Figure 3-5 The right boundary of the window is extended outward. The extended portion of the window will inhibit the loading of Segment 3 into the line buffer and.?



05098B 3-6

Figure 3-6 The left boundary of the window is extended outward. The extended portion of the window will overlay some of the right portion of Segment 1's data.

must account for the characters in the segment with the Ignore Character attribute set. The CRTC needs this information in order to overlay windows correctly. For debugging purposes, the ignored characters can be displayed by setting the DH-bit (Display Hidden DH=1) in Mode Register 1. Displaying ignored characters in a segment will increase the number of displayable characters in the segment. This may cause windows to overlay incorrectly.

Attribute Invoking

The CRTC supports a demand attribute fetch to save memory space and to reduce the bus occupancy of the CRTC. The CRTC scans the fetched characters for attribute invoking characters. A character is an attribute invoking character when it matches the Value programmed in the Attribute Flag Register. Each time a match occurs an attribute word is fetched from the attribute string. Certain bits of the character code can be masked off by the Mask, programmed in the same register. The CRTC supports three basic options as shown in Figure 3.11.

In the straightforward Option 1, each character invokes an attribute. In this case, the Latched/Unlatched attribute is ignored since latched attributes apply only to characters not invoking attributes. To enable this scheme, the Attribute Flag Register is programmed with 00xx$_H$ where "x" is a "don't care."

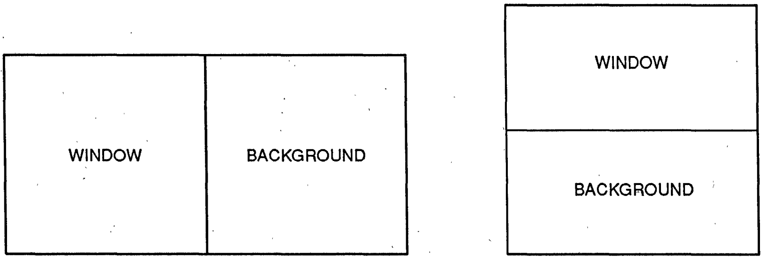


Figure 3-7 Horizontal and Vertical Split Screens

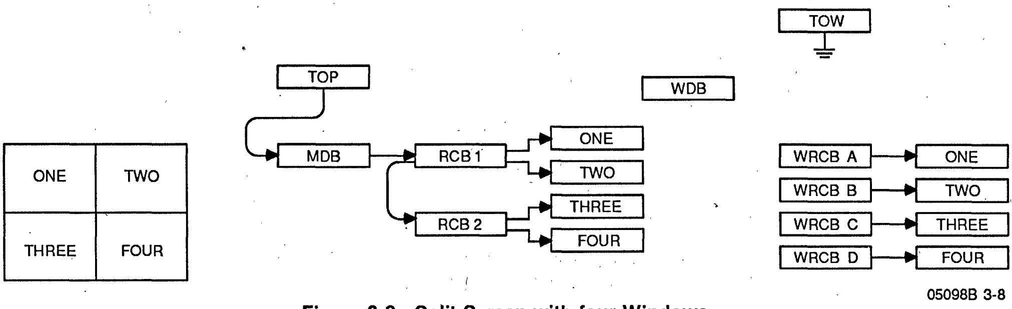


Figure 3-8 Split Screen with four Windows

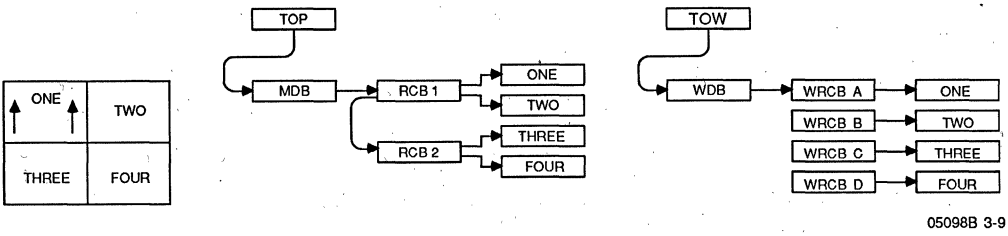
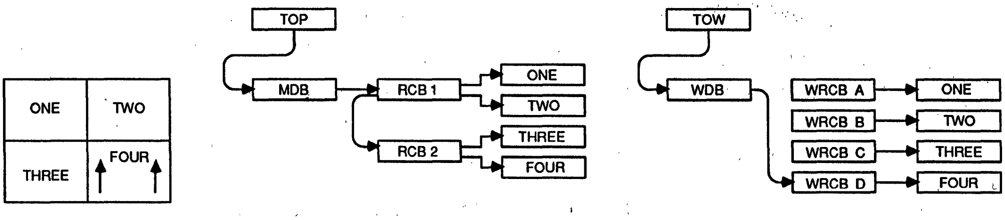


Figure 3-9 Scrolling Window "ONE"

In Option 2, only the characters with the most significant bit of character code set invoke an attribute. Therefore, the Attribute Flag Register is programmed with 8080_H. A Mask of 80_H specifies that only the most significant bit of the character code must match the most significant bit of the Value (here: "1"). The attribute invoking character is displayed if the Invisible Attribute Flag in Mode Register 1 is not set. If the Invisible Attribute Flag is set, the attribute invoking character is not displayed and the fetched attribute applies to the next character.

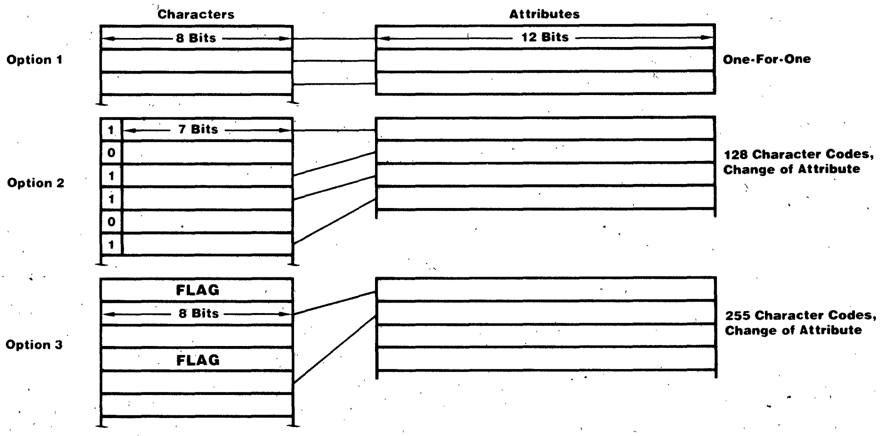
In Option 3, only one specific character code (the Flag) invokes an attribute. The Invisible Attribute Flag is set to disable the display of these characters. The Mask of the Attribute Flag Register is loaded with FF_H to specify that the character code must match exactly the Value to invoke an attribute. To program the character code 10_H to be the Flag, the Attribute Flag Register is loaded with $FF10_H$.

Certain attribute port lines may be disabled (they stay Low) by loading a pattern into the Attribute Port Enable Register. For example, a value of $607F_H$ in the Attribute Port Disable Register will enable all the predefined attributes and disable all the user-definable attributes. The internal processing of the predefined attributes may be disabled by using the Attribute Redefinition Register. This yields up to 11 user-definable attributes. The predefined attributes Reverse and Highlight are not processed internally, so they can be treated as user-definable attributes. The processing of these attributes takes place in the Video System Controller (Am8152A). To display the attribute invoking character, the IAF-bit in Mode Register 1 must be reset.



05098B 3-10

Figure 3-10 Scrolling Window "FOUR"



03684B-6

Figure 3-11 Attribute Fetch Options

Latched and Unlatched Attributes

A latched attribute applies to the attribute invoking character and all subsequent characters not invoking attributes. Latched attributes are not affected by window/background boundaries or screen boundaries. This means that the latched attributes in windows carried over to the background will carry over to the next frame. To avoid strange results in processing attributes, it is a good practice to have a latched attribute for the first character of each segment.

Examples of attribute processing

The characters A and B invoke attributes.
The display is A C C C B D D D.

A and B both invoke unlatched underline attributes (0002_H)

$\underline{\text{A}}\text{ C C C }\underline{\text{B}}\text{ D D D}$

A invokes a latched underline attribute (8002_H), B invokes an unlatched superscript attribute (0010_H).

$\underline{\text{A C C C}}^{\text{B}}\,\underline{\text{D D D}}$

A invokes a latched underline attribute (8002_H), B invokes a latched superscript attribute (8010_H).

$\underline{\text{A C C C}}^{\text{BDDD}}$

A invokes a latched underline attribute (8002_H), B invokes a latched null attribute (8000_H).

$\underline{\text{A C C C}}\text{ B D D D}$

A invokes a latched underline attribute (8002_H), C invokes a latched null attribute (8000_H), B invokes a latched underline attribute (8002_H), and D invokes latched null attribute (8000_H).

$\underline{\text{A}}\text{ C C C }\underline{\text{B}}\text{ D D D}$

The FAT-Bit

Setting the FAT-bit (Fill Code Attribute bit in the Main Definition Block) will cause the CRTC to fetch an attribute for the first Fill Code character in a Fill Code string. If the Fill Code attribute is unlatched, then it only applies to the first Fill Code character. If the Fill Code attribute is latched, then it applies to the whole Fill Code segment. The first valid character after the Fill Code segment should unlatch the previously latched attribute; this prevents the attribute from being carried past the Fill Code segment.

The CRTC loads Fill Code characters into its internal row buffer if either one of the three conditions below is true:

- The character code pointer of a segment is zero; the CRTC will fill the current segment with Fill Code. The size of the segment is defined by Visible #. The Fill Code Attribute is fetched from the address defined by the Attribute Pointer.
- The total number of characters fetched for a window is less than the horizontal width of the window (End Window Character #–Start Window Character #). The remaining part is filled with the Fill Code. The CRTC increments the current attribute pointer to fetch the Fill Code attribute; this means, the Fill Code attribute follows the last fetched character attribute.
- The total number of characters fetched for a row is less than that defined by the SLIM-bit in Mode Register 1 (96 or 132 characters). The remaining part is filled with the Fill Code. The Fill Code attribute is fetched from the location following the last fetched character attribute.

3.6 VERTICAL SMOOTH SCROLL

Vertical Smooth Scroll moves the text in fraction of rows up or down; the effect is more eye-pleasing than hard scrolling. The number of scan lines the text is moved per frame is programmable in 16 steps. The programmable rate ranges from very slow motion, where the viewer sees the text jumping in steps of scan lines (lowest rate), to a scroll rate where the text moves faster than the eyes of the viewer can follow (highest rate).

The CRTC performs smooth scrolling by adding a variable offset to the initial scan line count of the top most row. The offset is decremented or incremented, on a frame basis, for scrolling up or down, respectively. For example, if the scroll rate is one scan line per four frames, then the CRTC will scroll the text one scan line in one frame and waits for three frames before scrolling another scan line. In this manner, each character row appears to move upward smoothly, as opposed to the jerky motion of hard scrolling. The CRTC controls the smooth scroll process with minimum CPU intervention. The CPU only needs to update the linked list each time an entire row is scrolled in or out. All other operations that take place are transparent to the user.

The background and windows can each scroll independently, but not simultaneously. Either the background or window(s) can scroll at any given time. When multiple windows are to be scrolled simultaneously they do so synchronously, with the same rate, and in the same direction. The information on this type of scrolling is defined in the Main Definition Block. Windows can be scrolled independently by enabling window smooth scrolling in the Main Definition Block and setting the Smooth Scroll Window bits in the Window Definition Blocks of the windows to be scrolled.

Smooth Scrolling Up and Down

Flipping between two Main Definition Blocks (MDBs) or two Window Definition Blocks (WDBs), when scrolling background or windows, avoids screen flickering caused by scanning partially updated definition blocks. The Top of Page/Window Smooth Register alternately points to two different definition blocks. The CPU always updates the definition currently not processed by the CRTC. On relink request, the CPU toggles the pointer in the Top Of Page/Window Smooth Register. Initially, the TOPS/TOWS Register points to the definition block linking to the Row Control Block (RCB) of the topmost row. Figure 3.12 illustrates this process.

Background and Window Smooth-Scroll

To smooth-scroll the background, only the scroll bits in MD_4 of the Main Definition Block need to be set. To smooth-scroll a window, the scroll bits in MD_4 and the SCW bit in the scrolling window's definition block must be set. When a background text is scrolled past a window text, a common TSLC must exist between the window row and the background row that it overlays (Figure 3.13). If a background row is scrolled past a window row with their TSLC being unequal then distortion to the display will occur.

It is essential that for any scrolling activity, the TSLC in MD_6 of the MDB must be equal to the TSLC of the first RCB. To scroll a background-only dis-play with variable TSLCs on each row, the TSLC in MD_6 of the MDB must be equal to the TSLC of the top-most row. Consequently, MD_6 must be constantly updated while the background is scrolling. The update of MD_6 must occur before the new pointer is written to the Top of Page Register.

The interaction between the CPU and the CRTC may be coordinated using one of three techniques: polling, non-vectored interrupt, or vectored interrupt.

Polling

The CPU may test the IPS-bit (Interrupt Pending Smooth-Scroll bit in Mode Register 2) frequently to verify the time when the CRTC requires CPU intervention. The CRTC issues two types of interrupts (setting the interrupt pending bit) distinguished by the Scroll In Progress bit (SIP-bit in Mode Register 2). When the SIP-bit is set on interrupt the CRTC likes to have the Top Of Page/Window Smooth Register updated. When the smooth-scrolling is finished, the CRTC issues an interrupt with the SIP-bit reset to notify the CPU that it is done. After servicing the requested action, the CPU must reset the interrupt pending by software.

Non-Vectored Interrupt

A less time-consuming and more efficient way of requesting CPU interventions is to use hardware interrupts. If the Interrupt Enable Smooth-Scroll bit (IES-bit in Mode Register 2) is set the CRTC will also activate the INT line each time the IPS-bit is set. The INT line may be connected to the non-vectored interrupt input of the CPU or to a dedicated interrupt controller such as the 8259A or Am9519. In the end of the interrupt service routine the IPS-bit must be reset to enable further interrupts.

Vectored Interrupt

The most elegant way of synchronizing CPU interventions is to use vectored interrupts. Therefore the No Vector bit (NV of Mode Register 2) must be reset. Similar to non-vectored interrupts the CRTC also activates the INT line when IPS-bit is set. When the CPU acknowledges the interrupt by asserting the INTACK line the CRTC strobes out an 8-bit interrupt vector. Usually, this pointer addresses indirectly via a vector table the interrupt service routine. Bit 1 of the interrupt vector reflects the status of the SIP-bit so that testing the SIP-bit in the interrupt service routine becomes obsolete. The CPU may execute different interrupt service routines for both types of interrupts. Asserting the INTACK line also sets the Interrupt Under Service Smooth Scroll bit (IUSS-bit in Mode Register 2). Note, that unlike the implementation in some Z8000-type peripherals the interrupt acknowledge does not reset the interrupt pending bit. Both the IPS-bit and the IUSS-bit must be reset by software in the end of the interrupt service routine.

3.7 EDITING THE LINKED-LIST

All text data is organized in a linked-list structure simplifying editing tasks. The host CPU only needs to modify the pointers in order to swap pages, insert lines, delete lines, or display windows. Pages can be swapped simply by reloading either the Top of Page Register pointer or the pointer in the Main Definition Block linking to the top most row. Since the pointers have both an upper and a lower part (two 16-bit values), a problem arises when the host CPU has to update both for a new pointer value; the CRTC might use a partially updated pointer in the case where the CPU has loaded only either the upper or lower pointer, and the CRTC gains the bus mastership right after this load. This problem occurs when updating both pointers in Top of Page/Window Register or the pointers in Main Definition Block.

Row Control Block Memory

The user can prevent the problem by synchronizing the host CPU updates with the CRTC linked-list scanning, via the vertical interrupt feature. For example, the vertical interrupt may be set to occur after loading the first row to signal that the Main Definition Block may be modified without any risk of running into the above mentioned problems. If only the lower part of the pointers is to

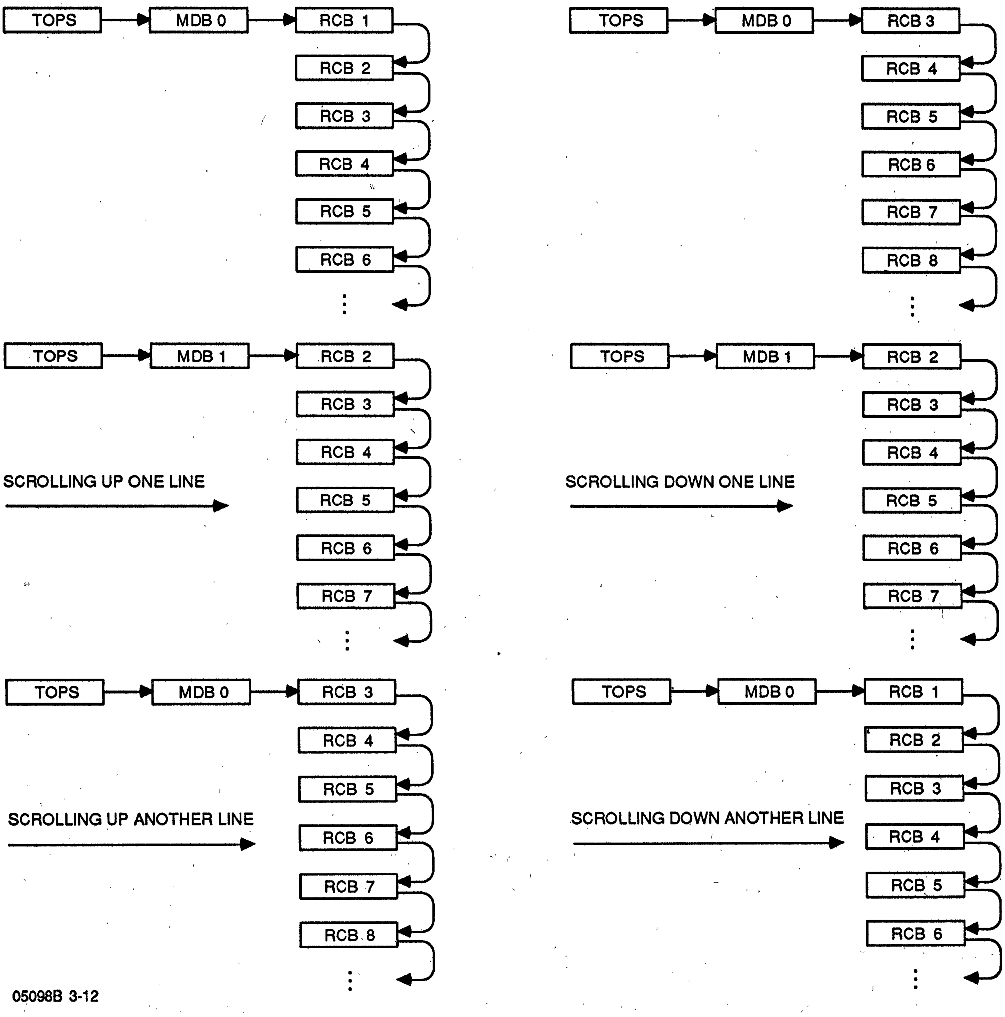


Figure 3-12 MDB Swapping Simplifies Scrolling

be modified, the problem does not occur; pointers can be modified at any time.

Row Insertion

First, link the new row to the subsequent row (Step 1 in Figure 3.20), then link the previous row to the new row (Step 2 in Figure 3.20). When operating only with the lower half of the pointers, this type of modification can be done, at any time, without any concern of synchronization to the CRTC operation.

Row Deletion

A row is deleted simply by linking the pointer in the previous Row Control Block to the next Row Control Block (Step 1 in Figure 3.21).

Character Code and Attribute Pointers

The least significant bit of the linked-list pointers in registers or memories is "don't care." The CRTC resets this bit when operating with the pointer. Consequently, all addresses put out by the CRTC are even. Since characters are 8-bit quantities which can be located at either odd or even addresses, the user has to take into consideration that character code strings always start at even addresses. This might become a restriction if the background characters are stored in a linear list and this list has to be split up into segments in order to overlay windows. Since the character code pointers are always even, the background list can be split only at even addresses. The number of choices can be increased by interleaving characters with the Ignore Attribute Set.

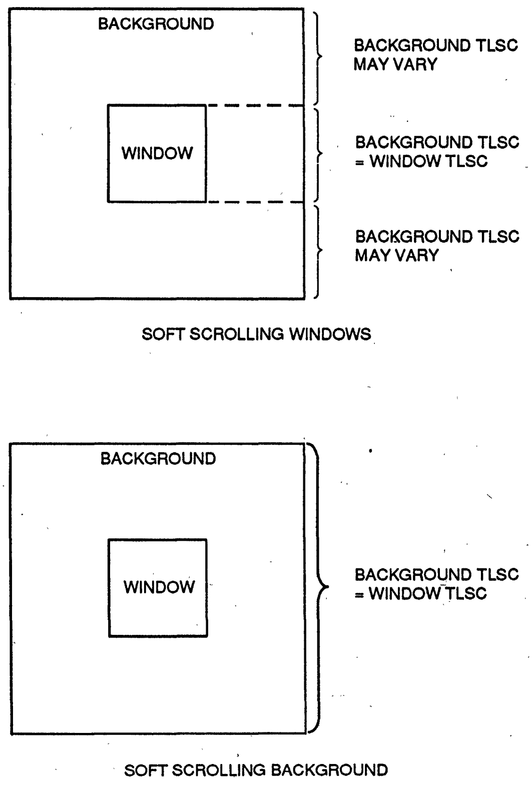
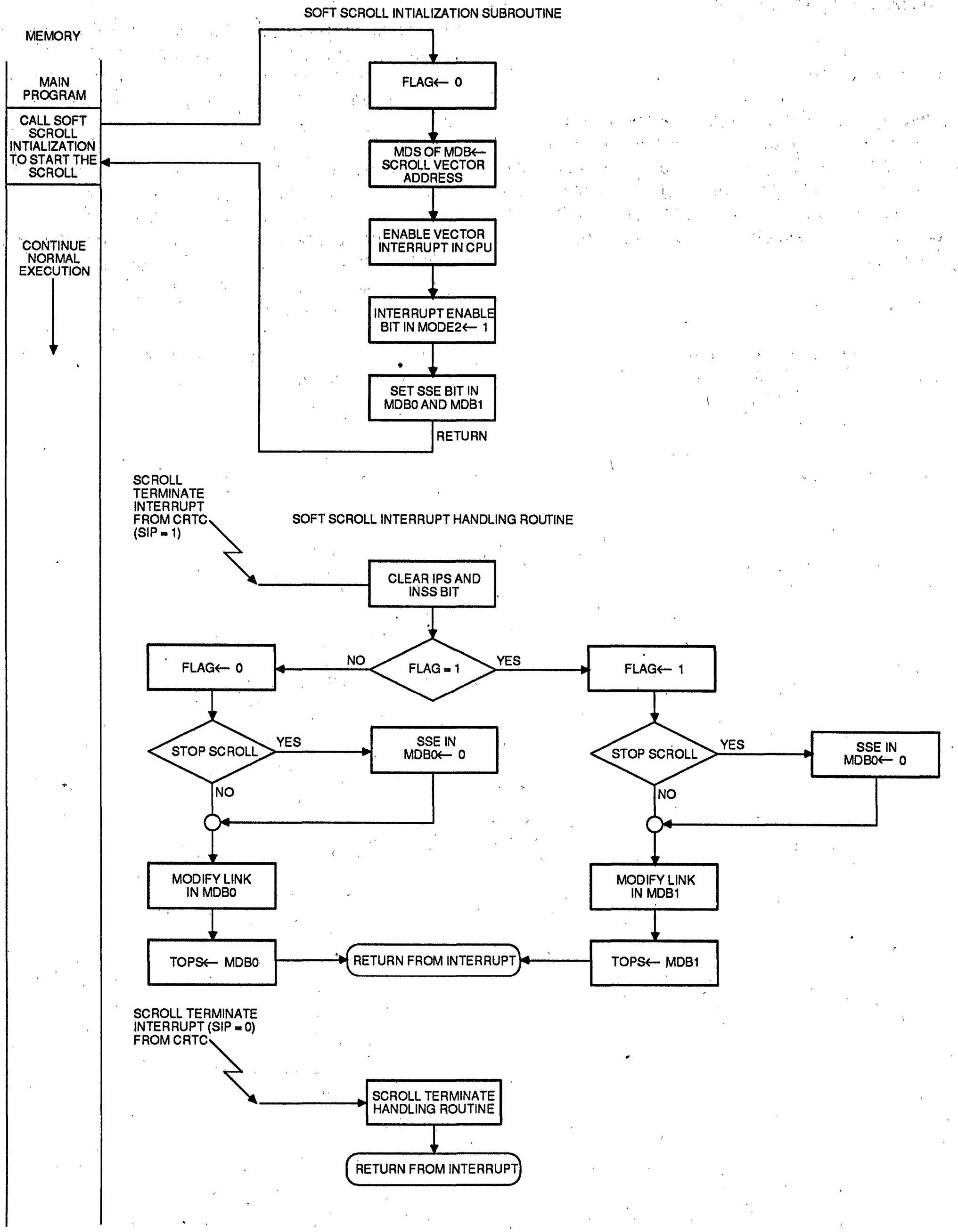
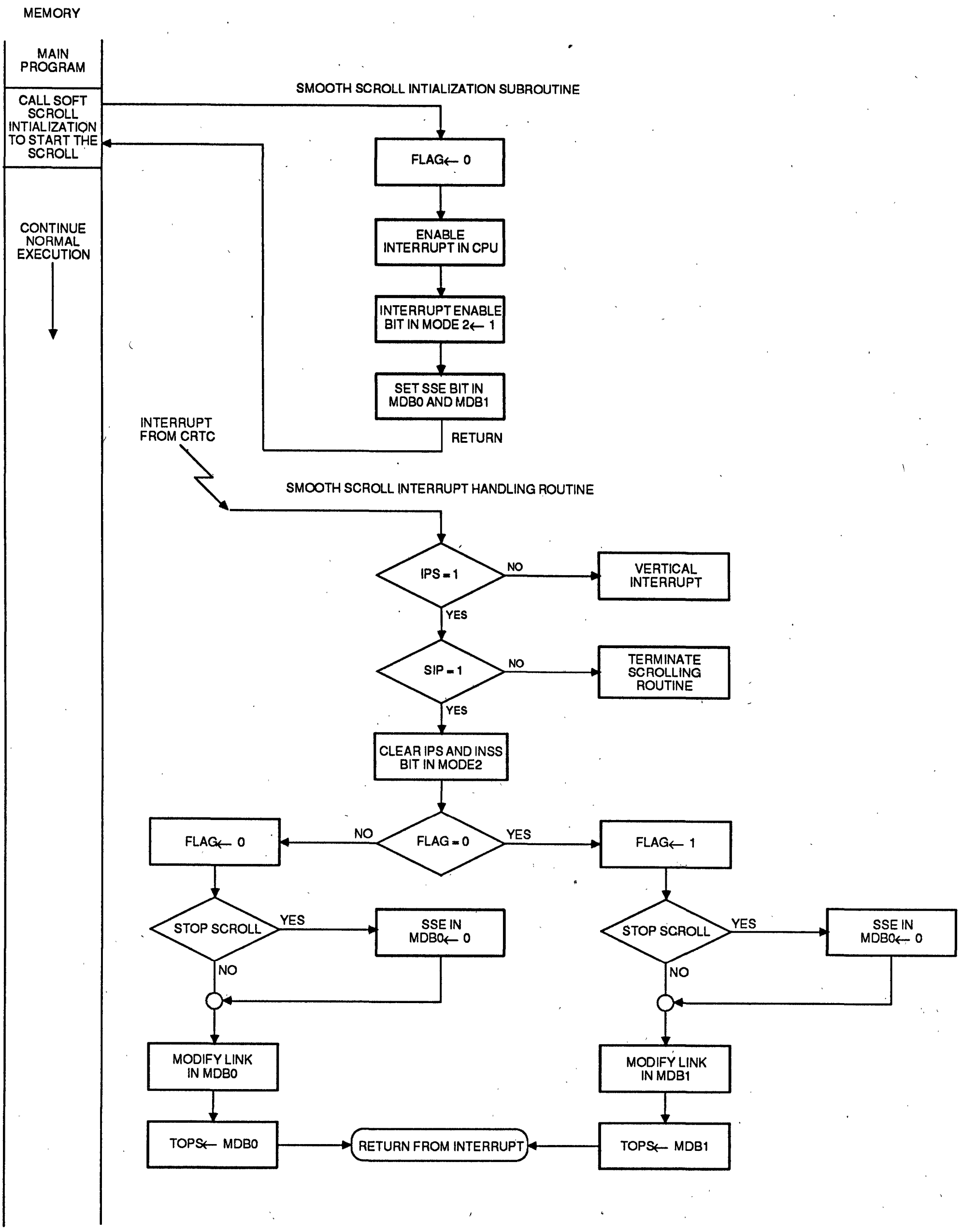


Figure 3-13 Background rows overlayed by window must have the same TSLC than the window rows.


SOFT SCROLL INTIALIZATION SUBROUTINE
MEMORY
MAIN PROGRAM
CALL SOFT SCROLL INTIALIZATION TO START THE SCROLL
CONTINUE NORMAL EXECUTION
FLAG← 0
MDS OF MDB← SCROLL VECTOR ADDRESS
ENABLE VECTOR INTERRUPT IN CPU
INTERRUPT ENABLE BIT IN MODE2← 1
SET SSE BIT IN MDB0 AND MDB1
RETURN
SCROLL TERMINATE INTERRUPT FROM CRTC (SIP = 1)
SOFT SCROLL INTERRUPT HANDLING ROUTINE
CLEAR IPS AND INSS BIT
FLAG = 1
NO
YES
FLAG← 0
FLAG← 1
STOP SCROLL
YES
SSE IN MDB0← 0
NO
STOP SCROLL
YES
SSE IN MDB0← 0
NO
MODIFY LINK IN MDB0
MODIFY LINK IN MDB1
TOPS← MDB0
RETURN FROM INTERRUPT
TOPS← MDB1
SCROLL TERMINATE INTERRUPT (SIP = 0) FROM CRTC
SCROLL TERMINATE HANDLING ROUTINE
RETURN FROM INTERRUPT

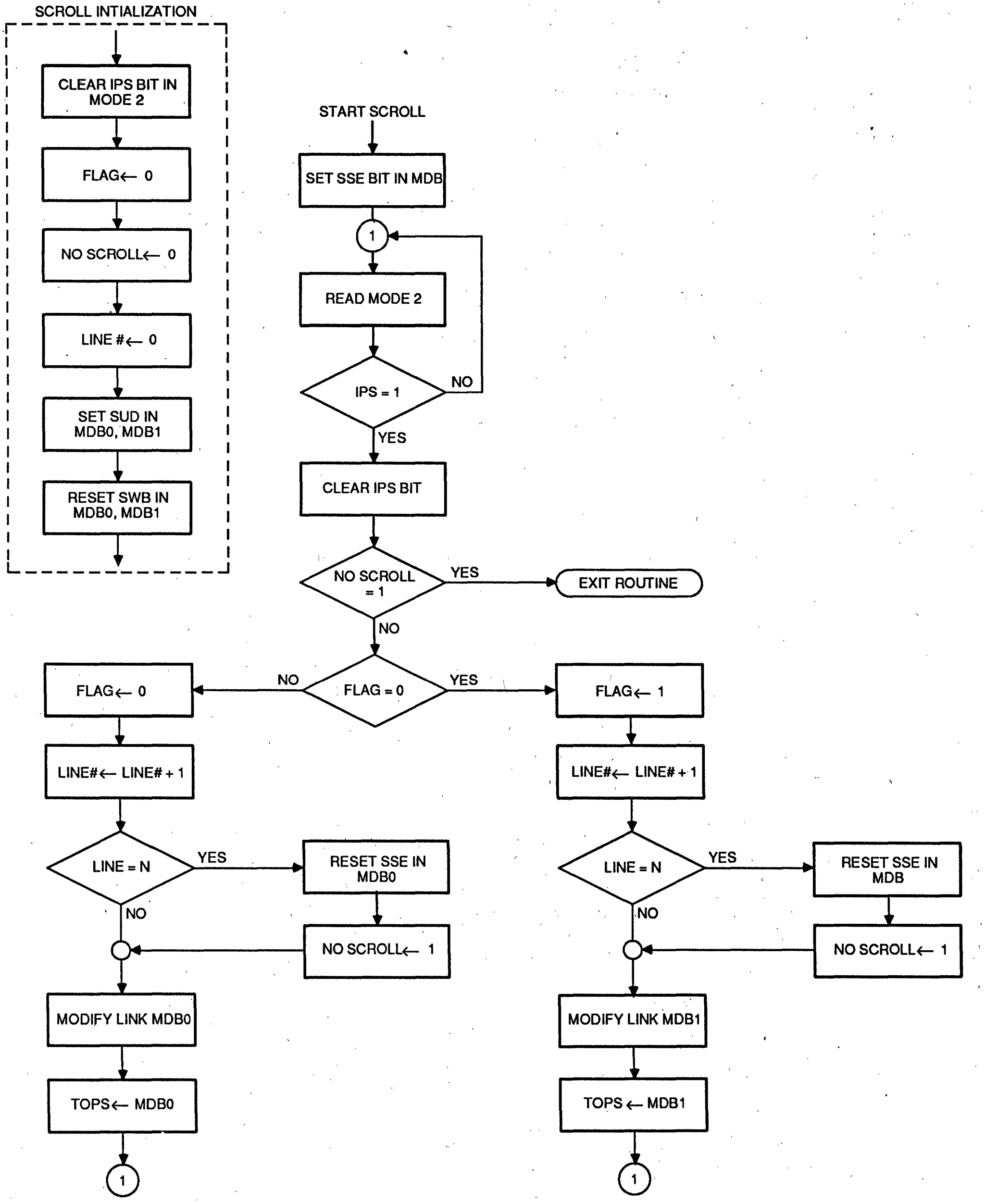
05098B 3-14

Figure 3-14 Flowchart for scrolling up the background using vectored interrupts



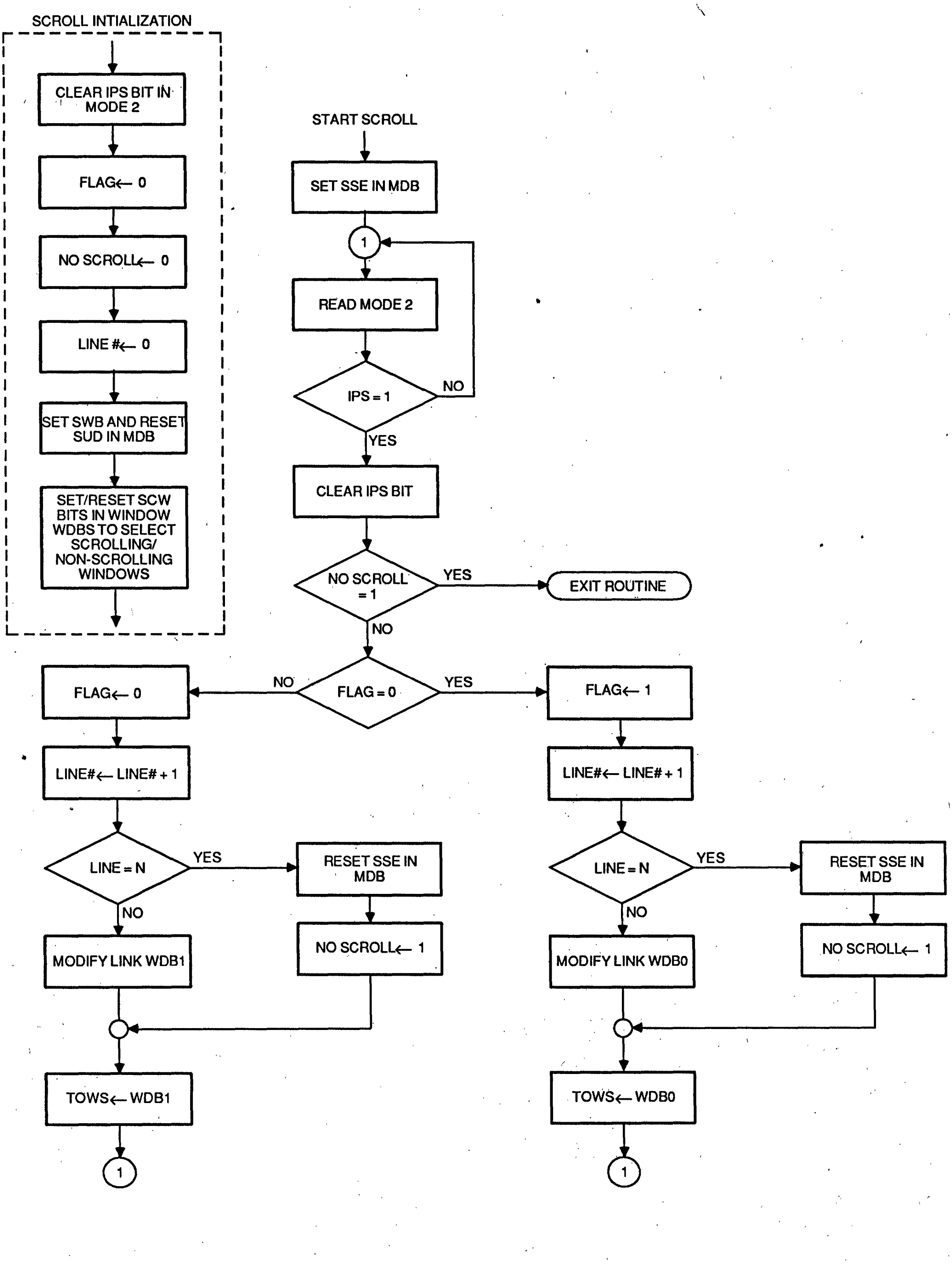
05098B 3-15

Figure 3-15 Flowchart for scrolling up the background using non-vectored interrupts



05098B 3-16

Figure 3-16 Stop scrolling up of the background after N lines



05098B 3-17

Figure 3-17 Stop scrolling up of a window after N lines

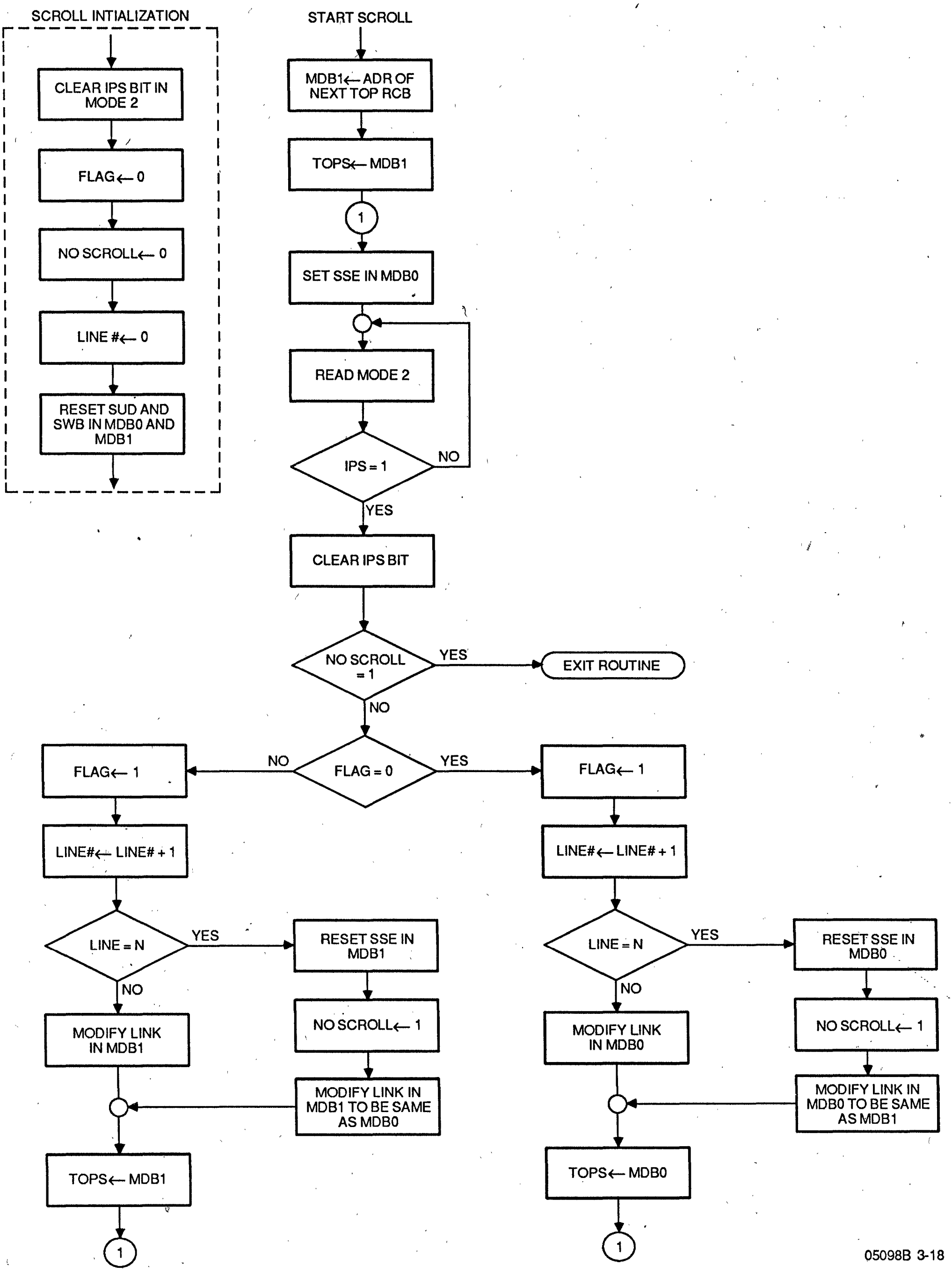


Figure 3-18 Stop scrolling down of the background after N lines

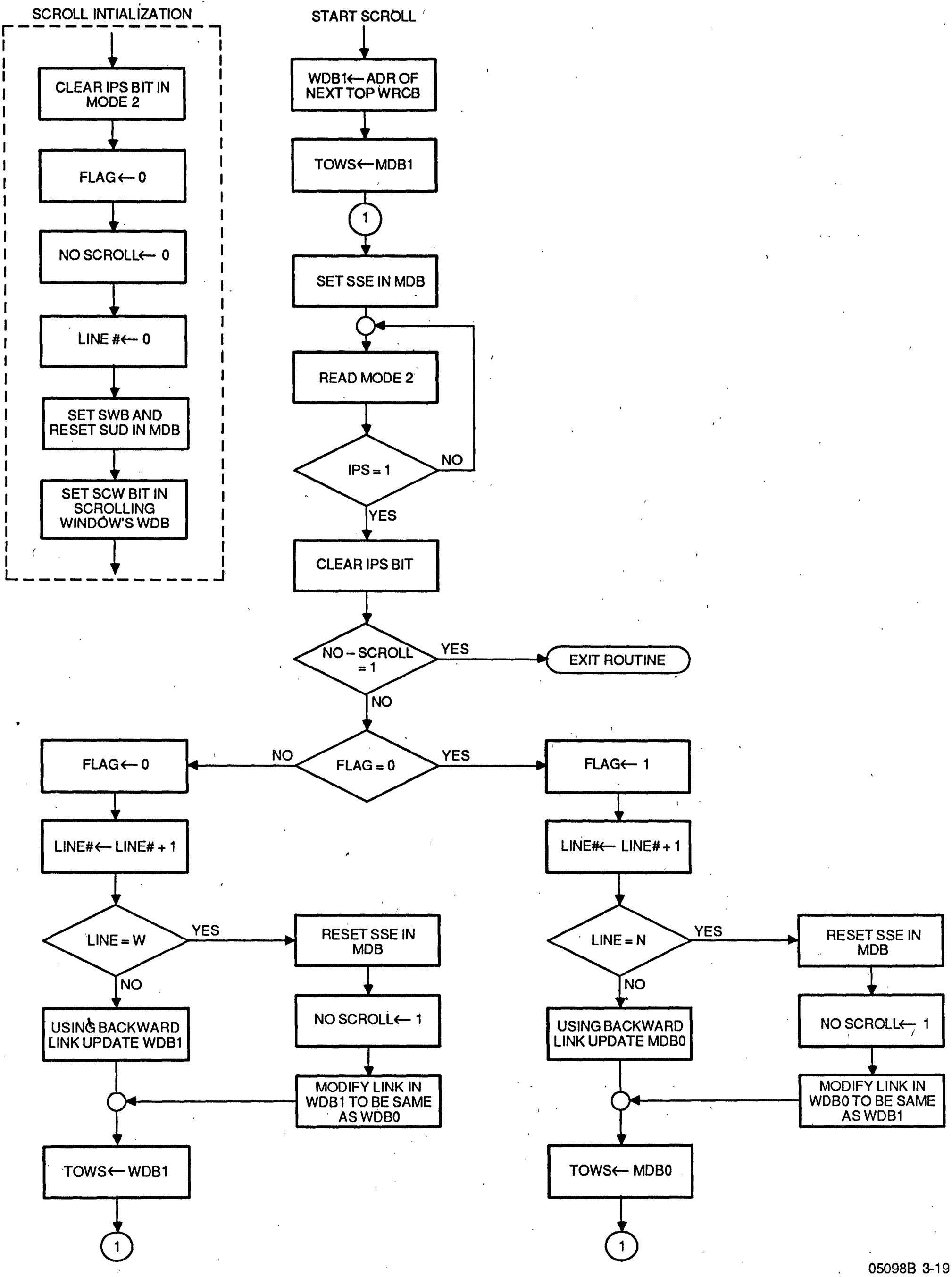
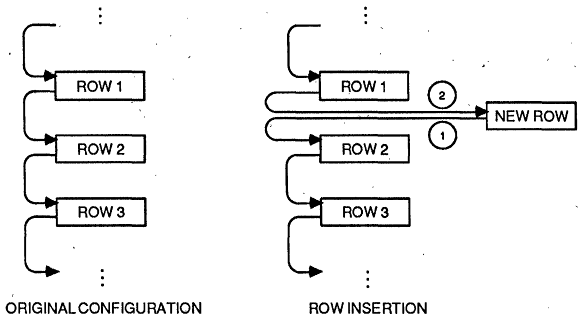
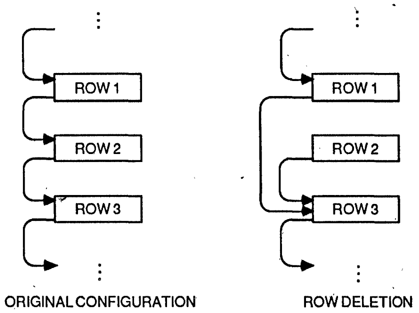


Figure 3-19 Stop scrolling down of a window after N lines



05098B 3-20

Figure 3-20 Pointer manipulation inserts ROW



05098B 3-21

Figure 3-21 Pointer manipulation deletes ROW

CHAPTER 4

VIDEO SYSTEM APPLICATIONS

4.0 INTRODUCTION

This chapter outlines three system applications of the Am8052 and the Am8152A. The first application describes a typical design with 8 pixels per character and a 40 MHz pixel rate. In the second application, the character width is increased to 12 pixels and it will be shown how the 9-bit-wide input of the Am8152A is multiplexed to load the wider character slice. The third application, proportional-spacing, discusses pipelining of the data flow, which becomes necessary at high character clock rates.

4.1 TYPICAL APPLICATIONS

Figure 4.1 shows a non-proportional spacing application operating the video system at 40-MHz pixel rate. The character matrix is 7 x 9 pixels in a character cell of 8 x 14 pixels. The rightmost pixel is blanked. The Character Clock defining the rate of characters being shifted out can be determined by dividing the pixel rate by the horizontal width of the character cell:

$$40\ \text{MHz} / 8 = 5\ \text{MHz}.$$

Since this video system employs only a single Video System Controller (VSC), which does not need to be synchronized to an external dot clock, the internal crystal oscillator can be used. The crystal frequency can be determined as

$$40\ \text{MHz} / 5 = 8\ \text{MHz}.$$

Since the CLK_2 frequency is constant, the $Clock_2$ Divide Ratio inputs ($CLK_2DR<3:0>$) may be hardwired to High or Low, respectively, instead of generating new values on a character-by- character basis as in the case of proportional spacing. Since no trailing blanks are used, TB<1:0> are tied Low. The formula for calculating the appropriate $Clock_2$ Divide Ratio is shown below:

$$N = n + TB + 2$$

N	=	Number of pixels/character
n	=	CLK_2DR programming
TB	=	Number of Trailing Blanks
2	=	adjust range to 2..17 pixels/character

In this example, "n" becomes 8 – 0 – 2 = 6. Since the character matrix is 7-bits wide horizontally, inputs DD7 and DD8 can be grounded. The 256 different characters are addressed by the 8-bit Character Code (usually an ASCII code). The 14 scan lines, per character cell, are addressed by the 4-bit Scan Line Address. Altogether 12 bits are used to select a particular character slice, which implies using an 8K x 8(7)-bit Character Font Generator (usually ROM, PROM, or EPROM).

A 5-MHz CLK_2 translates to a 200 ns character clock period. The following calculation shows how the maximum allowable data access time for the Character Font Generator is determined. The Am8052 strobes out the Character Code (CC<7:0>), and Scan Line Addresses (R<4:0>) with a propagation delay to the Character Clock ($MCLK_2$). The character slice data addressed needs to be valid before the next rising edge of the Character Clock to allow the VSC to latch it. Therefore, the propagation delay of the Am8052 plus the maximum access time of the Character Font Generator plus the set-up time required by the VSC must be less than one character clock period. Assuming the Am8052 propagation delay from $MCLK_2$ to CC and R is 55 ns (6-MHz spec), and $TCLK_2$ to $MCLK_2$ delay is 8 ns, and the set-up time required for the data to $TCLK_2$ is 20 ns, the maximum access time becomes:

$$200\ \text{ns} - 55\ \text{ns} - 8\ \text{ns} - 20\ \text{ns} = 117\ \text{ns}.$$

Am27S43 (4K x 8) PROMs satisfy this requirement (55 ns maximum).

4.2 MULTIPLEXING THE DATA INPUTS

This application features a system of 12-bit-wide, non- proportional spaced characters at 60-MHz dot rate. It is illustrated in Figure 4.2. Similar to Figure 4.1, the on-chip crystal generator can be used to generate the Dot Clock. The crystal frequency is 60 MHz/5 = 12 MHz. The inputs specifying the number of Trailing Blanks to be added are grounded (no Trailing Blanks). Having a non-proportional spaced set of characters means that there is no use for the Trailing Blanks; therefore, their inputs are grounded.

The CLK_2 Divide Ratio inputs are hardwired to High and Low to provide a constant divide ratio. The Dot Clock is divided by 12 to generate the

Character Clock. The inputs are programmed as:

$$12 - 0 - 2 = 10\ (1010_B).$$

Given the 60-MHz dot rate and the 12-pixel-wide character cells the CLK_2 frequency can be calculated as

$$CLK_2 = 60\ MHz / 12 = 5\ MHz.$$

The character clock Period becomes 200 ns. Since the character cell is wider than the data input path of the VSC, the data must be pipelined. With the rising edge of the clock, the right 9 pixels are loaded. DD0 is the rightmost pixel. With the next falling edge of the clock, the VSC latches the left 8 pixels. In this application, only 3 bits are loaded with the second clock edge.

The CRTC outputs the Character Code (CC_{0-7}) and Scan Line Addresses (R_{0-4}) with a propagation delay of 55 ns to the rising edge of $MCLK_2$. The maximum skew between $TCLK_1$ and $MCLK_1$ are 8 ns and 12 ns for rising and falling edges respectively. Similar to the application shown in Figure 4.1, the maximum allowable access time is:

$$200\ ns - 8\ ns - 20\ ns - 55\ ns = 117\ ns.$$

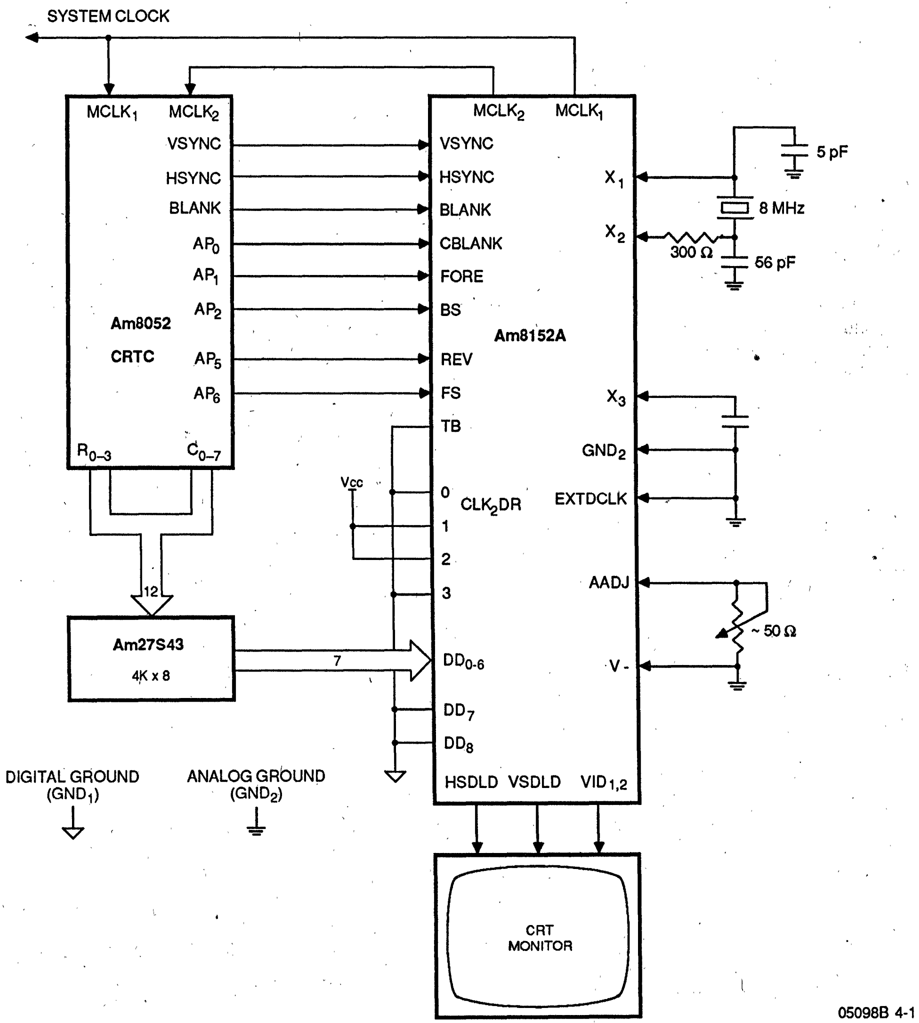


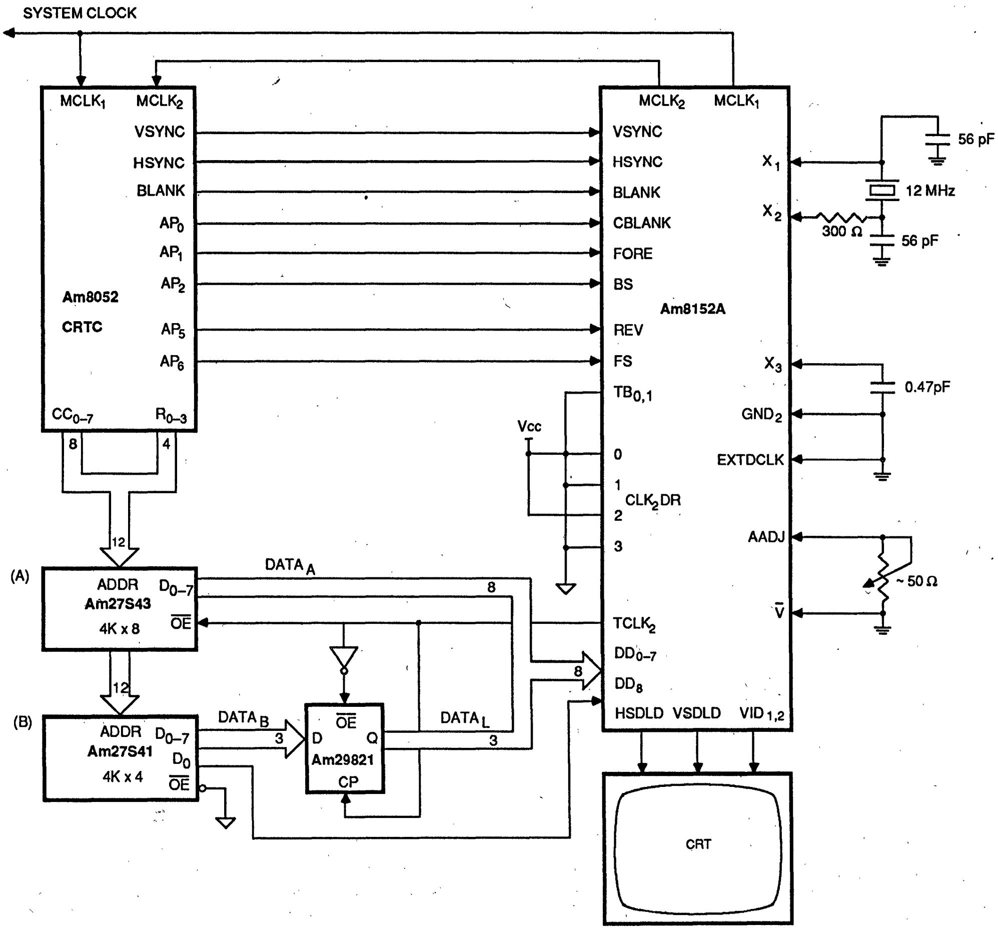
Figure 4-1 Non-proportional Spacing System

Since PROM B has to present the data at the inputs of the register with a set-up time of 2 ns (Am29821 parameter), the access time of PROM B can be calculated as:

$$200\ \text{ns} - 55\ \text{ns} - 8\ \text{ns} - 2\ \text{ns} = 135\ \text{ns}.$$

The multiplexing of the data is as follows:

The CRTC outputs the character and scan line information for the characters synchronously to MCLK$_2$. The Character Code and the Scan Line Address select a particular character slice. Since the VSC expects 9 bits of data on the rising edge of TCLK$_2$, and PROM A supplies only 8 bits, PROM B provides the 9th bit; it is connected to DD$_8$. Enabling PROM A with TCLK$_2$ ensures that the first 8 bits are present at the VSC data inputs prior to the rising edge of TCLK$_2$. PROM B is permanently enabled, therefore, the 9th bit is available at the rising edge of TCLK$_2$ but is ignored on the falling edge. The remaining 3 bits (12-bit character width) are loaded on the falling edge of TCLK$_2$ at which time the Am8052 has already selected the next character. Therefore, the output of PROM B has to be registered (Figure 4.3).



05098B 4-2

Figure 4-2 Multiplexed data path to load wider character slices

4.3 CHARACTER PIPELINING

At high character clock rates, or in proportional-spacing applications, the character data path needs to be pipelined to relax, as much as possible, the access time requirements for the Character Font Generator. Assuming a 8-MHz clock rate and taking the approach of the examples in Figure 4.1 and Figure 4.2 would require an access time of:

125 ns – 8 ns – 45 ns – 20 ns = 52 ns.

The following analysis points out how this access time can be relaxed (Figure 4.4).

The clock to output delay of the Am29821 register is specified at 12 ns. The set-up time is 2 ns. This calculates a worst case access time of:

125 ns – 12 ns – 2 ns = 111 ns.

Pipelining both input and output data gains about 50 ns (Figure 4.5).

If only the input data is pipelined than the requirement becomes:

125 ns – 12 ns – 20 ns = 93 ns.

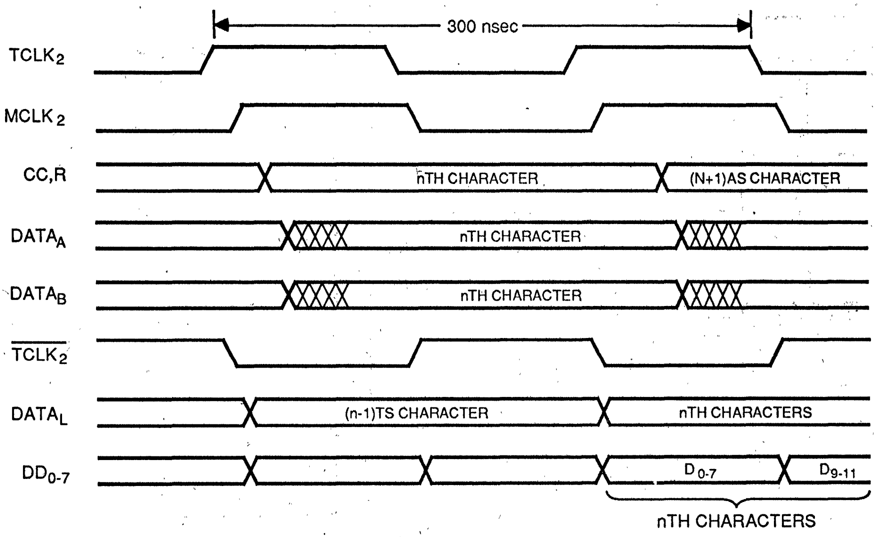
This approach still gains 41 ns.

The CRTC allows programming the skew between Character Code and Attribute output or Control Signal (HSYNC, VSYNC, and BLANK) output. (See Mode Register 1 description in Chapter 2) This skew can be used advantageously in this case by advancing the Character Code and Scan Line Address by one or two CLK_2 cycles so that the rest of the signals do not need to be pipelined externally.

4.4 CHARACTER/SYSTEM CLOCK SYNCHRONIZATION

In proportional-spacing applications, the Character Clock defining the Character Output Rate and the System Clock defining video timing (VSYNC, HSYNC, BLANK) must be synchronized at the left edge of the display in order to avoid a jagged edge. The VSC synchronizes both clocks when the SSEL (Synchronization Select) is tied High. If SSEL is Low, no synchronization occurs.

Synchronization ensures that HSYNC and BLANK change synchronously to CLK_2, resulting in a straight and smooth left border of the display. The right edge of the screen also is straight and



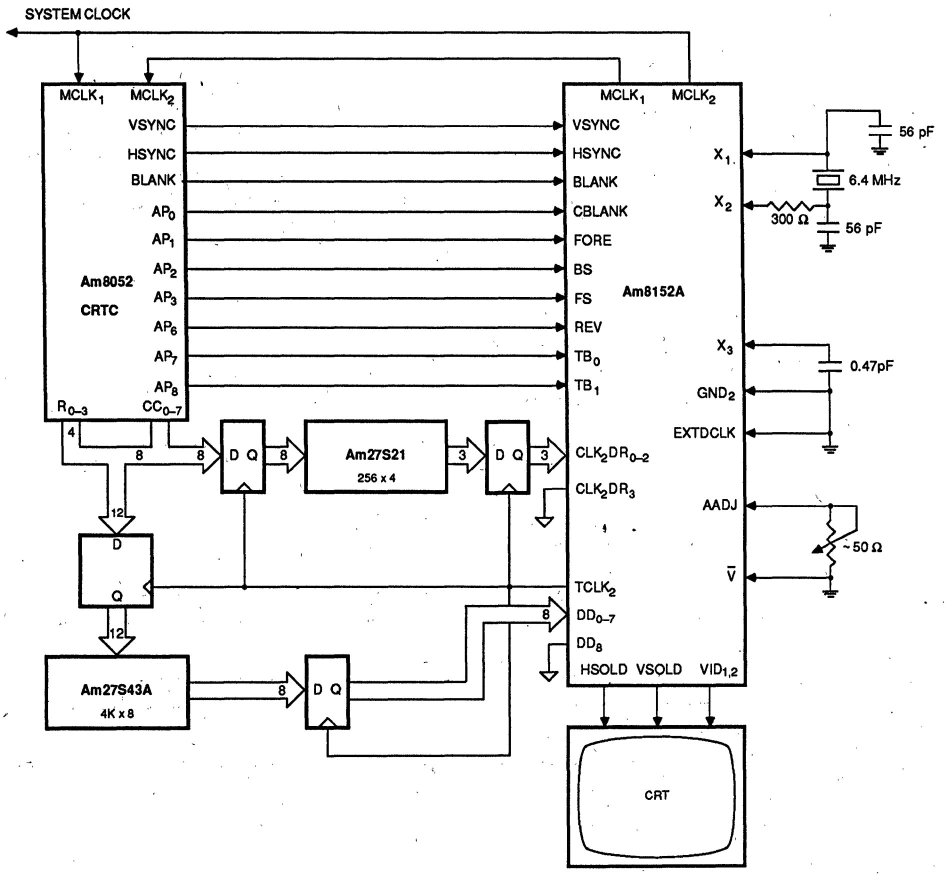
05098B 4-3

Figure 4-3 Multiplexed character data timing

smooth since the width of the display is a multiple of the fixed-rate System Clock (CLK_1). Note, that it is the system designer's responsibility to ensure that the last characters in any line are blank, so a valid character is not truncated due to the asynchronism of CLK_1 and CLK_2 at the end of a scan line.

The synchronization process of CLK_1 and CLK_2 takes place in the beginning of HBLANK. The VSC holds CLK_2 Low for several CLK_1 cycles then toggles in phase to CLK_1 until it recognizes HBLANK going Low (inactive). From then on CLK_2 is generated as controlled by the divide ratio inputs.

Additionally, the VSC delays HSYNC and VSYNC so that they change synchronous with the Video Data (VID_1 and VID_2). The internal delay buffers are clocked by CLK_2 when SSEL is Low, and by CLK_1 when SSEL is High. Since these delays match the video delay when SSEL is Low, these buffers can be used to latch any other video attribute the user might chose to use, in addition to the given attributes (FS, BS, REV, etc.).



05098B 4-4

Figure 4-4 Character pipelining in proportional spacing systems

4.5 CRYSTAL OSCILLATOR LAYOUT

The VSC has two power supplies: a digital power supply (V_{CC1} and GND_1) and an analog power supply (V_{CC2} and GND_2). This split enables the system designer to keep the analog supply as clean as possible. A low-noise analog supply is essential for a reliable operation of the crystal oscillator and the phase-lock-loop (PLL) multiplying the crystal frequency; especially if the operation of the PLL is a direct function of the noise-level on the supply.

The PC-board should be laid out in such a way that the lines from the pins of the VSC to the external capacitors, resistors and crystal are as short as possible. These passive circuits are connected to the analog ground (GND_2).

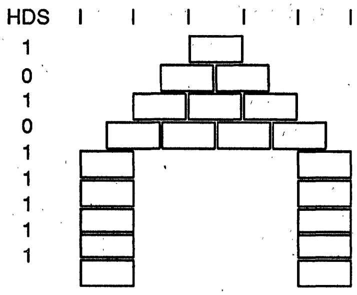
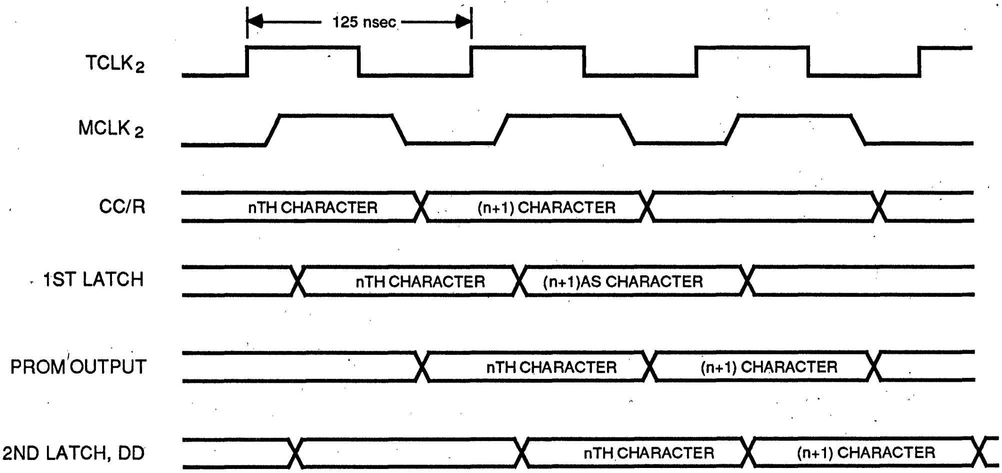


Figure 4-6 Half Dot Shift

4.6 HALF DOT SHIFT WITH THE Am8152A

To increase the display quality, character slices can be shifted half a dot as shown in Figure 4-6. One character font bit enables or disables this feature. This bit is delayed by two D-flip-flops to compensate for the delay in the Am8152A (Figure 4-7). The AND-gates route the output of the Am8152A (VID2) either triggered flip-flop or to the negative edge triggered flip-flop. If Half Dot Shift is activated, the appropriate character slice is shifted half a dot to the left.



05098B 4-5

Figure 4-5 Pipelining Timing Diagram

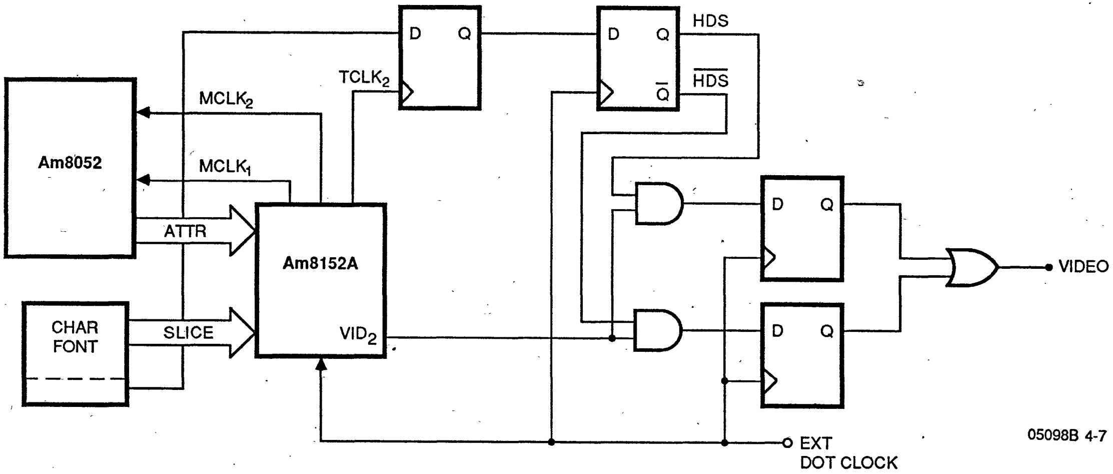


Figure 4-7 Half Dot Shift Diagram

CHAPTER 5

GENERAL APPLICATIONS

Some applications for alphanumeric CRT systems require a dynamically programmable character-set to be able to modify the character font, to add special characters used in some foreign languages, or to provide semi-graphic characters. In this chapter, three application notes for the CRTC are introduced. These applications examples by no mean imply to cover solutions for all types of applications; however, they serve to motivate designers to use their imagination and creativeness in finding the ideal solution for his or her particular application design.

5.1 LOADABLE CHARACTER GENERATOR FOR AN Am8052 SYSTEM

This application note describes a Loadable Character Generator for an Am8052 based alphanumeric CRT system, implementing the unique approach when the Am8052 itself loads the character font. It assumes that the reader is familiar with the Am8052. For background information, refer to Section 2. An alternate approach is described in the chapter on low cost, smart terminals.

There are two basic approaches to the design of a Loadable Character Generator:

(i) The "usual" way of designing a Loadable Character Font Generator (RAM) is to implement it as a dual-port memory where the CPU has direct access. An address multiplexer is then inserted at the Address Bus of the Character Generator (CC_{0-7} and R_{0-4}), connect the output via a bus driver to the System Data Bus, and control both the multiplexer and the driver by arbitration logic. To prevent screen flickering, the Character Generator should only be accessed during horizontal or vertical retrace.

The advantage of this approach is that the character RAM can be read and written directly by the CPU. Also, the Font RAM can be altered rapidly.

The disadvantage is that a large number of TTL support parts is required to build the two-port RAM control logic.

(ii) The second approach utilizes the Am8052 for loading the Character Generator. Most of the pins of the Character Generator are already connected to the Am8052. Only a path to the data bus of the Character Generator must be set up; a few additional TTL devices are needed to implement this feature. The Character Generator information is stored in the linked list.

Advantage of this approach is the small amount of support logic required.

The disadvantage is that more sophisticated software is required to control the loading process, and the character font cannot be read back.

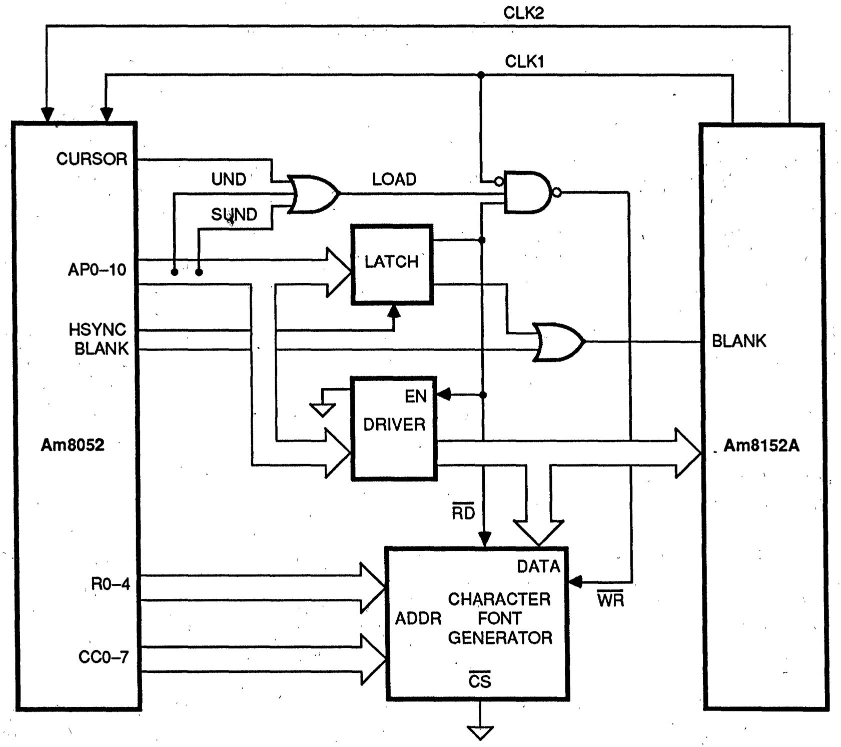
This application note focuses on the second approach, utilizing the Am8052 (Figure 5-1).

A blank part of the screen is utilized to load the Character Font Generator. In the initialization phase, this space can be the entire screen; during display time, it may be a blank space at the bottom of the screen. The number of characters per frame which can be reloaded is directly proportional to the space allocated.

The screen is divided into two parts (Figure 5-2): the visible part of the screen displays the normal text; the invisible, lower part hides the rows used to load the Character Font Generator. In this example, there are 18 scan lines at the bottom of the screen that are used to load a character box of 7 X 9 pixels. These scan lines are located between normal-vertical-blank active and vertical-sync active. The rows are hidden by setting a user-definable Row Attribute Bit that externally blanks the video. Each character of the rows invokes an attribute word. As in the usual display mode, the character code addresses a character box in the Character Generator. However, the purpose of the attribute word changes; now, it contains the data of the character slice to be loaded.

Detailed Description

The Am8052 provides user definable data during horizontal retrace. This data is stored as a row attribute word in the Row Redefinition Block. It can be latched with the falling edge of HSYNC. In this design, two bits are used to control the load operation. One bit blanks the screen to hide the rows containing the Character Generator data; the second bit disables the Read input of the Character Generator and enables the attribute bus driver. The bus driver connects the attribute port



05098B 5-1

Figure 5-1 Using the Am8052 to Load the Character Generator

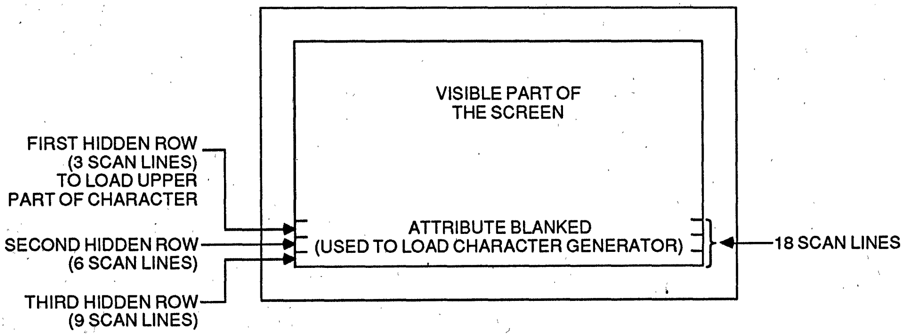

VISIBLE PART OF THE SCREEN
FIRST HIDDEN ROW (3 SCAN LINES) TO LOAD UPPER PART OF CHARACTER
SECOND HIDDEN ROW (6 SCAN LINES)
THIRD HIDDEN ROW (9 SCAN LINES)
ATTRIBUTE BLANKED (USED TO LOAD CHARACTER GENERATOR)
18 SCAN LINES

Figure 5-2 Screen Layout

to the data bus of Character Generator. Since this design assumes a 7 X 9 character box, only 7 bits of the attribute are connected to the Character Generator; the 8th bit is grounded at the input of the driver. Any character fonts size can be supported in order to accommodate design changes.

Two bits of the attribute port and the cursor output are used to enable the loading of specific character slices. These 3 bits have a common feature. The character part where these attributes are active is programmble on a character row basis. "Underline" and "Shifted Underline" are active during one scan line in the character cell. The scan line number, where these two attributes are active, is specified in the Row Redefinition Block. The values can be changed on a row basis by specifying a Row Redefinition Block for each row. "Cursor", is an attribute which is active during part of a character. "Start" and "End" values for this attribute is specified in the Row Redefinition Block. If these values are identical, the attribute is active only during one programmed scan line (see Tables 1 and 2).

The 3 attributes determine which slice of the selected character is loaded. The attribute string layout of Figure 5-4 assumes that the Row Redefinition Blocks contain the values of Tables 1 and 2. Each attribute word activates one of these 3 attribute bits to select a specific character slice. The character slice is loaded with the 7 bit value contained in the attribute word. Three consecutive attribute words in which each activates a different attribute bit (Figure 5-4) so that the upper 3 slices are reloaded in the end. In the next row, the row attributes are redefined to enable loading of the middle part of the characters. A third row loads the remaining lower part.

When one of the 3 attribute output pins is activated by the attribute word, and when a latched row attribute bit disables Read, then the Character Generator receives a Write pulse to Strobe in the character slice (Figure 5-3)

Seven attribute bits must be programmed in the Attribute Redefinition Register as user-definable attributes. In this design, a maximum of 44 characters-per-frame can be reprogrammed. This number is determined by:

- The length of the row buffers (132 characters)
- 18 scan lines are used for loading the Character Generator
- Each character has 9 slices (9 character positions in the row buffer).

Modifications to support character font generators wider than 7 bits:

Loading can be done in steps. A character box which is 12 pixels wide can be loaded in two steps, each loading 6 pixels. The 7th bit of the attribute now selects the left or right part. An alternative is to use a latched attribute bit (an output of the latch in Figure 5-1) to select the parts. Note that these attributes are constant in the entire row, therefore, different parts cannot be loaded if a latched attribute is used.

Scan line count can be reduced when less attributes are used to select character slices. Note that the minimum scan line count of a row is determined by the time the CRTC needs to fill the row buffer.

An arbitrary number of attributes ("n") are utilized to select slices. The first row loads the upper "n" character slices and has a minimum scan line count of "n." The second row loads the next "n" slices and has a scan line count of 2 • n. A third row loads

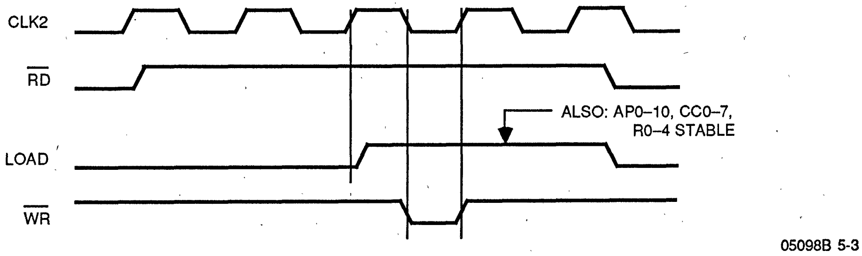


Figure 5-3 Write Timing

subsequent "n" slices and has 3 • n scan lines. In this example of a 7 X 9 character box and 3 slice attributes, 2 rows are needed to load all 10 slices. The first row loads the upper 3 slices and contains 3 scan lines, the second row has 4 scan lines and loads the middle 3 slices. The third row has 9 scan lines and loads the lower 3 slices.

The "old" vertical blank active time must be reprogrammed to allocate space for the character-load rows. An attribute bit will blank this part of the screen so that there is no visually detectable difference on screen.

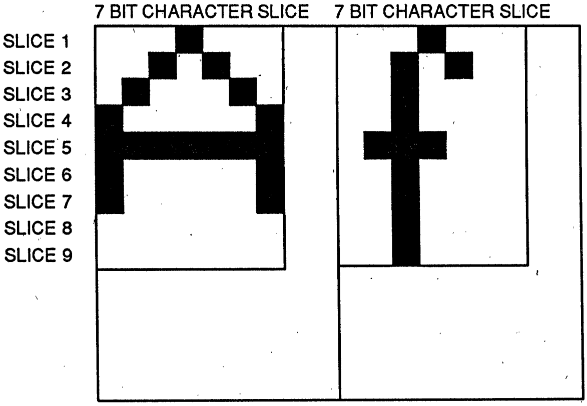
Figure 5-5 shows two 7 X 9 character cells containing an "A" and a "F". Figure 5-6 shows parts of the linked list data strings specifying the data to load the character fonts of these characters.

A 7 X 9 character set of 256 characters fits into an 8K X 8 RAM. The maximum access time depends on the resolution of the display (high resolution => about 60 ns).

CURSOR	1	0	0	1	0	0	1	0	0	...
UND	0	1	0	0	1	0	0	1	0	...
SUND	0	0	1	0	0	1	0	0	1	...
ATTRIBUTE BITS	SLICE #1	SLICE #2	SLICE #3	SLICE #1	SLICE #2	SLICE #3	SLICE #1	SLICE #2	SLICE #3	...
CHARACTER CODE	CHARAC #1	CHARAC #1	CHARAC #1	CHARAC #2	CHARAC #2	CHARAC #2	CHARAC #3	CHARAC #3	CHARAC #3	...

05098B 5-4

Figure 5-4 Character and Attribute List



05098B 5-5

Figure 5-5 7 x 9 Character Box

Table 1 Parameters for Row Redefinition Block of 1st Row

TSLC	=	4	The first row loads only the upper five character slices
NCS	=	0	Normal character start on scan l ine 0
NCE	=	4	Normal character end on scan line 4
CURS	=	0	Cursor start
CURE	=	0	Cursor end
UND	=	1	Underline active on line 1
SUND	=	2	Shifted underline active on line 2

Table 2 Parameters of Row Redefinition Block of 2nd Row

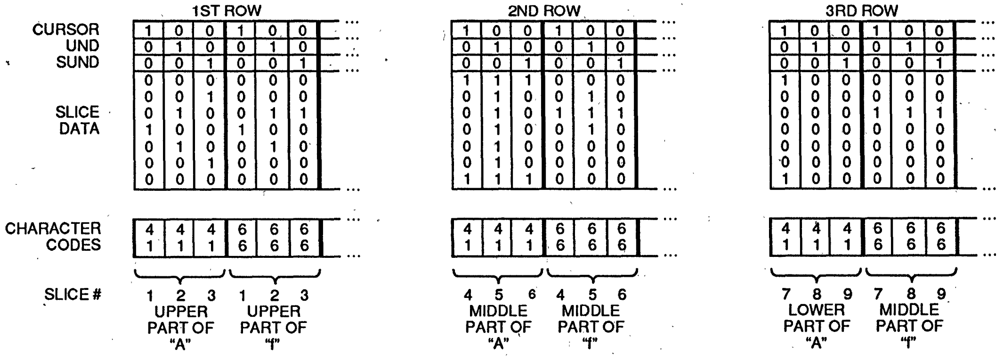
TSLC	=	9	The second row loads the lower five character slices (scanning the first five lines a second time is an unavoidable overhead)
NCS	=	0	Character position starts at 0 and
NCE	=	9	ends at 9
CURS	=	5	All other values are incremented by 5 to
CURE	=	5	access lower 5 scan lines
UND	=	6	
SUND	=	7	

5.2 HORIZONTAL SMOOTH SCROLL

Vertical screen scrolling on standard terminals is done by replacing the text line by line; the text appears to jump up or down the screen. A more desirable and ergonomic approach is to smooth scroll the text. The Am8052 alphanumeric CRT controller (CRTC) can achieve this effect by replacing the scrolled line on a scan line basis. The text moves in steps of line partitions (scan lines). This produce smaller jumps and is almost unnoticeable to the viewer; it appears to be a continuous, smooth, upward or downward movement of the text on screen. The scrolling itself is executed without CPU interventions.

In applications that involve displaying text running off the screen horizontally requires scrolling the text accordingly. Once the user has experienced vertical smooth scroll, the demand for horizontal smooth scroll will come naturally. Similar to vertical smooth scroll, horizontal smooth scroll can be done by replacing characters on a pixel basis. Although the CRTC does not have a built-in mechanism to control horizontal smooth scrolling, this application note provides some ideas for a practical implementation. External MSI logic and CPU interventions are required to control the scrolling process.

The basic idea behind this scrolling technique is to place a dummy character in front of the line. This character is made invisible by delaying the horizontal BLANK with external logic. The entire line is then moved by modifying the width of this dummy character, utilizing the proportional character capability of the Am8152A Video System Controller (VSC). The blank delay covers the entire dummy character when it is programmed for full width. By reducing the width of this character, the first visible character moves left and gets partially covered. Characters seem to enter the screen on the right side and leave on the left side. Figure 5-7 diagrams the process.



1ST ROW

CURSOR	1	0	0	1	0	0	...
UND	0	1	0	0	1	0	...
SUND	0	0	1	0	0	1	...
SLICE DATA	0	0	0	0	0	0	
	0	0	1	0	0	0	
	0	1	0	0	1	1	
	1	0	0	1	0	0	
	0	1	0	0	1	0	
	0	0	1	0	0	0	
	0	0	0	0	0	0	...
CHARACTER CODES	41	41	41	66	66	66	...
SLICE #	1	2	3	1	2	3	
	UPPER PART OF "A"			UPPER PART OF "f"			

2ND ROW

CURSOR	1	0	0	1	0	0	...
UND	0	1	0	0	1	0	...
SUND	0	0	1	0	0	1	...
SLICE DATA	1	1	1	0	0	0	
	0	1	0	0	1	0	
	0	1	0	1	1	1	
	0	1	0	0	1	0	
	0	1	0	0	0	0	
	0	1	0	0	0	0	
	1	1	1	0	0	0	...
CHARACTER CODES	41	41	41	66	66	66	...
SLICE #	4	5	6	4	5	6	
	MIDDLE PART OF "A"			MIDDLE PART OF "f"			

3RD ROW

CURSOR	1	0	0	1	0	0	...
UND	0	1	0	0	1	0	...
SUND	0	0	1	0	0	1	...
SLICE DATA	1	0	0	0	0	0	
	0	0	0	0	0	0	
	0	0	0	1	1	1	
	0	0	0	0	0	0	
	0	0	0	0	0	0	
	0	0	0	0	0	0	
	1	0	0	0	0	0	...
CHARACTER CODES	41	41	41	66	66	66	...
SLICE #	7	8	9	7	8	9	
	LOWER PART OF "A"			MIDDLE PART OF "f"			

05098B 5-6

Figure 5-6 Character Code and Attribute sequence to load "A" and "f"

Detailed Description:

Here it assumes a non-proportional spacing environment with a character width of 8 pixels, and a dummy character width of 10 pixels; there is no restriction to these values. External logic hides the dummy character and the first visible character by delaying BLANK (10 pixels). The delayed BLANK masks off the serial video stream put out by the VSC (Figure 5-8).

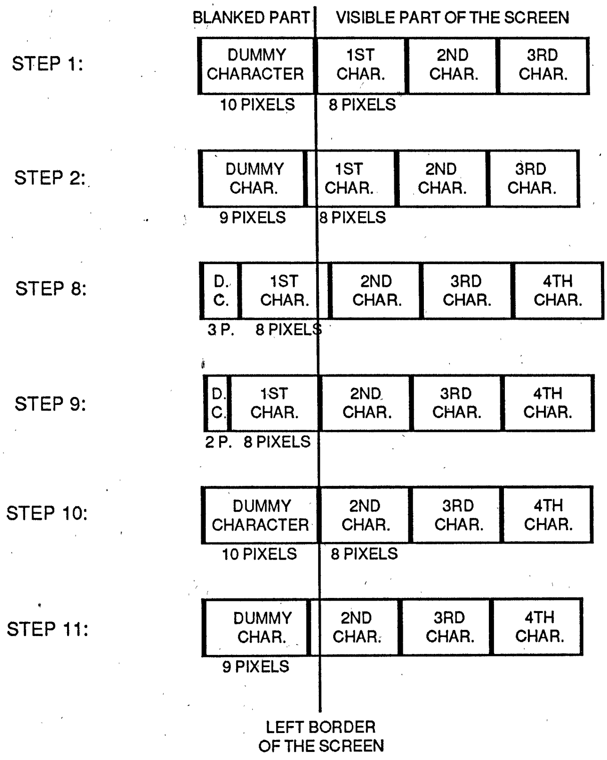
By reducing the character width of the dummy character from 10 to 2 pixels in 8 steps, the leftmost character is moved out. The dummy character has to be wider than the widest visible character in order to hide a dummy character (2 pixels minimum width) as well as the leftmost character in the blanked space (Step 9 of Figure 5-7). The width of the dummy character can be controlled by using several methods described in the following paragraphs. Step 9 of Figure 5-7 is optional, it is shown to clarify the entire process. The user can expand the dummy character to its full size (10 pixels) when only one pixel of the leftmost character is left visible (Step 8 to Step 10).

Horizontal smooth-scrolling can be made frame-synchronous by incorporating the Vertical Interrupt of the CRTC. This interrupt is issued once per frame. The scroll rate can range from as low as one pixel per several frames to several pixels per frame. This is similar to the programmable scroll rate for vertical smooth scrolling. For additional information refer to Section 2 of this handbook.

External Blanking

One of the criterion for this application is to find a simple way of delaying BLANK to the appropriate number of pixels (example 10) to hide the dummy character.

A practical approach is to delay BLANK by feeding it through two D-flip-flops clocked by the system clock CLK1. This requires CLK1 period to be



05098B 5-7

Figure 5-7

larger than the character clock period (CLK2) and that CLK1 has the appropriate pixel width (Figures 5-8 and 5-9).

Another approach is to use a counter to delay the BLANK for the appropriate number of pixels. The counter is to be clocked by the DOT Clock and enabled by the first edge of CLK1 or CLK2, after BLANK active. The problem with this approach is that an external DOT Clock must be available. Most applications make use of the built-in PLL of the VSC and consequently an external DOT Clock in unavailable.

Width Control

The width of the dummy character can be modified by using the proportional character display capabilities of the VSC. In proportional character

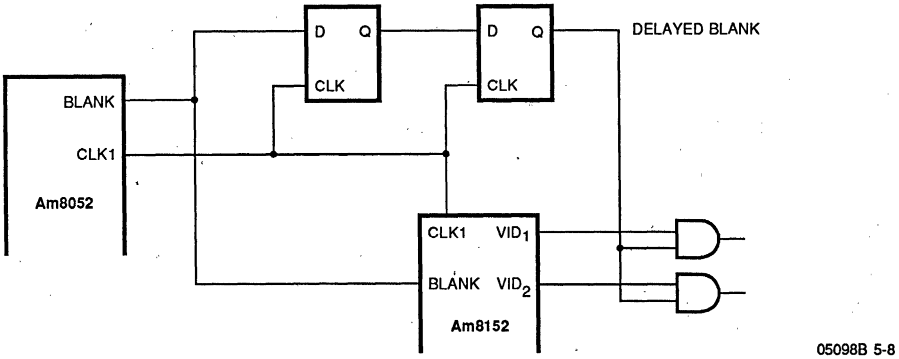


Figure 5-8 Delaying BLANK

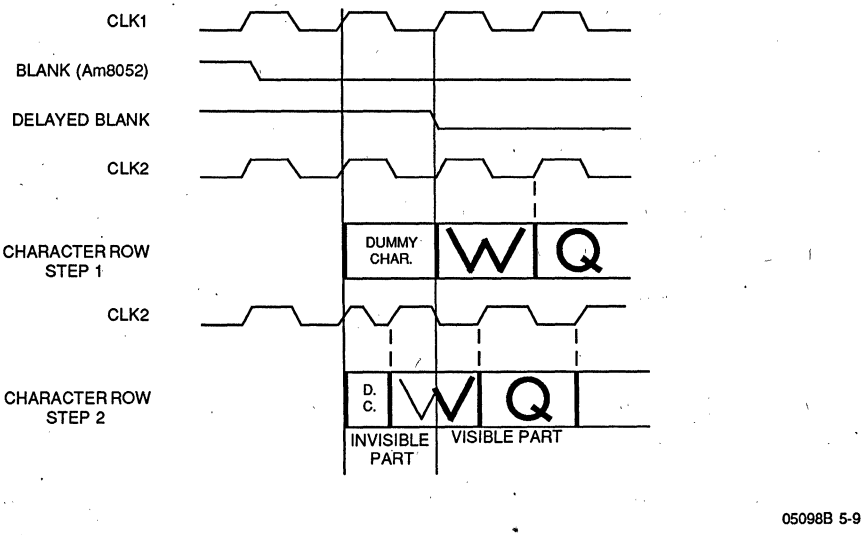


Figure 5-9 Delayed BLANK Timing

applications, where the character font generator already contains a set of characters with widths between 2 and 10, no special hardware is necessary (Figure 5-10). The CPU changes the dummy character for each scrolling step. The new character has either a decreased or increased width, depending on the scrolling direction. Decreasing the width causes a left scroll; increasing the width causes a right scroll. The row data list has to be updated after scrolling an entire character.

In proportional character applications, the user has to keep track of the width of the.character inserted or deleted when updating the row data list. The modification of the width of the dummy character is a function of the width of the inserted or deleted character.

In applications with a fixed character-width, it might be practical to add a character-font width generator to implement a character set of different widths for the dummy character.

Another approach in controlling the width of the dummy character is to include the bias of the row in the Row Attribute Word. This attribute word is put out during horizontal retrace, and can be latched by HSYNC (Figure 5-11). The character attribute AP9 is only activated during scanning the dummy character to switch the multiplexer. The multiplexer normally guides the Character Font Generator output to the VSC CLK2-Divider inputs. Only during scanning the dummy character the 4-bit width stored in the Row Attribute Word is used. This approach is advantageous when the linked-list contains only one Row Redefinition Block common to all rows. In that case, the CPU only has to update one word to move the screen horizontally. In the other approaches the CPU has to update one character per row.

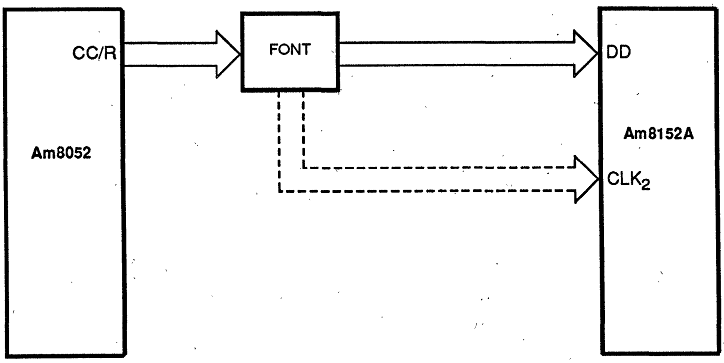
5.3 BIT-MAPPED GRAPHICS WITH Am8052

This section outlines a second approach in using the Am8052 for bit-mapped graphic. The design discussed in the reprints of magazine articles in the appendices dealt with graphic information stored in a special x-y addressed display memory. The linked-list interpreted by the Am8052 provides only the address information and not the display information itself. The approach presented in this section involves linked-list providing all display information including the pixel data. The software-oriented implementation requires only one external 8-bit multiplexer (minimum configuration) whereas the hardware oriented implementation of the design outlined in the magazine article requires multiplexers, a separate display memory including refresh circuitry, and bus arbitration logic to let either the host CPU or the CRTC access the display memory. The advantages of this scheme over the design shown in the magazine articles can be summarized as follows:

- less external circuitry
- no dual-ported display memory, pixel and text data is stored in system memory

However, this approach has some trade-off and limitations compared to the design in the last chapter.

- mixed text and graphics only on horizontally split screens (entire scan lines are allocated for either text or graphics)



05098B 5-10

Figure 5-10 Variable Character Widths

- heavily increased system bus utilization (up to 100%) when displaying graphics, therefore dual-bus architecture appropriate

Both designs provide the same resolution for text and graphics (same dot clock). Both designs take advantage of the linked-list architecture of the CRTC system and thereby allow easy and fast page swapping, block moves. Further on, the graphic page can be vertically smooth scrolled in both designs.

Pixel Generation

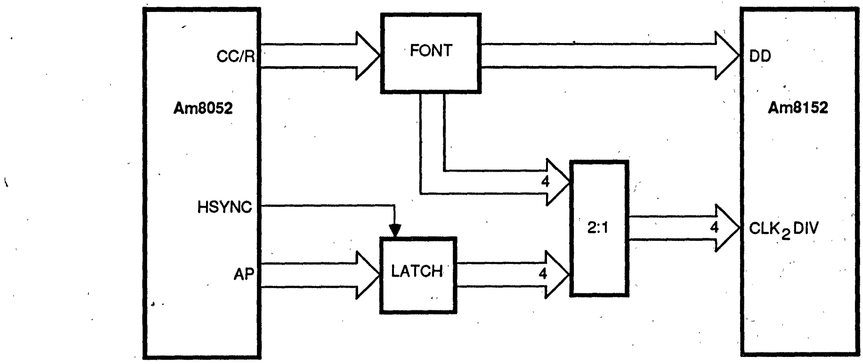
A standard CRT controller strobes out the character code (usually the ASCII-code for the character to be displayed) on a character clock basis. This character code and the scan line address select a particular character slice address in the character font generator. The character font generator then provides the character slice data which is serialized by the video shift register.

In this bit-mapped graphic approach the character font generator is bypassed and the character code is shifted out directly. Since the character code can have any 8-bit pattern, it can define any slice of 8 pixels. Since subsequent scan lines on bit-graphic displays are usually unique bit patterns, each scan line is described by its own sequence of character codes. This means that, in bit-mapped graphic mode, character rows contain only one scan line compared to a scan line count of 8..16 in text mode. Consequently, the bus utilization of the CRTC increases drastically. In fact, the screen resolution is limited by the data transfer capability of the system bus.

In bit-mapped graphic mode the Total Scan Line Count (TSLC) is set to 00_H in the Row Redefinition Block (RRB) for the first graphic scan line. Additionally, one bit of the 10-bit row attribute switches the multiplexers to graphic mode. This row and all succeeding rows will maintain that attribute until another RRB is invoked by the linked-list of Row Control Blocks. In this fashion alphanumeric and bit-mapped presentations can be intermixed on the display device.

Figure 5-12 shows the linked-list data structure upon which this application is based. For the rows/scan lines defined to be bit-mapped, the hardware will be made to display the 8 pixels per character slice directly out of the CRTC instead of using a character font memory as an indirect look-up mechanism. This switching mechanism is implemented with the row attribute information normally outputted by the CRTC during horizontal retrace. In this application only one bit is used to differentiate between text and bit-mapped graphic mode. The other bits can be used for other purposes such as implementing a soft loadable character font generator.

The major design consideration is that the CRTC's on-chip DMA controller is given enough bus time to complete loading the row buffers contained in the chip before the information is displayed. The CRTC must be able to load one character row in less than a horizontal SYNC cycle (one scan line). In the limit, this can take all of the available bus time and would, therefore, lock out the host CPU from processing. For this reason, it is expected that only small portions of the total display will be bit-mapped such as in business applications to display small charts or graphs.



05098B 5-11

Figure 5-11

To minimize the bus utilization of the CRTC the linked-list describing the graphic should be straightforward: no windows and no segmentation.

System Performance

To improve the system performance a dual-bus architecture may be implemented. The display information is stored in local memory shared by the host CPU and the CRTC. Additionally, the CPU has system memory to perform the other tasks. In this scheme the host CPU is only slowed down when it actually accesses the display data while the CRTC still uses nearly the entire bus bandwidth provided by the local memory.

To calculate the DMA time for a row four factors must be considered. The performance data is based on the 8-MHz CRTC.

- Each DMA cycle takes 3 ticks of the CLK1 clock assuming operating without Wait states. A bus cycle therefore will be 3/8 Mhz or 375 ns. This implies, using a transparent address latch, that a total of Parameter 4 + Parameter 42 = 30 + 185 = 215 ns is the maximum access time to system/local memory that will be used for bit-mapped data.

Each row's data consists of:

- Row Control Block (RCB) information (7 words or bus cycles)
- The data to be displayed (Two bytes per bus cycle)
- Any attributes that the data invokes (one attribute per bus cycle)

The performance calculations consider three different resolutions for the bit-mapped portion of the display: 512, 768, and 1024 dots horizontally. The former and latter represent low and high end applications; the middle resolution is typical for many present CRT systems and fits into the hardware of the CRTC in a particularly convenient way.

For all of these screen resolutions many common considerations will first be discussed. To simplify both the software which generates the bit-map data and to optimize the bus utilization, it is desirable to place all of the data within one contiguous 64K segment of CPU address space. In terms of the CRTC this means that the upper address does not need to be updated, eliminating Am8052 Bus Master Write cycles and thereby saving bus time. For this same reason it is desirable to have all of the RCB's for the rows of the bit-map in this same address space.

To minimize the bus request and bus release overhead due to handshaking involved it is desirable to have the DMA burst as long as possible. The maximum length for a DMA burst is one character row. It is programmed when the Burst Space value in the Burst Register is set to 00_H.

Since character attributes are not used in graphic mode it is desirable to turn the attribute fetches off. (see Attribute Flag Register).

A word of explanation is appropriate at this point concerning the values to be put into the Row Control Block word RA_2. The "HIDDEN #" and the "VISIBLE #" are used by the DMA to ascertain the maximum number of characters to be fetched into the internal row buffers. For each segment of a row, the DMA will fetch a number of characters equivalent to the sum of these two parameters. In graphic mode "HIDDEN #" should be set to zero and "VISIBLE #" to 64, 96, or 128 for screen resolutions of 512, 768, or 1024, respectively. The remaining row buffer entries are filled with the programmed fill code.

Timing Calculations

The DMA must fetch the control information and character data for graphic row in less than the horizontal scan time. The number of DMA cycles per row can be determined as:

N = R / 16 + C + B
N = number of DMA cycles per row
R = screen resolution in pixels
C = 7 words for Row Control Block
B = 7 bus cycles for bus exchange and Idle DMA Cycles

The data (character string) must be word aligned. The CRTC takes additional time internally to fill in the row buffer with the default data byte specified by the MDB's character fill code. Internally each row buffer has 132 entries. However, for screen size equal to or less than 96 characters per row, the SLIM bit in Mode Register 1 may be set to reduce the time taken to do the fill operation. This "magic number" was used as the basis for the medium resolution selection to reduce the required fill time to zero. The time to fill the remaining part of a row it takes a system clock (CLK_1) cycles per fill character.

The Maximum horizontal frequency supported by each of the three resolutions is as follows (Am8052 at 8 MHz):

512 pixels/line

$[512/16 + 14] \bullet 3 + (96 - (512/8))$
$= \ [32 + 14] \bullet 3 + (96 - 64) = 138 + 32$
$= \ 170$ ticks of CLK1 = 0.021 ms
$F_{max} = 47$ kHz

768 pixels/line

$[768/16 + 14] \bullet 3 + 0 = [48 + 14] \bullet 3$
$= \ 186$ ticks of CLK1 = 0.023 ms
$F_{max} = 43$ kHz

1024 pixels/line

$[1024/16 + 14] \bullet 3 + (132 - (1024/8))$
$= \ [64 + 14] \bullet 3 + (132 - 128) = 234 + 4$
$= \ 238$ ticks of CLK1 = 0.029 ms
$F_{max} = 34$ kHz

Hardware Implementation

A latch (Figure 5-11) stores the row attribute data the CRTC outputs during horizontal retrace. The Am29841 latch is ideal for this purpose as it contains 10 bits worth of storage in the convenient 24-pin slim package and has the correct polarity of clock.

A multiplexer feeds either the 8-bit character code (graphic mode) or the character slice data provided by the character font generator (text mode) to the parallel input of the Video Shift Register.

Another Multiplexer selects the character width from the character font generator (as for proportionally-spaced characters) or is set to 0110_B to indicate eight pixels per character clock when in graphic mode.

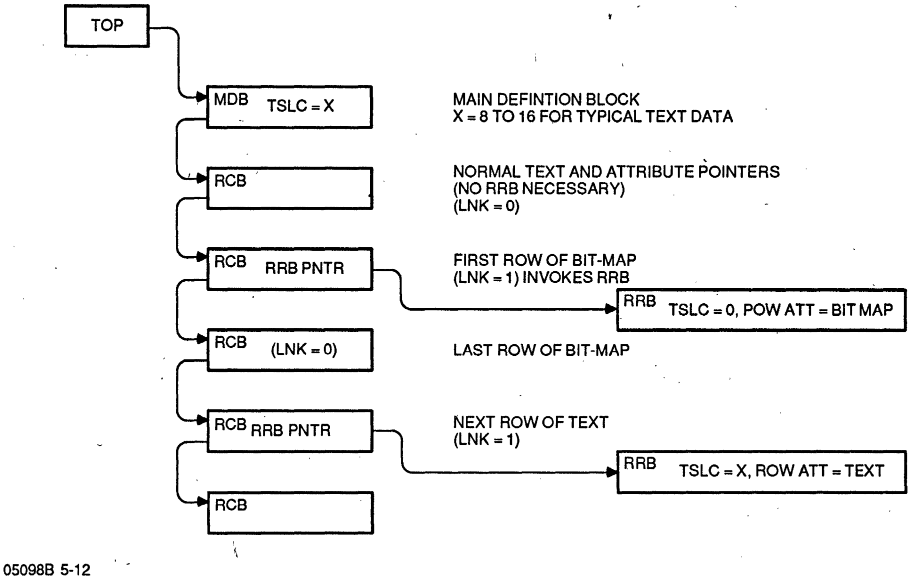


Figure 5-12 "Linked List"

CHAPTER 6

Am8052 BUS INTERFACE GUIDE

6.0 INTRODUCTION

The Am8052 is a general-purpose controller for raster scan CRT displays. Its link-oriented data manipulation provides sophisticated text display without imposing undue overhead on the host CPU. The versatility of this device covers a wide range of applications from medium performance up to very-high performance displays.

A wide variety of systems will be able to take advantage of its features, turning them into powerful display controllers with a minimum of chip count. This application note covers the area in a system outlined in Figure 6-1. It should provide designers with application hints and information on how to interface the device to some of the popular CPUs.

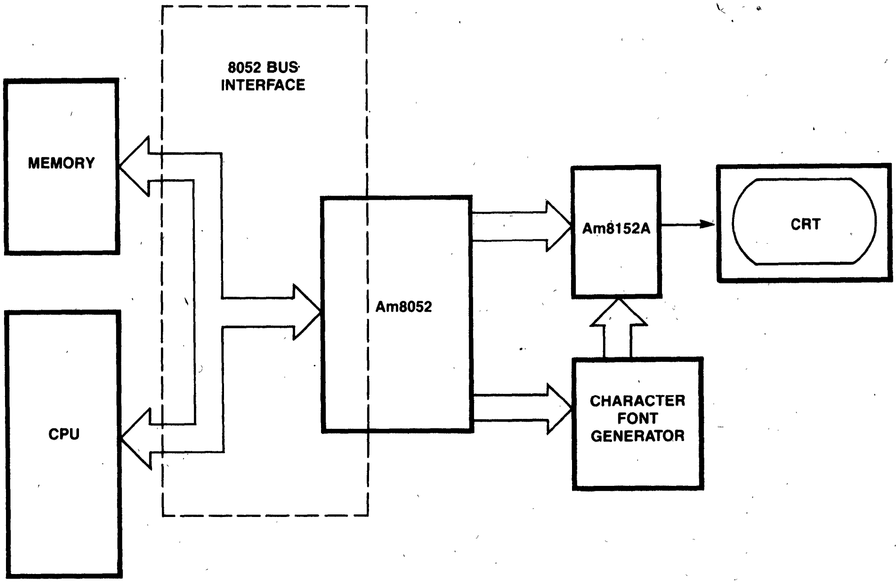
6.1 PERFORMANCE DECISIONS

When designing a display subsystem, the system designer makes multiple decisions to acheive the most cost-effective design. The designer finds the best compromise between performance and cost; the cost mainly consists of hardware/software development and manufacturing. The following shows the trade-off between software development cost and hardware cost.

The basic factors that influence the performance of a display system are:

1. Single/dual bus architecture
2. System clock rate
3. Number of wait states
4. DMA burst length
5. Full/reduced attribute fetches

The hardware designer defines the first three factors. The fourth factor is determined by system constraints such as real-time response time or multi-master bus sharing. The fifth is set by the software designer. The demand attribute fetch feature of the Am8052 can be used to reduce bus traffic by about 50%.



06178A 6-1

Figure 6-1 8052 Bus Interface

How is system performance measured? First of all, system performance is defined here as the response time of tasks executed by the local intelligence. It is assumed, also, that this response time is directly proportional to the remaining bus bandwidth in the CPU. Therefore, parameters such as Wait States can be very important in the determination of system performance.

The various factors affecting system performance are analyzed in the following.

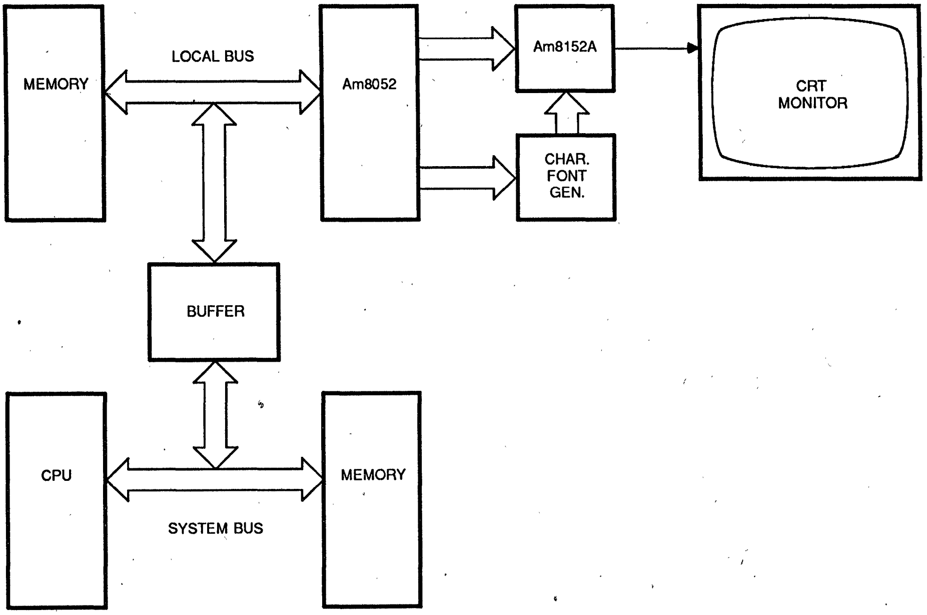
Single/Dual Bus Architecture

The single most important decision the system designer makes is to implement either a single or dual bus architecture.

In the single bus architecture, (Figure 6-1) the CPU, the system memory, and the peripheral devices are interfaced via a single bus, the System Bus. With the Am8052, all display information are stored in the system memory and the Am8052 self-loads this data via the system bus. Consequently, the more data the Am8052 transfers, the smaller the CPU bus bandwidth and lower performance. However, this is the simplest approach and requires no additional hardware.

The the dual bus architecture (Figure 6-2) is implemented in higher performance systems where peripheral devices do not claim a share of the bus bandwidth. Each peripheral device has its local memory and interfaces via its local bus. The Am8052 stores all display data in this local memory. Thus, the self-load no longer burdens the System Bus and Am8052 bus traffic becomes insignificant. This set-up does not affect the overall system performance.

When interfacing the Am8052 to synchronous buses, performance can be increased if the on-chip (Am8052) bus arbitration logic is not used. Instead, an external, synchronous arbitration logic is used to arbitrate the System Bus on a cycle-by-cycle basis. In this mode, BAI is tied Low to allow the Am8052 to perform its transaction at any time.



05098B 6-2

Figure 6-2 Dual-Bus Architecture

An active $\overline{\text{DS}}$ (Data Strobe) is treated as an cycle request. $\overline{\text{WAIT}}$ is pulled Low for as long as it is necessary to hold the Am8052. Upon release of $\overline{\text{WAIT}}$, the actual bus cycle is performed.

System Clock Rate

The Am8052 was originally designed as a Z8000 peripheral, one that has three clocks per machine cycle; this means, performance-wise, a 6MHz Am8052 can cope with an 8MHz 8086, or 68000, or one of the MOS microprocessors that operates on four clock cycles per machine cycle. It is obvious, therefore, that, if the clock rate of the Am8052 is high, the Am8052 requires less of the System Bus bandwidth and gives a higher performance.

In order to optimize the system performance with the Am8052, the CPU should be operated asynchronous to the Am8052. However, since some dynamic memory controller operate synchronously to the System Clock, the design should be simplified to operate both the CPU and the Am8052 synchronously. The disadvantage of this approach is that it requires a faster Am8052.

Wait States

A single Wait State increases, by 33%, the bus bandwidth used by the Am8052. The two examples in the following show cases in which whether or not Wait States are inserted made an important difference.

In the first example, the Am8052 occupies 6% of the bus bandwidth. Inserting a single Wait State raises it to 8%, two Wait States raises it to 10%. The overall system performance is basically not affected.

Example 1

	Am8052	DMA	CPU	Relative Performance
no Wait State	6%	–	94%	1.00
1 Wait State	8%	–	92%	0.98
2 Wait States	10%	–	90%	0.96

The difference would be drastically increased if the Am8052 occupies a more significant share of the bus bandwidth and other DMA devices are also taking their share of it. The following table shows the difference in relative performance when DMA devices are involved.

Example 2

	Am8052	DMA	CPU	Relative Performance
no Wait State	45%	15%	40%	1.00
1 Wait State	60%	15%	25%	0.625
2 Wait States	75%	15%	10%	0.250

Here, the insertion of two Wait States reduces the relative system performance to a quarter of the one with no Wait State.

DMA Burst Length

The purpose of performing bus transactions in burst is, on one hand, to minimize the effect of bus exchange overhead (burst as long as possible) and, on the other hand, to limit the time the Am8052 occupies the bus to allow real-time responses of the CPU or other peripherals.

The DMA burst length is another factor which affects the system performance. This is due to bus arbitration and bus release overhead. After the Am8052 has asserted Bus Request ($\overline{\text{BRQ}}$ Low), the system will acknowledge the bus request by asserting $\overline{\text{BAI}}$ Low. However, in most systems this exchange involves a bus dead time of a few clock cycles (overhead). Furthermore, it takes the Am8052 about eight clock cycles to perform the first bus cycle after receiving bus acknowledge.

Considering these facts, the bus exchange overhead decreases if the burst length is increased (less bus exchanges). In the best case Burst Space is set to zero. Here, the bus is exchanged only once per character row being loaded. In the worst case Burst Count is set to "2". Here, single bus cycle DMA bursts are performed which maximize the bus exchange overhead.

An analysis has shown the overhead involved due to bus exchanges is neglectable if the burst length exceeds 64.

Full/Reduced Attribute Fetch

The amount of attribute fetches also directly affect the system performance. In lower performance systems the software designer can choose to employ the full attribute fetch mode. This means the Am8052 fetches an attribute for each character being loaded. The advantage is that this is the most simple software scheme which can be implemented. There is a fixed relationship between characters and their attributes.

The required bus bandwidth can be reduced by about a factor of two when implementing the reduced or demand attribute fetch mode. Here, attributes are loaded when required. However, this scheme involves a more sophisticated software since the relationship of characters and their attributes becomes variable.

6.2 GENERAL SYSTEM BUS APPLICATION HINTS

The following outlines the unique observations of the Am8052 bus interface.

Upper Address Writes

The Am8052 updates the upper address on a demand basis to minimize bus overhead. In upper address write cycles (Bus Master Writes), $\overline{\text{AS}}$ and R/$\overline{\text{W}}$ are both Low. This is the only time the Am8052 pulls the R/$\overline{\text{W}}$ Low. In both segmented and linear mode, the upper address (7 or 8-bit, respectively) are strobed out on the lower half of the address/data bus (AD_{0-7}). Note, that it is not possible to OR $\overline{\text{AS}}$ and R/$\overline{\text{W}}$ in order to enable a transparent latch (Figure 6-3). Since R/$\overline{\text{W}}$ propagates into the Bus Master Read cycle following the write cycle (timing parameter 10), ORing R/$\overline{\text{W}}$ and $\overline{\text{AS}}$ may generate a glitch. Therefore, it is preferable to take an approach similar to Figure 6-4.

The upper address is stored in a register such as the Am29823. The register is enabled when the CRTC is bus master ($\overline{\text{BAI}}$=Low, and $\overline{\text{BAO}}$=High) and R/$\overline{\text{W}}$ is Low. The register is strobed by the trailing edge of $\overline{\text{AS}}$. The CRTC timing guarantees that R/$\overline{\text{W}}$ settles before that edge.

Slave Transfers

The CRTC supports two slave data transfer modes: the latched and the unlatched mode. The latched mode may be selected for systems with a multiplexed address/data bus such as the 8086 and Z8000. The CRTC latches Chip Select ($\overline{\text{CS}}$) and Control/Data (C/$\overline{\text{D}}$) with the trailing edge of address strobe. C/$\overline{\text{D}}$ indicates to the CRTC that the CPU is going to address one of the internal registers (C/$\overline{\text{D}}$=High), or that the CPU is going to transfer data to or from a previously addressed register (C/$\overline{\text{D}}$=Low). With the subsequent data strobe, either the pointer or the data word is transferred. The leading edge of data strobe latches R/$\overline{\text{W}}$. The entire cycle may be asynchronous to CLK_1 or CLK_2.

The unlatched mode may be chosen for systems with demultiplexed address/data bus such as the 68000. Address strobe being Low enables an internal transparent latch to pass $\overline{\text{CS}}$ and C/$\overline{\text{D}}$ through to slave select logic. Therefore, both $\overline{\text{CS}}$ and C/$\overline{\text{D}}$ must be stable for the entire cycle. $\overline{\text{AS}}$ is connected to a flag that signals the bus has stabilized, that is, the address is valid. $\overline{\text{CS}}$ is the

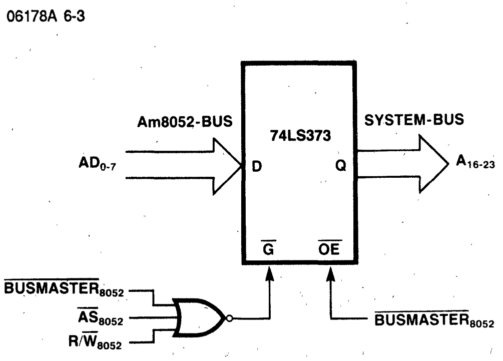


Figure 6-3 Incorrect Implementation—Latch Stores Upper Address

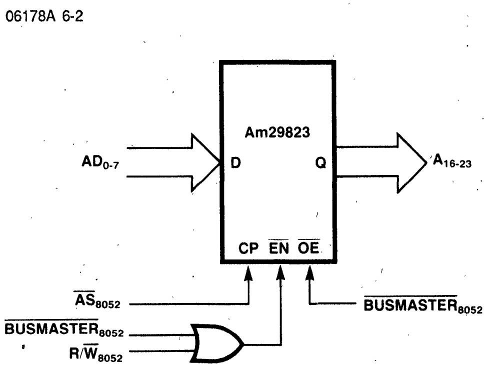


Figure 6-4 Correct Implementation—Register Stores Upper Address

decoded I/O address. C/$\overline{D}$ usually connects to A_1 of the system bus. (A_1 is the least significant address in 16-bit microprocessor systems; A_0 is "don't care".) Similar to the latched mode, data strobe latches R/$\overline{W}$, and transfers either the pointer or the data.

Clock Input Requirements

All inputs except the two clock inputs (CLK_1, CLK_2) have the normal TTL input voltage/capacitance specification. The two clock inputs require a lower Input Low Voltage, a higher Input High Voltage; and they have an increased input capacitance. The companion part, Am8152A, provides clock signals satisfying these requirements. Applications not employing the Am8152A can either use CMOS clock drivers or the discrete circuit in Figure 6-5. To increase output drive capability and improve rise and fall times, CMOS drivers can be connected in parallel.

Interrupt Acknowledge

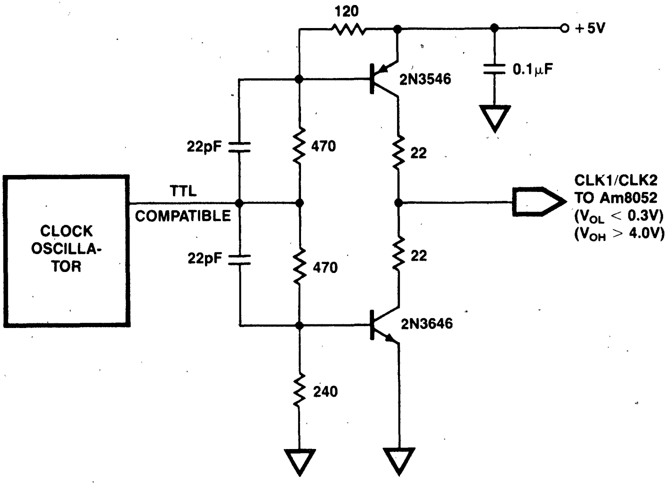
The Am8052 provides an interrupt acknowledge input to support vectored interrupts. For normal operation this input has to be tied high. Note that, as long as $\overline{INTACK}$ is Low or floating the device will not respond to any slave transactions, or will not execute any master transfers.

Wait Synchronization

It is very important, that $\overline{WAIT}$ is synchronized to the clock (CLK_1), especially when software Wait States are enabled. When the number of software Wait States is set to zero, and the setup and hold times of $\overline{WAIT}$ to CLK_1 are violated, the Am8052 either misses $\overline{WAIT}$ going High and inserts an additional Wait State (not a problem), or it goes meta-stable (a seldom case, but a real problem, since meta-stable consequences are not predictable). If the $\overline{WAIT}$ setup and hold timing is violated and the number of software Wait States is 1, 2, or 3, an additional problem occurs. In that case the Am8052 does not insert the programmed software Wait States, and scans the $\overline{WAIT}$ input in the subsequent T1 cycle. If $\overline{WAIT}$ is Low in this T1 state, the Am8052 will hang up this T1 state, characterized by $\overline{AS}$ toggling with the frequency of CLK_1.

Bus Turn-Around

The bus turnaround times when going from the address output ($\overline{DTEN}$ Low) to data input ($\overline{DREN}$ Low) should be analyzed carefully. Slow driver turn-off times in conjunction with fast turn-on delays might cause bus contention on the multiplexed address/data bus. Therefore, combinatorial delays between the transceiver control outputs of the Am8052 ($\overline{DREN}$, $\overline{DTEN}$) and the transceiver inputs should be avoided (use transceivers with



06178A 6-4

Figure 6-5 CLK1/CLK2 Driver

receive/transmit control such as Am2949). Note that, in Master Read cycles, the Am8052 does not require a data hold time to $\overline{\text{DREN}}$ or $\overline{\text{DS}}$, whichever goes inactive first. So either $\overline{\text{DREN}}$ or $\overline{\text{DS}}$ may be used to enable/disable the data.

6.3 Am8052 AND AN 8-Bit MICROPROCESSOR INTERFACE

There are two fundamental issues associated with mixing devices that communicate over different-sized buses. The first problem is allowing the two devices to communicate on a "common" data bus. Consider, for example, a 16-bit system utilizing 8- and 16-bit peripherals. Overcoming the mismatched data paths requires some form of controlled multiplexing/demultiplexing of the different data paths. In addition, extra control signals for partitioning the 16-bit word into 8, and 16-bit units may be required. Today, most of the 16-bit CPU based systems that use 8-bit peripherals usually use just the lower half of the data bus to transfer data to and from the peripheral. However, this scheme does not work when interfacing 16-bit peripherals to 8-bit CPUs, especially when these peripherals have bus master capability.

Data Funnelling

When a 16-bit peripheral attempts to transfer data over an 8-bit bus (memory write cycle or slave read cycle), the 16-bit data has to be broken down into two bytes and transferred sequentially. First, the

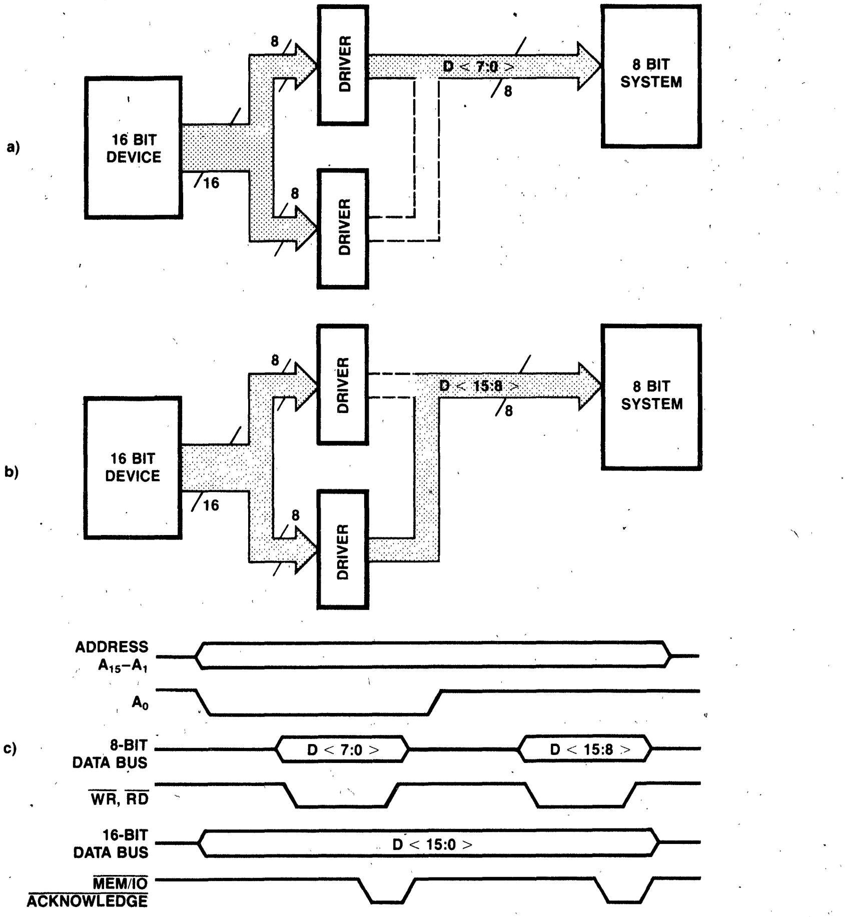
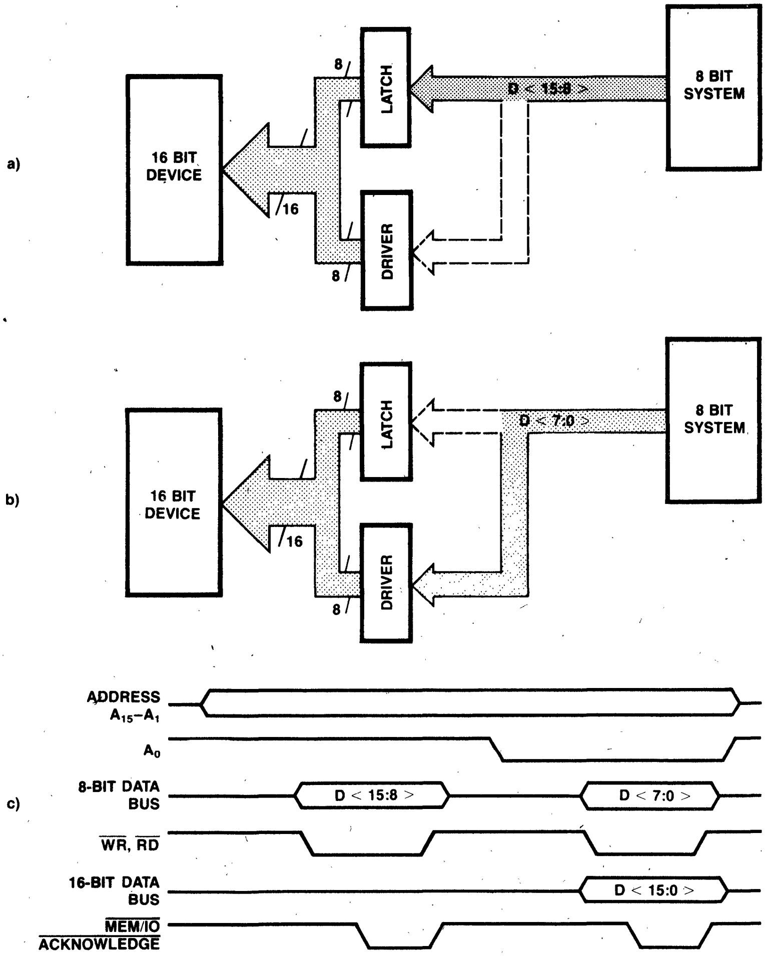


Figure 6-6 Bus Master or Slave Read Operation

lower 8-bits are transferred out on the bus (Figure 6-6a), and then in the next transfer cycle the upper 8-bits of the 16-bit word are sent out (Figure 6-6b). The generalized bus timing for such an operation is shown in Figure 6-6c. Figures 6-7a, 6-7b, and 6-7c show the opposite case; a bus read operation from an 8-bit bus to a 16-bit peripheral. Here, the first byte read from the system must be latched. Once the second byte has been fetched, the 16-bit peripheral reads in the assembled 16-bit (2-byte) word. Additionally, provisions may need to be made for the case when the 16-bit peripheral accesses single bytes.

Interruptions of the two cycle transfer must be analyzed very carefully. Master transfers must not be interrupted by slave accesses while being in the middle of a two-cycle transaction. Similar, slave accesses must not be interrupted by master transfers. While the interfacing funnels the data, the current bus cycle needs to be stretched. When the peripheral is bus master, as shown in Figures 6-6a, 6-6b, and 6-6c, the 16-bit peripheral is holding its data available for what would normally be two complete bus transfer cycles. This stretch can be achieved by delaying the transfer acknowledge signal to the peripheral, causing it to wait ($\overline{\text{WAIT}}$ asserted).

In slave mode, the 8-bit CPU would have to make two consecutive read operations to examine a 16-bit peripheral status register. The peripheral must not become bus master in between the first and second read operations since this invalidates the



06178A 6-10

Figure 6-7 Bus Master Read or Slave Write Operation

results of the first read operation. This function can be handled in two different ways: if the CPU has a bus lock instruction (for example, like the iAPX family of CPUs), then the programmer uses one of these before the CPU accesses the peripheral. Alternately, the CPU can disable the arbitration logic while it is performing the critical uninterruptible slave transfer.

Developing the Control and Data Transfer Interface

Designing the control interface to allow mixing 8 and 16-bit peripherals requires an analysis of the data and control flow. The data flow automatically defines the data path design (see Figures 6-6 & 6-7). The bus master operation by the peripheral is relatively straightforward. During a write operation, the data is written out sequentially: the lower byte first and then the upper byte (or vice-versa). During a read operation, the data is fetched sequentially. The byte fetched first is latched, to hold the data until the peripheral can read it. In the second byte read cycle, the remaining byte is fetched, the 16-bit word is assembled from the two bytes, and the 16-bit word is loaded into the peripheral. Similarly, $\overline{\text{WAIT}}$ is asserted until the second byte read cycle can be terminated.

The slave mode of operation works almost identically to the peripheral bus master mode. The master read cycle is similar to the slave write cycle, and the master write cycle is similar to the slave read cycle. In general, if the peripheral puts data on the narrower system bus, the peripheral can keep the data active in both sequential system bus cycles. On the other hand, if data is loaded into the peripheral, the interface logic has to latch the data of the first fetch cycle, whereas the data of the second cycle can be loaded directly into the peripheral (no latching required).

When defining the interface, the designer must make a conscious choice about which byte (upper or lower) to latch during peripheral read operations (or conversely, slave peripheral write operations). Once this decision has been made, the CPU must always access the latched data byte first (during a slave write) and then access the non-latched byte to complete the transfer. This restriction is a minor one with no extra software overhead; yet it could affect the ease of the programmer's coding if not handled properly. For example, if the programmer uses a compiler to generate the software for the system, extra care may be necessary to ensure the compiler generates the correct addressing sequence. An alternative to this solution would be to latch both the upper and lower data bytes. In that case, the cost of the interface would be increased, as would the complexity, with no gain in performance.

The state diagram (Figure 6-8) illustrates the control sequence implemented in the 8/16-bit bus control logic. It also depicts how uninterrupted word transfers will occur and how the addresses for upper and lower bytes are generated. In addition, the specific bus timing of the peripheral and the data bus must be examined to quantify the state control flow and provide information on data latching, read/write control strobes, and addressing to and from the peripheral. The state control flow is broken down into three parts: bus master read, slave read, and slave write operations.

The three control signals that must be be generated by the 8/16-bit control unit are: Address bit 0 (A_0), peripheral hold ($\overline{\text{WAIT}}$), and bus read ($\overline{\text{RD}}$). The A_0 line is generated by the control logic to indicate which byte is to be transferred in bus master modes only. Otherwise, the A_0 generated by the system is used to indicate which byte is being accessed. The $\overline{\text{WAIT}}$ line holds up the peripheral during transfers. The $\overline{\text{RD}}$ line is required to indicate successive transfer cycles on the bus. The peripheral's control signals will only strobe active once, because the two cycle transfer should be kept hidden from the peripheral.

The slave transfer flows are almost identical, except the CPU is generating the bus signals and the transfer directions are reversed, that is, a bus write goes into the peripheral.

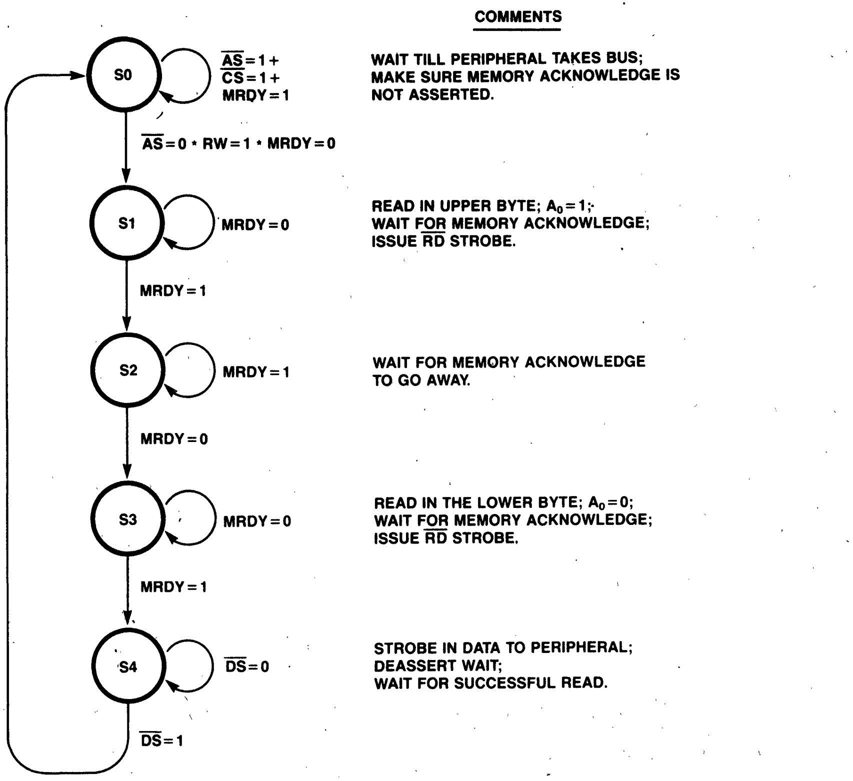
The conceptual logic for the 16-to 8-bit data flow example is shown in Figure 6-9. The data on the upper byte is latched when data is being read (as bus master) and read or written (as a bus slave). Although this interface must latch data coming from the 8-bit data bus into the peripheral, it also needs to act as transceiver when the peripheral is sending data out to the system. The ideal part to accomplish such an interface would be one that has a three-stated output, with an 8-bit wide latch, in one direction and a three-stated driver in the other direction. The Am2952 8-bit bidirectional I/O port provides a close match to the targeted logic and allows the combining of the upper data bus latch and upper data driver chips into one IC. It provides two 8-bit clocked I/O ports, each with three-state output controls and individual clocks and clock enables. An Am2949 bidirectional bus transceiver completes the logic required to buffer the data path.

The state flow control requires logic capable of sequentially moving from state to state, holding in a particular state, and being reset or initialized back to a predefined state. This design integrates the state machine generator into the same Programmable Array Logic device (PAL) as the control signal logic.

The bus control logic required to generate the data-path flow logic and the bus control signals is considerable. This is especially true if the peripherals and CPUs have different signal conventions (for example, $\overline{AS}$, $\overline{DS}$, and R/$\overline{W}$ versus ALE, $\overline{RD}$, and $\overline{WR}$). Conversion between different signal conventions, signal polarity changes, and extra functions (such as generating A_0) requires quite a bit of logic-synthesis ability. If the peripheral has bus master capability, additional information, such as bus arbitration controls, must be fed into the next state determination logic to decide what control sequence to follow.

Assembling a 8-bit CPU/16-bit peripheral interface combines all the individual components discussed above. Figure 6-10 shows a typical 8/16-bit control interface. The state machine and the bus and latch controls have to be tightly coupled in order to transfer data between the 8-bit and 16-bit buses. The generalized machine is designed under the assumption that the peripheral has bus master capability. If this is not the case, the design can be vastly simplified.

Since the CRTC does not modify system memory, no provision for a bus master write operation needs to be provided. This provision is important because it eliminates the need to generate a system write control signal ($\overline{WR}$). In addition, the control and display information has to be aligned on word boundaries. This additional requirement relieves the 8/16-bit control logic from worrying about funneling the bytes and performing odd/even byte transfers. It also saves control inputs from the Am8052 because all transfers are words; there is no need for upper and lower data strobes or byte high enable inputs/outputs.


COMMENTS
S0
$\overline{AS}$=1+
$\overline{CS}$=1+
MRDY=1
WAIT TILL PERIPHERAL TAKES BUS;
MAKE SURE MEMORY ACKNOWLEDGE IS
NOT ASSERTED.
$\overline{AS}$=0 • RW=1 • MRDY=0
S1
MRDY=0
READ IN UPPER BYTE; A_0=1;
WAIT FOR MEMORY ACKNOWLEDGE;
ISSUE $\overline{RD}$ STROBE.
MRDY=1
S2
MRDY=1
WAIT FOR MEMORY ACKNOWLEDGE
TO GO AWAY.
MRDY=0
S3
MRDY=0
READ IN THE LOWER BYTE; A_0=0;
WAIT FOR MEMORY ACKNOWLEDGE;
ISSUE $\overline{RD}$ STROBE.
MRDY=1
S4
$\overline{DS}$=0
STROBE IN DATA TO PERIPHERAL;
DEASSERT WAIT;
WAIT FOR SUCCESSFUL READ.
$\overline{DS}$=1

06178A 6-11

Figure 6-8 Bus Master Read State Flow Control

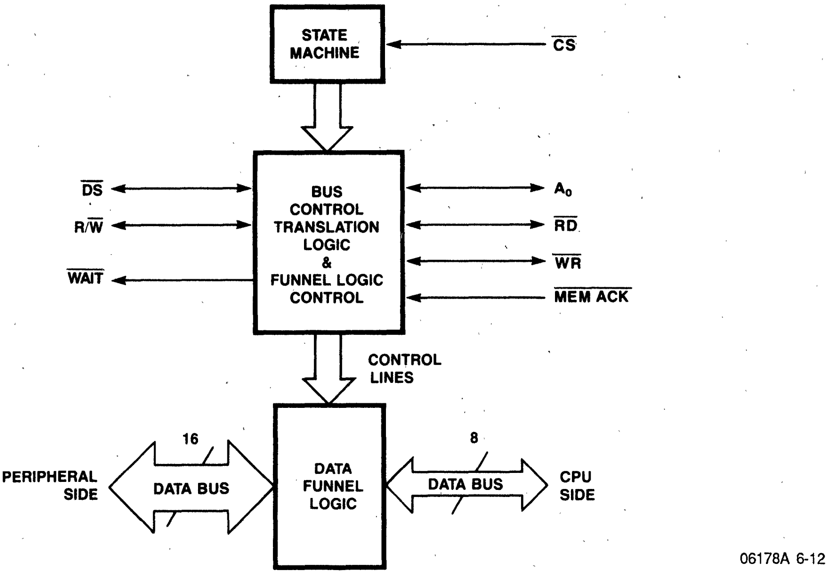


Figure 6-9 Conceptual 16/8-Bit Conversion Logic

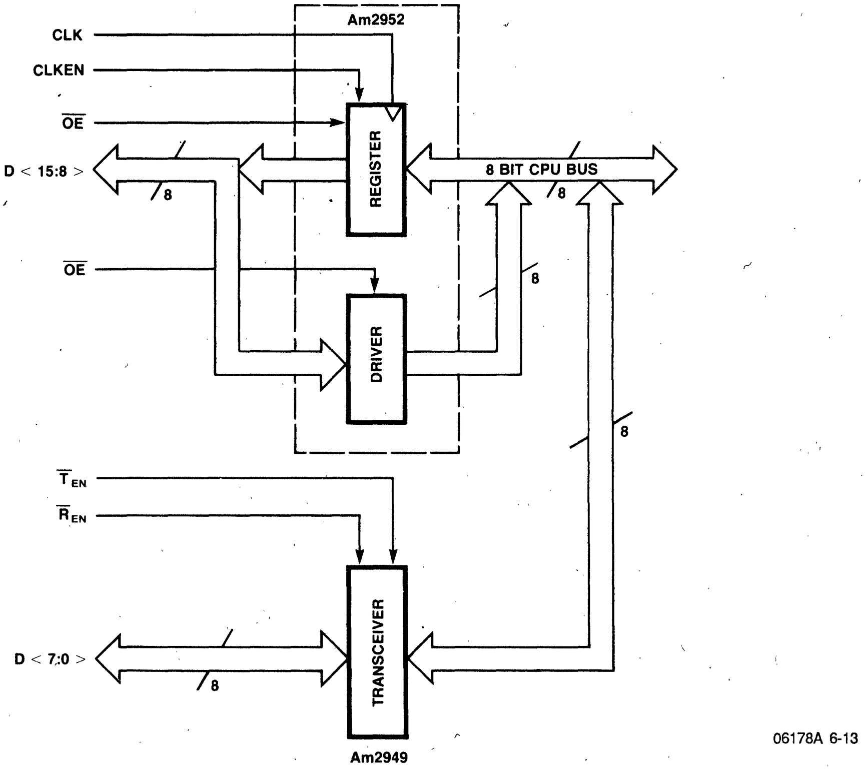
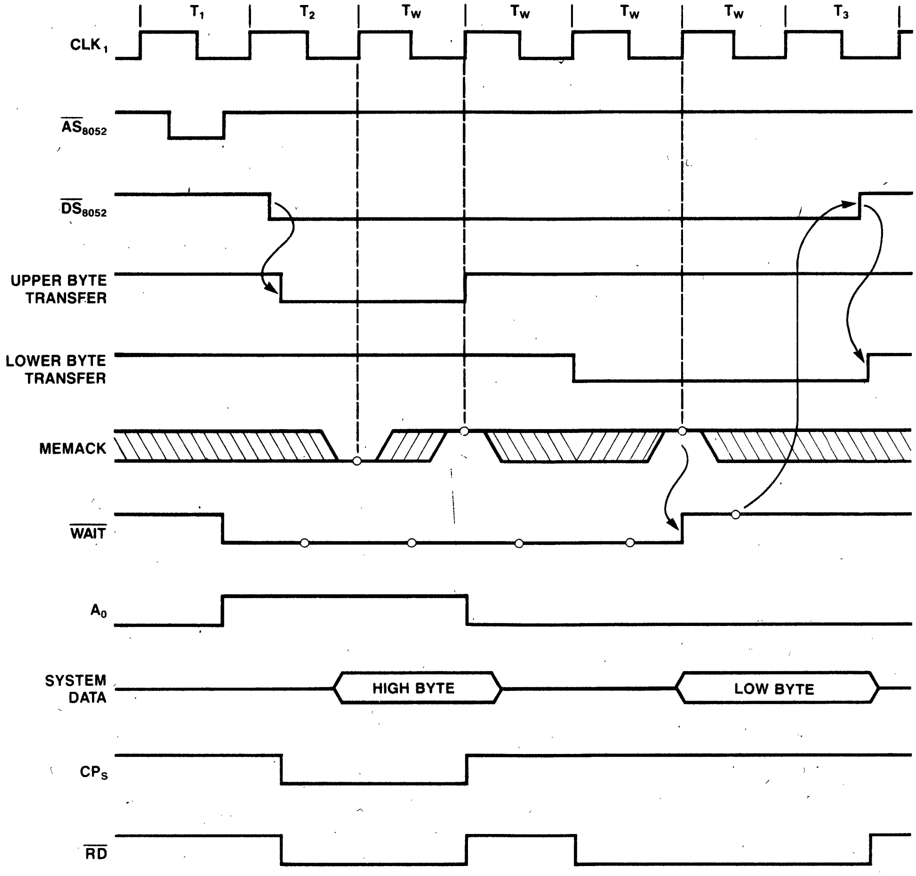


Figure 6-10 Data Funnel Logic

The slave accesses by the CPU are either pointer writes (to select the desired control/status register) or 16-bit data read/write operations. The pointer write operation is really an 8-bit operation because only the lower 8 bits of the data form the register address. This is illustrated in the flow diagram by the path that bypasses half the slave read/write states if the command/data (C/D̄) line is High. These state flow diagrams are derived directly from the timing diagrams of the Am8052. The three different transfer timings are shown in Figures 6-11, 6-12, and 6-13.

Two special conditions have been incorporated into the state flow diagrams whenever a transfer is first initiated. Before a new transfer cycle is attempted (that is, the state machine is waiting in S0), the memory acknowledge must be inactive. This prevents any interference from the last transfer. The second special condition occurs when the Am8052 asserts the R/W̄ line to indicate a write operation. Whenever the Am8052 updates the upper 8 bits of the 24-bit address latch, the R/W̄ line indicates a write operation (in conjunction with AS̄). The Am8052 is not actually performing a system data write, only an address latch update. Hence, the state flow reflects this fact by not starting a sequence if the R/W̄ line is active Low from the Am8052.

These simplifications allow the Am8052 to 8-bit CPU control interface to be synthesized in a single AmPAL22V10 device (Figure 6-14). In addition, the bus control signals are converted from AS̄, DS̄,



06178A 6-14

Figure 6-11 Bus Master Read Timing Diagram

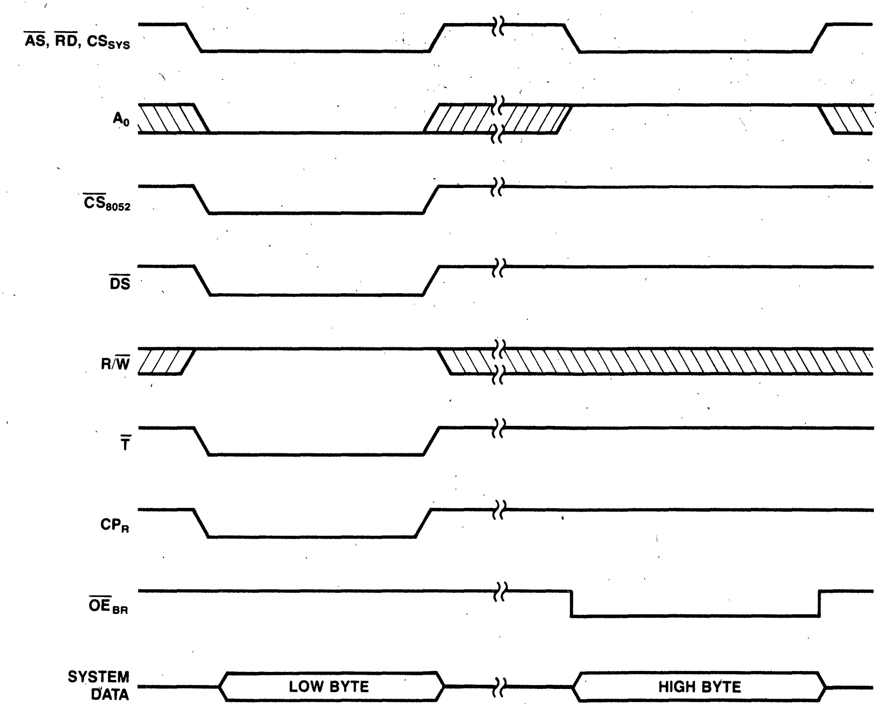


Figure 6-12 Slave Read Timing Diagram

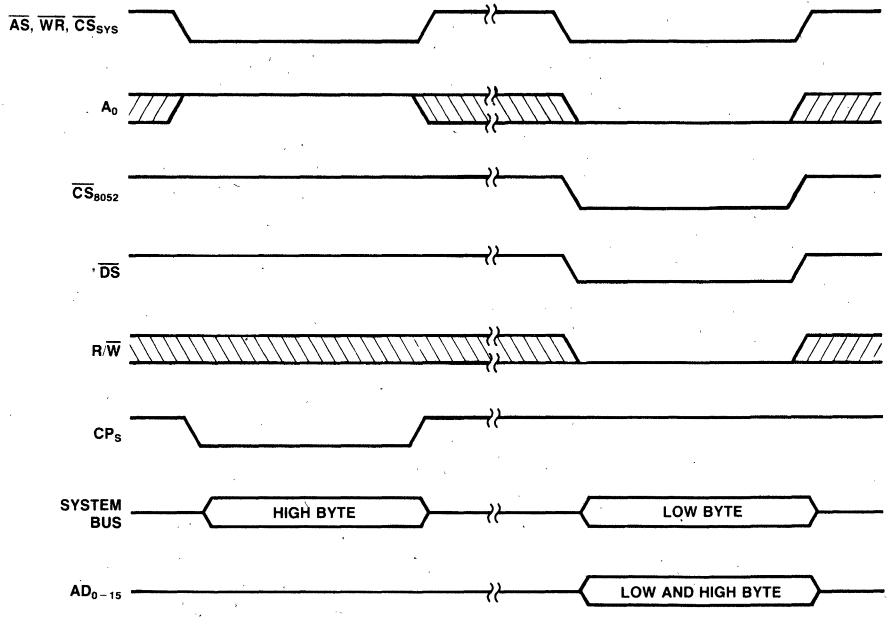
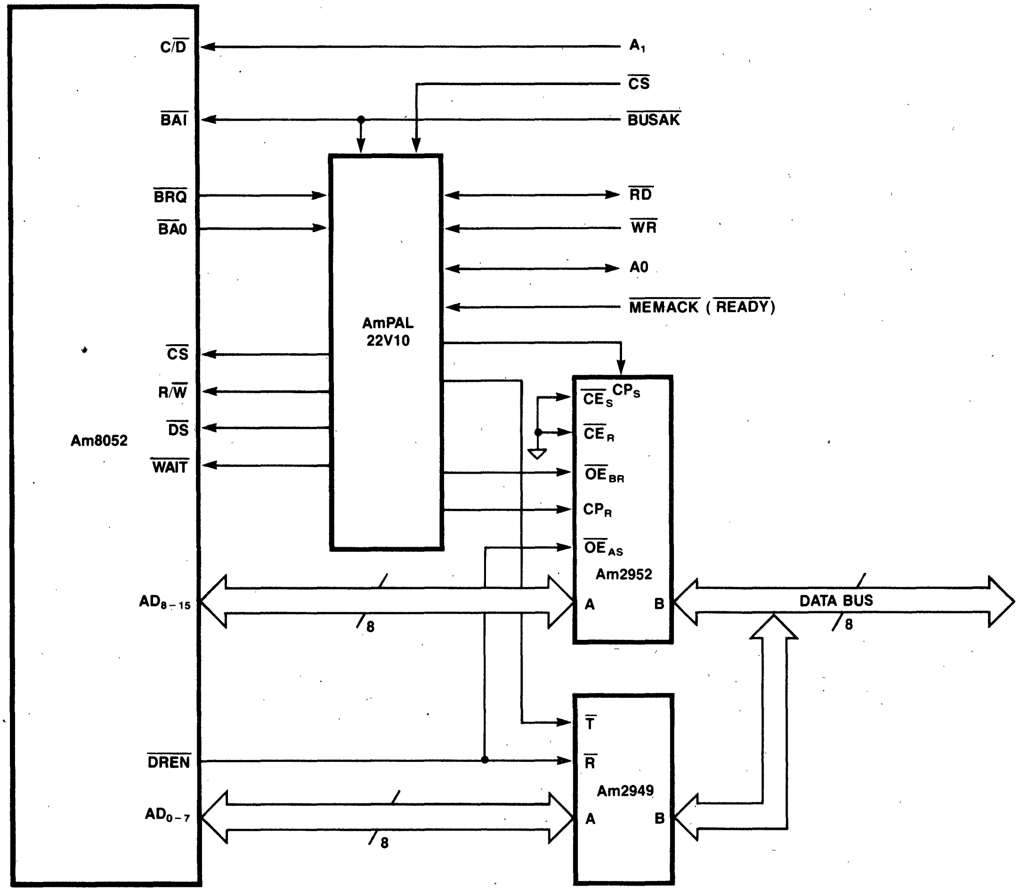


Figure 6-13 Slave Write Timing Diagram

and R/W to RD and WR. Figure 6-14 shows the assembled control and data transfer logic for this interface. The minimum Am8052 and bus control signals that have to be generated are RD, A_0, DS, R/W. Although DS and R/W are used as inputs during a bus master operation by the Am8052, the AmPAL22V10 must convert the CPU RD and WR signals to DS and R/W for slave I/O operations. The signals A_0 and RD are generated by the control logic when the Am8052 is performing a read access to the system. The WAIT (or not READY) signal to the Am8052 also needs to be generated by the control logic. Additionally, the four control signals of the bidirectional port and transceiver are generated.

Trade-offs and Limitations

In a design dramatically affecting the I/O of the system, a number of trade-offs and limitations should be noted. The most obvious limitation in using 16-bit peripherals on an 8-bit bus is that the 16-bit peripheral will be under-utilized. The speed of all I/O operations will be cut by 50%. Consequently, the bus utilization will go up if the 16-bit peripheral represents a significant factor of the bus usage. A CRT controller like the Am8052 might use 5% to 10% of the bus bandwidth for display information when using 16-bit I/O. Converting to 8-bit I/O would double bus usage to 10% to 20%, or more.



06178A 6-17

Figure 6-14 Am8052 8-Bit Interface

Another factor that might affect the bus usage is the efficiency of the 8- to 16-bit conversion control logic. If the state machine designed to perform the 8/16-bit conversion (or 16/32-bit) is improperly designed, then extra transfer overhead may be introduced. This could mean a sequential transfer of two 8-bit values takes longer than two single 16-bit transfers. The system designer must weigh the cost of the extra overhead on a case-by-case basis. However, as previously mentioned, the benefits may well justify these limitations; the bus is self-limiting, but the device characteristics allow for value-added designs. In addition to bus degradation for certain configurations, extra logic and design effort are involved. However, most interfaces outside a system's immediate family require some kind of extra interface logic anyway. Therefore, by optimizing the control signals and incorporating them into programmable logic devices such as the AmPAL22V10, the IC count can be dramatically reduced.

6.4 THE Am8052 AND 8086 INTERFACE IN MIN MODE

The 16-bit multiplexed address/data bus of the 8086 is directly connected to the multiplexed address/data lines of the Am8052, Figure 6-15. The upper address (7 bit for segmented mode or 8

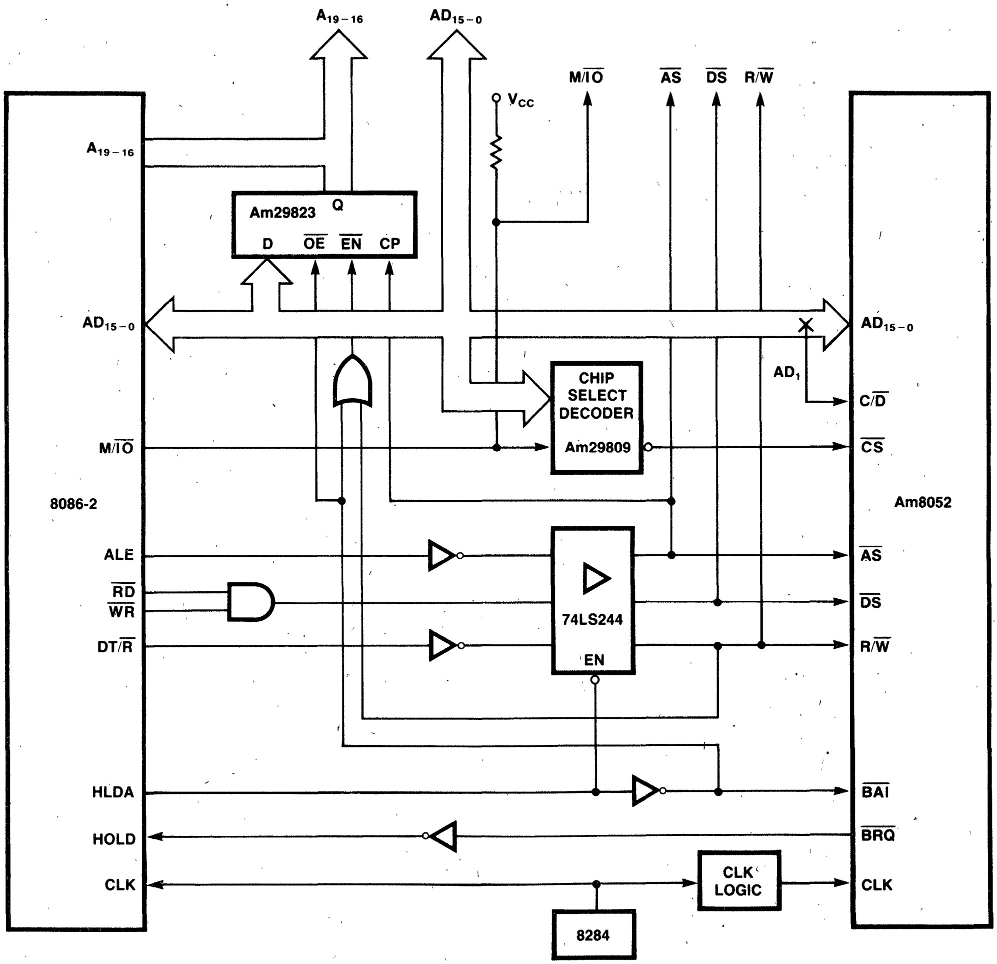


Figure 6-15 8086–Am8052 Interface

bit for linear mode) is strobed out on the lower half of the bus (AD$_{0-6}$ or AD$_{0-7}$) and is stored in a register (Am29823). The Am8052 may be programmed for segmented or linear mode depending on whether address roll-over is desired. The register output is enabled ($\overline{OE}$=Low) when the Am8052 is bus master. Clocking is enabled ($\overline{EN}$=Low) when R/$\overline{W}$ is Low while the Am8052 is bus master (upper address update cycle). The trailing edge of Address Strobe clocks the register.

$\overline{RD}$ and $\overline{WR}$ from the 8086 are logically ORed to generate $\overline{DS}$. ALE is inverted and connected to $\overline{AS}$ of the Am8052. DT/$\overline{R}$ is also inverted to form R/$\overline{W}$. All three signals are passed through a three-state buffer which is enabled when the 8086 is bus master. Memory/IO (M/IO) is pulled High when the Am8052 is bus master since the Am8052 only addresses memory.

Bus Clock

The Bus Master timing is synchronized to the bus clock (CLK$_1$) of the Am8052. In order to get a similar and synchronous bus timing when the 8086 or the Am8052 are driving the bus, the Am8052 bus clock can be connected to the 8086 bus clock. However, in proportional spacing applications, the video timing must be derived from the bus clock and therefore the bus clock must be synchronized to the character clock (CLK$_2$).

For these applications the Am8152A provides the synchronized clocks (CLK1,CLK2) with the right timing and DC specification.

In non-proportional spacing applications, the Am8052 can operate with the 8086 bus clock if the duty cycle is adjusted. In this case, the Am8152A cannot be used as the clock driver, and a separate clock driver needs to be provided. This clock driver must provide a clock satisfying the special clock input specification (MOS specification) such as clock High and Low width and voltage, and input capacitance. Most CMOS drivers or a discrete clock driver shown in Figure 6-5 satisfies these specifications. This design must be changed for different frequencies. Figure 6-16 shows circuitry which adjusts the duty cycle for the Am8052. The required delay time needs to be adjusted for the chosen bus clock frequency.

At high bus clock frequencies (e.g., ≥ 8 MHz) Bus Request of the Am8052 must be synchronized to the clock, to generate a synchronized HOLD for the 8086.

Detailed Timing Analysis

The following timing analysis is based on an 8-MHz 8086-2 and an 8-MHz Am8052. At this frequency the minimum clock High (TCHCL) and Low (TCLCH) times for the 8086-2 become 43 ns and 68 ns, respectively. Some of the subsequent calculations are based on these values for TCHCL and TCLCH.

Slave Reads and Writes

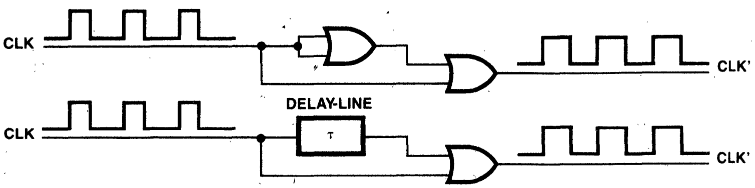
#21 $\overline{CS}$ set-up time to the trailing edge of $\overline{AS}$ (minimum 0 ns). The 8086-2 provides a set-up time of 28 ns of AD$_{0-15}$ before the trailing edge of ALE. Let us assume 0 ns of minimum propagation delay since neither the inverter nor the driver specifies one. The maximum propagation delay allowed for the decoder is, therefore, 28 ns (68 ns–40 ns). The decode time for the Am29806/809 decoders is 13 ns.

#22 $\overline{CS}$ hold time after the trailing edge of $\overline{AS}$ (minimum 25 ns). The 8086-2 provides a minimum address hold time of 33 ns.

#23 C/$\overline{D}$ set-up time before the trailing edge of $\overline{AS}$ (minimum 0 ns). The 8086-2 provides an address set-up time of 28 ns.

#24 C/$\overline{D}$ hold time after the trailing edge of $\overline{AS}$ (minimum 25 ns). The 8086-2 provides a minimum address hold time of 33 ns.

#25 Delay from $\overline{CS}$ to $\overline{DS}$ (minimum 30 ns). The



06178A 6-8

Figure 6-16 Duty Cycle Adjustment for the Am8052

worst case (shortest delay) can be calculated as:

(TCLCH – TCHLL) + TCLRL
+ (28 ns – 13 ns)
= (68 ns – 55 ns) + 10 ns + (28 ns – 13 ns)
= 37 ns.

#26 Access time (maximum 150 ns). The 8086-2 expects an I/O access time no longer than:

2 • TCLCL – TCLRL – TDVCL
= 2 • 125 ns – 100 ns – 20 ns
= 130 ns.

This means that one Wait State must be inserted.

#27 Data hold time (minimum 10 ns). The 8086-2 requires a max. data hold time of 0 ns, i.e., no hold time.

#28+ 29 R/$\overline{\text{W}}$ to $\overline{\text{DS}}$. Since DT/$\overline{\text{R}}$ is connected to the R/$\overline{\text{W}}$ input of the CRTC, this timing is not guaranteed by design.

#32 Data hold time during slave writes (minimum 20 ns). The 8086-2 provides at least 38 ns.

#33 Data set-up time in slave writes (minimum 90 ns). The 8086-2 provides more than one clock period (125 ns) data set-up time.

#34 The Am8052 requires a minimum Data Strobe pulse width of 100 ns. The 8086-2 provides

TWLWH
= 2 • TCLCL – 40 ns
= 210 ns.

#35 Recovery time (minimum 330 ns). The 8086-2 provides more than 3 clock periods = 375 ns.

6.5 Am8052 AND 68000 INTERFACE

One of the designer's most challenging tasks is to interface two generically different Bus Masters. Such as the 68000 microprocessor and the Am8052 CRT Controller. Both Bus Masters support a 16-bit-wide data bus and a 24-bit linear addressing space (if the Linear/Segmented bit in the Am8052 Mode Register 1 is set to "1"). The control bus signals of the Am8052, however, differ from that of the 68000's and need to be translated bidirectionally. Figure 6-17 shows the interface schematics.

Slave Mode

The Am8052 provides two basic slave modes: the latched mode for systems with multiplexed address/data buses and the unlatched mode for systems with demultiplexed address/data buses. In this interface application, the Am8052 operates in the unlatched mode because the address and data buses of the 68000 are demultiplexed. In this mode, Address Strobe ($\overline{\text{AS}}$) is kept asserted throughout the entire bus cycle, making the internal latches for Chip Select ($\overline{\text{CS}}$) and Control/Data (C/$\overline{\text{D}}$) transparent. AS is driven Low by an open collector inverter connected to $\overline{\text{BAI}}$. This forces $\overline{\text{AS}}$ to go Low whenever the Am8052 is not in control of the bus.

Slave Access Timing Analysis:

The Am8052 timing parameters are analyzed in ascending numerical order.

#25 The set-up time of Chip Select ($\overline{\text{CS}}$) to Data Strobe ($\overline{\text{DS}}$) must be at least 30 ns in order to guarantee the minimum access time (#26). Violation of this specification could happen if Parameter 26 is lengthened, as shown below.

#26 When $\overline{\text{CS}}$ and $\overline{\text{DS}}$ are asserted simultaneously, the access time increases from 150 ns (#26) to 180 ns (150 ns + 30 ns). The 68000 requires an access time of 175 ns (2.5 • 125 ns – 60 ns – 15 ns) to operate without Wait States. No such Wait States are necessary for slave reads.

#27 The data hold time requirement of 0 ns (68000 read operation) is easily met; the Am8052 provides a minimum of 10 ns.

#28 The R/$\overline{\text{W}}$ setup time requirement of 0 ns before $\overline{\text{DS}}$ (Am8052) is is guaranteed by the 68000 (1 clock cycle).

#32 The data hold time (20 ns) in slave write is provided by the 10-MHz or slower 68000s.

#33 The data set-up time before the trailing edge of data strobe (80 ns) is provided by the 8-MHz 68000 (145 ns min).

#34 The minimum guaranteed write pulse width of the 8-MHz 68000 is 115ns. The Am8052 requires at least a 100 ns pulse. Similar to #26, smaller values for #25 cause the $\overline{\text{DS}}$ pulse width (#34) to be widened. In order to satisfy this parameter, either the set-up time

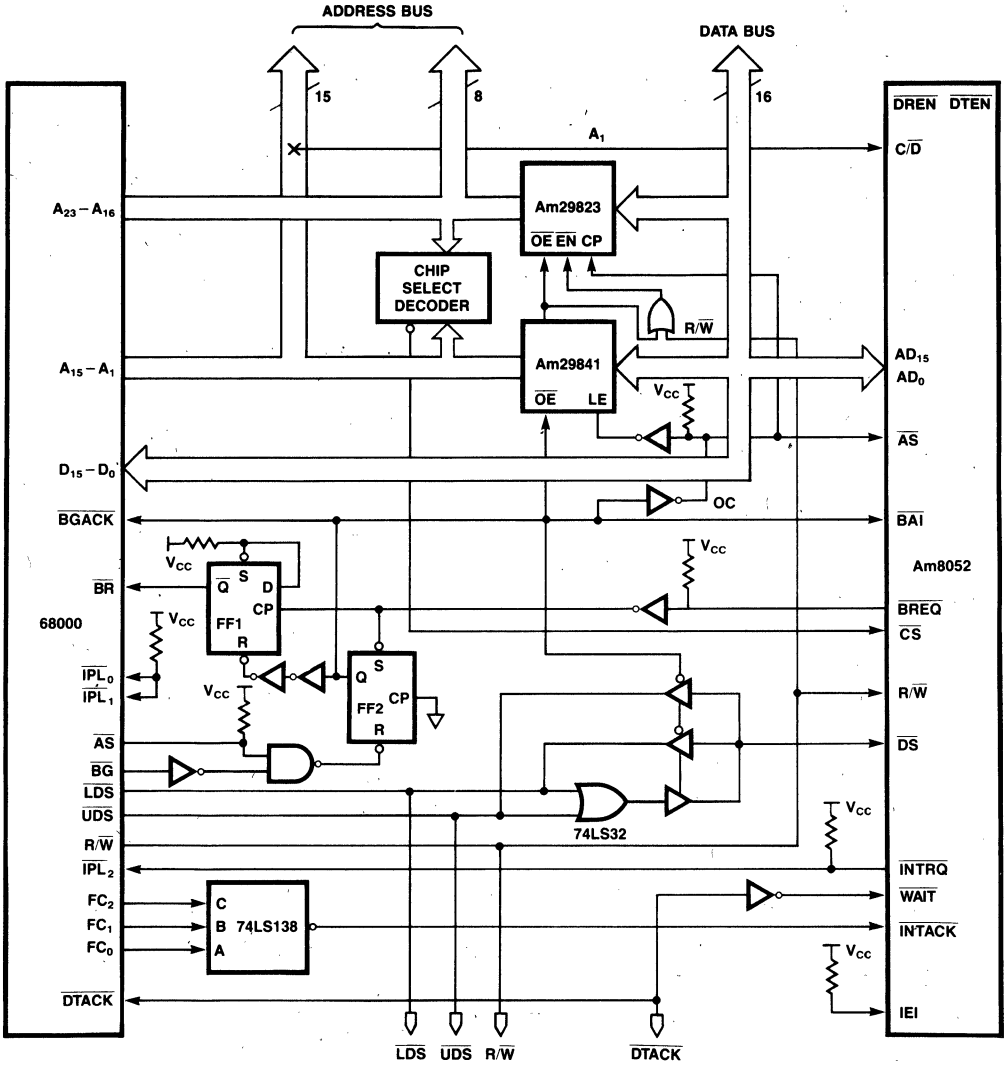
#24 must be at least 15 ns or one Wait State (68000) must be inserted. The 15 ns setup time demands a fast chip select decoder.

#36 The $\overline{CS}$ to $\overline{DS}$ hold time (5 ns) is satisfied by the address hold time of the 68000 (30 ns min).

#37 Same as #36.

Data Strobe

The Am8052 in slave mode can only be accessed as a 16-bit peripheral (word transfers only). This means that both Data Strobes of the 68000 ($\overline{LDS}$ and $\overline{UDS}$) must be active simultaneously. It is only then that the OR gate asserts DS for the Am8052. The driver is enabled when the 68000 is Bus Master (BAI High). In Master Mode, both data strobes are driven by the Am8052 because it does only word transfers.



06178A 6-5

Figure 6-17 68000–Am8052 Interface

Master Mode

After the Am8052 is initialized and the display is enabled, the Am8052 asserts Bus Request ($\overline{\text{BREQ}}$ Low) to request the system bus. The bus arbitration scheme between the Am8052 and the 68000 is discussed in the paragraph below. To avoid bus contention at the end of Bus Master read cycle, the data bus transceiver (not shown) must be turned off before the Am8052 starts driving the address for the next cycle. Timing Parameter 11 allows a turn-off time of 25 ns which is sufficient for the Am29863 transceiver.

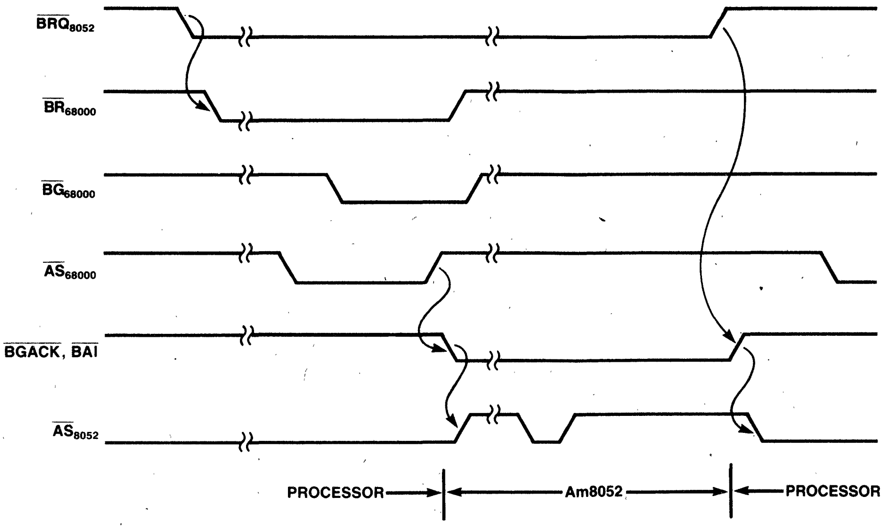
Bus Arbitration

The 68000 CPU supports a three-wire bus arbitration mechanism. A peripheral requesting bus mastership asserts a Bus Request ($\overline{\text{BR}}$ Low), see Figure 6-18. The CPU, in response, asserts a Bus Grant ($\overline{\text{BG}}$ Low). At the end of the current bus cycle, the requesting peripheral goes on the bus. The end of the current CPU bus cycle is signaled by the Address Strobe going inactive. The combination of Bus Grant active and Address Strobe inactive asynchronously resets FF2 (see Figure 6-17), thereby asserting $\overline{\text{BAI}}$ for the Am8052 and Bus Grant Acknowledge ($\overline{\text{BGACK}}$). Resetting FF2 also resets FF1 asynchronously, which deactivates BR. In response to $\overline{\text{BR}}$ becoming inactive, the 68000 deactivates $\overline{\text{BG}}$. Note that $\overline{\text{BR}}$ must be Low for at least 20 ns after $\overline{\text{BGACK}}$ to prevent rearbitration. The inverters and the delay through FF1 must meet this requirement. $\overline{\text{BGACK}}$ and $\overline{\text{BAI}}$ stay asserted until the Am8052 terminates its DMA burst and releases $\overline{\text{BREQ}}$. At that time FF2 is asynchronously set and $\overline{\text{BGACK}}$ and $\overline{\text{BAI}}$ are deactivated, and the 68000 resumes operation.

The bus arbitration mechanism does not yet support DMA preemption. However, Am8052 DMA preemption by external devices can simply be supported by setting FF2 on preemption. The preemption DMA can grant the bus after the Am8052 has released the bus by deactivating $\overline{\text{BREQ}}$. In this case, $\overline{\text{BAI}}$ being Low is no longer sufficient to flag that the Am8052 has been granted the system bus. For proper DMA preemption support, the data strobe drivers and the open collector driver for $\overline{\text{AS}}$ must be controlled by a signal which flags that the Am8052 is on the bus. (Note: For the time between preemption ($\overline{\text{BAI}}$ High) and bus release ($\overline{\text{BREQ}}$ High), the Am8052 is still in control of the system bus).

Interrupt Acknowledge

The Am8052 supports vectored interrupts if the No Vector bit in Mode Register 2 is disabled (NV=0). The vector is put out in Interrupt Acknowledge cycles ($\overline{\text{INTACK}}$ Low, IEI High, and $\overline{\text{DS}}$ Low).


$\overline{\text{BRQ}}_{8052}$
$\overline{\text{BR}}_{68000}$
$\overline{\text{BG}}_{68000}$
$\overline{\text{AS}}_{68000}$
$\overline{\text{BGACK}}$, $\overline{\text{BAI}}$
$\overline{\text{AS}}_{8052}$
PROCESSOR
Am8052
PROCESSOR

06178A 6-6

Figure 6-18 Bus Exchange Timing

6.6 Am8052 AND 80188 INTERFACE WITH DUAL BUS ARCHITECTURE

With today's predominantly 16-bit systems, some new designs still evolve around the 8-bit structure. The underlying reason is cost. Systems designed for specific control operations can usually be satisfied with state-of-the-art 8-bit CPUs such as the 80188. They do not require the slightly higher performance 16-bit CPUs such as the 80186. The 8-bit system design requires less memory devices (EPROMs, RAMs) and less MSI-devices (address latches, data bus drivers). Board layout is simpler as well.

With all the attractiveness of an 8-bit system design, interfacing such a system with the Am8052 must maintain the low cost level. The additional cost of designing an 8-bit system interfacing with a 16-bit device must be kept as low as possible.

The interface design outlined below contains only low cost, off-the-shelf devices such as the AmPAL16L8, byte-wide registers, drivers and transceivers (Am2947, Am2956, and Am2959) and a few standard TTL devices.

Data Path (Figure 6-19)

The previous section analyzed the strategy and general problems associated with designing the Am8052 into an 8-bit system. There the Am8052 interfaces with the byte-wide memory and microprocessor via a 16-bit to 8-bit data funneling logic. The drawback of that design is a significant system performance degradation due to the Am8052 DMA activity.

The design discussed here avoids this drawback by implementing a dual bus architecture. The Am8052 fetches the display information from a local memory, without affecting the operation of the microprocessor. This local memory is implemented in two static memory devices (e.g. 8K • 8 static CMOS RAMs). The bus arbitration logic controls CPU accesses to the local bus, preempting the Am8052 whenever necessary. Depending on whether the Am8052 is bus master or bus slave, the bus arbitration logic has to take actions listed in the following, in order to grant the local bus to the CPU.

If the Am8052 is in slave mode, the arbitration logic prevents the Am8052 from granting the local bus by blocking Bus Acknowledge ($\overline{\text{BAI}}$ stays High). The CPU then accesses the local memory without asserting any Wait States. Since the Am8052 typically uses about 5 to 20% of the bus bandwidth (80 to 95% of the time the Am8052 is off the bus), this can be considered to be the normal case.

If the Am8052 is bus master, the CPU transfer cycle is stopped temporarily by inserting Wait States (ARDY Low). To minimize the wait time, the Am8052 DMA is immediately preempted ($\overline{\text{BAI}}$ High). As soon as the Am8052 releases the bus ($\overline{\text{BRQ}}$ High) the CPU transfer cycle is terminated (ARDY High).

Control Logic

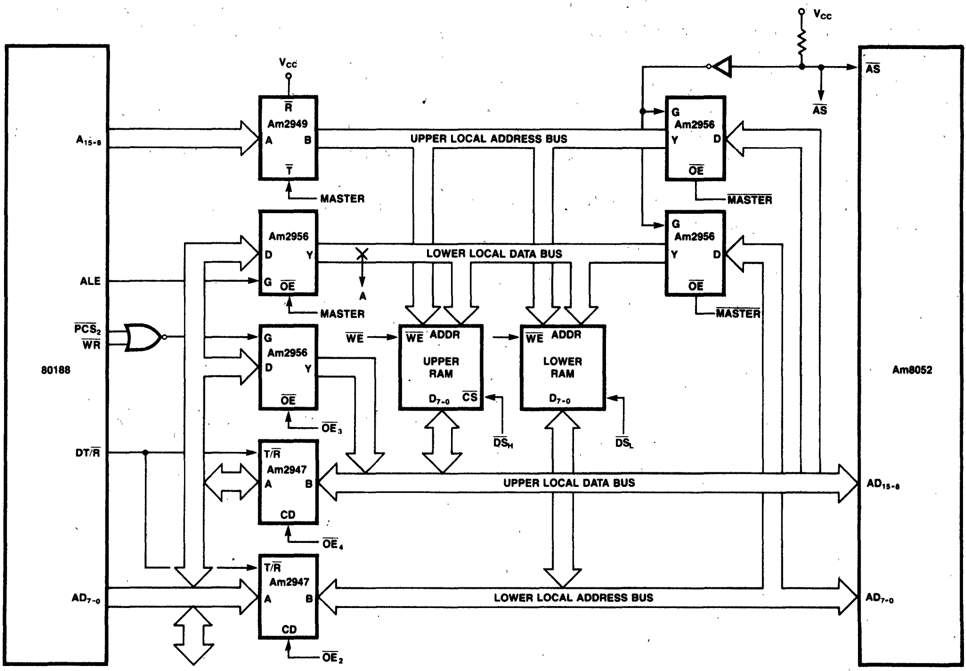
The control logic consists of three separate units: The "Master" unit (Detail A in Figure 6-20), the "Bus Arbiter" (Detail B in Figure 6-20) and the PAL device (Figure 6-22), converting the CPU-Am8052 signals and generating the various control signals for the data path logic (Figure 6-19).

The "Master" Unit

The "Master" unit generates a signal $\overline{\text{MASTER}}$, which indicates if the Am8052 has granted bus mastership on the local bus ($\overline{\text{MASTER}}$ Low). $\overline{\text{MASTER}}$ is the output of a flip-flop built out of OR/AND gates. Master is asserted when the Am8052 receives a bus acknowledge ($\overline{\text{BAI}}$ Low), after it has requested the bus ($\overline{\text{BRQ}}$ Low). $\overline{\text{MASTER}}$ then stays active until the Am8052 releases the bus ($\overline{\text{BRQ}}$ High). In applications not involving DMA preemption, $\overline{\text{MASTER}}$ can be generated simply by OR'ing $\overline{\text{BRQ}}$ and $\overline{\text{BAI}}$. This simplified logic does not generate a correct $\overline{\text{MASTER}}$ signal in case of DMA preemption, because the Am8052 is bus master while $\overline{\text{BRQ}}$ is Low and $\overline{\text{BAI}}$ is High (time between preemption and bus release).

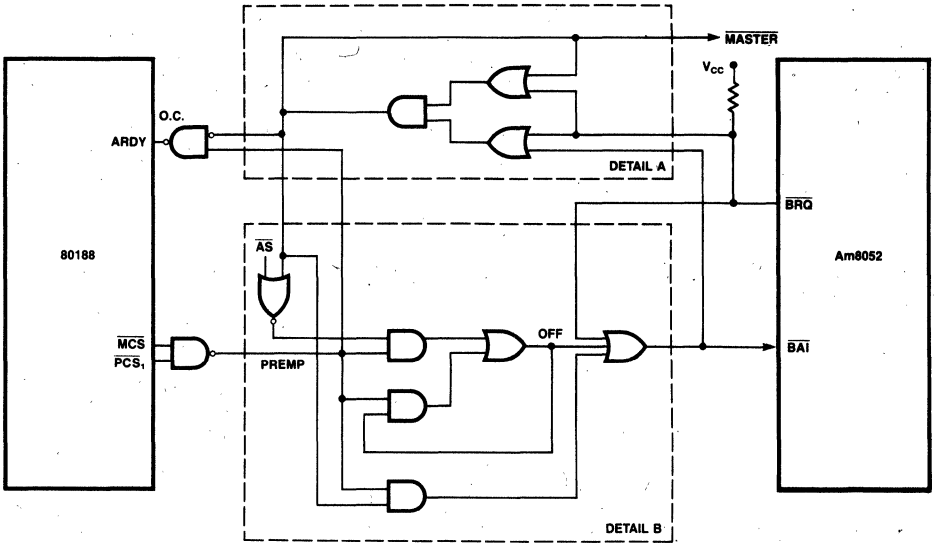
The "Bus Arbiter"

This simple logic arbitrates between the CPU and the Am8052 where the CPU has higher priority. When the Am8052 is in slave mode and the CPU accesses the local bus ($\overline{\text{MCS}}$ Low or $\overline{\text{PCS}}$ Low), ARDY becomes High and $\overline{\text{BAI}}$ is blocked from going Low, in order to prevent granting the Bus to the Am8052. When the Am8052 is bus master and the CPU accesses the local bus, ARDY is asserted and DMA preemption is initiated. This forces the Am8052 off the bus. To avoid glitches on $\overline{\text{BAI}}$, and satisfy the minimum width requirement for $\overline{\text{BAI}}$, DMA preemption is delayed until the next address strobe ($\overline{\text{AS}}$ Low).



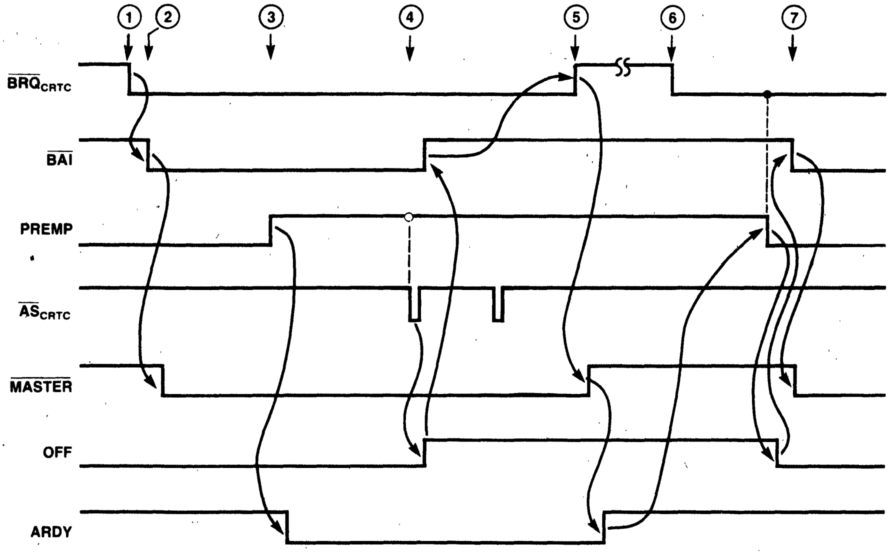
06178A 6-18

Figure 6-19 80188–Am8052 Interface



06178A 6-19

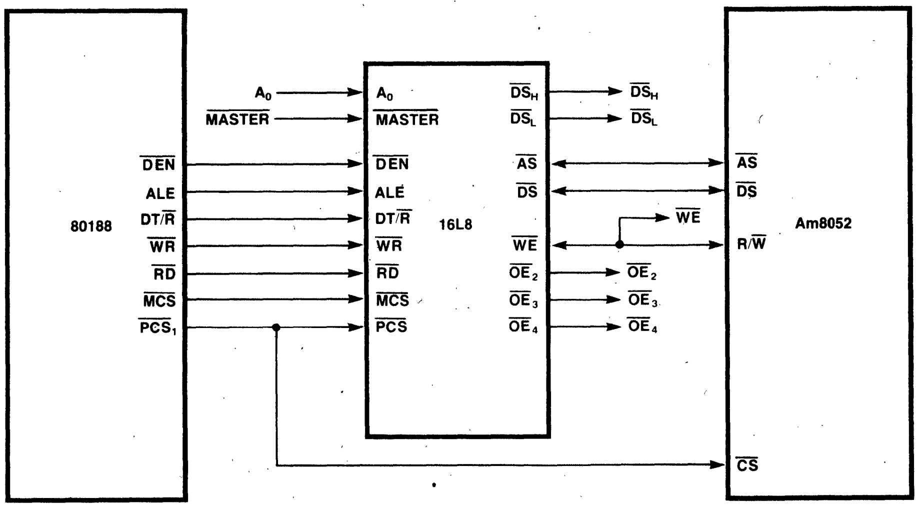
Figure 6-20 Bus Arbitration Logic



1 Am8052 requests bus ($\overline{BRQ}$ ↓)
2 Am8052 receives bus acknowledge ($\overline{BAI}$ ↓)
3 80188 requests Am8052 DMA preemption ($\overline{MCS}$ ↓ or $\overline{PCS}_2$ ↓)
4 Preemption request to Am8052 is delayed until $\overline{AS}_{CRTC}$ ↓ to guarantee min width of $\overline{BAI}$ (> 4 CLK_1 cycles)
5 Am8052 gets off the bus ($\overline{BRQ}$ ↑)
6 Am8052 requests bus again
7 Am8052 receives bus acknowledge after CPU finishes access

06178A 6-20

Figure 6-21 Bus Arbitration Timing Diagram



06178A 6-21

Figure 6-22 Control Logic

Slave Access Sequence

The CPU loads internal registers of the Am8052 in two cycles. First, it strobes the upper data byte into a latch by asserting $\overline{PCS_2}$. Next, both data bytes are loaded into the Am8052 by asserting $\overline{PCS_1}$.

To minimize interface logic, this application does not support read accesses of the upper byte of the internal registers. Only the lower byte can be read. Contents of control registers can be tracked by software in memory, therefore it is not necessary to be able to read these registers. All status bits except the "Scroll In Progress" (SIP) bit are located in the lower byte and can be read. However, the SIP-bit can be scanned while using vectored interrupts, because it is included in the interrupt vector.

When the Am8052 is in slave mode, the least significant CPU address line (A_0) selects the memory device for the upper (A_0 Low) or lower byte (A_0 High) and the appropriate transceivers.

PAL Design Specification

```
AMPAL16L8                                               PAL DESIGN SPECIFICATION
PAT007                                                  H.-J. Ruehl     1/15/85
Interface 80188 - Am8052
Advanced Micro Devices, Stuttgart, West Germany

/PCS1    /MCS    A0     /DEN    ALE     /RD     /WR     DTR/    MASTER    GND
RW       /DSH    /DSL   /AS     /DS     /WE     /OE2    /OE3    /OE4      VCC

DSH  =  MCS*DS*A0*/MASTER + DS*/WE*MASTER

DSL  =  MCS*DS*/A0*/MASTER  + DS*/WE*MASTER

IF (/MASTER) THEN AS  =  ALE

IF (/MASTER) THEN DS  =  RD + WR

IF (/MASTER) THEN WE  =  DTR

OE3  =  /MASTER*PCS1*DTR

OE4  =  MCS*DEN*A0*/MASTER

OE2  =  MCS*DEN*/A0*/MASTER + /MASTER*PCS1*DEN
```

CHAPTER 7
LOW-COST SMART TERMINAL DEMO SYSTEM

7.1 INTRODUCTION

This project was initiated to demonstrate that a low-cost, but high performance terminal can be built based on the Am8052/Am8152A CRT controller chip'set. It shows that it is possible to design a high-performance display system with limited amount of memory (just 16 kBytes) and a low-cost CPU (the Am8051) (Figures 7.1, 7.2, and 7.3). The architecture of the Am8052 allows display updates and editing tasks to be performed with a minimum load on the local CPU (mostly pointer changes rather than block moves). However, by providing more memory or a faster CPU, the overall system performance can be further improved.

Note. The hardware design and the corresponding software package are the property of Advanced Micro Devices Inc., Sunnyvale. However, since this project is intended to be a promotion tool for the Am8052, the complete (or any part of the) hardware or software may be copied and used in other designs. Soucre code and listing files are made available on IBM PC compatible floppy-disks.

The complete demo set consists of:

- Hardware description (Section 7.4)
- Software users manual (Section 7.5)
- Comparison to other terminals (Section 7.6)
- Source files (2 floppy-disks)
- Listing files (2 floppy-disks)
- Demo program (1 floppy-disk)
- Am8052 Terminal Board (IBM PC form factor)
- Cable for async communication port

The following items are required, but not provided:

- Power supply: IBM PC or ext. power supply
- IBM PC monochrome monitor plus AC power cable
- IBM PC/XT/AT with async port (COM1)

7.2 DEMO SET-UP

Take the following steps to set up the demo:

- Turn off the power to the IBM PC/XT/AT or compatable
- Open chassis by removing five screws located on the back side of the system
- Insert the Am8052 Terminal board into one of the empty slots
- Connect the bottom 9-pin D-Connector (J4) to the async port of the PC (COM1). The cable is supplied.
- Connect the upper 9-pin D-Connector (J3) to the monochrome monitor. The cable is attached to the monitor.
- Connect the monitor to AC power. A special cable is required, but not supplied. A spare IBM PC power cable can be used.
- Turn on the PC power. After a few seconds a cursor should show up at the top left corner of the display. Also, the PC should boot up. If either item does not happen ,turn off power and re-check the connections.
- Insert demo disk into the PC and execute demo by typing the following command sequence:

```
BASICA          (to load basic interpreter)
LOAD "DEMO"     (to load source of demo)
RUN             (to execute demo)
```

- If the demo disk contains the compiled (faster) version of the demo called "DEMO.EXE", it may be executed by typing:

```
DEMO            (to load and execute
                demo)
```

- Various parts of the demo may be executed by selecting items of the main demo menu.

Speed

The terminal board operates at 9600 baud. The baud rate may be changed by reprogramming EPROM addresses 3FF0H and 3FF1H. For example, to set the baud rate to 19200 the value at 3FF0H (DblBaudOpt) should be set to 80H, and the value at 3FF1H (BaudRatCnt) to FDH.

The demo program written in BASIC supplies characters at a lower rate than equivalent to 9600 baud. To show higher screen update rates, the following command may be executed:

```
COPY A:DEMO.BAS COM1:
```

This command copies the source file of the demo program to the terminal board. The font loading performance may be shown by down-loading the file "8052FONT.DOC" to the terminal board. It will define the 120-character-per-line font.

```
COPY A:8052FONT.DOC COM1:
```

The Am8052 can currently load one new character font matrix (7*12 pixel) per frame (about 60 chars/sec). Defining the characters using the ANSI standard it takes about 50 bytes to describe a single character. At 9600, baud about 1000 bytes/sec can be down loaded. This results in an update rate of 20 characters/sec which is limited by the data rate of the async line.

The terminal has been speed optimized. The character placement and CR/LF routine have been speeded up as much as possible. The result is, that this board can operate at 19200 baud without interface handshake (no control signal, no XON/XOFF) as long as no escape sequences are sent to the terminal board.

7.3 BUILDING PROCEDURE

There are nine assembly source files supplied on two IBM PC compatible disks. The files are listed below:

C_BASE	Interrupt Handlers
C_INIT	Initialization
C_SWITCH	Dispatch Control
C_TABLES	Control Tables (easy expandable)
C_WORK	Control Routines
C_UTIL	System Utilities
C_FONT	80 Character-Per-Line Font
C_CONFIG	Configuration
C_MemMap	Included Definitions

Each of these files is down loaded to the HP 64110A Logic Developement System. The first eight files are assembled with the Am8051 Cross Assembler. The resulting object files must then be linked together. Both C_BASE and C_CONFIG contain absolute addresses. C_BASE also contains relocatable program memory as do the remaining six modules. All eight modules should be specified together in the link with the base of the relocatable program segment set to 0040H.

The absolute file produced by the linker can than be uploaded to a PROM programmer. The baud rate is defined in the C_CONFIG module. the locations "DblBaudOpt" and "BaudRatCnt" correspond to the special function registers PCON and TH1, respectively. The Am8051 timer 1 is used to generate the serial communications clock described in the 8051's users manual. Only the most significant bit of "DblBaudOpt" (corresponds to SMOD) is relevant.

Keyboard Interface

The keyboard logic is copied from the IBM PC/XT Technical Manual. It is provided as an example only. The hardware is not tested. In fact, if U15 is installed the system will not operate. The current software does not support the keyboard interface.

7.4 HARDWARE DESCRIPTION

While the the cost for VLSI is decreasing, the so called "dumb" terminals take over more and more features of their smart companions. Performance, features, and ergonomics are the important considerations for todays generation of low cost terminals. Large eye-saving, operator friendly non-glare screens, which can be tilted or swiveled to suit the user, combined with high resolution smooth scrolling displays highlight the ergonomic features. Functional enhancements include user programmble function keys, programmble screen formats (80 or 132 columns), a stationary 25th status line with the time of day. High screen update rates, and text editing speed are characteristic of these high performance terminals.

First generation alphanumeric CRT controllers such as the 6845 (Motorola) or the 8275 (Intel) became the standard for low cost systems. However, as the demand for enhanced features increases, these very low cost controllers lose their attractiveness. Implementing additional features with external logic would raise the cost. Second generation controllers such as the Am8052 are becoming more cost effective since these controllers integrate enhanced functions in a single device. Furthermore, drastic price reductions made possible by die shrinks and cost saving packaging techniques (i.e. PLCC–Plastic Leaded Chip Carrier) now match the requirements of this very cost sensitive market.

An Am8051/8751 micro-controller is chosen as the local intelligence. It receives display commands from the host system via an asynchronous communication channel and interprets them, eventually generating the display list for the Am8052. Both the CRT controller and the micro-controller share a 16kbyte static RAM array which stores this display data. The Am8051 controller views this memory as 16kbytes (8-bit interface) while the Am8052 views it as 8k words (16-bit interface). Four standard latches (74LS373) and a PAL device demultiplex the address buses and implement the data funneling logic to interface the 8-bit and 16-bit bus masters.

Since the Am8052 off-loads display and editing tasks from the processor, little CPU activity is required. With the Am8052, editing tasks such as swapping pages, inserting/deleting lines or characters are implemented via pointer manipulation rather than data block moves. The simple, inexpensive Am8051/8751 micro-controller is, therefore, capable of executing all display fast and efficiently.

The distinctive characteristics are listed below:

- two display formats (selected by software)
  80 • 24 characters with 9 • 14 pixels/char cell
  120 • 30 characters with 6 • 10 pixels/char cell

- optionally up to three trailing blanks may be appended to simplify text right justification

- windowing and vertical smooth scrolling

- proportional spacing

- highlight, superscript, subscript, reverse, underline, overscore, blinking, multiple cursors

Additional features requiring extra hardware:

- soft loadable character font generator (single port RAM)

- horizontal smooth scroll

- italic characters generated by hardware

- Kanji/Chinese character set

System Interface

Addressing

Two transparent address latches (74LS373) demultiplex the 16-bit address/data bus of the Am8052 and, in addition, the 8-bit address/data bus on Port 0. Both latches are enabled if either ALE of the Am8051 (gated with $\overline{\text{BAI}}$) or $\overline{\text{AS}}$ of the CRTC are active. The output of the lower latch is always enabled, the output of the upper latch is only enabled if the CRTC is bus master ($\overline{\text{BAI}}$ Low). Otherwise, the upper address is directly driven by the Am8051. Port 2 (upper address byte of the Am8051) cannot be connected to the inputs of the upper address latch, because this would result in bus contention, when the Am8051 reads the upper RAM.

Am8051 Address Map

The Am8051 addresses data memory (IC3 and IC4), the internal registers of the Am8052, and the keyboard logic. These cycles are flagged by $\overline{\text{BAI}}$ being inactive, and by either $\overline{\text{RD}}$ or $\overline{\text{WR}}$ being active. The PAL device perform the decoding task. The address map is listed below:

0000_H–$3FFF_H$	keyboard logic (odd addresses only!)
4000_H–$7FFF_H$	Am8052 internal registers
8000_H–$BFFF_H$	data memory (IC3 and IC4)
$C000_H$–$FFFF_H$	reserved

Note, that reading even addresses activates the output of IC1. The keyboard logic must, therefore, be accessed by odd addresses only. The I/O address space is defined as follows:

0001_H	keyboard latch (IC21) (read only)
4000_H	Am8052 register data access (high byte, R/W)
4001_H	Am8052 register data access (low byte, R/W)
4003_H	Am8052 register pointer (low byte, write only)

The proper sequence of accessing both halves of the Am8052 registers is crucial. Before performing any register access the pointer must be loaded. When writing a register first the high byte is latched (even address), then the low byte (odd address) is provided. In the second cycle, the interface controller supplies both bytes to the Am8052. When reading a register the two cycles are performed in the reverse order. First, the low byte is read (odd address), then the high byte (even address) is read.

Bus Arbitration

The Am8051 performs the bus arbitration in software. The bus request of the Am8052 ($\overline{\text{BRQ}}$) interrupts the Am8051. In the following interrupt service routine the Am8051 three-states Port 2 (upper address bus) and Port 0 (lower address/data bus). Then it acknowledges bus request by granting the bus to the Am8052 by pulling P1.2 Low ($\overline{BAI}$ Low) and P1.3 High ($\overline{AS}$ High). P1.2 controls the bus acknowledge input ($\overline{BAI}$) directly. P1.3 pulls the address strobe line of the Am8052 ($\overline{AS}$) Low whenever a slave access is planned. For Am8051 memory accesses P1.3 must be High to allow ALE to propagate to the address latch ($\overline{AST}$ Low). A High on any port 3 pins is equivalent to a floating output since each of these pins has an open-drain driver with internal pull-up resistors.

The Am8051 scans the level on the interrupt input frequently to determine when the CRTC releases the bus. In response, the Am8051 removes Bus Acknowledge (P1.2 High and P1.3 High). This design can support DMA preemption, since the Am8051 can preempt the Am8052 whenever appropriate by removing $\overline{BAI}$. The Am8051 program loop executed while the Am8052 controls the bus, must be located within program memory internal to the Am8051.

Am8051 Memory Access

The 8-bit Am8051 accesses the 16-bit RAM in byte mode. For even addresses (A_0 Low) IC16 is selected, for odd addresses (A_0 High) IC24 is selected. IC14 latches the lower address byte. Port 2 provides the upper address byte directly. IC5 and IC13 are both disabled, since data will go directly to Port 0. The lower RAM (IC16) is selected ($\overline{CS_4}$ Low); IC24 is disabled. In read cycles the output is enabled ($\overline{OE}$ Low). Write is enabled ($\overline{WE}$ Low) during a write cycle.

In read cycles when A0 is Low IC13 is enabled ($\overline{OE}$ Low, G High) to pass the data from the upper RAM (IC24) to the data port of the Am8051 (Port 0). In write cycles IC5 is enabled (OE Low, G High) to pass the data in the opposide direction from the Am8051 to the RAM. IC5 and IC13 can be replaced by a single, bidirectional latch (such as the 74LS646). For memory accesses only the transparent (driver) function is required. However, the latching function is required when the Am8051 accesses the 16-bit registers of the Am8052 (see below).

Am8052 Memory Access

The Am8052 performs only word read accesses. This means $\overline{WE}$ stays inactive High. Also, both RAMs are selected, and A_0 is disregarded. IC5 and IC13 are disabled.

Register Write

The Am8052 registers are accessed in two cycles. The first write cycle latches a pointer to the register to be accessed (C/$\overline{D}$ High). In subsequent write cycles the actual data transfer to the register can take place (C/$\overline{D}$ Low). C/$\overline{D}$ is connected to A1 of the Am8051. Otherwise, control or data write cycles are identical.

How does the Am8051 load the 16-bit register via its 8-bit data bus? To accomplish this task, the Am8051 first latches the upper byte in IC5 (A_0 Low, $\overline{OE2}$ High, G2 High pulsed). In the next cycle, the Am8051 accesses the CRTC and loads both bytes into the Am8052 (A_0 High, $\overline{OE2}$ Low, G2 Low). The upper byte is supplied by IC5, the lower byte is supplied by Port 0.

Register Read

The Am8051 reads a 16-bit register in the reverse sequence. First it accesses the CRTC, to read both bytes. The lower byte is loaded into the Am8051 immediately, the upper byte is temporarily latched in IC13 (A_0 Low, $\overline{OE1}$ High, G1 High pulsed). In a subsequent cycle the Am8051 can read the upper byte from IC13 (A_0 High, $\overline{OE1}$ Low, G1 Low).

Port 1 Allocation

P1.2 and P1.3 are High when the Am8051 controls the system bus. P1.4 and P1.5 control the keyboard logic. For normal operation these lines should be Low. $\overline{OEN}$ is active once per active scan line and may be used to determine the beam position. Therefore, it is connected to the counter/timer input of the Am8051 (T0).

Control Logic

Most of the control logic is integrated in a single PAL device, a PAL16L8, which controls the memory selection, write enable, and output enable, the control for the data funneling (IC5 and IC13), and the bidirectional data strobe ($\overline{DS}$) for the Am8052.

Timing

The Am8051 and the CRTC operate asynchronously. The Am8051 should be operated at its maximum frequency to achieve maximum performance. The CRTC is driven by the clocks provided by the Am8152A (Video System Controller). CLK$_1$ specifies the bus clock (DMA operation). CLK$_2$ determines the character clock rate. To support various screen formats and, optionally, proportional spacing CLK1 controls the video timing. Both clocks are derived from the dot clock, and digitally synchronized during HBLANK to avoid screen jitter. The dot clock is 16 MHz. CLK1 is 4MHz (divide ratio of four). CLK2 cycle width varies from 4 to 12 dots, thus also resulting in a maximum frequency of 4MHz.

Video Interface

Basic Configuration

The basic configuration consists of the Am8052 (IC3), the Am8152A (IC12), a JEDEC pin-compatible character font generator ROM (IC1), the dot clock oscillator (Y2), and the video cable driver (IC19). All the remaining logic shown is optional and implements the special functions outlined below.

Horizontal Smooth Scroll

The Am8052 only supports vertical smooth scroll directly. Horizontal smooth scroll can however be implemented quite easily. A dummy character is placed at the start of each character row. This dummy character is made invisible by blanking it externally. The actual smooth scrolling is performed by modulating the width of this character. By shortening it, the character row moves left. Eventually, the leftmost character will disappear. At that time the first character is linked

```
PAL SPECIFICATION PROGRAM

PAL16L8                                              PAL DESIGN SPECIFICATION
PAT020                                                                6/21/85
8051-Am8052 INTERFACE CONTROLLER                          JUERGEN STELBRINK
ADVANCED MICRO DEVICES, SUNNYVALE CA

A0   /RD    /BAI   MEM   IO  NC   /WR   NC    NC    GND
NC   /CS3   /CS4   /OE2  G1  G2   /DS   /OE   /CS   VCC

OE    =  RD + DS*BAI                              ; OUTPUT ENABLE OF RAMS
CS3   =  /A0*RD*MEM*/IO*/BAI +                    ; UPPER (EVEN) RAM
         /A0*WR*MEM*/IO*/BAI +
         DS*BAI
CS4   =  A0*RD*MEM*/IO*/BAI  +                    ; LOWER (ODD) RAM
         A0*WR*MEM*/IO*/BAI  +
         DS*BAI
OE2   =  /A0*WR*MEM*/IO +                         ; IC2, CPU WRITES EVEN RAM
         A0*WR*/MEM*IO                            ; IC2, CPU WRITES AM8052
/G1   =  /MEM*/A0 + MEM*A0  +                     ; IC1
         /IO*/MEM + IO*MEM  +
         /A0*IO + A0*/IO    +
         /RD
/G2   =  /MEM*IO + MEM*IO   +                     ; IC2
         /WR + A0
CS    =  /MEM*IO*A0                               ; AM8052 CHIP SELECT
IF (/BAI) DS = RD + WR                            ; BIDIRECTIONAL DATA STROBE

DESCRIPTION:

The non-inverted equations for G1 and G2 are listed below:

G1    =  /A0*RD*MEM*/IO +                         ; CPU READS EVEN RAM
         A0*RD*/MEM*IO                            ; CPU READS AM8052
G2    =  /A0*WR*MEM*/IO +                         ; CPU WRITES EVEN RAM
         /A0*WR*/MEM*IO                           ; CPU WRITES AM8052
```

out, and the width of the dummy character is increased to it's original size. Then the smooth scroll process is continued until the second character is scrolled out completely, etc.

The digital delay line consisting of four D-Flip-Flops (IC22) delays BLANK to mask off the video stream (IC23). The delay is set to four CLK1 cycles (16 dot clocks). This covers the maximum length of the dummy character (12 dots) plus a delay of one CLK2 cycle (the first CLK2 cycle is 4 dot clocks).

Since the Am8152A involves one further dot clock propagation delay, the rightmost pixel of the dummy character is not masked off by the delayed BLANK. This pixel is blanked by loading a blank pixel ("1") into the 12th position of the video shift register.

The upper half of the video shift register is loaded with the falling edge of CLK2. While CLK2 is High, the character font generator output is three-stated (IC1). So, the pull-up resistors supply a High to the parallel input port, causing the Am8152A to always latch "1"s with the falling edge of CLK2. Since the character font is implemented in negative logic (for normal video REVERSE is active), "1"s are represented as blank pixels.

Horizontal smooth scroll is discussed in more detail in a separate application note.

Soft Loadable Character Font Generator

Once horizontal smooth scroll is implemented, it takes only one additional latch (IC7) to integrate a soft loadable character font generator. Note, that this implementation differs from the method discussed in an earlier AMD application note. This implementation requires less hardware and also boosts the loading performance. Here, one slice of one character may be loaded per character row resulting in a loading rate of about two to three full character cells per frame (100 to 200 characters cells per second) assuming that 24 character rows are diplayed and that a cell contains between 8 and 12 slices.

In this implementation, the dummy character at the start of each character row performs one more task. It enables the loading process as well as providing all necessary information to perform the process itself.

The character code of the dummy character specifies the character to be loaded. The upper eight bits of the 10-bit row attribute word contained in the Row Redefinition Block provides the pixel pattern of the character slice to be loaded. The cursor attribute selects the scan-line to be loaded. Therefore, the Row Redefinition Block defines the scan-line number to be loaded (the cursor position within the character cell). The values for cursor start and end must be equal to activate this attribute for a single scan line only.

Finally, the cursor attribute bit within the character attribute word of the dummy character enables the loading process itself.

Character Code Graphic

An alphanumeric display system can implement bit-mapped graphic directly. One graphic implementation in an alphanumeric system treats the character code directly as bit-map. Each character code specifies eight consecutive pixels within a scan line. Therefore, the character code bypasses the character font generator via IC2 and supplies the pixel pattern to the parallel input of the shift register. Since each character row now consists of only one scan line, the Am8052 bus traffic is increased significantly and must be analyzed carefully.

Row attribute bit 1 enables/disables this mode. For normal operation this bit is set to "0."

Italic Characters

Italic type characters could obviously be supported by an additional (or larger) character font ROM or by reloading the character font RAM. But a small amount of special hardware can change straight characters to slanted characters.

The italic mode is turned on by placing a unique shaped blank character into the character string. This character is wide on the top and narrow on the bottom. Once this character is placed in a character string all following characters will be tilted according the programmed shape of the "Italic On" character. The italic mode is turned off by placing a blank character with the reversed shape into the character string.

IC11, a 256•8 bit PROM implements this feature. The 8-bit address is assembled from the 4-bit scan line address and the 4-bit CLK2 divide ratio supplied by the Am8052. For the standard divide ratios from 4 to 12 the PROM just passes the supplied ratio through to the Am8152A (normal character mode). For four other ratios this device becomes active. There, it modulates the width of the character with the scan line address to build the uniquely shaped characters. The following table lists the width values and shows how they affect the character diplay.

Value	Function
0000	Italic On (9 pixels/character)
0000	Italic Off (9 pixels/character)
0010	normal character (4 pixels wide)
1010	normal character (4 pixels wide)
1110	Italic On (6 pixels/character)
1111	Italic Off (6 pixels/character)normal

The italic mode is automatically reset at the end of a character row. Both characters controlling italic mode have the same width as the standard characters. So, no width computations like in proportional spacing applications are required.

Italic mode is turned on by placing a blank (20_H) in the string. This blank has the width value: 0000 or 1110 depending on the chosen screen format. The italic mode is terminated by inserting a blank with a width value: 0001 or 1111.

7.5 USER'S MANUAL FOR THE LOW-COST, SMART TERMINAL

Displays

Background Display	
30	usable rows stored in memory
128	characters/row stored in memory
80 • 24	characters in "normal" mode
120 • 30	characters in "compressed" mode
	Scrolls vertically and horizontally

Message Display	
1	row (visible only when selected)
128	characters/row
	Scrolls only horizontally

Window Display	
14	usable rows (7 visible when selected)
40	characters/row (40 visible when selected)
	Scrolls only vertically

There is only one cursor in the terminal; it is always in the active display. It may not be visible (e.g. beyond the currently visible bounds, or under the (visible) window while in the background display). The active position (i.e. cursor) indicates where the next graphic character which this system receives will be stored.

Controls

There are five classes of controls: normal ASCII control characters, escape sequences, extended control characters, standard control sequences, and private control sequences.

Normal C0 Control Characters

These are the subset of the ASCII X3.4 control characters which we have implemented.

Backspace (BS)

Moves the active position one column left in the active row, except when the cursor is already in the leftmost column in the display. This control does not cause scrolling.

Carriage Return (CR)

Moves the active position to the first column in the active row. This control does not cause scrolling.

New Line (NL)

Moves the active position to the first column in the next row downward from the active row. If the active row is the bottom row of the display then a blank row is inserted at the bottom of the display and the top row is deleted. This has the appearance of scrolling the entire display upward one row. The next row, to which the active position is moved, is the new bottom row.

This control has no effect when the message display is active.

Escape (ESC)

Introduces escape sequences defined in the ANSI X3.64 extension.

Escape Sequences

These are seqences defined in ANSI X3.64 that consist of an escape character followed by a final character. They are parameterless controls.

Reset to Inital State (RIS)—ESC c

Resets the terminal to a blank background display, in small display mode, scrolled all the way up and to the right, with the active position at the first column in the seventh row (top row on the monitor screen). The graphic rendition, character blink rate, smooth scroll rate and cursor appearance are given their initial values. The Vertical Editing Mode (VEM), Display Width Mode (AMDDWM), Scroll Mode (AMDSCM) and Screen Polarity Mode (AMDSPM) are all reset. The message and window displays are also blanked as well as being made invisible. The background display is active and the character generator is reloaded with its initial patterns.

Control Sequence Introducer (CSI)—ESC[

Introduces control sequences defined in the ANSI X3.64 extension. It also introduces the private control sequences that are implemented in accor=dance with that standard.

Extended Control Characters

The Control Sequence Introducer (CSI) is also available as a single, 8-bit control character (x'9B'). It performs the same function as the escape sequence described above.

Extended Control Sequences

These are sequences introduced with the Control Sequence Introducer (CSI) described above. They may contain parameters and intermediate characters and end with a final character. Parameters may be interpreted either as decimal numbers or as special selectors that depend on the particular control for their meaning. A default parameter is one that is missing or is specified with a value of zero.

Cursor Backward (CUB)—CSI Pn D

Moves the active position left by the number of columns specified by the single numeric parameter, without altering its vertical position. An attempt to move the active position beyond the leftmost column in the display leaves it at the leftmost column. A default parameter causes movement one column leftward, except from the leftmost column. This control does not cause scrolling. It may move the cursor to a position where it is invisible.

Cursor Down (CUD)—CSI Pn B

Moves the active position downward the number of rows specified by the single numeric parameter, without altering its horizontal position. An attempt to move the active position beyond the bottom row of the display leaves it at the bottom row. A default parameter causes movement one row downward, except from the bottom row. This control does not cause scrolling. It may move the cursor to a position where it is invisible.

Cursor Forward (CUF)—CSI Pn C

Moves the active position the number of columns rightward specified by the single numeric parameter, without altering its vertical position. An attempt to move the active position beyond the rightmost column in the display leaves it at the rightmost column. A default parameter causes movement one column rightward, except from the rightmost column. This control does not cause scrolling. It may move the cursor to a position where it is invisible.

Cursor Position (CUP)—CSI Pn ; Pn H

Moves the active position to the row and column specified by the two numeric parameters. The first parameter specifies the row; a default causes movement to the top row. An attempt to move the active position beyond the bottom row in the display leaves it at the bottom row. The second parameter specifies the column; a default causes movement to the leftmost column. An attempt to move the active position beyond the rightmost column in the display leaves it at the rightmost column. This control does not cause scrolling. It may move the cursor to a position where it is invisible.

Cursor Up (CUU)—CSI Pn A

Moves the active position upward the number of rows specified by the single numeric parameter, without altering its horizontal position. An attempt to move the active position beyond the top row of the display leaves it at the top row. A default parameter causes movement one row upward, except from the top row. This control does not cause scrolling. It may move the cursor to a position where it is invisible.

Delete Line (DL)—CSI Pn M

Deletes the number of rows specified by the single numeric parameter. If the Vertical Editing Mode is reset then the active row and rows below it are discarded and any remaining rows at the bottom of the display are shifted upward with blank rows being shifted into the display below them. The active position remains in the same horizontal position within the highest row that was shifted (which may be blank). If VEM is set then the active row and rows above it are discarded and any re-maining rows at the top of the display are shifted downward with blank rows being shifted into the display above them. The active position remains in the same horizontal position within the lowest row that was shifted (which may be blank). An attempt to delete more rows than is possible blanks the display from, and including, the active row through the bottom or top row, depending on the state of VEM. A default parameter causes one row to be deleted.

Erase In Display (ED)—CSI Ps J

Blanks a region, of the display, specified by the selective parameter. A default parameter causes the region from, and including, the active position through the end of the display to be blanked. The active position does not change.

Parameter	Meaning
0	Blanks the active position and all positions to the end of the display
1	Blanks from the beginning of the display up to, and including, the active position
2	Blanks the entire display

Parameters other than those listed above are ignored.

Erase In Line (EL)—CSI Ps K

Blanks a region of the active row specified by the selective parameter. A default parameter causes the region from, and including, the active position through the end of the row to be blanked. The active position does not change.

Parameter	Meaning
0	Blanks the active position and all positions to the end of the row
1	Blanks from the beginning of the row up to, and including, the active position
2	Blanks the entire row

Parameters other than those listed above are ignored.

Insert Line (IL)—CSI Pn L

Inserts the number of blank rows specified by the single numeric parameter. If the Vertical Editing Mode (VEM) is reset then the active row and all rows below it are shifted downward. The active position remains in the same horizontal position within the first (highest) blank row. If VEM is set then the active row and all rows above it are shifted upward. The active position remains in the same horizontal position in the last (lowest) blank row. An attempt to insert more rows than are being shifted blanks the display from, and including, the active row through the bottom or top row, depending on the state of VEM. Rows shifted out of the display are discarded. A default parameter causes one blank row to be inserted.

Reset Mode (RM)—CSI Ps l

Resets the modes indicated by the selective parameters to their initial states. Four modes have been implemented. When Vertical Editing Mode (VEM) is reset the Insert Line (IL) and Delete Line (DL) controls operate below the active row. When Display Width Mode (AMDDWM) is reset the normal display mode (80 characters per row and only 24 rows displayed) is in effect. When Scroll Mode (AMDSCM) is reset then jump (i.e. non-smooth) scrolling is affected. When Screen Polarity Mode (AMDSPM) is reset then normal characters are shown as light on dark. A sequence with no parameters has no effect.

Parameter	Meaning
7	VEM (insert/delete below active row)
?3	AMDDWM (normal display mode)
?4	AMDSCM (jump scrolling)
?5	AMDSPM (light on dark characters)

Parameters other than those listed above are ignored.

Scroll Down (SD)—CSI Pn T

Scrolls the display downward the number of rows specified by the single numeric parameter. An attempt to scroll the top row of the display downward beyond the top row on the screen leaves it at the top row. A default parameter causes the display to scroll down one row, unless the top row of the display is already at the top row on the screen.

This control has no effect when the message display is active.

Select Graphic Rendition (SGR)—CSI Ps m

Selects the attributes, with which subsequent characters will be displayed, as specified by the selective parameters. A choice between two fonts is also selectable. A sequence with no parameters does not change attributes.

Parameters	Meaning
0	Initial rendition: steady, normal intensity, not underlined, not crossed out, normally aligned, positive image, primary font
1	Bold or increased intensity
4	Underlined
5	Blinking
7	Negative image
9	Crossed out (legible but marked as to be deleted)
10	Primary font
11	Secondary font
22	Normal intensity
24	Not underlined
25	Steady (not blinking)
27	Positive image
29	Not crossed out
?91	Superscript alignment
?92	Subscript alignment
?93	Normal alignment

Parameters other than those listed above are ignored.

Scroll Left (SL)—CSI Pn SP @

Scrolls the display leftward the number of columns specified by the single numeric parameter. An attempt to scroll the rightmost column of the display leftward beyond the rightmost column on the monitor screen leaves it at the rightmost column. A default parameter causes the display to scroll left one column, unless the rightmost column of the display is already at the rightmost column on the monitor screen.

This control has no effect when the window display is active.

Set Mode (SM)—CSI Ps h

Sets the mode indicated by the selected parameters to their alternate states. Two modes have been implemented. When Vertical Editing Mode (VEM) is set the Insert Line (IL) and Delete Line (DL) controls operate above the active row. When the Display Width Mode (AMDDWM) is set the compressed display mode (120 characters per row and all 30 rows displayed) is in effect. When the Scroll Mode (AMDSCM) is set then smooth scrolling is used. When the Screen Polarity Mode (AMDSPM) is set then normal characters are shown dark on light. A sequence with no parameters has no effect.

Parameters	Meaning
7	VEM (insert/delete above active row)
?3	AMDDWM (compressed display mode)
?4	AMDSCM (smooth scrolling)
?5	AMDSPM (dark on light characters)

Parameters other than those listed above are ignored.

Scroll Right (SR)—CSI Pn SP A

Scrolls the display rightward the number of columns specified by the single numeric parameter. An attempt to scroll the leftmost column of the display rightward beyond the leftmost column on the monitor screen leaves it at the leftmost column. A default parameter causes the display to scroll right one column, unless the leftmost column of the display is already at the leftmost column on the monitor screen.

This control has no effect when the window display is active.

Scroll Up (SU)—CSI Pn S

Scrolls the display upward the number of rows specified by the single numeric parameter. An attempt to scroll the bottom row of the display upward beyond the bottom row on the monitor screen leaves it at the bottom row. A default parameter causes the display to scroll up one row, unless the bottom row of the display is already at the bottom on the monitor screen.

This control has no effect when the window display is active.

Private Control Sequences

These are sequences that are introduced by the Control Sequence Introducer (CSI). They may contain parameters just like the standard sequences, but their final characters are in the set which the standard has reserved for private use.

Character Blink Rate (AMDCBR)—CSI Psu

Selects the rate and duty cycle, for characters displayed with the blink attribute, as specified by the selective parameters. Currently blinking characters, as well as those subsequently displayed, will reflect the selection made by this control. A default parameter selects the fastest blink rate and a 25%–75% duty cycle.

Parameters	Meaning
0	Initial character blink: fastest, 25%–75% cycle
11	Blink with 50% active, 50% inactive cycle
12	Blink with 25% active, 75% inactive cycle
20	Fastest blink rate
21	Fast blink rate
22	Slow blink rate
23	Slowest blink rate

Parameters other than those listed above are ignored.

Load Font Cell (AMDLFC)—CSI Pn ... Pn~

Programs one cell of the character generator with the pattern specified by the numeric parameters. When in "normal" display mode the 7•9 display cells are programmed, otherwise the small display cells (5x7) are programmed. The first parameter specifies which character cell is to be programmed. There are 256 chracter cells specified in the range 0 through 255, inclusive. All cells except that at location 32 can be programmed; this is always a blank and cannot be changed. A default for this parameter will cause this control to be ignored. The second parameter specifies at which character cell slice programming is to begin. Slices are numbered downward beginning with zero. Slices above the first slice are automatically blanked. A default for this parameter causes the programmed pattern to begin at the top slice in the character cell. The rest of the numeric parameters each represent a slice of the character pattern. They are decimally encoded byte values for the desired eight-bit slices, with the most significant bit at the left side of the character and the least significant bit at the other side of the character cell. In small display mode, the entire slice (all eight bits) are shown with an additional blank pixel after each character. In large display mode, only the most significant six bits are shown and there is no additional blank pixel. A default for a pattern parameter causes the slice to be blanked. As many slices are programmed as there are parameters supplied, down to the bottom of the character cell. Any unprogrammed slices below the last programmed slice are automatically blanked.

Select Active Display (AMDSAD)—CSI Psp

Makes one of the background, message or window displays the active display. The active display is where the characters being received are stored and where the controls being received perform their functions. The displays each have their own active position and current graphic rendition. The cursor is shown at the active position of the active display, provided that active position is visible. A default parameter makes the background display the active display. This control does not affect message and window display visibility.

Parameter	Meaning
0	Makes the background display active (default)
1	Makes the message display active
2	Makes the window display active

Parameters other than those listed above are ignored.

Select Cursor Appearance (AMDSCA)—CSI Ps v

Selects the appearance of the cursor, which marks the active position, as specified by the selective parameters. The fundamental form of the cursor, as well as whether or not it blinks and at what rate, can be changed. A default parameter selects a steady, reversed block covering the entire character cell.

Parameters	Meaning
0	Initial cursor: steady, reversed, full block
1	Reversed block covering entire character cell
2	Reversed block covering lower half of character
3	Solid block covering lower half of character
4	Underscore
5	Thick underscore
10	Steady, non-blinking
11	Blink with 50% active, 50% inactive cycle
12	Blink with 25% active, 75% inactive cycle
20	Fastest blink rate
21	Fast blink rate
22	Slow blink rate
23	Slowest blink rate

Parameters other than those listed above are ignored.

Smooth Scroll Rate (AMDSSR)—CSI Ps t

Selects the rate at which both vertical and horizontal smooth scrolling occurs as specified by the selective parameters. If more than one parameter is specified then the last one has precedence. A default parameter selects one scanline/pixel per frame.

Parameter	Meaning
0	Initial scroll rate: one scan line/pixel per frame
1	One scan line/pixel per frame
2	Two scan lines/pixels per frame
3	Three scan lines/pixels per frame
4	Four scan lines/pixels per frame
5	Five scan lines/pixels per frame
6	Six scan lines/pixels per frame
7	Seven scan lines/pixels per frame
8	Eight scan lines/pixels per frame
12	One scan line/pixel every two frames
13	One scan line/pixel every three frames
14	One scan line/pixel every four frames
15	One scan line/pixel every five frames
16	One scan line/pixel every six frames
17	One scan line/pixel every seven frames
18	One scan line/pixel every eight frames

Parameters other than those listed above are ignored.

Select Window Visibility (AMDSWV)—CSI Ps r

Makes the window display either visible or invisible as specified by the selective parameter. A default parameter makes the window display invisible. This control does not affect which display is active.

Parameter	Meaning
0	Makes the window display invisible (default)
1	Makes the window display visible

Parameters other than those listed above are ignored.

Select Message Visibility (AMDSMV)—CSI Ps q

Makes the message display either visible or invisible as specified by the selective parameter. A default parameter makes the message display invisible. This control does not affect which display is active.

Parameter	Meaning
0	Makes the message display invisible (default)
1	Makes the message display visible

Parameters other than those listed above are ignored.

7.6 LOW-COST TERMINAL COMPARISONS

This document contains two tables comparing the features of four terminals with the implemented Low-Cost, Smart Terminal based on the Am8052/Am8152A chip set. The purpose is to clarify the relationship of this terminal to other well known alphanumeric terminals. The tables include the DEC VT100 and VT220 and the IBM 3101. All but the IBM terminal are ANSI X3.64 compatible terminals. The IBM terminal claims to adhere to an earlier ANSI and ISO specification; it is similar in some respects to the ADDS Viewpoint or the DEC VT52.

It is very important to understand that the ANSI specification does not define the characteristics of any specific terminal, nor does it require any minimum implementation. Rather, it defines the method of encoding control information which may be sent to, or received from, a terminal. Consequently, a terminal may conform to ANSI X3.64 whether or not it has the ability, for example, to insert a line in a display. If an ANSI X3.64 compatible terminal does have the ability to insert a line in a display, however, then the control which is sent to perform a line insertion must be encoded as specified in the ANSI standard.

In a practical sense, a user of ANSI terminals can write software which performs the most elementary operations (such as cursor positioning) with confidence that they will work on any conforming terminal. There are some slightly more advanced operations (such as insertion and deletion) which may or may not be included in a given terminal, but if present will always be encoded in the standard manner. The user may write "portable" programs which make use of these functions only if he checks carefully for their support on any terminals he wishes to use. Finally, there will be many unique operations for a given terminal (such as window support) which will be represented by "private" extensions that conform to the ANSI standard. User programs which make use of such operations become bound to a particular terminal or its emulators.

From the user's viewpoint, it would be better if there were some truly standard specification of a terminal, for which he could write programs with the expectation that such programs would then be completely "portable" among ANSI compatible terminals. Unfortunately, this is not the situation. Only the method of encoding control information is standardized, not the characteristics or capabilities of a terminal. Still, this is better than the complete absence of standardization. Programs can be written which are reasonably portable and standard modules for sending controls to a terminal can be developed. Furthermore, high-level software simulations of more advanced features, which may be missing in some terminals, can be written to use the simpler features which are present. For these reasons, it is appropriate for developers of new terminals to conform to the ANSI X3.64 standard.

The Low-Cost Smart Terminal, implemented with the Am8052/Am8152A chip set on an IBM-PC board, does have ANSI X3.64 compatible control definitions. Its relationship to other terminals can only be determined by detailed analysis of the characteristics of these terminals. The two tables which form the bulk of this document provide a first level analysis. The first table is a summary of groups of features. The second is a detailed listing of individual controls.

In viewing this comparison, certain general statements can be made. These are:

1. The implemented terminal handles the most common forms of cursor positioning and character display as do all the other terminals.

2. The implemented terminal includes advanced, yet fairly common features such as character assigned attributes, row insertion and deletion, smooth scrolling and a window. The criteria for including these features was that they should relate directly to capabilities of the Am8052. No advanced features have been included "for their own sake" or for compatibility with any other terminal. Such features, since they do not relate to the Am8052, would be primarily a software exercise.

3. The implemented terminal includes some "private" controls for the purpose of demonstrating unique hardware capabilities such as varying the rate of smooth scrolling, smooth scrolling either window or background without affecting the other and horizontal smooth scrolling.

The comparison reveals the original design intent, that it should demonstrate the applicability of the Am8052 to a low-cost terminal while also revealing the advanced features that the use of an Am8052 could bring to such a product.

SUMMARY TABLE

		Am8052	VT100	VT220	IBM
1.	Simple cursor movement and positioning	YES	YES	YES	YES
2.	Additional cursor movement	–	IND & RI only	IND & RI only	within a row
3.	Cursor tabulation movements	–	fore hrz only	fore hrz only	fore & back hrz
4.	Tabulation control	–	hard setup only	simple set & clear	simple set & clear
5.	Insert and Deletes by Row	YES	–	YES	YES
6.	Insert and Deletes by Character	–	–	YES	YES
7.	Unconditional Erasures	display & line	display & line & chr	display & line & chr	display line & field
8.	Conditional Erasures	–	–	display & line	display
9.	Vertical Scrolling	smooth only	smooth & jump	smooth & jump	jump only
10.	Horizontal Scrolling	smooth only	–	–	–
11.	Superscripts and Subscripts	YES	–	–	–
12.	Modes	some	well stocked but most hardware dependent	well stocked but most hardware dependent	–
13.	Character Display Attributes	YES	YES	YES	YES
14.	Selectable Fonts	YES	YES	YES	YES
15.	Alterable Fonts	YES	–	YES	–
16.	Windows	single fixed	simple scrolling region	simple scrolling region	–
17.	Am8052 Dependent Features	special controls defined	–	–	–
18.	Double Height/ Double Width Characters	–	YES	YES	–
19.	Diagnostics and Reports	–	YES	YES	cursor pos only
20.	Miscellaneous	reset	reset comm & specials	reset comm & specials	comm

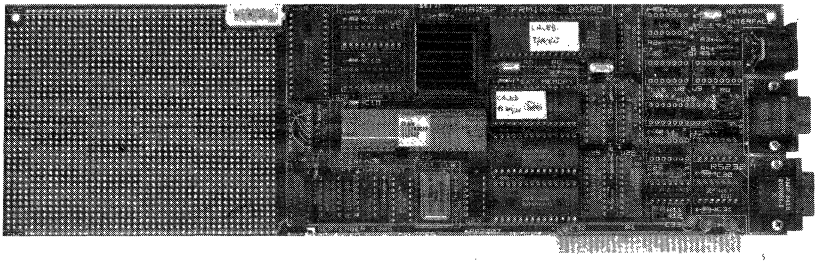
DETAILED TABLE

	Am8052	VT100	VT220	IBM
1. Simple cursor movement and positioning				
Cursor Back	YES	YES	YES	YES
Cursor Down	YES	YES	YES	YES
Cursor Forward	YES	YES	YES	YES
Cursor Position	YES	YES	YES	YES
Cursor Up	YES	YES	YES	YES
Backspace	YES	YES	YES	YES
Carriage Return	YES	YES	YES	YES
New Line	YES	YES	YES	YES
Line Feed	YES	YES	YES	YES
Horz Vert Pos	1)	YES	YES	–
2. Additional cursor movement				
Horz Pos Abs	1)	–	–	YES
Index	1)	YES	YES	–
Reverse Index	1)	YES	YES	–
3. Cursor tabulation movements				
Horizontal Tab	1)	YES	YES	YES
Cursor Backward Tab	1)	–	–	YES
4. Tabulation control				
Clear Tab	1)	–	YES	YES
Set Horz Tab	1)	–	YES	YES
5. Insert and Deletes by Row				
Delete Line	YES	–	YES	YES
Insert Line	YES	–	YES	YES
6. Insert and Deletes by Character				
Insert Character	1)	–	YES	YES
Delete Character	1)	–	YES	YES
7. Unconditional Erasures				
Erase Display	YES	YES	YES	YES
Erase Line	YES	YES	YES	YES
Erase Field	1)	–	–	YES
Erase Character	1)	–	YES	–
8. Conditional Erasures				
Erase Display	1)	–	YES	YES
Erase Line	1)	–	YES	–
9. Vertical Scrolling				
Scroll Down	YES	YES	YES	–
Scroll Up	YES	YES	YES	YES
10. Horizontal Scrolling				
Scroll Left	YES	–	–	–

	Am8052	VT100	VT220	IBM
Scroll Right	YES	–	–	–
11. Superscripts and Subscripts				
Partial Line Down	YES	–	–	–
Partial Line Up	YES	–	–	–
12. Modes				
Reset Mode	YES	YES	YES	–
Set Mode	YES	YES	YES	–
Send-Receive	1)	–	YES	–
LineFeed/NewLine	1)	YES	YES	–
Insert/Replace	1)	–	YES	–
ANSI/VT52	1)	YES	YES	–
Auto Repeat	1)	YES	YES	–
Cursor Key Usage	1)	YES	YES	–
Keypad usage	1)	YES	YES	–
Origin Location	1)	YES	YES	–
Normal/Reverse Display	1)	YES	YES	–
Interlace Display	1)	YES	–	–
80/132 Column Display (120)	YES	YES	YES	–
Jump/Smooth Scroll	YES	YES	YES	–
AutoWrap	1)	YES	YES	–
Print Form Feed	1)	–	YES	–
Print Extent	1)	–	YES	–
Text Cursor	1)	–	YES	–
13. Character Display Attributes				
Select Grph Ren	YES	YES	YES	–
Start Field	1)	–	–	YES
14. Selectable Fonts				
Shift Out	YES	YES	YES	YES
Shift In	YES	YES	YES	YES
Single Shift Two	1)	–	YES	–
Single Shift Three	1)	–	YES	–
Select Character Set	2)	YES	YES	–
15. Alterable Fonts				
Load Font	YES	–	YES	–
16. Windows				
Write to Window	YES	–	–	–
Make Window Visible	YES	–	–	–
Make Window Invisible	YES	–	–	–
17. Am8052 Dependent Features				
Character Blink Rate	YES	–	–	–
Select Cursor Style	YES	–	–	–
Smooth Scroll Rate	YES	–	–	–
18. Double Height/Double Width Characters				
Double-Width Line	1)	YES	YES	–
Double-Height Line	1)	YES	YES	–

	Am8052	VT100	VT220	IBM
19. Diagnostics and Reports				
Screen Alignment	1)	YES	YES	–
Identify Terminal	1)	YES	YES	–
Confidence Test	1)	YES	YES	–
Cursor Position	1)	YES	YES	YES
Report Term Params	1)	YES	YES	–
Request Term Params	1)	YES	YES	–
20. Miscellaneous				
Reset Init State	YES	YES	YES	–
Bell	2)	YES	YES	YES
Enquiry	1)	YES	YES	–
Xon	1)	YES	YES	YES
Xoff	1)	YES	YES	YES
Cancel	1)	YES	YES	YES
Substitute	1)	YES	YES	–
Device Attribute	1)	YES	YES	–
Restore Cursor	1)	YES	YES	–
Save Cursor	1)	YES	YES	–
Load LEDs	1)	YES	–	–

Notes:
1) software driver not implemented, but can be easily added
2) requires additional hardware support
–) not supported



Low-Cost Smart Terminal Demo Board

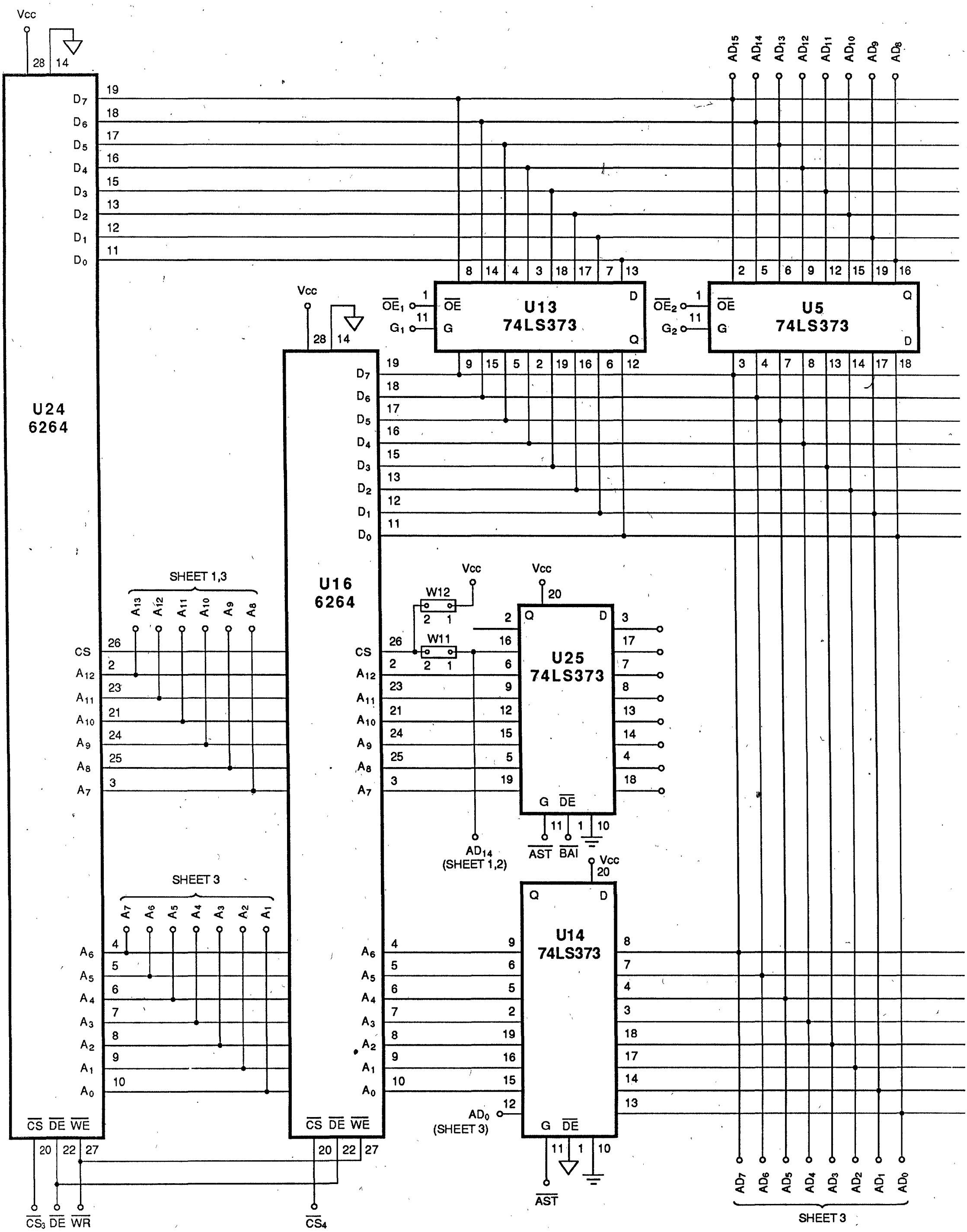
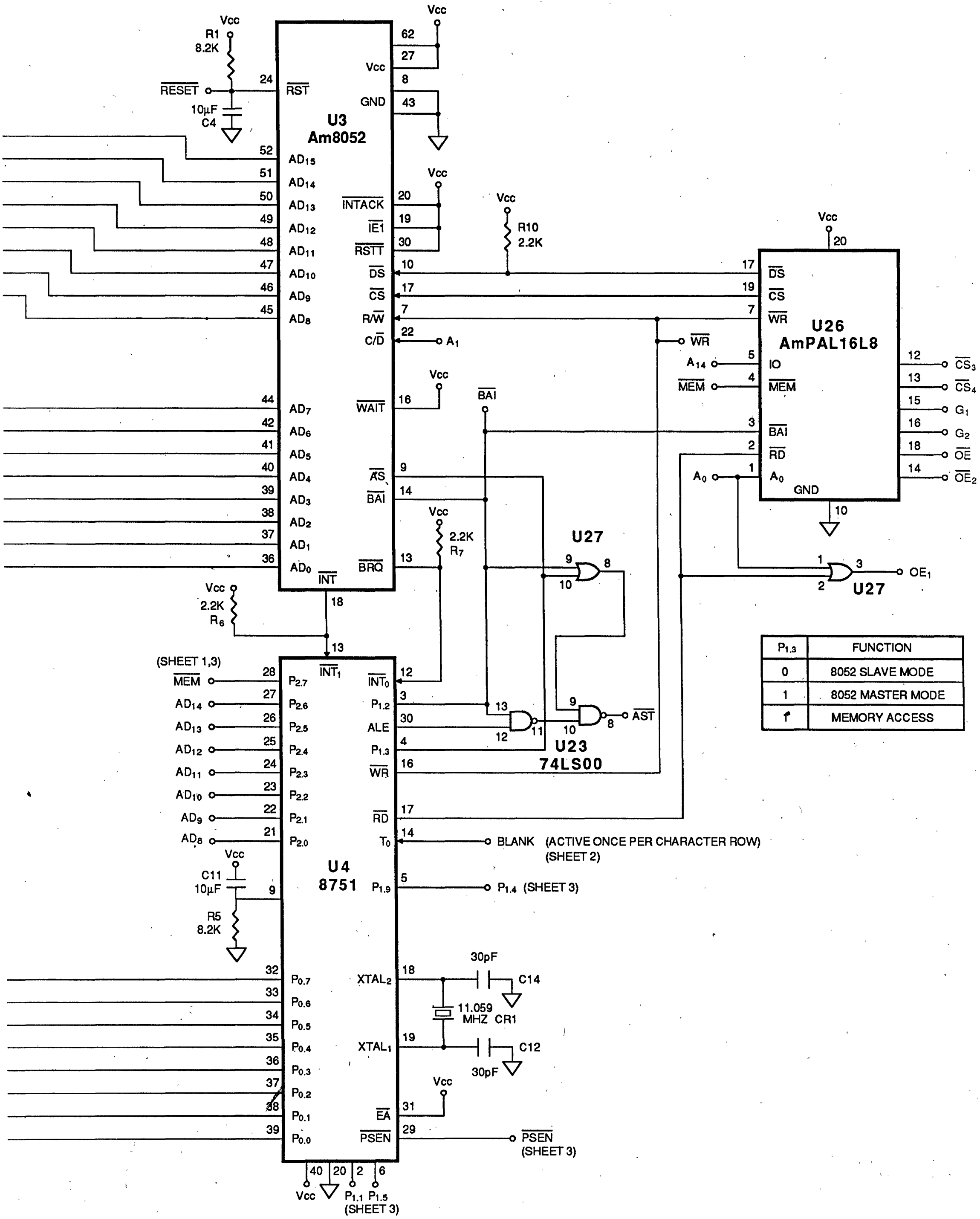


Figure 7-1 Am8052 Terminal Board System Interface



P1.3	FUNCTION
0	8052 SLAVE MODE
1	8052 MASTER MODE
↑	MEMORY ACCESS

Figure 7-1 Am8052 Terminal Board System Interface (Continued)

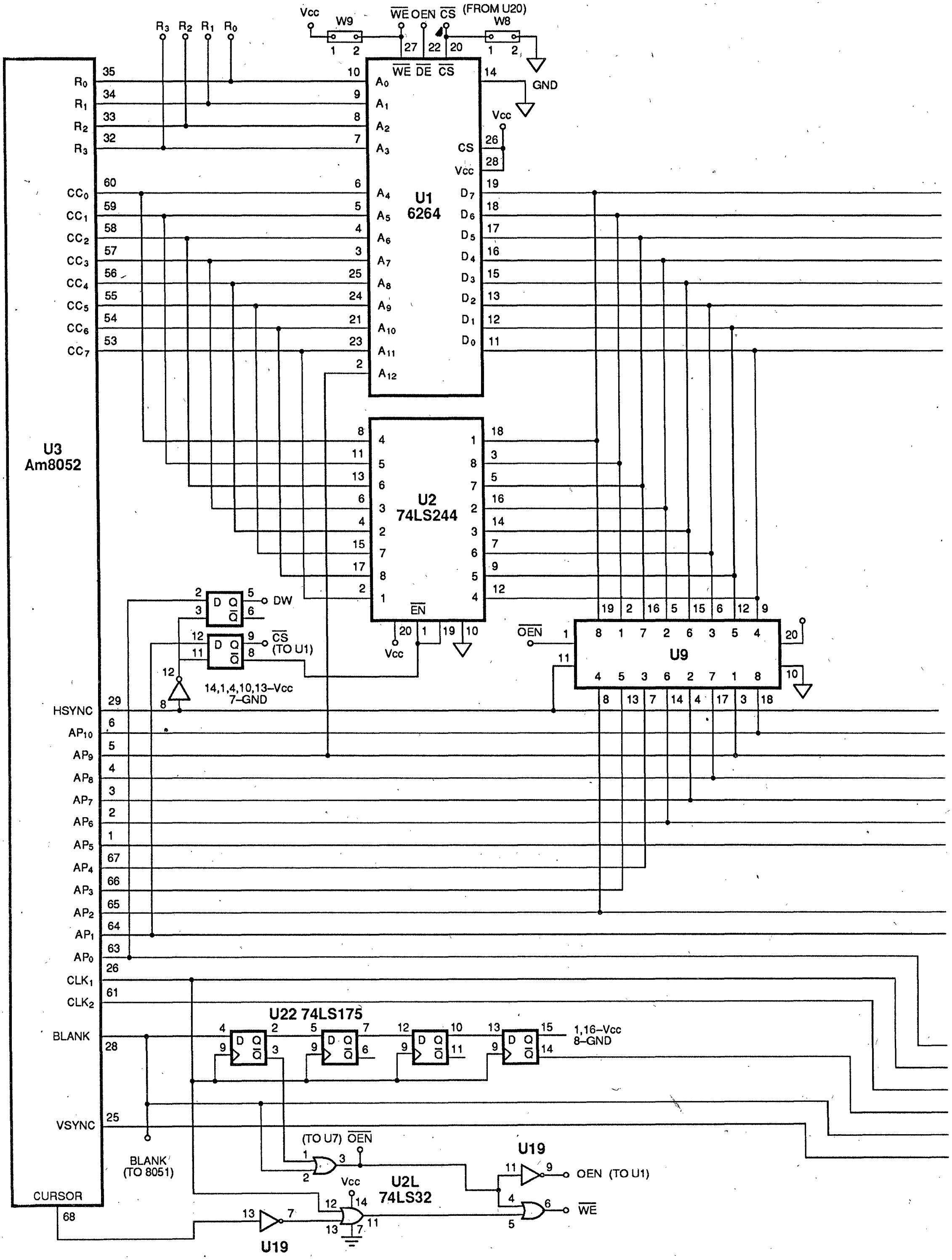


Figure 7-2 Am8052 Terminal Board Video Interface

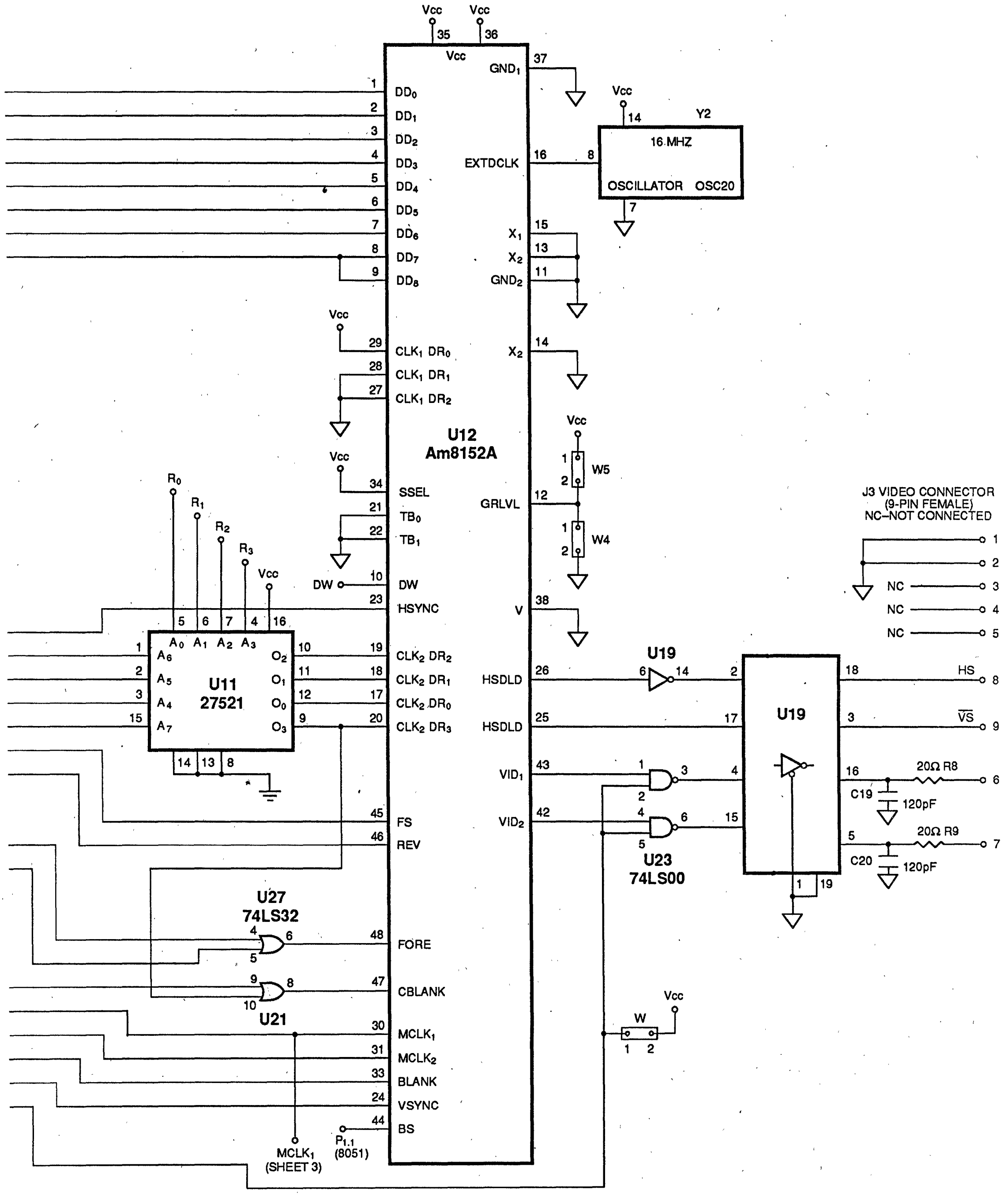


Figure 7-2 Am8052 Terminal Board Video Interface (Continued)

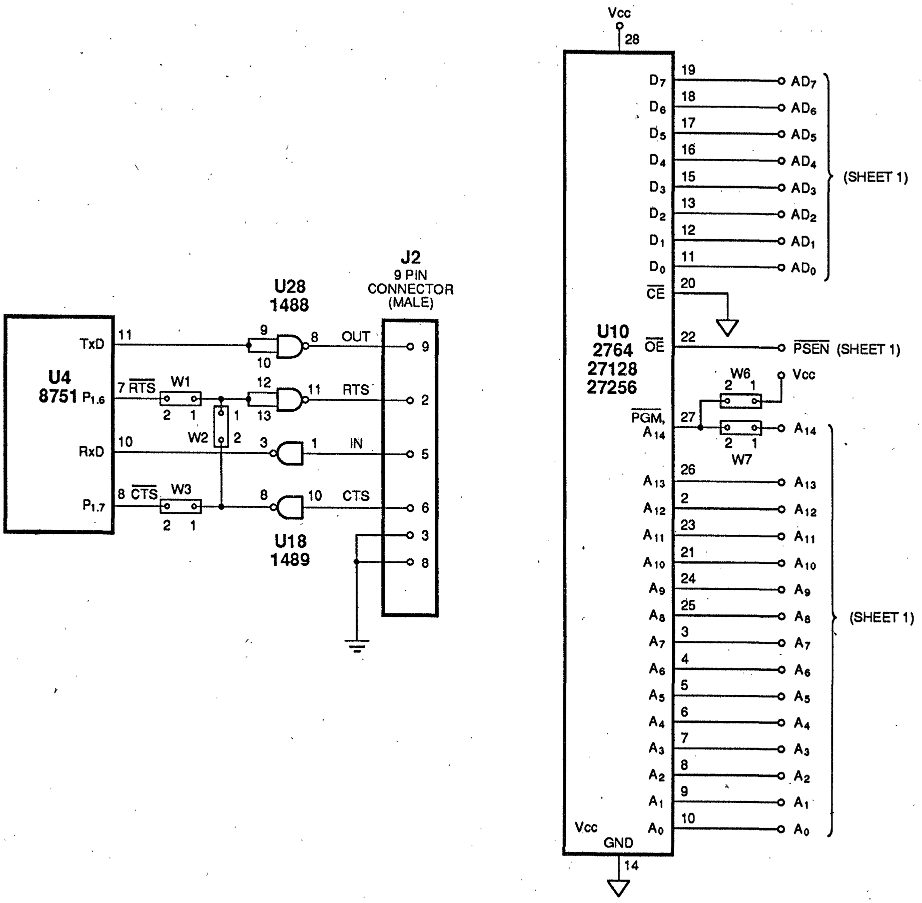


Figure 7-3 Am8052 Terminal Board EPROM and Keyboard Interface

APPENDIX A

Mixing Data Paths Expand Options In System Design

- Mark S. Young and James R. Williamson

MIXING DATA PATHS EXPANDS OPTIONS IN SYSTEM DESIGN

Chip designers are creating powerful CPUs and peripherals with 16- and 32-bit parts. Mixing these with 8-bit parts overcomes limitations imposed by established designs, incomplete families, and software incompatibility.

by Mark S. Young and James R. Williamson

Integrating 16- and 32-bit peripherals and CPUs into 8-bit designs, at the simplest level, means separating the control and data paths from new peripherals and the systems. Mixing different data path widths and control protocols, however, makes possible major improvements in function, performance, and cost.

The price/performance curve of VLSI chips, for example, allows designers to obtain more and better functions for the same amount of money every year. Alternately, the functionality of a device can remain constant while the price falls.

Moreover, these new devices with wider data paths can extend the life of older designs. For example, many of the most popular personal computers today use the 8088 microprocessor and, therefore, are constrained to an 8-bit data path. Designers of add-on accessories for these personal computers prefer the newer 16-bit peripherals. These peripherals will let users preserve their software investments, improve performance, and stave off obsolescence.

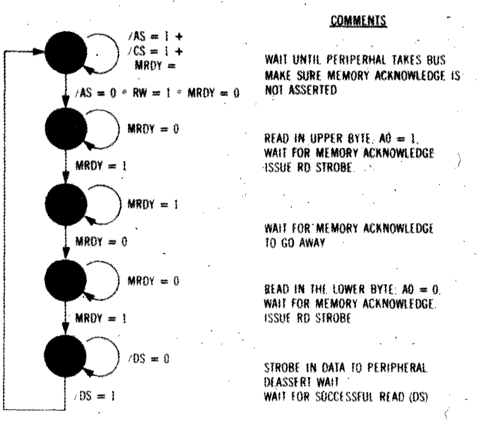
Mixing different data path widths can also enhance new designs. For example, it is less expensive to use an 8-bit bus in a new design because the memory requirements are generally cheaper. Only half as many dynamic RAMs are necessary for the same number of kilobytes of memory. In addition, an 8-bit bus needs much less control and support logic. Designers can mix smaller data path peripherals with wider data path CPUs. This allows them to introduce systems based on the newer, more powerful 32-bit CPUs even before 32-bit peripherals are available.

Designers can use this mixing method to obtain wider data paths from existing designs until a new system design is warranted. They can also use parts in unexpected applications. For example, cost-conscious terminal manufacturers might want to use the Am8052/8152A chip set (the 8052 is an advanced CRT controller and the 8152A is a video system controller) in new terminals based on the relatively inexpensive 8051 microprocessor. Mixing the 8-bit, single-chip microprocessor with the 16-bit CRT controller allows designers to maximize the cost/performance ratio of the terminal.

Mixed data path widths can improve bus utilization as well. A 16-bit peripheral in a 32-bit system only occupies half the data bus for data transfers. If the designer mixes the data paths correctly, however, the 16-bit peripheral could transfer data as

Mark S. Young is a product planning engineer at Advanced Micro Devices, Inc (Sunnyvale, Calif). He holds a BA in computer science from the University of California at Berkeley.

James R. Williamson is an applications engineer at AMD. He holds a BS in electrical engineering from the California State Polytechnic University, Pomona.


COMMENTS
/AS = 1 +
/CS = 1 +
MRDY =
WAIT UNTIL PERIPERHAL TAKES BUS
MAKE SURE MEMORY ACKNOWLEDGE IS NOT ASSERTED
/AS = 0 * RW = 1 * MRDY = 0
MRDY = 0
READ IN UPPER BYTE, A0 = 1.
WAIT FOR MEMORY ACKNOWLEDGE
ISSUE RD STROBE.
MRDY = 1
MRDY = 1
WAIT FOR MEMORY ACKNOWLEDGE
TO GO AWAY
MRDY = 0
MRDY = 0
READ IN THE LOWER BYTE; A0 = 0.
WAIT FOR MEMORY ACKNOWLEDGE.
ISSUE RD STROBE
MRDY = 1
/DS = 0
STROBE IN DATA TO PERIPHERAL
DEASSERT WAIT
WAIT FOR SUCCESSFUL READ (DS)
/DS = 1

The state flow control diagram for a bus master read operation illustrates the control sequence employed by the 8/16-bit bus control logic.

32-bit chunks and improve bus efficiency by 100 percent for that peripheral.

Two central concerns stem from mixing devices that communicate over different-sized buses. The first problem results when two devices communicate on a "common" data bus. Consider, for example, a 32-bit system utilizing 8- and 16-bit peripherals. Overcoming the mismatched data paths requires some form of controlled multiplexing/demultiplexing of the different data paths. In addition, extra control signals for partitioning the 32-bit word into 8-, 16-, and 32-bit chunks may be required.

Many 16-bit CPU-based systems that use 8-bit peripherals normally use just the lower 8 bits of the data bus to transfer data to and from the peripheral. This method does not work in systems using 16-bit peripherals and 8-bit CPUs, however, and it tends to break down in systems with 8-bit peripherals having bus master capability.

A bus multiplexing method involves multiple transfers when taking data from or adding data to a mismatched data bus. For example, before a 16-bit peripheral can transfer data over an 8-bit bus, the 16-bit data must be divided into two 8-bit chunks. It is then transferred sequentially. First, the lower 8 bits are transferred out on the bus. Then, in the next transfer cycle, the upper 8 bits of the 16-bit word are sent out. The major difference in the opposite case—a bus read operation from an 8-bit bus to a 16-bit device—is that the first byte read from the system must be latched. Once the second byte has been fetched, the 16-bit peripheral reads in the assembled 16-bit (2-byte) word. Additional provisions may be needed when the 16-bit peripheral only wants to access a single byte.

The other major problem in mixed data path transfers is the actual data read/write operation. The nature of the multiple transfer forces designers to guarantee that the stretched transfer will occur and that it will not be interrupted. Two aspects of stretching the transfer cycle from or to the peripheral illustrate the complexity of this problem.

The first case, when the peripheral is the bus master, is the simplest. A 16-bit peripheral holds its data available for what normally would be two complete bus transfer cycles. This function can be performed when the transfer acknowledge signal to the peripheral is delayed. If the data was latched instead of holding the peripheral in a multiple word transfer, however, the device could try to send the next 16-bit data word and its "new" address. The procedure of latching the data and releasing the peripheral should not be used, therefore, because it may interfere with the addressing of the remaining (pending) 8-bit transfer.

Whenever a device acts as a bus slave to a CPU that cannot access the device's natural word width in a single operation, a different constraint appears. The sequence must be set up so the peripheral cannot obtain the bus while the CPU is in the middle of a slave read/write operation. In a typical system, the CPU is the last device in the interrupt queue. It is possible for the peripheral to become bus master between the first and second read operations and invalidate the results of the first read operation in a realtime system. This is because an 8-bit CPU would have to perform two consecutive read operations to examine a 16-bit peripheral control register.

This function can be handled two different ways. If the CPU has a bus lock instruction, as in the iAPX family of CPUs, the programmer must use one of these instructions before the CPU accesses the peripheral. Alternately, the CPU needs to disable the arbitration logic while it is performing the uninterruptible access with the 16-bit peripheral.

Crucial cycle

The uninterruptible word transfer cycle is crucial for maintaining the integrity of the data transferred. When either the CPU or a peripheral on the bus makes an access using the 8/16-bit control logic, it must complete the larger device's word access before relinquishing the bus. If this requirement is not met, a transfer's integrity can be violated easily by some other device. This interrupts the transfer, and corrupts or aborts the multiplexing sequence.

To illustrate this point, consider a system consisting of an 8-bit CPU and several 8- and 16-bit peripherals. Assume one of the peripherals is executing a block transfer of 16-bit data onto the 8-bit bus. If the CPU interrupted the transfer in order to poll the peripheral during a half-word transfer, two undesirable events would occur. Either the multiplexing sequence would be damaged irreparably when the CPU polled the peripheral, or the CPU would read garbage from the peripheral.

Designing the control interface to allow mixing of 8- and 16-bit peripherals requires attention to the data and control flow. During a write operation, the data is written out sequentially: the lower byte comes before the upper byte (or vice versa). The read operation differs only because the data bus is 8 bits and because it forgets the last byte transferred; it knows the current byte only. Hence, the interface requires that one of the bytes be latched until the full 16-bit word has been assembled.

The slave mode of operation works almost the same as the peripheral bus master mode. The single exception is the slave write operation. When the interface is defined, the designer must make a conscious choice about which byte (upper or lower) to latch during peripheral read operations (or conversely, slave peripheral write operations). Once this decision has been made, the CPU must always access the latched data byte first (during a slave write) and then access the non-latched byte to complete the transfer. This restriction is minor, requiring no extra software overhead. It could affect the ease of the programmer's coding if not handled properly, however. For example, if the programmer used a compiler to generate the software for the system, extra care may be necessary to ensure the compiler generates the correct addressing sequence.

An alternative solution would be to latch both the upper and lower data bytes. In this case, however, the cost of the interface would increase, as would the complexity, with no appreciable gain. The control flow in these designs derives from two different sources: the state control flow itself and the 16-bit peripheral interfacing with the 8-bit bus. A state diagram can be used to specify how uninterrupted word transfers will occur and how the upper and lower byte address is generated.

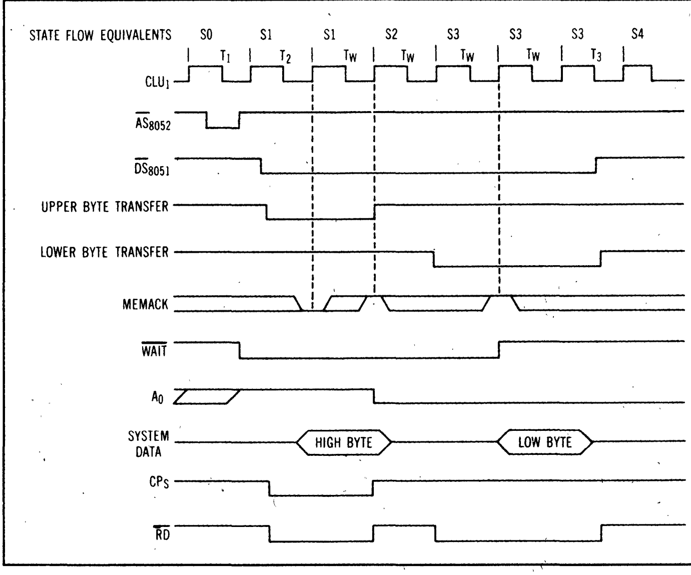
In addition, the specific bus timing of the peripheral and the data bus must be examined to quantify the state control flow. These timing specifics also provide information on data latching, read/write control strobes, and addressing to and from the peripheral. The state control flow is divided into four operations: bus master read, bus master write, slave read, and slave write.

For a bus master read/write operation from a 16-bit peripheral device operating on an 8-bit bus, four control signals must be generated by the 8/16-bit control unit: address bit 0 (A0), peripheral hold ($\overline{\text{WAIT}}$), bus read ($\overline{\text{RD}}$), and bus write ($\overline{\text{WR}}$). The A0 line is generated by the 8/16-bit control logic to indicate which byte is to be transferred in bus master modes only. Otherwise, the A0 generated by the system is used to indicate which byte is being accessed. The $\overline{\text{WAIT}}$ line holds up the peripheral during transfers. The $\overline{\text{RD}}$ and $\overline{\text{WR}}$ lines are required to indicate successive transfer cycles on the bus.

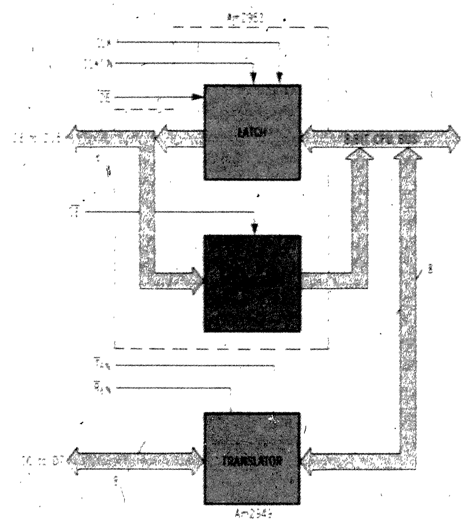
Hidden transfers

The peripheral's signals will only strobe active once because it does not know that two transfers are being executed. The slave transfer flows are almost identical, except the CPU is generating the bus signals and the transfer directions are reversed (ie, a bus write goes into the peripheral).

For this 16- to 8-bit data flow example, the data on the upper byte only needs to be latched when data



In addition to a state flow diagram, a timing diagram can be used to describe such data read/write operations as a master bus read.



The 16- to 32-bit conversation logic diagram indicates the complexity of bus and funnel logic control. It must convert between different signal conventions and polarities as well as generate extra functions and bus arbitration control signals.

is being read (as bus master) or written (as a bus slave). An interface to handle this operation needs to latch data coming from the 8-bit data bus into the peripheral, it also needs to act as transceiver when the peripheral is sending data out to the system. A device with a clocked, tri-state output that has an 8-bit wide latch in one direction and a tri-state transceiver in the other direction would be ideal for accomplishing such an interface.

The Am2952 8-bit bidirectional I/O port provides a good enough match to the logic and allows the upper data bus latch and upper data transceiver chips to be combined on one IC. It provides two 8-bit clocked I/O ports, each with tri-state output controls and individual clocks and latch enables. An Am2949 bidirectional bus transceiver completes the logic required for the data path function.

The state flow control requires logic that can move sequentially from state to state, hold in a particular state, and be reset or initialized back to a predefined state. Depending on the number of states required (generally less than 16 distinct states for a design of this complexity), a 3- or 4-bit counter should be able to solve the problem nicely.

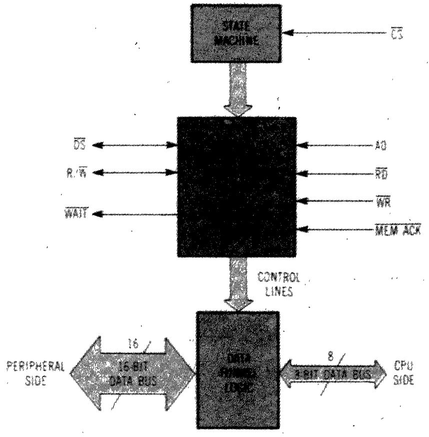
Considerable bus control logic is required to generate the data path flow logic and the bus control signals. This is especially true if the peripherals and CPUs use different signal conventions (eg, when $\overline{AS}$, $\overline{DS}$, and R/W use address latch enable, $\overline{RD}$, and $\overline{WR}$). Conversion from one signal convention to another, changes in signal polarity, and provision for extra functions (such as generating A0) require a lot of logic synthesis ability. If the peripheral has bus master capability, such additional information as bus arbitration controls must be fed into the next state determination logic in order to decide what control sequence to follow.

Customized interface minimizes cost

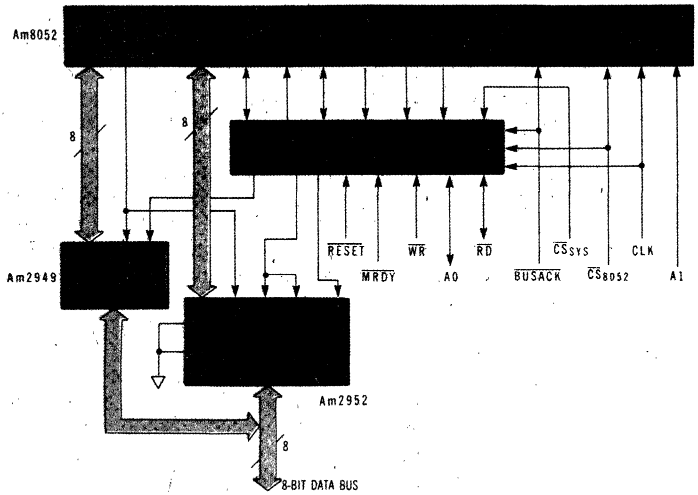
An 8/16-bit control interface between the Am8052 CRT controller and an 8-bit CPU provides a good example of how customizing a general interface can reduce costs. (The CRT controller is designed with a 16-bit data interface.) The onboard DMA unit fetches data from system memory and the CPU polls the CRT-controller's internal status and control registers. Because the CRT controller does not modify system memory, however, a bus master write operation is unnecessary. Thus, there is no reason to generate a system write control signal (WR).

In addition, the control and display information must be aligned on word boundaries. This requirement relieves the 8/16-bit control logic from funneling the bytes and performing odd/even byte transfers. It also saves control inputs from the CRT controller because all transfers are words; that is, no need exists for upper and lower data strobes or byte high enable inputs.

The bus master read operations are standard 16-bit data transfers divided into two 8-bit transfers. The CPU's slave accesses are either pointer writes (to select the desired control/status register) or 16-bit data read/write operations. (Pointer write operations



The state machine and the bus and latch controls have to be coupled in order to transfer data between the 8- and 16-bit buses. This generalized machine is designed with the assumption that the peripheral has bus master capability. If this is not the case, the design can be greatly simplified.



The logic for control and data transfer between an Am8052 and 8-bit CPU has the control interface implemented in an AmPAL22V10.

are actually 8-bit operations because only the lower 8 bits of the data form the register address.) The bus master read operation can be represented by a state flow diagram or a timing diagram. Conceptually, state flow diagrams are easier to understand, but timing diagrams usually convey more information. Other state flow diagrams can be derived directly from the timing diagrams of the CRT controller to 8-bit interface.

Simplifications allow synthesis on one device

Two special conditions must be met in the state machine implemented in the 8/16 interface. First, before a new transfer cycle is attempted (when the state machine is waiting in the initial state, S0), memory acknowledge ($\overline{MRDY}$) must be inactive. This prevents interference from the last transfer.

The second special condition occurs when the CRT controller asserts the R/$\overline{W}$ line to indicate a write operation. Although the CRT controller does not write data into system memory, when it updates the upper 8 bits of the 24-bit address latch the R/$\overline{W}$ line indicates a write operation (in conjunction with $\overline{AS}$). The CRT controller is not actually performing a system data write, only an address latch update. The state machine, therefore, must not start a bus sequence if the R/$\overline{W}$ line is held active low by the CRT controller during a bus master operation.

These simplifications in design allow the CRT controller to 8-bit CPU control interface to be synthesized in a single AmPAL22V10 programmable logic array device. In addition, the bus control signals are converted from $\overline{AS}$, $\overline{DS}$, and R/$\overline{W}$ to $\overline{RD}$ and $\overline{WR}$. The minimum CRT controller and bus control signals that must be generated are $\overline{RD}$, A0, $\overline{DS}$, and R/$\overline{W}$. Although the CRT controller uses $\overline{DS}$ and R/$\overline{W}$ as inputs during a bus master operation, the PAL device must convert the CPU $\overline{RD}$ and $\overline{WR}$ signals to $\overline{DS}$ and t/$\overline{W}$ for slave I/O operations.

The signals A0 and RD are generated by the control logic when the CRT controller is performing a read access to system. The $\overline{WAIT}$ (or not READY) signal to the CRT controller must also be generated by the control logic. The data flow controls require six additional controls to load and strobe the latch, and to enable transceivers to pass data to and from the 8-bit bus. Theoretically, 4 more bits (outputs) are required to represent all the control states needed to manipulate the 8/16-bit control logic. This means the design appears to need 14 output logic units in a PAL device to perform the required task.

Reducing the 14 output cells to the 10 cells available in the PAL device requires a closer look at the timing and output switching functions. The A0 and $\overline{RD}$ control lines are in effect part of the system bus control and, therefore, cannot be multiplexed easily. The $\overline{DS}$ and R/$\overline{W}$ lines to the CRT controller are also fixed because they must be valid throughout the entire transfer cycle as well.

This leaves 6 of the 10 output logic cells of the PAL device to represent the remaining 10 identified control lines. This method of minimization involves careful state synthesis, analysis of the signal switching functions during the transfers, and utilization of several control pins on the CRT controller. By using the $\overline{BREQ}$, $\overline{BACKI}$, $\overline{BACKO}$, $\overline{CS}$, and C/$\overline{D}$ inputs to the PAL device, we can reduce the number of unique states required to 8 instead of 15. This reduces the number of logic cells required for the state machine from 4 to 3 bits.

At this stage, the design requires seven control signals to manipulate the data transfer registers and $\overline{WAIT}$ line. The two latch enables ($\overline{CE}_S$ and $\overline{CD}_R$) on the Am2952 bidirectional I/O port can be permanently enabled. By controlling the clock signal to the latches, the controls required for three pins can be reduced to one. The interface control state machine will only use the correct side of the dual latches on the bidirectional I/O port.

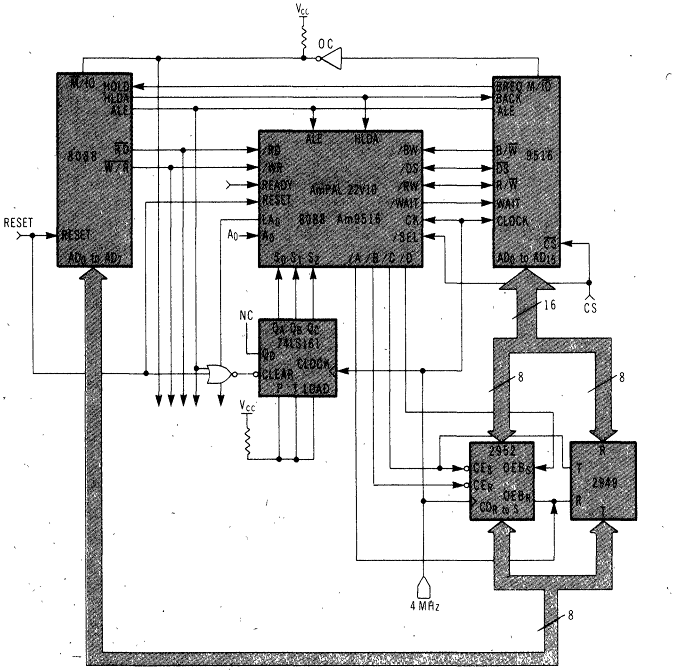
The Am8052 CRT controller helps considerably with its own control bus interface. Two signals provided by the CRT controller, $\overline{\text{TBEN}}$ and $\overline{\text{RBEN}}$, switch the data transceivers in the correct direction regardless of the type of data transfer (as a bus master or bus slave). When the controller is a bus master performing a read operation, or when it is a bus slave undergoing a write operation, therefore, the $\overline{\text{RBEN}}$ signal is strobed to obtain the correct polarity. By using this line, two of the remaining six control lines can be eliminated ($\overline{\text{REN}}$ on the Am2949 and $\overline{\text{OE}}_{\text{AS}}$ on the Am2952). Although the $\overline{\text{TBEN}}$ line performs a similar function, it does not function correctly in a 16- to 8-bit multiplexed bus environment.

Two of the remaining control lines ($\overline{\text{OE}}_{\text{AS}}$ on the Am2952 and 10 on the bidirectional bus transceiver) must be generated by individual cells in the PAL device. The two clock enables on the Am2952 are permanently enabled. The two Am2952 clocks are tied together to minimize the amount of logic required in the PAL device used to generate clock strobes to the latches.

This leaves the design with three logic cells and four output functions (the $\overline{\text{WAIT}}$ line to the CRT controller and the 3 state bits). Careful analysis of the state flows and timing diagrams indicates that the $\overline{\text{WAIT}}$ line is only asserted in 4 of the 8 states. A clever assignment of state numbers to the state flow sequence allows the $\overline{\text{WAIT}}$ line to be absorbed into the 3 state encoding bits. The logic equations for the AmPAL22V10 device can be derived directly from the timing diagrams.

An unusual problem might occur when a peripheral device operates as a bus slave on a smaller data bus, such as a 16-bit peripheral to 8-bit CPU. During the first slave write operation, the chip select $\overline{\text{CS}}$ is enabled by the bus master making the access. No actual data—just the data latch—is strobed into the peripheral, however. After the first byte of data has been written, the second access causes the full 16-bit data to be strobed into the peripheral.

If the designer is using a common $\overline{\text{CS}}$ function to both the peripheral and the 8/16-bit control logic, the controller logic must be designed not to glitch or strobe any of the control lines to the peripheral (it must prevent $\overline{\text{DS}}$, R/$\overline{\text{W}}$ from being enabled, for example). For some peripheral devices, glitches on the control lines might cause the register to be written accidentally onto a register that will be overwritten in the next write cycle anyway. With other peripherals this might be a catastrophic event. Many devices acting as bus slaves have write recovery time requirements (ie, a certain minimum interval between consecutive write operations). Glitches on the control lines might force the next (and final) write operation to be delayed—or cause a violation of the



The data bus and control interface between an 8-bit 8088 CPU and a 16-bit Am9516 DMA controller uses an AmPAL22V10 for control, and a 74LS161 for state sequencing along with a bidirectional I/O port and transceiver.

device specifications. Glitches might evade any special addressing/register accessing scheme used in the peripheral. This might occur, for example, if the slave device requires the user to write the address of the register that was accessed immediately before the register was written. In this case, glitches or useless control strobes could wreck the sequence.

The problem can also be solved by using two lines. In this solution, one of the lines would go to the peripheral device and the other would connect to the 8/16-bit controller. The chip select to the peripheral is activated each time a slave read occurs (for both upper and lower byte accesses), or when a slave write operation occurs and the unlatched 8-bit data is being written. The chip select function to the 8/16-bit controller is chosen each time the peripheral is selected normally (for slave read/writes on both upper and lower 8-bit data transfers). This problem is bypassed completely when two separate chip select functions are used: one for loading up the Am2952 latch during a slave write/read and one to strobe the Am8052 controller into action when it is needed by the 8-bit CPU.

```
PIN
          CK      = 1     /RD    = 23
          S[0:2]  = 2:4   /WR    = 22
          A0      = 5     LA0    = 21
          /SEL    = 6     /DS    = 20
          ALE     = 7     /RW    = 19
          HLDA    = 8     /WAIT  = 18
          /BW     = 9     /A     = 17
          READY   = 10    /B     = 16
          RESET   = 11    /C     = 15
                          /D     = 14;
BEGIN
          IF (RESET) THEN ARESET( );
This section defines the wiggles when the Am9516 is bus master

          IF    (HLDA) THEN ENABLE( );
          IF    (/S[2] * HLDA) THEN BEGIN
                IF (S[1] * /S[0]) THEN
                      LA0    = /CK * BW + /BW * A0 *
                               ALE + / BW * LA0 * /ALE  ;
                ELSE
                      LA0    = BW + /BW * A0 *
                               ALE + /BW * LA0 * /ALE   ;
          END;
          IF    (HLDA) THEN
                (CASE) (S[2:0])
                      BEGIN
                        1) BEGIN
                        RD   = /RW * DS                 ;
                        A    = /BW * /RW * /CK          ;
                        WR   = /BW * RW * DS            ;
                        C    = /BW * RW                 ;
                        WAIT = 1                        ;
                      END;
                        2) BEGIN
                        RD   = /RW * DS                 ;
                        B    = BW                       ;
                        A    = /BW * /RW                ;
                        WR   = /BW * RW * DS            ;
                        C    = /BW * RW
                        WAIT = /BW
                      END;
                        3) BEGIN
                        RD   = /RW * DS * B             ;
                        B    = BW * CK                  ;
                        A    = /BW * RD                 ;
                        WR   = /BW * RW * DS            ;
                        C    = /BW * RW                 ;
                        WAIT = BW                       ;
                      END;
                        5) BEGIN
                        RD   = /RW * DS                 ;
                        A    = /BW * /CK                ;
                        WAIT = BW                       ;
                      END;
                        6) BEGIN
                        RD   = /RW * DS                 ;
                        A    = /BW                      ;
                      END;
                        7) BEGIN
                        RD   = /RW * DS                 ;
                        A    = /RD                      ;
                      END;
          END;
  This section defines the wiggels when the 8088 is bus master
          BEGIN
          LA0= A0 * ALE * SEL + LA0 * /ALE * SEL
          B  = LA0 * WR * SEL
          A  = /LA0 * WR * SEL
          DS = A + /LA0 * RD * SEL
          C  = /LA0 * RD * SEL
          D  = LA0 * RD * SEL
          END;
  END.
```

This PLPL file implements an interface between the 8-bit 8088 and the 16-bit Am9516.

Bus conversion maximizes flexibility

A data bus and control interface to an 8088 8-bit microprocessor and Am9516 16-bit DMA controller can be created using four devices: an AmPAL22V10 for the control block, a 74LS161 counter for the state sequencer, an Am2952 bidirectional I/O port, and an Am2949 bidirectional transceiver.

This design incorporates certain simplifications. The DMA controller requires word accesses only during command chaining and for slave register accesses. The 8/16-bit data transfer interface for bus master operations (ie, DMA data transfer functions) is handled automatically as a programmable option. During slave write operations, the first byte output to the DMA controller must have an odd address and the following second byte an even address. Conversely, during a slave read cycle, the first byte read from the DMA controller must be at an even address and the second at the next higher odd address.

Furthermore, for bus master operations, the system must use the latched address line A0 (LA0) from the AmPAL22V10 as its sole A0. Because the logic is already available, the system does not have to provide this function. LA0 now becomes the system address bit 0 with full 24-mA drive capability.

Deciding on a means for controlling the funneling of the data stream—that is, transforming 16-bit data into 8-bit data and vice versa—was the first step in deriving this example. As mentioned earlier, simply dividing each 16-bit access into two 8-bit data transfer cycles presents one way of doing this. On outgoing accesses (16-bit path from the DMA controller) during the first cycle, the upper half of the 16-bit path is latched while the lower half passes through

Programming the PAL and the counter

In writing the Programming Language for Programmable Logic (PLPL) file to control the operation of the AmPAL22V10 and the 74LS161 counter, the inputs to the PAL device from the counter are assigned S0, S1, and S2, respectively. Then, it is possible to apply a "sculptured design" technique to the entire timing diagram (see figure in Panel, "A matter of timing") by using the Case statement from PLPL. By assigning combinatorial equations to only one binary partition or column at a time (Case), the designer can ignore all other aspects of the design for the time being and generate simple equations directly from the timing waveforms.

During clock time T1 of the Am9516's word read cycle the state of the 74LS161 (S0, S1, S2) is cleared to 000 by the assertion of address latch enable (ALE). LA0 is the only output control signal from the CRT controller asserted during this period. This signal is handled as a special case, however. During time T2 of the DMA controller's word read cycle, the $\overline{RD}$ and $\overline{WAIT}$ outputs from the CRT controller must be asserted. This time partition corresponds to the state inputs S2, S1, S0 = 001. Therefore, the first Case equations are

```
CASE    (S[2:0])
  BEGIN
  1) BEGIN
       RD = /RW*DS  ; Transform Control
                    ; Signals /RW and DS
                    ; into Intel /RD

       WAIT = 1     ; Assert     Wait
                    ; unconditionally

  END;
```

During time T2 of the DMA controller's byte read cycle, $\overline{A}$ is the only additional output not already accounted for in the Case statement. This signal allows a byte of data to flow through the bidirectional bus transceiver into the DMA controller during byte read operations. Some additional constraints are placed on this signal, however: it must only be asserted in time T2 on byte read operations (the B/W input) and it must be delayed by a half clock period from the rising edge of T2 ($\overline{CK}$ signal). Thus the Case statement becomes

```
CASE  (S[2:0])
  BEGIN
  1)  BEGIN
        RD = /RW*DS          ;
        A = /BW*/RW*/CK      ; enable the
                             ; receiver

        WAIT = 1

  END;
```

Finally, by examining the last time T2 elements ($\overline{WR}$ and C) during the DMA controller's byte write cycle, the remaining terms in Case 1 are derived. With the exception of LA0, the remaining equations were developed in the same fashion. Clearly, this "sculptured" technique is a very simple and methodical means for arriving at the Boolean requirements for a logic block.

As the PLPL listing shows, the signal LA0 was handled slightly differently from the previously discussed method. The number of product terms generated via the Case statement made this approach necessary. The number exceeded the upper limit (16 terms) for a programmable logic array. As a practical matter, therefore, it was necessary to optimize this signal manually. However, it should be noted that this step will not be necessary once the fully optimized version of PLPL becomes available.

a tri-state buffer onto the 8-bit bus. During the second cycle, the tri-state buffer is turned off and the previously latched half of the data is driven onto the bus. On incoming accesses (8-bit path to 16-bit path), the process is reversed.

The control mechanisms that perform this cycling depend on the WAIT and R/W signals passing to and from the DMA controller, and on the ability to enable or disable the latches and transceivers selectively. The Am2952 bidirectional I/O port was chosen because of its dual registers and its flexible control. The AmPAL22V10 device was chosen to match the required number of control pins and functions. Since the complexity of this design requires the use of all of the PAL's I/O pins for control functions, however, it was necessary to use a 74LS161 counter to provide the state sequencer function.

Programming with PLPL

It has long been the logic designer's "art" to merge the often very different concepts and notations of timing information with Boolean logic. Yet, the evolution of a syntax to fully express this art has taken a long time. AMD recently developed such a language for programming the AmPAL22V10, however.

"Programming Language for Programmable Logic," or PLPL, allows the designer to specify a design using multiple input formats. This specification flexibility supports the variety of design approaches necessary to express different design problems efficiently. These formats range from simple sum-of-products Boolean equations to high level constructs. PLPL also supports the input specifications for many types of AND/OR based devices, including all of the current AMD programmable logic array and PROM devices,

PLPL is block structured, and includes the high level language constructs If-Then-Else, Case, and For; all familiar to many programmers of the C and Pascal languages. Macros, functions, constants, and variables may also be used in PLPL. The language also facilitates use, clarity, and self-documentation.

Such current programmable logic technology and associated programming languages as PLPL allow

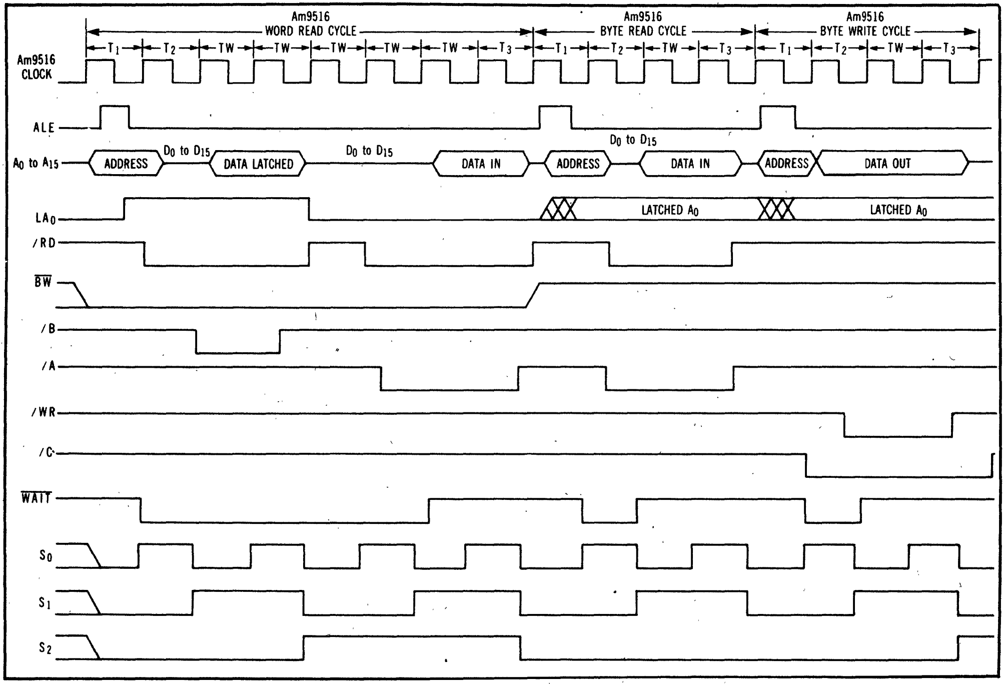
A matter of timing

The complex AmPAL22V10 design used the accompanying timing diagram to correspond to the desired waveforms. They are partitioned by the respective binary state (or count) from the counter.

The desired timing requirements during the period when the DMA controller is bus master appears below. During time T1, address latch enable (ALE) is asserted by the DMA controller to denote the beginning of the cycle; a short time later, an address is driven onto the bus. This address is valid at the falling edge of ALE. The control signal LAO (latched A0), therefore, must be valid at this time, as well. In this phase of the cycle, it must also be high to enable the odd byte from memory to be loaded into the bidirectional I/O port. In addition, the assertion of ALE performs the function of resetting the 74LS161 counter to 0000 in order to synchronize the cycle.

During time T2, the DMA controller will assert its $\overline{\text{DS}}$ signal. The timing for this signal, in conjunction with the R/$\overline{\text{W}}$ signal (asserted in T1) must be transformed into an 8088-equivalent $\overline{\text{RD}}$ signal. During a word read cycle, this $\overline{\text{RD}}$ signal also must be artificially negated and then reasserted to accomplish a double byte read. At the same time, the DMA controller must be "parked" in order to multiplex or assemble a word. Thus, the $\overline{\text{WAIT}}$ signal is also asserted at time T2. During time TW (S2, S1, S0 = 010), the receiver clock enable control signal $\overline{\text{B}}$ must be asserted in order to allow the next system clock's rising edge to strobe the upper byte into the bidirectional I/O port. This is accomplished during the next TW period (S2, S1, S0 = 011).

During the remainder of the word read cycle, $\overline{\text{RD}}$ is negated and then reasserted after LA0 has been forced low to address the even byte. $\overline{\text{A}}$ is then asserted to allow both the previously latched upper byte and the current lower byte to be driven onto the DMA controller's pins. And finally, the $\overline{\text{WAIT}}$ signal is negated, allowing the DMA controller to finish its read cycle by strobing in the 16 bits of command data on its data pins.



highly organized application-oriented control blocks to be formed easily. These tools can conceptually raise the designer above the details of the design at the logic level and directly translate the necessary response characteristics from a timing diagram.

This approach can be referred to as a "sculptured design" technique because it is analogous to the way solid stone is formed according to an artist's image. Raw logic can be transformed directly into useful control functions from the desired timing information.

The AmPAL22V10 is, in essence, a fuse-programmable gate containing up to 22 inputs and 10 outputs. It can define and program that architecture of each output on a pin by pin basis. Thus, the designer is free to optimize the design mix between registered and combinatorial functions as needed.

The AmPAL22V10 is programmed by opening fusable links in any or all of its 10 output macrocells, as well as in its AND gate array. The AND gate structure is very similar to other PAL devices; therefore

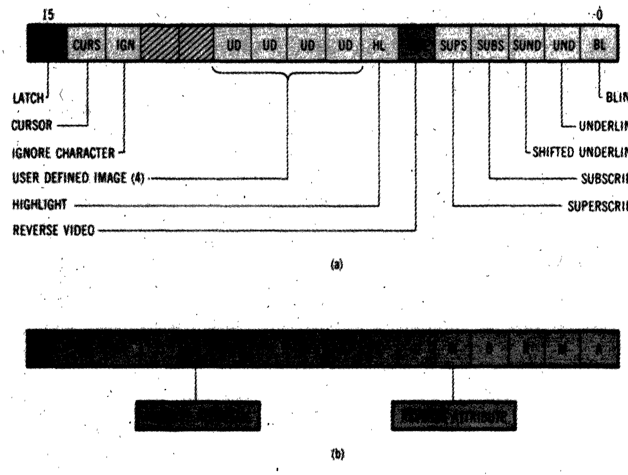


Fig 4 A 16-bit character attribute affects each individual character as it is output from the CRT controller (a). In memory, however, each new character need not invoke a new attribute. In example (b), the latch attribute, in conjunction with the reverse attribute, allows a string of characters to be displayed in reverse video without each character having to be individually reversed.

displayed white on black. Proportional spacing is achieved by altering the CLK2 input to the Am8052. The CLK2 spacing can be made to be as narrow as 2 pixels, or as wide as 17, assigning each character a width value that can be used to program the CLK2 output of the Am8153. Proportionally spaced video characters allow the screen to be formatted similar to the output of a proportionally spaced printer. Thus, proportionally spaced text can be composed accurately on the screen, prior to printing.

The CLK2 output of the Am8153 can be further modified by trailing blanks. Any number of blank pixels, between 0 and 3, can be inserted after the visible character. This allows the user to implement a smooth right justification of text, without inserting blank characters between consecutive words.

In addition to handling characters, the controller chip applies innovative techniques to the raster scan. It provides programmable horizontal synchronous (HSYNC), vertical synchronous (VSYNC), and BLANK signals, and accepts an external synchronization input. This input allows the frame to be synchronized to some external source such as line frequency, which prevents annoying interference display patterns known as "swimming."

Beyond supporting the more common noninterlaced and interlaced modes of operation, the chip also has a repeat field interlace feature that has each character row effectively repeated and offset by the scan line. This has the effect of making a vertical stroke on the screen look more solid, to match the horizontal strokes.

Reprinted with permission from COMPUTER DESIGN

ADVANCED
MICRO
DEVICES, INC.
901 Thompson Place
P.O. Box 3453
Sunnyvale,
California 94088
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL FREE
(800) 538-8450

APPENDIX B

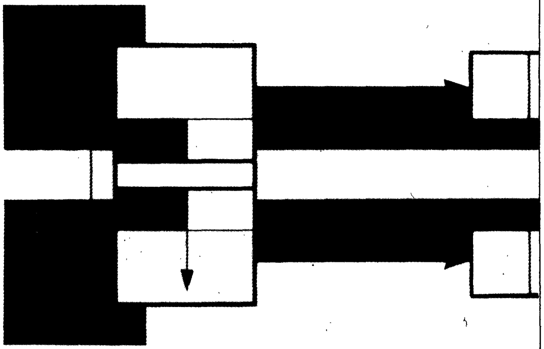
Chip Set Gives A Smooth Scroll In CRT Displays

-Steven Dines and Mohammad Maniar

CHIP SET GIVES A SMOOTH SCROLL IN CRT DISPLAYS

Two large scale integration chips and a read only memory font generator interface 16-bit processors with CRTs directly to control scrolling in multiple windows and to space characters proportionally.

by Steven Dines and Mohammad Maniar



Marrying state-of-the-art display technology and computational capability in today's terminal requires a large data handling capability. Features such as a noninterlace flicker-free frame refresh and a full-page graphics representation dictate high dot update rates in the 100-MHz range. This speed can only be handled by emitter coupled logic chips with all of their attendant problems. Similarly, embedded local editing intelligence places severe constraints on a terminal's microprocessor subsystem, which must efficiently handle such interactive tasks as insertions and deletions.

These and many other obstacles have been solved by a 2-chip cathode ray tube (CRT) controller set that combines the advantages of N-channel metal oxide semiconductor and bipolar technologies. The two chips, together with an offchip font generation circuit, form a complete CRT interface between the microprocessor bus and the monitor (Fig 1). In this application, the Am8052 CRT controller is used as a direct memory access (DMA) controller. This has two advantages: first, it eliminates a separate DMA controller, thereby keeping costs down and saving space in the CRT terminal. Second and more significant, the DMA channel on the CRT controller can be customized to facilitate the controller's editing functions. Thus, a font-control read only memory allows a full video subsystem to be built that matches display data formats with printed information.

Steven Dines is currently a department manager at Advanced Micro Devices Inc, 901 Thompson Pl, Sunnyvale, CA 94086, where he is responsible for microprocessor peripheral product planning. He holds a BSEE from the University of Leeds and an MSEE from the University of Manchester, England.

Mohammad Maniar is supervisor of MOS microprocessor design engineering at Advanced Micro Devices. He holds a BS in electrical engineering from NED Engineering College, Pakistan, and an MSEE from the University of California, Berkeley.

The DMA channel is configured as a linked-list processor, which sets up the display data with minimal editing overhead. This channel fetches data into onboard buffers that store three rows of character information. Incorporating triple row buffers onchip solves a major impediment to a pleasant-looking display: it allows the user to scroll smoothly in a split-screen application, which has always been a major problem in screen formatting.

Parallel pixel data emerge from the font generator and are serialized by the CRT controller set's second chip, the Am8153. All clocks for the system are also generated here. These consist of a 100-MHz pixel or dot clock, and two subclocks, the Am8052 CLK1 bus clock and CLK2 character clock. Emitter coupled logic (ECL) outputs in the Am8153 obviate the need for peripheral ECL output devices. Thus, both analog and ECL video are output from the Am8153.

Smooth scrolling

Scroll has always been one of the main requirements of any display terminal. Usually data are moved on the screen on a character row by character row basis, which makes for poor viewing. In addition, using "hard" scroll to rapidly scan a document is prohibitive to use because the eye has a hard time following the staccato movement of the text.

Smooth scrolling allows the text to be scrolled gradually, scan line by scan line. Not only is this much more pleasing to the eye, but it also allows documents to be visually scanned very rapidly, in a manner similar to the way one scans a phone book for a particular entry. Implementing this scan line by scan line offset is fairly easy. The difficulty lies in holding part of the screen stationary while scrolling the remainder. The Am8052 supports both split screens (horizontal and vertical) and smooth scroll of a subscreen—a combination that has previously been impossible to implement economically. Window screens also create data structure problems since each scroll involves juggling large amounts of data. While this may be a difficult task for a local central processing unit (CPU), the Am8052 CRT controller integrated circuit (IC) fetches all its refresh data by means of a linked-list data structure.

In this structure, a top-of-page register contains the 24-bit memory address of the first component in the list, called the main definition block (MDB). The MDB, in turn, points to a sequence of row

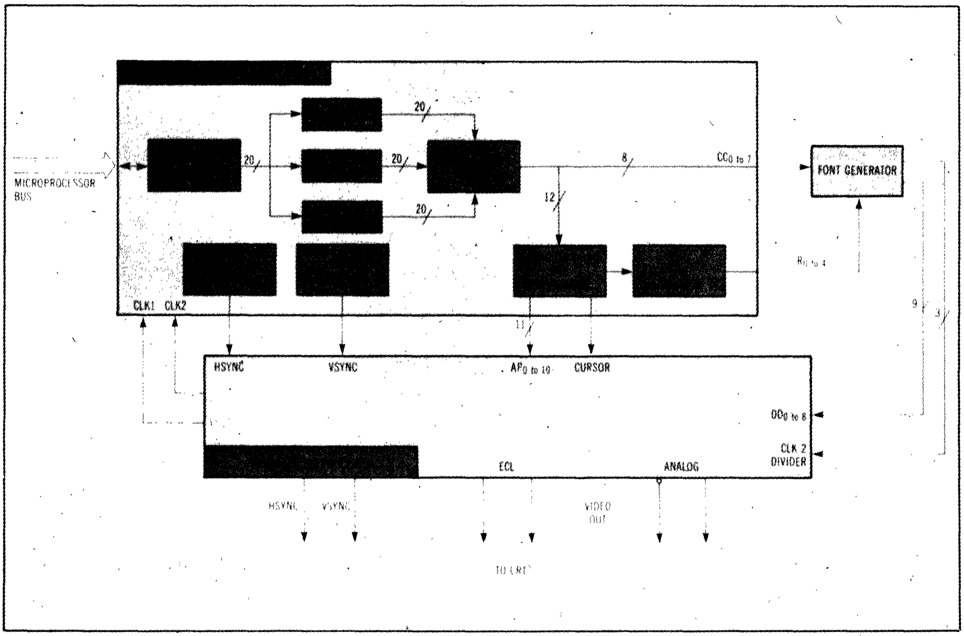


Fig 1 Two large scale integration chips and a font-generation read only memory form the interface between a 16-bit microprocessor bus and the CRT. Using three row buffers instead of the usual two ensures smooth scrolling in a split-screen application. The DMA channel fetches rows of characters into the three row buffers and outputs multiplexed data for attribute and cursor generation. The video processor chip serializes data for a video cutput and synchronizes the display with all the appropriate timing signals. The font generator can format the characters for proportional spacing to match the typical proportionally spaced characters of a printer output.

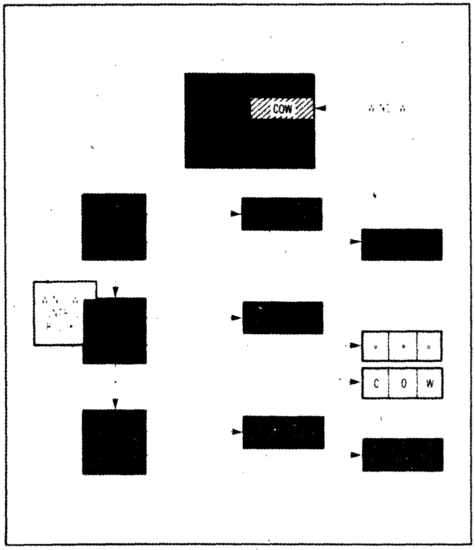


Fig 2 Windowing requires manipulation of a large amount of data. By using a linked-list data structure, the CRT controller chip can perform the windowing task at the CRT refresh rate. The chip maintains parallel control over the characters for both the full screen and the window. In this example, the three row control blocks keep track of their row entries in the background of the screen, while at the same time the window control block is used to insert the word "COW" in the appropriate window.

control blocks (RCBs). These blocks hold pointers to character and attribute lists for the appropriate row. The controller IC scans this complete list once per frame. Furthermore, the Am8052 keeps an eye on a second parallel list—the window data structure. This window linked list is used to overlay windows onto the screen. As the controller fetches screen data, it jumps from the screen to the window and vice versa to format the display (Fig 2).

After setting the display and one or more windows, the user can now issue a "scroll window" command to set the scroll in motion. When scrolling the screen, the user must ensure that the data structure is updated to reflect the new screen by modifying a pointer. Likewise, when scrolling one of multiple windows, the user must then update the window list in a similar fashion. In both cases, no complex data movements need occur. The Am8052 can scroll as slowly as one scan line every eight frames, and as fast as eight scan lines per frame—a significant spread in scroll rates. A system of interlocks protects the data from corruption during this scrolling.

A split-screen smooth scroll mandates three row buffers; a 2-row buffer configuration [Fig 3(a)] is acceptable for a single screen. Each of the rows is swapped or toggled with the other. Thus, while one row is being loaded, the other can be displayed. As long as each row buffer (ie, character row) is displayed for multiple scan lines, enough time is available to reload. However, for a split-screen smooth scroll, a character row can only be present in the frame for one scan line. This does not permit the alternate row buffer to be loaded and causes the screen to flicker. With three row buffers, however [Fig 3(b)], the problems of single scan line rows are averaged out, eliminating annoying screen flicker.

Character display generation

The Am8052 gives a flexible character capability to a video display terminal. Once the size (in scan lines) of a given character row is determined, the characters can then be placed in any position on the row. Further, row size can be varied on a row-by-row basis, and characters can be displayed as normal, superscripted, or subscripted, to allow flexible text.

Each character can be modified by an attribute word [Fig 4(a)] that is stored along with the character in the row buffers. Attribute words are fetched from memory, at the time the display is on, in a fashion similar to characters. The number of attributes fetched, however, can be programmed to be much smaller than the number of characters, thus reducing bus overhead. As in Fig 4(b), the string "CHANGED" is to be displayed in reverse video. By fetching a reverse attribute on the first "C" and a nonreverse attribute on the first "N" of "NORMAL," only two attributes are required to reverse the 7-character string.

The Am8052 attribute word on APO-AP10 can be used by the Am8153 to produce gray-level video from the font generator. For example, normal characters are displayed gray on white. If the highlight bit is set, however, the character will be

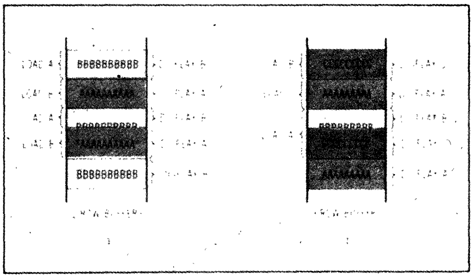


Fig 3 For split-screen scrolling applications, a character row could be displayed for only a single scan line. With two row buffers (a), this does not leave enough time for the reloading of the alternate row buffer, which results in a flashing screen. With three row buffers operating in a rotating fill-display mode (b), any single row buffer can be displayed for one scan line without any danger of screen flashing.

it allows the same powerful, yet familiar features. However, it is the AmPAL22V10's 10 output logic macrocells that give the designer substantial new design freedom. Moreover, at each macrocell output is a tri-state output buffer controlled by a separate output-enable AND gate.

These macrocells provide the AmPAL22V10's key features. They can be configured to make any or all of the I/O pins act either in sequence or in combination and have either active-high or active-low characteristics. Furthermore, the output enables can individually control the direction of the pins so they act as outputs, inputs, or bidirectional ports.

A number of trade-offs and limitations are apparent in a design that so dramatically affects the input and output of the system. The most obvious limitation stems from under utilization of 16-bit peripherals on an 8-bit bus—the speed of all I/O operations are cut in half. As a result, bus utilization will increase if the l6-bit peripheral represents a significant factor of the bus use. A CRT controller such as the Am8052 might use 5 to l0 percent of the bus bandwidth for display information when using 16-bit I/O. Converting to 8-bit I/O would double bus use to 10 to 20 percent. Another factor that might affect the bus usage is the efficiency of the 8- to 16-bit conversion control logic. If the state machine designed to perform the 8/16-bit (or 16/32-bit) conversion is improperly designed, extra transfer overhead might be introduced. This might mean a sequential transfer of two 8-bit values would take twice as long a single 16-bit transfer.

The design constraints might limit the use of the peripheral to byte-only operations during data transfers (as in the design using the DMA Am9516 controller), and slow it down by a factor of two during command operations. For such a DMA device as the Am95l6, the extra time required for command fetching is not usually a significant portion of bus time.

System designers will have to weigh the cost of the extra overhead on a case-by-case basis. The benefits may well justify these limitations—particularly when the bus is self-limiting, but the device characteristics allow for value-added designs. In addition to bus degradation for certain configurations, extra logic and design effort are involved. Most interfaces outside a system's immediate family require some kind of extra interface logic, however. By manipulating the signals and incorporating them into programmable logic devices such as the AmPAL22V10 device, therefore, most of this logic is free.

APPENDIX C

CRT Controllers Can Enhance Test Display And Simplify Editing

- Juergen Stelbrink

CRT Controllers Can Enhance Text Display And Simplify Editing

For screen editing the CPU normally has to move blocks of display data. This time-consuming task can be speeded up by use of a CRT controller.

by **Juergen Stelbrink,**
Advanced Micro Devices Inc.

As terminals become increasingly sophisticated, the designer is faced with many new problems in the areas of data manipulation and display. The high-resolution screen necessary to display a full-size 8½ × 11-in. page results in pixel rates exceeding 50 MHz. Additionally, the use of microprocessor technology in modern terminal designs has transferred the editing tasks from the host system to the terminal itself. Support for the latest text-display features available from letter-quality printers can be provided by CRT controllers.

Today's printers can support such text-display features as proportional spacing with block justification and double print. To adapt the word-processing task more fully to the human operator, workstations for word processing should be able to display edited text that looks like the printout of these letter-quality printers.

For example, instead of displaying the beginning and end of an underline with a special character sequence, the workstation should underline the string just as the printer does. Additionally, it should support features like highlighting (which is equivalent to double print in the case of a printer), character blinking, and multiple cursors to emphasize parts of the text.

Vertical smooth scroll will become a standard feature of future designs. Also helpful would be windows (overlaid on the displayed page) to provide temporary information about issued commands.

LINKED-LIST DATA STRUCTURE

In standard CRT subsystems, display data is organized as contiguous memory blocks associated with video frames and stored in video-refresh memory. To execute editing tasks like character or line insertion or deletion, the CPU has to move blocks of this data—a time-consuming operation that slows down the editing process.

Text editing would be faster and more elegant if a linked-list data structure were used. In a linked-list structure, display data is organized in small strings—usually rows—held together by pointers. The advantage becomes obvious when you consider execution speed: you can insert or delete a line by modifying one pointer instead of moving half the screen down (Fig 1). And you can swap pages simply by altering a pointer.

A second advantage is that when the display data is stored in the main system memory, the CRT controller can fetch the data directly from the list on which the word processor is operating, and there's no need to set up a special list of display data.

WINDOWS

Windows are text blocks overlaid in the background. Usually they're used to display temporary information. A

Reprinted by permission of the publisher from the Winter 1983 edition of COMPUTER TECHNOLOGY REVIEW

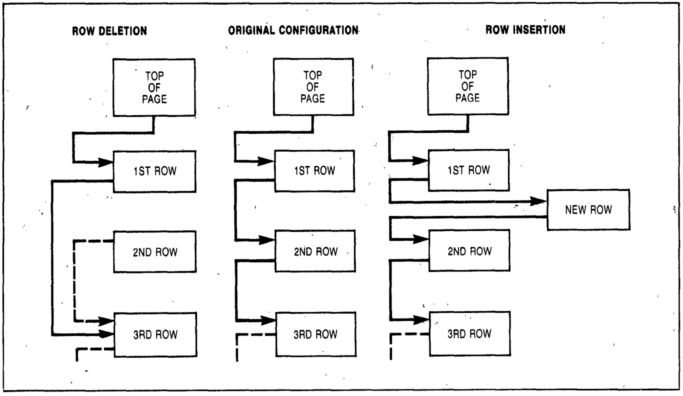


Fig 1 In a linked-list structure, data is organized in small strings held together by pointers. A line can be inserted or deleted by modification of a single pointer.

word processor, for example, might use the windows to display command tables while the background still shows the edited text. After the user has chosen a command from the table, the window is removed to make the overlaid text visible again.

Multitasking systems might use a window for each task currently active. In order to keep the window-processing overhead small, the data structure of the window should be similar to the background data structure so that you can display or remove windows without modifying the background data structure.

SOFT SCROLLING AND ATTRIBUTES

Vertical soft scrolling is the gradual replacement of a character row on a scan-line by scan-line basis. The displayed effect is more eye-pleasing than hard scrolling (where entire rows are replaced) and will become a key feature in future terminal designs. The smooth scroll of the entire screen is a relatively easy task and can be accomplished with a minimum of hardware.

However, soft scrolling of an overlaid window or soft scrolling of the background while windows are displayed is a much more sophisticated task. If a window is smooth-scrolled, text seems to appear and disappear within it while the background remains stable. If, on the other hand, the background is scrolled, background text will appear to pass under the window.

There are three kinds of attributes, distinguished by the number of characters they correspond to:

- **Screen attributes** affect the text display of the entire screen and represent screen information that might vary from page to page. Smooth-scroll rate, cursor blink rate, and cursor layout are all attributes of this kind.
- **Row attributes** modify text on a row basis. The height of a row and the positioning of normal, subscripted, and superscripted characters are some examples.
- **Character attributes** modify certain characters or strings. Examples are highlight, underline, blinking, subscript, and superscript.

Many CRT controllers treat characters and attributes in the same fashion. They fetch one attribute word per character. To minimize the bus occupancy of the CRT controller, the number of attribute fetches should be minimized. A fundamental difference between the changing rate of characters and attributes is that characters are typically uncorrelated along a character string and attributes are highly correlated, since features like reverse video affect a character string rather than individual characters. For this reason, a flexible correspondence between characters and attributes saves memory space and reduces the bus occupancy.

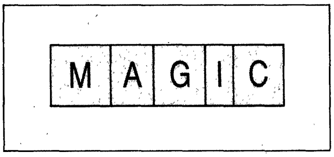


Fig 2 In proportional spacing, letters vary in the amount of line space they occupy. An "M", for example, is wider than an "I".

In a demand-attribute mode, an attribute is only loaded when the attribute characteristics need to be changed. A flag is positioned in the character string to make the CRT controller fetch a new attribute word, which could apply either to the next character or to all following characters. This flag could be a specific character that is not displayed on the screen, or it could be any bit of the character code. The first option would allow a 255-character set with a small bus overhead when attributes are fetched. The second option would halve the character set but eliminates overhead for attribute incorporation.

PROPORTIONAL SPACING AND CURSOR

Proportional spacing is now a standard feature of high-performance letter-quality printers. The CRT system should be able to support proportional spacing in order to display a text on the screen similar to the printed text on paper.

Proportional spacing means that narrow characters like "I" use less space in a character row than wider characters like "M" (Fig 2). The screen is no longer divided into a raster of character fields. The number of characters that can be put into one line is now a function of the characters themselves. Right justification in proportional-spacing applications requires a user-definable number of blank pixels to follow each character so that the text will have a straight right-hand edge.

Two kinds of cursors are imaginable. A cursor could be programmed to appear on an X-Y coordinate. This type of cursor would be tied to the screen. When scrolling, the cursor still appears on the same location but applies to a new character. The sec-

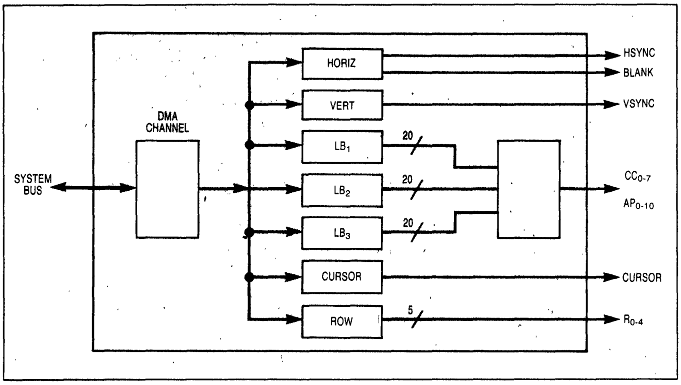


Fig 3 A cathode-ray tube controller (CRTC) uses three line buffers for smooth scrolling of windows and provides the character code and scan-line address for the character-font generator.

ond way to specify cursors is to use the attribute word. In this case, the cursor would be fixed to a character, so any scrolling of the screen would move the cursor and character both. A system usually has only one X-Y cursor, since each X-Y cursor needs a pair of coordinates. There is no restriction in the number of attribute cursors because this information is a part of the attribute word.

The cursor layout should be very flexible. Examples of cursor styles are:

- Static or blinking underline.
- Blinking by switching between normal display and blank.
- Blinking by switching between normal display and reverse.
- Reverse character.

The X-Y cursor and the attribute cursor can have different styles to distinguish them. For example, the X-Y cursor could be a blinking underline and the attribute cursor could reverse the character.

SILICON IMPLEMENTATION

These features are all supported by the Am8052 and Am8152/53 CRT controller (CRTC) chip set. To make editing tasks simpler and faster, the set supports a display data structure organized as a linked list in system memory. By adding an external character-font generator to these chips, you can build a complete subsystem that talks to the system bus on one side and generates a high-speed analog or digital video signal on the other. Other features, such as horizontal soft scroll and a loadable character-font generator, can be implemented by the addition of a few more medium-scale integration (MSI) devices and support software.

The first element of this design, the CRT controller (Am 8052, CRTC) (Fig 3), fetches the display data via the built-in DMA controller, interprets the linked list, and handles attributes, windows, and soft scrolling. It has three line buffers to support flicker-free smooth scrolling of windows and provides the character code and scan-line address for the character-font generator. Its maximum character-output rate is 14 MHz.

The second element in the chip set, the Video System Controller (VSC), is basically a parallel-to-serial video-shift register (Fig 4). It accepts the character font from the character generator, and the attribute words supplied by the CRTC, and generates a high-speed pixel stream. The video output provides a 4-level analog signal that can directly drive a 75Ω load or a 2-bit digital signal. The video system controller (VSC) can handle video rates of 40 MHz (TTL outputs) or 100 MHz (ECL outputs), allowing high-resolution flicker-free displays.

The CRTC handles the linked-list management, the windows, soft scrolling, cursor, and attribute processing. The display data is stored in system memory to be easily accessible by the host CPU during its execution of display-editing tasks. The display data consists of characters and their attributes, both of which are grouped into segments. One or more segments are tied together by a list of pointers—the row-control block—to form a row. Row-control blocks are connected via a linked list, each block pointing to its successor. The CRTC interprets the linked list and transfers the character strings and attributes sequentially to the character generator.

The terminal processor loads the top-of-page register (Fig 5) to notify the CRT controller of the beginning of the linked list.The main definition block at the beginning of the linked list contains screen attributes like cursor style and cursor blink rate, and a pointer to the first row-control block. The row-control block holds information relevant to one row displayed on the screen. It contains pointers to the succeeding row-control block and pointers to segments containing character and attrib-

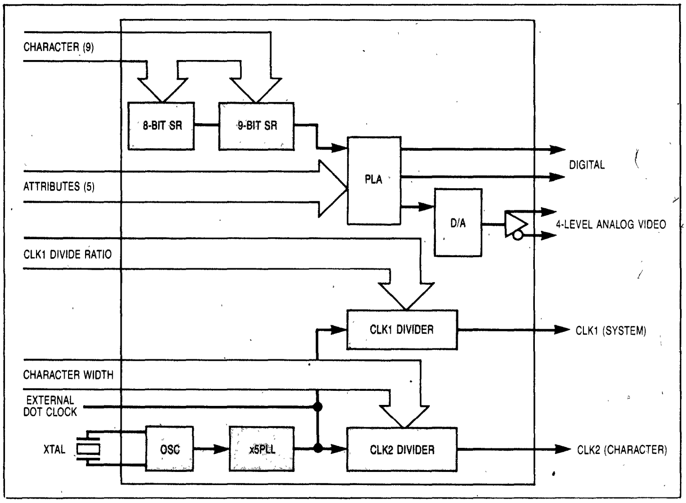


Fig 4 A parallel-to-serial video-shift register generates a high-speed pixel stream from data supplied by the character generator and the CRTC.

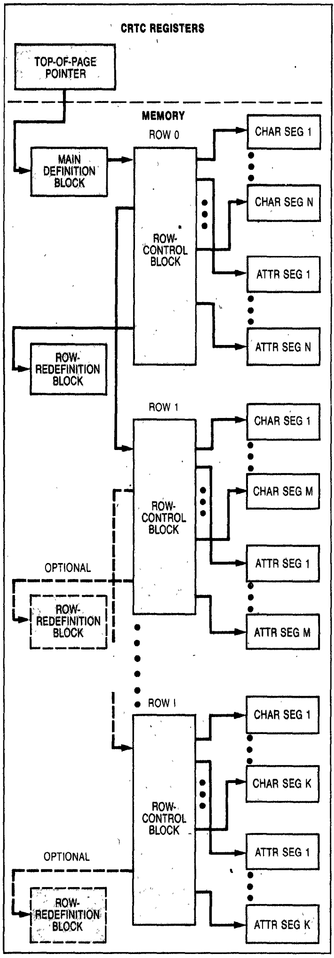


Fig 5 The top-of-page register points to the begin of the linked list. The main definition block contains screen attributes and the row-control block holds information for one row of the display.

ute strings. Positioning of subscript, superscript, and normal characters in the row and the number of scan lines per row is optionally redefinable on a row-by-row basis.

The display data structure representing the layout of windows is similar to the data structure of the background. Vertical soft scrolling of the background or of windows requires little interaction with the CPU. The CRTC only interrupts the CPU when a row is totally scrolled in or out, to make it relink the data structure. The scroll rate is programmable and can range from one scan line per eight frames (low-speed scroll) to eight scan lines per frame (high-speed scroll).

ATTRIBUTE PROCESSING

The CRTC allows flexible attribute processing. Attributes are handled in 16-bit quantities and fetched on demand, in order to reduce bus occupation for direct memory access (DMA). Seven attribute bits are predefined and four are user-definable. However, the internal attribute processing can be partially or totally deactivated to satisfy specific application requirements so that the designer can interface external attribute-processing logic. The predefined attributes are:

- Highlight. Characters are made brighter.
- Reverse. The colors of the background and the foreground are exchanged.
- Superscript. The character is shifted up a defined number of scan lines.
- Subscript. The character is shifted down a defined number of scan lines.
- Underline. The character is underlined; the position of the underline is programmable.
- Strike through (shifted underline). The affected character is struck through.
- Blink. The affected character blinks at a programmable rate and duty cycle.

The attributes mentioned above control an attribute port of the CRTC. A special character-font generator can be used to display smaller subscript or superscript characters. Two attributes are used for internal processing only. They are:

- Ignore. The character is not loaded into the line buffer and, consequently, not displayed. You can erase a character by setting this attribute bit.
- Latched. This attribute word is latched by the CRTC and therefore applies to a character string.

The VSC serializes the character stream, processes the attributes, handles proportional spacing, and generates the system timing. In proportional-spacing applications, the character generator consists of two parts: one part stores the font of the characters; the other holds the character width—a 4-bit value. The character width is passed to the VSC to determine the divide factor for the character clock, which is connected to the CRTC to specify the character-output rate. In addition, the VSC has logic to allow you to justify text by the insertion of up to three blank pixels between characters. This technique allows smooth, virtually unnoticeable line stretching.

The CRTC can easily be interfaced to 16-bit system buses. In slave mode, the CPU initializes the CRTC by programming the registers for the timing parameters. After the CRTC is activated, it tries to gain mastery of the bus to fill the line buffers, and then starts displaying. The CRTC bus-interface

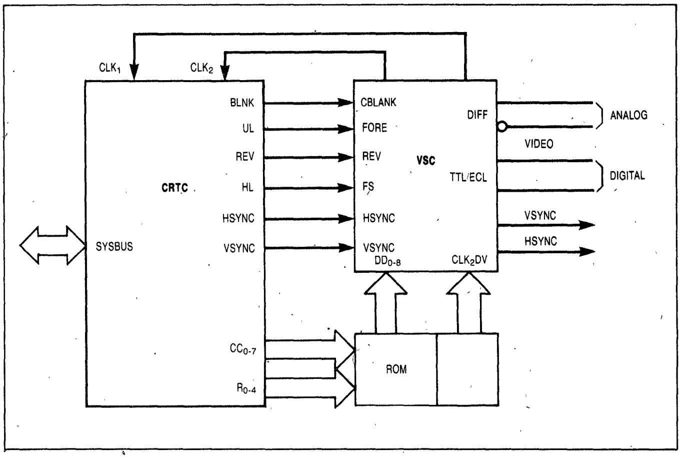


Fig 6 In a standard proportional-character application, the CRTC's 8-bit character-code (CC_0 through CC_7) and the 5-bit scan-line count (R_0 through R_4) address the character-font generator. The VSC can serialize character slices up to 17 bits wide.

architecture supports 24-bit linear address buses (68000, 8086) and 23-bit segmented address buses (Z8000). To make sure that the system still can respond to interrupts in real time, the CRTC has a burst-length register that controls the maximum length of a DMA block read and a burst-space register to have a minimum delay between two DMA cycles.

Fig 6 shows a standard proportional-character application employing the CRTC, the VSC, and a character-font generator. The 8-bit character code, usually ASCII code, allows a set of up to 256 characters. The 5-bit scan-line address can distinguish 32 scan lines. The VSC can serialize up to 17-bit-wide character slices, so that the maximum achievable character box is 17 × 32 pixels.

Since the CRTC fetches all the data needed for the display refresh from system memory, the controller uses a significant part of the bus bandwidth. For each frame, it fetches the control, character, and attribute blocks. The bus overhead caused by the video refresh is a function of the number of displayed characters and invoked attributes.

In systems where the CPU is involved in editing tasks, it might be intolerable for the CRTC to use a major part of the bus bandwidth. This problem can be solved by utilizing a dual bus system. The main memory where the display data is stored has two ports. One is connected to the main system bus; the other passes the data via a local bus to the CRT controller.

In this configuration, the CRT DMA transfer doesn't slow the system down. Instead, an arbitration logic controls system and CRT access to the display memory. The data path from the main bus to the local bus is used to access the CRTC directly to alter register contents.

The structure of the CRTC allows you to add special features that aren't directly supported. The implementation of horizontal soft scroll is a good example of the flexibility of the controller's design (see Box). Horizontal scroll moves the entire page left or right in order to display characters that are hidden because the text row is wider than the row that can be displayed on the screen. Similar to vertical soft scroll, horizontal soft scroll moves text on a pixel basis rather than on a character basis, so the viewer notices very smooth movements. ■

Juergen Stelbrink, *applications engineer for Advanced Micro Devices, has his MS degree in computer engineering from the RWTH Aachen, West Germany.*

Implementation of Horizontal Soft Scroll

The basic idea behind this implementation is to place in the front of the line a dummy character that's rendered invisible by external logic that delays the horizontal BLANK. You move the entire line by using the VSC's proportional-spacing capability to modify the width of this dummy character.

When the dummy character is programmed for full width, the delayed BLANK covers it. When you reduce the width of this character, the first visible character moves left and gets partially covered. Characters seem to enter the screen on the right side and seem to leave it on the left.

The detailed description that follows assumes a nonproportional-spacing application, a character width of 8 pixels, and a dummy character width of 10 pixels. There is no restriction on these values, but reference to a specific environment makes the description easier.

By reducing the width of the dummy character from 10 to 3 (steps 1 through 7 in Fig 1) and a modification of the character-segment pointer in the row-control block (step 8), the left-most character is moved out. Each scroll step the CPU modifies the width of the dummy character one pixel. Decreasing the width causes a left scroll; increasing the width causes a right scroll. The horizontal soft-scroll speed can be similar to the vertical soft-scroll speed (scrolling one pixel per 8 frames to 8 pixels per frame). It is supported by the CRTC interrupt on a vertical event issued once per frame.

The width of the dummy character is controlled by providing an appropriate value at the character-clock divider inputs of the VSC. This value can be supplied in several ways:

- The width can be controlled by the four user-definable attribute bits of the attribute word corresponding to the dummy character.
- Bits of the row-attribute word can determine the width. This attribute word is put out during horizontal retrace and can be latched by HSYNC.
- In proportional-spacing applications, the character-font generator can be programmed to contain a set of characters with widths from 3 to 10.

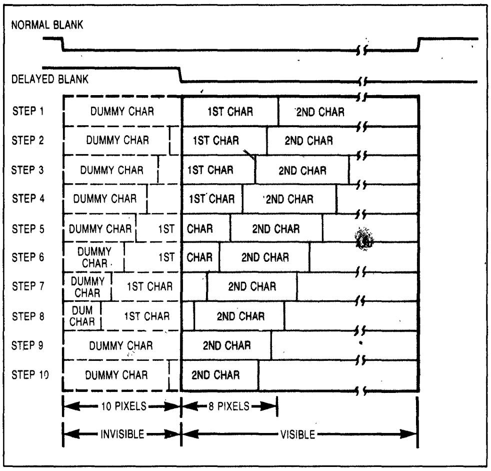


Fig 1 In horizontal soft-scroll, the proportional-character capability is used to reduce the width of an invisible dummy character placed at the front of each line. As the width changes, the first visible character moves left and gets partially covered.

The second task the designer is confronted with is to find a simple solution to delay BLANK. If the system-clock cycle is wider than the character-clock cycle, BLANK can be delayed by being fed through two D-flipflops clocked by the system clock (CLK1) (Fig 2).

Another approach is to use a counter to delay BLANK the appropriate number of pixels. The counter is clocked by the dot clock and enabled by the first edge of CLK1 or CLK2 after BLANK inactive. ■

APPENDIX D

Source Code For The Low-Cost Smart Terminal Board

```
"8051"
   TITLE "    CALEB 0.00    Interrupt Handlers"
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; C_Base                                CALEB 0.00
;
;         Copyright 1985 Advanced Micro Devices, Inc.
;
; This file contains the reset and interrupt entrypoints as well as the
; interrupt handlers.

   NAME  "Interrupt Handlers"

;--------------------------------------------------------------------------------

   GLB   CopyrightMsg                   ; EEPROM resident claim

;--------------------------------------------------------------------------------

   EXT   Reset                                              ; in C_Init

   EXT   PlcCsr,ShwWnd                                      ; in C_Util
   EXT   ScrlRtOne,ScrlLtOne,SetForScrlUp,SetForScrlDn      ; in C_Util
   EXT   SetAftScrlDn,SetWndPos                             ; in C_Util
   EXT   WrAm8052Reg,RdAm8052Reg                            ; in C_Util

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
   INCLUDE C_MemMap

   SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; This is the base of the Am8052/8152 Low Cost Terminal demonstration firmware.
; The entrypoints for the reset and five interrupt sources are defined here.
; There are only eight bytes between interrupt entrypoints, so when a larger
; handler than that is required the entrypoint must transfer control elsewhere.
; This is the case for most of the interrupt handlers we have implemented.
```

```
;--------------------------------------------------------------------------------

   ORG    00000H                            ; Reset entrypoint

; The 8751 reset condition begins execution here.  This entrypoint will only
; be entered once, immediately after power is supplied to the board.
  LJMP                                      ; Go to the reset procedure

;--------------------------------------------------------------------------------

   ORG    00003H                            ; External interrupt 0 entrypoint

; The external interrupt 0 entrypoint is defined below.  The 8751's INTO*
; input is connected to the Am8052's bus request (BRQ*) output.  Therefore,
; this interrupt occurs when the Am8052 desires control of the display
; memory bus for performing video refresh.

   PUSH  P2                                     ; Save port 2 contents and keep
   MOV   P2,#0FFH                               ;    it from interfering w/Am8052
   LJMP  BusReqHdl                              ; Go to actual handler

;--------------------------------------------------------------------------------

   ORG    0000BH                            ; Timer 0 interrupt entrypoint

; The timer 0 interrupt entrypoint is defined below.  The 8751's T0 input
; is connected to the Am8052's BLANK output.  This has the affect of counting
; visible scan lines.  The counter is reloaded during vertical retrace so
; that the interrupt occurs twenty-eight (28) scan lines before the vertical
; blanking period begins at the bottom of the monitor screen.

   PUSH  PSW                                    ; Save normal flags
   SETB  RS0                                    ; Change register bank for
   SETB  RS1                                    ;    high priority interrupt
   AJMP  EndFrmHdl                              ; Go to actual handler (which
                                                ;    must be in first 2K of code)

;--------------------------------------------------------------------------------
   SKIP
;--------------------------------------------------------------------------------
```

```
 ORG   00013H                        ; External interrupt 1 entrypoint

; The external interrupt 1 entrypoint is defined below.  The 8751's INT1*
; input is connected to the Am8052's INT* output.  This interrupt occurs
; for the vertical event or when the soft-scroll (smooth scroll) process
; in the Am8052 requires attention.

 PUSH  PSW                                 ; Save normal flags
 SETB  RS0                                 ; Change register bank for
 SETB  RS1                                 ;    high priority interrupt
 AJMP  Am8052Hdl                           ; Go to actual handler (which
                                           ;    must be in first 2K of code)

;-----------------------------------------------------------------------
 ORG   0001BH                        ; Timer 1 interrupt entrypoint

; Timer 1 is used to provide the clock for serial communications with the
; host; therefore, the timer 1 interrupt is disabled and this entrypoint
; should never be executed.  As a precaution, we put a jump-to-self here
; for use while debugging.  We also included other code, as if this were
; a valid interrupt, so that it would be possible to continue.

 PUSH  PSW                                 ; Save normal flags
 LJMP  $                                   ; Stick right here
 POP   PSW                                 ; Restore normal flags
 RETI                                      ; Exit from interrupt

;-----------------------------------------------------------------------
 ORG   00023H                        ; Serial port interrupt entrypoint

; The serial port interrupt entrypoint is defined below.  The 8751's serial
; port capability is used for communications with the host.  Currently, only
; reception is implemented since CALEB does not generate output.  The addition
; of ANSI X3.64 report capabilities or the inclusion of a keyboard will make
; transmission necessary.

 PUSH  PSW                                 ; Save normal flags
 SETB  RS0                                 ; Reg bank for low priority intr
 AJMP  HstComHdl                           ; Go to actual handler (which
                                           ;    must be in first 2K of code)
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  PROG                                    ; Begin relocatable program here

;----------------------------------------------------------------------------
; The following ensures that the 8751's EEPROM contains a copyright claim.

CopyrightMsg:

  DB    " Copyright 1985 Advanced Micro Devices, Inc. "
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

BusReqHdl:

; Handles the bus request interrupt from the Am8052.  The bus acknowledge
; signal is output until the Am8052 no longer desires the bus then it is
; returned to its inactive state.  The contents of port 2 are saved and
; restored so that the port can be configured as all inputs during Am8052
; bus transactions (any pins configured as outputs will interfere with the
; signals on the bus).  Port 2 reconfiguration has already been done by
; this time.

  CLR   Am8052BusAckFlg                  ; Acknowledge the bus request
  JNB   Am8052BusReqFlg,$                ; Stay here 'til BRQ* is released
  SETB  Am8052BusAckFlg                  ;    then remove bus acknowledge
  POP   P2                               ; Restore port 2 contents
  RETI                                   ; Exit from interrupt

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

EndFrmHdl:

; Handles the timer 0 interrupt which occurs near the end of the frame (at
; the 28th visible scan line from the bottom of the monitor screen).  It
; sets a flag (which is reset by the Am8052 interrupt handler) to signal
; the start of this end-of-frame processing time.  This handler also does
; all changes to display memory to support horizontal smooth scrolling.
```

```
    SETB  EndFrmFlg                    ; Set end-of-frame flag and
    JNB   HrzScrlFlg,EFH0              ;    get out if not horz. scroll
    DJNZ  HrzFrmCnt,EFH0               ; Get out if no update for hz scr
    PUSH  ACC                          ; Save
    PUSH  DPH                          ;    special function
    PUSH  DPL                          ;    registers
    MOV   HrzFrmCnt,HrzFrmSet          ; Reset update count
    JNB   AMDDWMBit,EFH1               ; Jump if in normal mode
    MOV   R0,#6                        ; Char width in compressed mode
    SJMP  EFH2                         ;    and continue

EFH1:
    MOV   R0,#9                        ; Char width in normal mode

EFH2:
    JB    HrzDirFlg,EFH4               ; Jump if scrolling right
    MOV   A,HrzScrlCnt                 ; Check char scroll count and
    JZ    EFH6                         ;    jump if already at end

    CLR   C                            ; Clear carry for below
    MOV   A,R0                         ; Char width
    SUBB  A,HrzCurPxl                  ;    minus horz pixel offset is
    MOV   R1,A                         ;    amount to scroll in this chr
    MOV   A,HrzPxlShf                  ; Amount shifted each time
    SUBB  A,R1                         ;    minus amount left this chr
    JC    EFH3                         ;    skip if this char is enough
    MOV   HrzCurPxl,A                  ; Else store new pixel offset
    LCALL ScrlLtOne                    ;    and go to next character
    SJMP  EFH5                         ; Go check for end of scroll

EFH3:
    MOV   A,HrzCurPxl                  ; Current pixel offset
    ADD   A,HrzPxlShf                  ;    plus amount to shift gives
    SJMP  EFH7                         ;    new pixel offset; continue

EFH4:                              ; Right scroll
    CLR   C                            ; Clear carry for below
    MOV   A,HrzCurPxl                  ; Current pixel offset in char
    SUBB  A,HrzPxlShf                  ;    minus # shifted each time
    JNC   EFH7                         ; Continue if still in char
```

```
    ADD   A,R0                         ; Readjust to be in character
    MOV   HrzCurPxl,A                  ;    and store new pixel offset
    MOV   A,HrzScrlCnt                 ; Check char scroll count
    JZ    EFH6                         ;    and jump if already at end
    LCALL ScrlRtOne                    ; Else, get next character

EFH5:                              ; Check for end of scroll
    DJNZ  HrzScrlCnt,EFH8              ; Continue if more to scroll

    JB    HrzDirFlg,EFH8               ; Finish last char for scroll rgt

    MOV   HrzCurPxl,#0                 ; For left, set to char boundary
    MOV   HrzFrmCnt,#1                 ;    and wait one more frame time
    SJMP  EFH8                         ;    to actually finish

EFH6:                              ; Actual finish of horizontal scroll
    MOV   HrzCurPxl,#0                 ; Set pixel offset to char bound
    CLR   HrzScrlFlg                   ; Indicate no longer scrolling
    LCALL PlcCsr                       ; Place cursor (if possible)
    JB    MsgActFlg,EFH8               ; Get out if in message display
    LCALL SetWndPos                    ; Set window position if in bgd
    JNB   WndVisFlg,EFH8               ; Get out if window not visible
    LCALL ShwWnd                       ; Show window if it should be
    SJMP  EFH8                         ; Get out

EFH10:                             ; Set function char width and exit
    MOVX  A,@DPTR                      ; Get function attr (high byte)
    ANL   A,#0F8H                      ; Mask off old width bits and
    ORL   A,R0                         ;    put in new width
    MOVX  @DPTR,A                      ; Write new high byte of attr
    INC   DPL                          ; Point to low byte
    MOVX  A,@DPTR                      ; Get low byte of function attr
    ANL   A,#07FH                      ; Mask off old width bit and
    ORL   A,R1                         ;    put in new one
    MOVX  @DPTR,A                      ; Write new low byte of attr
    POP   DPL                          ; Restore
    POP   DPH                          ;    special function
    POP   ACC                          ;    registers
EFH0:                              ; Final exit
    POP   PSW                          ; Restore flags and reg bank
    RETI                               ; Exit from interrupt
```

```
EFH7:                                    ; In middle of character
   MOV    HrzCurPxl,A                        ; Keep new pixel offset
EFH8:                                    ; Set up for function character width
   SETB   C                                  ; Full
   MOV    A,#12                              ;    maximum width
   SUBB   A,HrzCurPxl                        ;    minus pixel offset
   DEC    A                                  ;    minus two (for Am8152)
   MOV    R0,A                               ; Keep new width
   SWAP   A                                  ; Most sig bit of width to bit 7
   ANL    A,#080H                            ;    and all else masked off
   MOV    R1,A                               ;    then keep for low attr byte
   MOV    A,#007H                            ; Mask off all but 3 low bits
   ANL    A,R0                               ;    of new width
   MOV    R0,A                               ;    then keep for high attr byte
   MOV    DPTR,#BgdFncAtr0                   ; Point to bgd function attribute
   JNB    MsgActFlg,EFH10                    ;    and use it unless in message

   MOV    DPTR,#MsgFncAtr                    ; Point to msg function attribute
   SJMP   EFH10                              ;    and use it

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Am8052Hdl:

; Handles the vertical event and smooth scrolling interrupts from the Am8052.
; The vertical event is set to occur during vertical retrace and is used to
; reset the visible scan line counter.  We also use this time to synchronize
; turning the cursor on.  The smooth scrolling interrupt also occurs during
; vertical retrace and is fully discussed in Am8052 technical documents.

   PUSH   ACC                                ; Save
   PUSH   DPH                                ;    special function
   PUSH   DPL                                ;    registers
   MOV    R1,#ModReg2Ind                     ; Read interrupt pending
   LCALL  RdAm8052Reg                        ;    status from Am8052
   MOV    A,R3                               ; Check vertical event pending
   JNB    ACC.3,AH1                          ;    and jump if not
```

```
   CLR    ACC.3                    ; Clear the condition
   MOV    R3,A                     ;    and keep it
   CLR    EndFrmFlg                ; Reset end-of-frame flag
   MOV    TH0,#END_FRM_CNT_HI      ; Reload
   MOV    TL0,#END_FRM_CNT_LO      ;    end-of-frame counter
   JNB    CsrShwFlg,AH0            ; Skip if not requesting cursor
   CLR    CsrShwFlg                ; Reset cursor request and
   SETB   CsrSetFlg                ;    defer actual action
   SJMP   AH1                      ;    until next frame

AH0:
   JNB    CsrSetFlg,AH1            ; Skip if no deferred cursor req
   CLR    CsrSetFlg                ; Reset deferred request flag
   XCH    A,R2                     ; Get byte with enable bit and
   SETB   ACC.7                    ;    set it (shows cursor)
   XCH    A,R2                     ;    then put that byte back

AH1:
   JB     ACC.0,AH3                ; Jump if a smooth scroll intr

AH2:
   LCALL  WrAm8052Reg              ; Update Am8052 status
   POP    DPL                      ; Restore
   POP    DPH                      ;    special function
   POP    ACC                      ;    registers
   POP    PSW                      ; Restore flags and reg bank
   RETI                            ; Exit from interrupt

AH3:
   CLR    ACC.0                    ; Clear smooth scroll condition
   MOV    R7,A                     ; Keep low
   MOV    A,R2                     ;    and high
   MOV    R6,A                     ;    bytes of status
   CLR    VrtScrlNewFlg            ; Signal extra row now available
   JB     ACC.0,AH6                ; Jump if scrolling continues
   JB     SudBit,AH5               ; Jump if scrolling up
   MOV    A,VrtScrlCnt             ; Check for late continuation of
   JZ     AH4                      ;    up scroll, jump if not
   INC    VrtScrlCnt               ; Allow for extra call when
   SJMP   AH7                      ;    scrolling up and continue
```

```
AH4:
   LCALL SetAftScrlDn                       ; Clean up after scroll up

AH5:
   CLR   VrtScrlFlg                         ; Indicate no longer scrolling
   LCALL PlcCsr                             ; Place cursor (if possible)
   SJMP  AH16                               ; Go restore status for exit

AH6:                                  ; Continue scrolling
   JNB   WndActFlg,AH10                     ; Jump if in background
   MOV   R0,#WndWDB0.AN.OFST+WDB_RowPag     ; Set up for window scrolling
   MOV   R1,#TOWSftLoInd
   MOV   R3,#WndWDB0.AN.OFST
   JB    CurWDBFlg,AH7
   SETB  CurWDBFlg
   MOV   R2,#WndWDB1.SR.PAGE
   SJMP  AH8

AH7:
   CLR   CurWDBFlg
   MOV   R2,#WndWDB0.SR.PAGE

AH8:
   JNB   CurMDBFlg,AH9
   MOV   R5,#BgdMDB1.AN.OFST+MDB_Scrl
   SJMP  AH12

AH9:
   MOV   R5,#BgdMDB0.AN.OFST+MDB_Scrl
   SJMP  AH12

AH10:
   MOV   R1,#TOPSftLoInd                    ;Set up for background scrolling
   MOV   R2,#BgdMDB0.SR.PAGE
   JB    CurMDBFlg,AH11
   SETB  CurMDBFlg
   MOV   R0,#BgdMDB1.AN.OFST+MDB_RowPag
   MOV   R3,#BgdMDB1.AN.OFST
   MOV   R5,#BgdMDB1.AN.OFST+MDB_Scrl
   SJMP  AH12
```

```
AH11:
   CLR   CurMDBFlg
   MOV   R0,#BgdMDB0.AN.OFST+MDB_RowPag
   MOV   R3,#BgdMDB0.AN.OFST
   MOV   R5,#BgdMDB0.AN.OFST+MDB_Scrl

AH12:
   JB    SudBit,AH14                        ; Jump if scrolling up
   JNB   VrtScrlFlg,AH12a                   ; Skip if first row in down scr
   LCALL SetAftScrlDn                       ; Clean up after a scroll down

AH12a:
   SETB  VrtScrlFlg                         ; Indicate scroll in progress
   DJNZ  VrtScrlCnt,AH13                    ; Jump if more after this
   MOV   DPH,R2                             ; Point to row pointer
   MOV   DPL,R0                             ;    in appropriate block
   MOV   A,TopRow                           ;    and make it point to top
   MOVX  @DPTR,A                            ;    visible row
   MOV   A,ScrlByt                          ; Get scroll control byte
   CLR   ACC.0                              ;    and set up to stop
   SJMP  AH15                               ;    after this last row

AH13:
   LCALL SetForScrlDn                       ; Set up to scroll another row
   MOV   DPH,R2                             ; Point to row pointer
   MOV   DPL,R0                             ;    in appropriate block
   MOV   A,R4                               ;    and make it point to top
   MOVX  @DPTR,A                            ;    visible row
   MOV   A,ScrlByt                          ; Get scroll control byte
   SJMP  AH15                               ;    and continue scrolling

AH14:
   SETB  VrtScrlFlg                         ; Indicate scroll in progress
   LCALL SetForScrlUp                       ; Set up to scroll another row
   MOV   DPH,R2                             ; Point to row pointer
   MOV   DPL,R0                             ;    in appropriate block
   MOV   A,R4                               ;    and make it point to top
   MOVX  @DPTR,A                            ;    visible row
   MOV   A,ScrlByt                          ; Get scroll control byte
   DJNZ  VrtScrlCnt,AH15                    ; Jump if more after this
   CLR   ACC.0                              ; Else set up to stop scroll
```

```
AH15:
   MOV   DPH,#BgdMDB0.SR.PAGE             ; Point to
   MOV   DPL,R5                           ;    appropriate MDB and put
   MOVX  @DPTR,A                          ;    in new scroll control byte
   LCALL WrAm8052Reg                      ; Write new block (MDB or WDB)
AH16:
   MOV   R1,#ModReg2Ind                   ; Ready
   MOV   A,R6                             ;    to restore status
   MOV   R2,A
   MOV   A,R7
   MOV   R3,A
   AJMP  AH2                              ; Go restore status

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

HstComHdl:

; Handles host communications using the 8751's on-chip asynchronous serial
; port feature.  Currently, only reception from the host is supported, but
; transmission can be easily added.

   PUSH  ACC                              ; Preserve accumulator
   PUSH  DPH                              ;    and
   PUSH  DPL                              ;    data pointer
   JNB   RI,HCH4                          ; Jump (to xmt) if no rcv intr

   CLR   RI                               ; Reset receiver intr condition
   MOV   DPH,#HstRcvBuf.SR.PAGE           ; Point to ring buffer
   MOV   DPL,HstRcvInsOff                 ;    insertion location
   CLR   C                                ; Ensure no interference w/SUBB
   MOV   A,HstRcvCnt                      ; Current number of chars in ring
   SUBB  A,#80                            ;    compared with maximum
   JNC   HCH3                             ; Jump (to exit) if ring is full

   ADD   A,#NEAR_FULL_CNT                 ; Check for nearly full ring
   JNC   HCH1                             ; Jump if plenty of room

   SETB  HstRcvBsyFlg                     ; Signal busy if nearly full
```

```
HCH1:                                     ; Read and store character
   MOV   A,SBUF                                    ; Get character from host
   MOVX  @DPTR,A                                   ;    and store it in ring buffer
   MOV   A,HstRcvInsOff                            ; Insertion location now
   INC   A                                         ;    incremented to next location
   JNZ   HCH2                                      ; Jump if still in buffer range

   MOV   A,#HstRcvBuf.AN.OFST                      ; Reset to start if past end
HCH2:                                     ; Finish receiver interrupt
   MOV   HstRcvInsOff,A                            ; Keep new insertion location
   INC   HstRcvCnt                                 ; New number of chars in ring
HCH3:                                     ; Common interrupt exit (rcv and xmt)
   POP   DPL                                       ; Restore data pointer
   POP   DPH                                       ;    and
   POP   ACC                                       ;    accumulator
   POP   PSW                                       ; Restore flags and reg bank
   RETI                                            ; Exit from interrupt

HCH4:                                     ; Transmitter interrupt handler
   CLR   TI                                        ; Reset transmit intr condition

; NOTE:  There is currently no software support for transmission to the host.
;        This part of the handler merely shows where actual code to support
;        this capability would be placed.

   SJMP  HCH3                                      ; Go to exit

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; end of C_Base
```

```
"8051"
   TITLE "    CALEB 0.00    Initialization"
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; C_Init                                  CALEB 0.00
;
;       Copyright 1985 Advanced Micro Devices, Inc.
;
;
;  This file contains the reset, memory test and initialization code.

   NAME  "Initialization"
   PROG

;-------------------------------------------------------------------------------
   GLB   Reset                            ; Reset procedure

;-------------------------------------------------------------------------------
   EXT   DisCon                                              ; in C_Switch

   EXT   WrFntCel,HidCsr,ShwCsr                              ; in C_Util
   EXT   DlyTilEndFrm,WrAm8052Reg,RdAm8052Reg                ; in C_Util
   EXT   HalfSwap                                            ; in C_Util

   EXT   Fnt_7x9,Fnt_5x7                                     ; in C_Font

   EXT   DblBaudOpt,BaudRatCnt                               ; in C_Config

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
   INCLUDE C_MemMap

   SKIP
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Reset:                                    ; Reset procedure

; This is the beginning of the reset procedure.  We get here either from a
; power-on condition (i.e. chip reset) or a Reset To Initial State (RIS)
; control from the host.
```

```
MOV    IE,#0                          ; Disable all interrupts
MOV    P1,#0EDH                       ; Ensure 7: HstXmtBsyFlg->input
                                      ;        6: HstRcvBsyFlg->busy
                                      ;        5: KbdRcvRdyFlg->input
                                      ;        4: KeybrdEnbFlg->disable
                                      ;        3: Am8052XfrBit->high
                                      ;        2: Am8052BusAck->high
                                      ;        1: AMDSPMBit    ->low
                                      ;        0: (unused)     ->input
MOV    P3,#0FFH                       ; Ensure special functions and
                                      ;    marking output to host
MOV    PSW,#0                         ; Ensure normal register bank
MOV    SP,#067H                       ; Base of 24-byte stack
MOV    R1,#ModReg1Ind                 ; Mode Register 1
MOV    R2,#0                          ;    gets zeroes
MOV    R3,#0                          ;    to
LCALL  WrAm8052Reg                    ;    disable the display
MOV    IP,#007H                       ; Bus request (INT0), end-of-
                                      ;    frame (T0) and Am8052 (INT1)
                                      ;    are high priority; serial
                                      ;    and unimplemented (T1) low
MOV    TMOD,#025H                     ; Timer 1 (mode 2) for baud rate;
                                      ;    timer 0 (counter, mode 1)
                                      ;    for end-of-frame interrupt
MOV    TCON,#055H                     ; Both timers on;
                                      ;    edge triggered interrupts
MOV    SCON,#050H                     ; Serial mode 1 (8-bit, variable
                                      ;    baud rate); receiver enabled
MOV    DPTR,#DblBaudOpt               ; Load double baud option for
CLR    A                              ;    PCON contents
MOVC   A,@A+DPTR                      ;    00H for normal speed
MOV    PCON,A                         ;    80H for doubled
MOV    DPTR,#BaudRatCnt               ; Load baud rate count
CLR    A
MOVC   A,@A+DPTR
MOV    TH1,A
MOV    TL1,A
MOV    TH0,#END_FRM_CNT_HI            ; End-of-frame interrupt occurs
MOV    TL0,#END_FRM_CNT_LO            ;    28 scan lines from bottom
```

```
; All of display memory will now be tested.  An alternating bit test is
; performed followed by an address test.  Here we begin to write the first
; pattern set for the alternating bit test.

   MOV   P2,#DspMemBas.SR.PAGE             ; Start at first byte of
   MOV   R0,#DspMemBas.AN.OFST             ;    display memory
   MOV   A,#0AAH                           ; Initial test pattern also
   MOV   MemTstTmp,A                       ;    saved for verification
   MOV   R7,#DSP_MEM_SIZ.SR.PAGE           ; Number of pages to test

MT1:                                 ; For each page
   MOV   R6,#4                             ; Number of groups per page

MT2:                                 ; For each group in a page
   MOV   R5,#PAG_SIZ/4                     ; Number of bytes per group

MT3:                                 ; For each byte in a group
   MOVX  @R0,A                             ; Write test pattern to memory
   INC   R0                                ;    then address next byte
   DJNZ  R5,MT3                            ; Loop until end of group

   CPL   A                                 ; Change pattern for next group
   DJNZ  R6,MT2                            ; Loop until end of page

   INC   P2                                ; Address next page
   DJNZ  R7,MT1                            ; Loop until end of memory

; Next, the patterns are verified.  As each byte is checked the complemented
; pattern is written back.  This section is performed twice so that each bit
; is tested with both a one and a zero.

MT4:                                 ; Verification (done twice)
   MOV   P2,#DspMemBas.SR.PAGE             ; Start at first page (R0 is 0)
   MOV   R7,#DSP_MEM_SIZ.SR.PAGE           ; Number of pages to test

MT5:                                 ; For each page
   MOV   R6,#4                             ; Number of groups per page

MT6:                                 ; For each group in a page
   MOV   R5,#PAG_SIZ/4                     ; Number of bytes per group
```

```
MT7:                                 ; For each byte in a group
   MOVX  A,@R0                             ; Read memory, check expected
   CJNE  A,MemTstTmp,RstErr                ;    pattern and quit on an error

   CPL   A                                 ; Change pattern and
   MOVX  @R0,A                             ;    write it to memory
   INC   R0                                ;    then address next byte
   DJNZ  R5,MT7                            ; Loop until end of group

   MOV   MemTstTmp,A                       ; Save next verification pattern
   DJNZ  R6,MT6                            ; Loop until end of page

   INC   P2                                ; Address next page
   DJNZ  R7,MT5                            ; Loop until end of memory
   CPL   A
   MOV   MemTstTmp,A
   CJNE  A,#0AAH,MT4                       ; Verify again, if first time

; The display memory has passed the alternating bit test; now the initial
; address test patterns will be written.  Each byte's offset address (within
; it's page) is exclusive-or'ed with it's page address.  This ensures a
; different pattern for each byte in a page and for each byte at the same
; offset in different pages.

   MOV   R2,#DspMemBas.SR.PAGE             ; Start at first page (R0 is 0)
   MOV   R7,#DSP_MEM_SIZ.SR.PAGE           ; Number of pages to test

MT8:                                 ; For each page
   MOV   P2,R2                             ; Address page

MT9:                                 ; For each byte in a page
   MOV   A,R0                              ; Make pattern from offset and
   XRL   A,R2                              ;    page address
   MOVX  @R0,A                             ; Write test pattern to memory
   DJNZ  R0,MT9                            ; Loop until page is finished

   INC   R2                                ; Prepare for next page
   DJNZ  R7,MT8                            ; Loop until end of memory
```

```
; Next, the address patterns are verified.  As each byte is checked a zero
; is written back.  This aids the verification process as well as providing
; a basis (all zero memory) for subsequent display memory initialization.

   MOV   R2,#DspMemBas.SR.PAGE              ; Start at first page (R0 is 0)
   MOV   R7,#DSP_MEM_SIZ.SR.PAGE            ; Number of pages to test

MT10:                                  ; For each page
   MOV   P2,R2                              ; Address page

MT11:                                  ; For each byte in a page
   MOV   A,R0                               ; Make pattern from offset and
   XRL   A,R2                               ;    page address then
   MOV   MemTstTmp,A                        ;    save for verification check
   MOVX  A,@R0                              ; Read memory, check expected
   CJNE  A,MemTstTmp,RstErr                 ;    pattern and quit on an error

   CLR   A                                  ; Write zero
   MOVX  @R0,A                              ;    to memory
   DJNZ  R0,MT11                            ; Loop until page is finished

   INC   R2                                 ; Prepare for next page
   DJNZ  R7,MT10                            ; Loop until end of memory

; Display memory is now tested and initialized to all zeroes.  We proceed with
; testing the Am8052.

   SJMP  AT1

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

RstErr:

; If some initialization error occurs then the following procedure is
; executed.

   SJMP  $                                  ; Currently we just stick here

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
AT1:

; NOTE:  There is currently no test of the Am8052.  A simple accessibility
;        test, which writes and verifies patterns in the read/write registers
;        of the Am8052 could be added here.  THIS TEST SHOULD LEAVE THE
;        Am8052 DISABLED AT ALL TIMES.

; The Am8052 is now known to be accessible.  We assume it works and begin
; it's initialization.  The display is already disabled; all other registers
; will be written except Mode Register 2.  This latter is deferred until after
; the display is enabled.

   MOV   R1,#AtrEnbInd                      ;Attribute Port Enable
   MOV   R2,#067H
   MOV   R3,#0FFH
   LCALL WrAm8052Reg
   MOV   R1,#AtrRdfInd                      ;Attribute Redefinition
   MOV   R2,#000H
   MOV   R3,#000H
   LCALL WrAm8052Reg
   MOV   R1,#TOPSftHiInd                    ;Top of Page Soft Pointer
   LCALL WrAm8052Reg
   MOV   R1,#TOPSftLoInd
   LCALL WrAm8052Reg
   MOV   R1,#TOWSftHiInd                    ;Top of Window Soft Pointer
   LCALL WrAm8052Reg
   MOV   R1,#TOWSftLoInd
   LCALL WrAm8052Reg
   MOV   R1,#AtrFlgInd                      ;Attribute Flag
   LCALL WrAm8052Reg
   MOV   R1,#TOPHrdHiInd                    ;Top of Page & Wind Hard Pointers
   LCALL WrAm8052Reg                        ;    high word = 0
   MOV   R1,#TOWHrdHiInd
   LCALL WrAm8052Reg
   MOV   R1,#TOPHrdLoInd                    ;Top of Page & Wind Hard ready
   MOV   R2,#ClrFntMDB.SR.PAGE              ;    for font load
   MOV   R3,#ClrFntMDB.AN.OFST
```

```
LCALL WrAm8052Reg
MOV   R1,#TOWHrdLoInd
MOV   R2,#ClrFntWDB.SR.PAGE
MOV   R3,#ClrFntWDB.AN.OFST
LCALL WrAm8052Reg
MOV   R1,#DMABstInd                     ;DMA Burst and Space
MOV   R2,#010H
MOV   R3,#040H
LCALL WrAm8052Reg
MOV   R1,#VrtWthInd                     ;Vertical Sync Width
MOV   R2,#002H                          ;    and Vertical Scan Delay
MOV   R3,#04FH
LCALL WrAm8052Reg
MOV   R1,#VrtActLneInd                  ;Vertical Active Lines
MOV   R2,#001H
MOV   R3,#067H
LCALL WrAm8052Reg
MOV   R1,#VrtTotLneInd                  ;Vertical Total Lines
MOV   R2,#001H
MOV   R3,#06CH
LCALL WrAm8052Reg
MOV   R1,#HsyncVIntInd                  ;Horizontal Synch Width
MOV   R2,#001H                          ;    and Vertical Event Row
MOV   R3,#020H
LCALL WrAm8052Reg
MOV   R1,#HDrvInd                       ;Horizontal Drive
MOV   R2,#000H
MOV   R3,#020H
LCALL WrAm8052Reg
MOV   R1,#HScnDlyInd                    ;Horizontal Scan Delay
MOV   R2,#000H
MOV   R3,#022H
LCALL WrAm8052Reg
MOV   R1,#HTotCntInd                    ;Horizontal Total Count
MOV   R2,#000H
MOV   R3,#0DBH
LCALL WrAm8052Reg
MOV   R1,#HTotDspInd                    ;Horizontal Total Display
MOV   R2,#000H
MOV   R3,#0D9H
LCALL WrAm8052Reg
```

```
; We next initialize a portion of display memory in a special way which
; is used only for initially blanking the character generator RAM.  This
; clear font display requires only a single main definition block, sixteen
; row control blocks (each with its own single character), and two attribute
; words and a row redefinition block which all RCBs use in common.  There is
; also a termination window definition block.

; First, the main definition block is written.  Since memory is known to
; contain all zeroes, only those parts of the MDB with non-zero values will
; be written.

   MOV   P2,#ClrFntMDB.SR.PAGE                ; Address page of the MDB at
   MOV   R0,#ClrFntMDB.AN.OFST+MDB_RowPag     ;    offset of top row pointer
   MOV   A,#ClrFntRCBBas.SR.PAGE              ; Point to page
   MOVX  @R0,A                                ;    of top row
   INC   R0                                   ;    and
   MOV   A,#ClrFntRCBBas.AN.OFST              ;    its offset
   MOVX  @R0,A                                ;
   INC   R0                                   ;
   MOV   A,#-1                                ; Impossible cursor position
   MOVX  @R0,A                                ;    entered for x
   INC   R0                                   ;    and
   MOVX  @R0,A                                ;    for y
   INC   R0                                   ;
   MOV   A,#001H                              ; Set the FAT bit to fetch
   MOVX  @R0,A                                ;    an attribute for fill chars
   MOV   R0,#ClrFntMDB.AN.OFST+MDB_Tslc       ; Set MDB's TSLC field to
   MOV   A,#15.SL.2                           ;    15 (which means 16 scan
   MOVX  @R0,A                                ;    lines per character row)

; Next, each of the sixteen row control blocks is initialized.  Again, only
; non-zero bytes are written.

   MOV   R2,#ClrFntRCBBas.SR.PAGE             ; Address page of first RCB
   MOV   R3,#ClrFntChrBas.AN.OFST             ; Address offset of character
   MOV   R4,#ClrFntAtr.SR.PAGE                ; Address page and
   MOV   R5,#ClrFntAtr.AN.OFST                ;    offset of attributes
   MOV   R6,#ClrFntRRB.SR.PAGE                ; Address page and
   MOV   R7,#ClrFntRRB.AN.OFST                ;    offset of row redef block
   MOV   R1,#16                               ; Number of RCBs to be made
```

```
CF1:
    MOV   P2,R2                                ; Address RCB at
    MOV   R0,#ClrFntRCBBas.AN.OFST+RCB_RdfLnk  ;    link bit offset (1st byte)
    MOV   A,#080H                              ; Set redef block link bit
    MOVX  @R0,A                                ;    to indicate RRB ptr present
    MOV   A,R0                                 ; Offset of RCB to be written
    MOV   R0,#ClrFntRCBBas.AN.OFST+RCB_RowOff  ;    as offset of next RCB
    MOVX  @R0,A                                ;    (all RCBs at same offset)
    MOV   R0,#ClrFntRCBBas.AN.OFST+RCB_1st+SEG_NumVis
    MOV   A,#1                                 ; One character specified per row
    MOVX  @R0,A                                ;    (rest are filled with null)
    MOV   R0,#ClrFntRCBBas.AN.OFST+RCB_1st+SEG_ChrPag
    MOV   A,R2                                 ; Put in page address
    MOVX  @R0,A                                ;    of char (same as its RCB)
    INC   R0                                   ;    and then
    MOV   A,R3                                 ;    its offset
    MOVX  @R0,A                                ;
    MOV   R0,#ClrFntRCBBas.AN.OFST+RCB_1st+SEG_AtrPag
    MOV   A,R4                                 ; Put in page address
    MOVX  @R0,A                                ;    of attributes
    INC   R0                                   ;    and then
    MOV   A,R5                                 ;    their beginning offset
    MOVX  @R0,A                                ;
    MOV   R0,#ClrFntRCBBas.AN.OFST+RCB_ClrRdfPag
    MOV   A,R6                                 ; Put in page address
    MOVX  @R0,A                                ;    of row redef block
    INC   R0                                   ;    and then
    MOV   A,R7                                 ;    its offset
    MOVX  @R0,A                                ;
    INC   R2                                   ; Prepare for next page
    MOV   R0,#ClrFntRCBBas.AN.OFST+RCB_RowPag  ; Put next page address
    MOV   A,R2                                 ;    into page address
    MOVX  @R0,A                                ;    for next RCB
    DJNZ  R1,CF1                               ; Loop until all RCBs are written
    DEC   A                                    ; Make the last RCB
    MOVX  @R0,A                                ;    point to itself

; Then, we initialize the attribute words.  The first one is set to force
; a load of character generator RAM.  The second is a latched but otherwise
; innocuous attribute which is fetched for the fill characters.
```

```
MOV   P2,R4                                ; Address page and
MOV   R0,#ClrFntAtr.AN.OFST                ;    offset of first attribute
MOV   A,#047H                              ; Set cursor bit and width to
MOVX  @R0,A                                ;    load data for 7x9 chars
INC   R0                                   ;    initially, also
MOV   A,#010H                              ;    set required superscript
MOVX  @R0,A                                ;    attribute
INC   R0                                   ;
MOV   A,#087H                              ; Second word is latched,
MOVX  @R0,A                                ;    nothing special attribute

; And now, the row redefinition block is initialized to load zeroes into each
; slice of each character.  This is done by leaving the row attribute fields
; all zeroes and forcing all slices of a character to be loaded with each row.

MOV   P2,R6                                ; Address page and
MOV   R0,#ClrFntRRB.AN.OFST                ;    offset of RRB
MOV   A,#15.SL.2                           ; Set 16 scan lines per row into
MOVX  @R0,A                                ;    RRB's TSLC field (1st byte)
MOV   R0,#ClrFntRRB.AN.OFST+RRB_SpcsLo_Spce; Set superscript start/end lines
MOV   A,#15                                ;    to 0 and 15 so that it spans
MOVX  @R0,A                                ;    the entire character row
MOV   R0,#ClrFntRRB.AN.OFST+RRB_CursLo_Cure; Set cursor start and end lines
MOV   A,#15                                ;    to 0 and 15 so that it spans
MOVX  @R0,A                                ;    the entire character row

; Finally, a window definition block is defined with its positioned near the
; bottom of the display.  It will be fetched by the Am8052 and show the first
; of the blanked character rows.

MOV   P2,#ClrFntWDB.SR.PAGE                ; Address WDB at
MOV   R0,#ClrFntWDB.AN.OFST+WDB_RowPag     ;    offset to top row pointer
MOV   A,#ClrFntRCBBas.SR.PAGE              ; Point to first RCB (just in
MOVX  @R0,A                                ;    case)
INC   R0                                   ;
MOV   A,#ClrFntRCBBas.AN.OFST              ;
MOVX  @R0,A                                ;
```

```
    MOV   R0,#ClrFntWDB.AN.OFST+WDB_NxtPag     ; Address pointer to next WDB and
    MOV   A,#ClrFntWDB.SR.PAGE                 ;    make it point to itself
    MOVX  @R0,A                                ;
    INC   R0                                   ;
    MOV   A,#ClrFntWDB.AN.OFST                 ;
    MOVX  @R0,A                                ;
    INC   R0                                   ;
    MOV   A,#20                                ; Set second from bottom row
    MOVX  @R0,A                                ;    for start
    INC   R0                                   ;    and
    MOV   A,#21                                ;    bottom partial row
    MOVX  @R0,A                                ;    for end

; We next set things in motion.  Interrupts are enabled and the display
; is enabled.  We need the Am8052 operating in order to load the character
; generator RAM.

    MOV   IE,#087H                             ; Enable interrupts (not serial)
    MOV   R1,#ModReg1Ind
    MOV   R2,#0C8H                             ; Enable the Am8052 display
    MOV   R3,#001H                             ;    operations
    LCALL WrAm8052Reg
    MOV   R1,#ModReg2Ind
    MOV   R2,#096H                             ; Enable Am8052 vertical
    MOV   R3,#0D2H                             ;    interrupt
    LCALL WrAm8052Reg
    LCALL DlyTilEndFrm                         ; Be sure that all is working

; Now we will zero the entire character generator.  This section is done
; twice; first for the 7x9 characters and then for the 5x7 characters.
; Sixteen character cells are cleared in each frame.

CF2:                                      ; For each set of chars (7x9 & 5x7)
    CLR   A                                    ; Start with null (char code 0)

CF3:                                      ; For each frame (group of 16 chars)
    MOV   P2,#ClrFntChrBas.SR.PAGE             ; Address page and
    MOV   R0,#ClrFntChrBas.AN.OFST             ;    offset of first character
    MOV   R7,#16                               ; Number of characters to load
    LCALL DlyTilEndFrm                         ; Wait for an auspicious omen
```

```
CF4:                                      ; For each character (row) in the frame
    MOVX  @R0,A                                ; Store code of char to be loaded
    INC   P2                                   ; Next page (next character) and
    INC   A                                    ;    next cell to be loaded
    DJNZ  R7,CF4                               ; Loop until frame is set up

    JNZ   CF3                                  ; Loop until back to null char

    LCALL DlyTilEndFrm                         ; Ensure that we are finished
    MOV   DPTR,#ClrFntAtr                      ; Check first attribute for
    MOVX  A,@DPTR                              ;    width of load character
    CJNE  A,#047H,CF5                          ; Skip if just loaded 5x7 chars

    MOV   A,#044H                              ; Else, set up to load
    MOVX  @DPTR,A                              ;    5x7 set and
    SJMP  CF2                                  ;    go do it

; Now that the character generator RAM is cleared we need to disable the
; Am8052 in preparation for initializing memory for actual operation.

CF5:                                      ; Finished clearing character generator
    MOV   IE,#0                                ; Disable all interrupts
    MOV   R1,#ModReg1Ind                       ; Using Mode 1 Register
    MOV   R2,#0CCH                             ;    blank display (VB=1)
    MOV   R3,#001H                             ;    but leave Am8052 enabled
    LCALL WrAm8052Reg                          ;

; The following code initializes all of memory, both internal and external,
; for normal operation.

MemInt:
    MOV   R1,#126                              ; Clear all but R0 and R1
    MOV   A,#00H
    MOV   R0,#02H
IntVar:                                   ; Loop point for clearing variables
    MOV   @R0,A
    INC   R0
    DJNZ  R1,IntVar
```

```
MOV    CurAtr,#00H                  ; Initial attribute is 00
MOV    ActCol,#00H                  ; Initialize to leftmost col
MOV    ActRow,#07H                  ; First window row is 7th in list
MOV    CurRow,#WndRCB7.SR.PAGE      ; Page value to active row
MOV    VisCol,#00H                  ; Always 0 in window
MOV    VisRow,#07H
MOV    BgnRow,#WndRCB0.SR.PAGE      ; Page value to beginning of list
MOV    TopRow,#WndRCB7.SR.PAGE      ; Page value to Am8052 bgn of lst
MOV    BtmRow,#WndRCB13.SR.PAGE     ; Page value to last visible row
MOV    RemRow,#WndRCB13.SR.PAGE     ; Page value to rows below dsp
MOV    EndRow,#WndRCB13.SR.PAGE     ; Page value of last row in list
MOV    ExtRow,#WndRCB14.SR.PAGE     ; Page value of extra
MOV    R3,#WndVarBuf.SR.PAGE
MOV    R4,#WndVarBuf.AN.OFST
LCALL  HalfSwap
MOV    CurAtr,#00H                  ; Initial attribute is 00
MOV    ActCol,#00H                  ; Initialize to leftmost col
MOV    ActRow,#00H                  ; First msg row is first in list
MOV    CurRow,#MsgRCB.SR.PAGE       ; Page value to active row
MOV    VisCol,#00H                  ; Start left aligned
MOV    VisRow,#00H
MOV    BgnRow,#MsgRCB.SR.PAGE       ; Page value to beginning of list
MOV    TopRow,#MsgRCB.SR.PAGE       ; Page value to Am8052 bgn of lst
MOV    BtmRow,#MsgRCB.SR.PAGE       ; Page value to last visible row
MOV    RemRow,#MsgRCB.SR.PAGE       ; Page value to rows below dsp
MOV    EndRow,#MsgRCB.SR.PAGE       ; Page value of last row in list
MOV    ExtRow,#MsgRCB.SR.PAGE       ; Page value of extra
MOV    R3,#MsgVarBuf.SR.PAGE
MOV    R4,#MsgVarBuf.AN.OFST
LCALL  HalfSwap
MOV    CurAtr,#00H                  ; Initial attribute is 00
MOV    ActCol,#00H                  ; Initialize to leftmost col
MOV    ActRow,#06H                  ; First bgrd row is 6th in list
MOV    CurRow,#BgdRCB6.SR.PAGE      ; Page value to active row
MOV    VisCol,#00H                  ; Start left aligned
MOV    VisRow,#06H
MOV    BgnRow,#BgdRCB0.SR.PAGE      ; Page value to bgn of list
MOV    TopRow,#BgdRCB6.SR.PAGE      ; Page value to Am8052 bgn lst
MOV    BtmRow,#BgdRCB29.SR.PAGE     ; Page value to last visible row
MOV    RemRow,#BgdRCB29.SR.PAGE     ; Page value to rows below dsp
```

```
   MOV    EndRow,#BgdRCB29.SR.PAGE        ; Page value of last row in list
   MOV    ExtRow,#BgdRCB30.SR.PAGE        ; Page value of extra
   MOV    R3,#BgdVarBuf.SR.PAGE
   MOV    R4,#BgdVarBuf.AN.OFST
   LCALL  HalfSwap
   MOV    DspWid,#80                      ;Set parameters used in program
   MOV    DspHgt,#24                      ;    Many are offsets into pages
   MOV    ColAdd,#1
   MOV    RowAdd,#0
   MOV    RcbOff,#BgdRCB0.AN.OFST
   MOV    ChrOff,#BgdChrBuf0.AN.OFST
   MOV    AtrOff,#BgdAtrBuf0.AN.OFST
   MOV    WndCol,#28
   SETB   CsrZonFlg
   MOV    CsrSiz,#00FH
   MOV    HstRcvInsOff,#HstRcvBuf.AN.OFST
   MOV    HstRcvExtOff,#HstRcvBuf.AN.OFST

; Initialize characters and attributes for the background and the message row.

   MOV    P2,#BgdRCB0.SR.PAGE             ;Background Row 0 page
   MOV    R2,#32                          ;count of rows (includes msg)
   MOV    A,#' '                          ;blank all characters
FilRow:                                ;row loop point
   MOV    R0,#BgdChrBuf0.AN.OFST          ;offset of first character
   MOV    R1,#128                         ;128 characters per row
FilChr:                                ;character loop point
   MOVX   @R0,A
   INC    R0                              ;next character
   DJNZ   R1,FilChr
                                       ;end of row
   INC    P2                              ;next row
   DJNZ   R2,FilRow
                                       ;P2 now points to attributes
   MOV    R2,#32                          ;32 rows again
   MOV    R6,#000H
   MOV    R7,#007H
FilAtrRow:                             ;row loop point
   MOV    R0,#BgdAtrBuf0.AN.OFST          ;offset of attributes
   MOV    R1,#128                         ;128 per row
```

```
FilAtr:                                        ;attribute loop point
   MOV    A,R7                                        ;set two bytes
   MOVX   @R0,A
   INC    R0
   MOV    A,R6
   MOVX   @R0,A
   INC    R0                                          ;next attribute
   DJNZ   R1,FilAtr
                                               ;end of row
   INC    P2                                          ;next row
   DJNZ   R2,FilAtrRow

; Initialize the background row control blocks.

   MOV    R2,#BgdRCB0.SR.PAGE                         ;page for row 0 control block
   MOV    R1,#31                                      ;only initializing background
IntBgd:                                        ;background RCB init loop point
   MOV    P2,R2                                       ;set page of RCB
   MOV    R0,#BgdRCB0.AN.OFST+RCB_RdfLnk              ;set flag to show row follows
   MOV    A,#080H
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_1st+SEG_Cont ;1st is not last seg
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_2nd+SEG_Cont ;2nd is not last seg
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_3rd+SEG_Cont ;3rd is not last seg
   MOVX   @R0,A
   MOV    A,#BgdFncChr0.SR.PAGE                       ;page for function character
   MOV    R0,#BgdRCB0.AN.OFST+RCB_1st+SEG_ChrPag     ;all func chars in 1 page
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_1st+SEG_AtrPag     ;same for attributes
   MOVX   @R0,A
   MOV    A,#1
   MOV    R0,#BgdRCB0.AN.OFST+RCB_1st+SEG_NumVis     ;1 function character (vis)
   MOVX   @R0,A
   MOV    A,#28                                       ;28 characters in 2nd segment
   MOV    R0,#BgdRCB0.AN.OFST+RCB_2nd+SEG_NumVis
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_1st+SEG_ChrOff     ;function char pos
   MOV    A,#BgdFncChr0.AN.OFST
```

```
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_1st+SEG_AtrOff     ;  and attributes
   MOV    A,#BgdFncAtr0.AN.OFST
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_2nd+SEG_ChrPag     ;R2 has page for this row
   MOV    A,R2
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_3rd+SEG_ChrPag
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_4th+SEG_ChrPag
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_2nd+SEG_ChrOff     ;set offset for char start
   MOV    A,#BgdChrBuf0.AN.OFST
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_2nd+SEG_AtrPag     ;   and attrib start
   MOV    A,R2
   ORL    A,#20H                                      ;set the attribute pages
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_3rd+SEG_AtrPag
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_4th+SEG_AtrPag
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_3rd+SEG_NumVis     ;40 visible in 3rd seg
   MOV    A,#40
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_3rd+SEG_ChrOff
   MOV    A,#BgdChrBuf0.AN.OFST+28                    ;starting 28 past first char
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_3rd+SEG_AtrOff
   MOV    A,#BgdAtrBuf0.AN.OFST+2*28                  ;attrib start 28*2 after 1st
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_4th+SEG_NumVis     ;60 visible in 3rd seg
   MOV    A,#60
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_4th+SEG_ChrOff
   MOV    A,#BgdChrBuf0.AN.OFST+28+40                 ;starting 28+40 after 1st
   MOVX   @R0,A
   MOV    R0,#BgdRCB0.AN.OFST+RCB_4th+SEG_AtrOff
   MOV    A,#BgdAtrBuf0.AN.OFST+2*(28+40)             ;attrib at 2*(28+40)
   MOVX   @R0,A
```

```
    MOV   R0,#BgdRCB0.AN.OFST+RCB_BgdRdfPag        ;all point to same
    MOV   A,#NrmRRB.SR.PAGE                        ;  row redef block
    MOVX  @R0,A
    MOV   R0,#BgdRCB0.AN.OFST+RCB_BgdRdfOff
    MOV   A,#NrmRRB.AN.OFST
    MOVX  @R0,A
    MOV   A,R2                                     ;next page
    INC   A
    MOV   R0,#BgdRCB0.AN.OFST+RCB_RowPag           ;is page in "next" link
    MOVX  @R0,A
    MOV   R2,A                                     ;and next for loop
    DJNZ  R1,IntBgd                                ;continue for 31 rows

; Initialize message Row Control Block

    MOV   R2,#MsgRCB.SR.PAGE                       ;P2 = R2 = msg page
    MOV   P2,R2
    MOV   R0,#MsgRCB.AN.OFST+RCB_RdfLnk
    MOV   A,#080H
    MOVX  @R0,A
    MOV   R0,#MsgRCB.AN.OFST+RCB_RowPag            ;"next" is last wnd RCB
    MOV   A,#WndRCB14.SR.PAGE
    MOVX  @R0,A
    INC   R0
    MOV   A,#WndRCB14.AN.OFST
    MOVX  @R0,A
    MOV   R0,#MsgRCB.AN.OFST+RCB_1st+SEG_NumVis    ;1 visible in function
    MOV   A,#1
    MOVX  @R0,A
    MOV   R0,#MsgRCB.AN.OFST+RCB_1st+SEG_Cont      ;1st seg is not last
    MOV   A,#080H
    MOVX  @R0,A
    MOV   R0,#MsgRCB.AN.OFST+RCB_1st+SEG_ChrPag    ;char is in function page
    MOV   A,#MsgFncChr.SR.PAGE
    MOVX  @R0,A
    INC   R0
    MOV   A,#MsgFncChr.AN.OFST                     ;char is function char
    MOVX  @R0,A
    MOV   R0,#MsgRCB.AN.OFST+RCB_1st+SEG_AtrPag    ;attrib is func attrib
    MOV   A,#MsgFncAtr.SR.PAGE
```

```
    MOVX  @R0,A
    INC   R0
    MOV   A,#MsgFncAtr.AN.OFST
    MOVX  @R0,A
    MOV   R0,#MsgRCB.AN.OFST+RCB_2nd+SEG_NumVis    ;128 visible in next segment
    MOV   A,#080H
    MOVX  @R0,A
    MOV   R0,#MsgRCB.AN.OFST+RCB_2nd+SEG_ChrPag    ;characters in RCB page
    MOV   A,R2
    MOVX  @R0,A
    INC   R0
    MOV   A,#MsgChrBuf.AN.OFST                     ;  at msg buffer offset
    MOVX  @R0,A
    MOV   R0,#MsgRCB.AN.OFST+RCB_2nd+SEG_AtrPag    ;attrib page calculated
    MOV   A,R2                                     ;  from RCB page
    ORL   A,#020H
    MOVX  @R0,A
    INC   R0
    MOV   A,#MsgAtrBuf.AN.OFST                     ;attrib offset
    MOVX  @R0,A
    INC   R0                                       ;then set row redef ptr
    INC   R0                                       ;  to std location
    INC   R0
    MOV   A,#NrmRRB.SR.PAGE
    MOVX  @R0,A
    INC   R0
    MOV   A,#NrmRRB.AN.OFST
    MOVX  @R0,A

;   We now initialize the Window memory.

    MOV   P2,#WndChrBuf0.SR.PAGE           ;P2 points to first wnd row
    MOV   R2,#15                           ;R2 has count of window rows
    MOV   A,#' '                           ;A has blank character
FilWndRow:                               ;window row loop point
    MOV   R0,#WndChrBuf0.AN.OFST           ;set character offset
    MOV   R1,#40                           ;R1 = character count
FilWndChr:                               ;window character loop point
    MOVX  @R0,A                            ;blank the character
    INC   R0                               ;next character
    DJNZ  R1,FilWndChr
```

```
   INC   P2                           ;next row
   DJNZ  R2,FilWndRow
                                ;done with window characters
   MOV   P2,#WndAtrBuf0.SR.PAGE       ;P2 = first wnd attrib page
   MOV   R2,#15                       ;R2 = count of rows
   MOV   A,#07                        ;A = initial attrib
FilWndAtrRow:                   ;window row loop point
   MOV   R0,#WndAtrBuf0.AN.OFST       ;R0 = ptr to attrib
   MOV   R1,#40                       ;R1 = attrib count
FilWndAtr:                      ;window attribute loop point
   MOVX  @R0,A                        ;set attrib
   INC   R0                           ;next attrib
   INC   R0
   DJNZ  R1,FilWndAtr

   INC   P2                           ;next row
   DJNZ  R2,FilWndAtrRow
                                ;done with window attributes
   MOV   R2,#WndRCB0.SR.PAGE          ;R2 = window row 0 page
   MOV   R1,#15                       ;R1 = window row count
IntWnd:
   MOV   P2,R2                             ;point to wnd page
   MOV   R0,#WndRCB0.AN.OFST+RCB_RdfLnk    ;indicate row follows
   MOV   A,#080H
   MOVX  @R0,A
   MOV   R0,#WndRCB0.AN.OFST+RCB_Seg+SEG_NumVis    ;one seg with 40 visible
   MOV   A,#40
   MOVX  @R0,A
   MOV   R0,#WndRCB0.AN.OFST+RCB_Seg+SEG_ChrPag    ;chars on same page
   MOV   A,R2
   MOVX  @R0,A
   MOV   R0,#WndRCB0.AN.OFST+RCB_Seg+SEG_ChrOff    ;  at buffer offset
   MOV   A,#WndChrBuf0.AN.OFST
   MOVX  @R0,A
   MOV   R0,#WndRCB0.AN.OFST+RCB_Seg+SEG_AtrPag    ;attrib page calculated
   MOV   A,R2                                      ;  from char page
   ORL   A,#010H
   MOVX  @R0,A
   MOV   R0,#WndRCB0.AN.OFST+RCB_Seg+SEG_AtrOff    ;attribute offset const
   MOV   A,#WndAtrBuf0.AN.OFST
```

```
   MOVX  @R0,A
   MOV   R0,#WndRCB0.AN.OFST+RCB_WndRdfPag         ;use the std row redef
   MOV   A,#NrmRRB.SR.PAGE
   MOVX  @R0,A
   MOV   R0,#WndRCB0.AN.OFST+RCB_WndRdfOff
   MOV   A,#NrmRRB.AN.OFST
   MOVX  @R0,A
   MOV   R0,#WndRCB0.AN.OFST+RCB_RowOff            ;next row at std offset
   MOV   A,#WndRCB0.AN.OFST
   MOVX  @R0,A
   MOV   A,R2                                      ;  on next page
   INC   A
   MOV   R0,#WndRCB0.AN.OFST+RCB_RowPag
   MOVX  @R0,A
   MOV   R2,A                                      ;next row
   DJNZ  R1,IntWnd

;  Initialize Termination Row Control block in last window row

   MOV   A,#WndRCB14.SR.PAGE                ;page of last window row
   MOV   TrmRow,A                           ;    is page of termination row
   MOV   P2,#BgdRCB29.SR.PAGE               ;make row 29 last in brgd
   MOV   R0,#BgdRCB29.AN.OFST+RCB_RowPag
   MOVX  @R0,A
   INC   R0
   MOV   A,#WndRCB14.AN.OFST                ;also set termination offset
   MOV   TrmOff,A
   MOVX  @R0,A

   MOV   R2,TrmRow                          ;R2 = P2 = termination row
   MOV   P2,R2
   MOV   R0,#WndRCB14.AN.OFST+RCB_RowPag    ;term row points to itself
   MOV   A,R2
   MOVX  @R0,A
   MOV   A,TrmOff
   INC   R0
   MOVX  @R0,A
   INC   R0
   CLR   A                                  ;term row has no hidden chars
```

```
     MOVX  @R0,A
     INC   R0
     MOV   A,#1                          ;    and one visible char
     MOVX  @R0,A
     MOV   R0,#WndRCB14.AN.OFST+RCB_Seg+SEG_AtrPag
     MOV   A,#TrmAtr.SR.PAGE             ;term attrib page
     MOVX  @R0,A
     INC   R0
     MOV   A,#TrmAtr.AN.OFST             ;term attrib offset
     MOVX  @R0,A

; Initialize Function Character and Attribute

     MOV   DPTR,#BgdFncChr0              ;function characters are blank
     MOV   A,#' '
     MOVX  @DPTR,A
     INC   DPTR
     MOVX  @DPTR,A
     INC   DPTR
     MOV   A,#002H                       ;1st function attrib
     MOVX  @DPTR,A
     INC   DPTR
     MOV   A,#090H
     MOVX  @DPTR,A
     INC   DPTR
     MOV   A,#004H                       ;2nd function attrib
     MOVX  @DPTR,A
     INC   DPTR
     CLR   A
     MOVX  @DPTR,A

;  Initialize Termination Attribute

     MOV   DPTR,#TrmAtr                  ;termination attrib
     MOV   A,#087H
     MOVX  @DPTR,A
     INC   DPTR
     CLR   A
     MOVX  @DPTR,A
```

```
;  Initialize Message Function Character and Attribute

     MOV   DPTR,#MsgFncChr                   ;function character is blank
     MOV   A,#' '
     MOVX  @DPTR,A
     INC   DPTR
     INC   DPTR
     MOV   A,#002H                           ;function attribute
     MOVX  @DPTR,A
     INC   DPTR
     MOV   A,#080H
     MOVX  @DPTR,A

;  Initialize Background Main Definition Blocks

     MOV   P2,#BgdMDB0.SR.PAGE               ;P2 = 1st bgrd main def
     MOV   R0,#BgdMDB0.AN.OFST+MDB_RowPag    ;R0 = MDB 1st row page ptr
     MOV   R1,#2                             ;R2 is count of main defs
InitMDB:                                   ;main def loop point
     MOV   A,TopRow                          ;1st row is Top Row
     MOVX  @R0,A
     INC   R0
     INC   R0
     MOV   A,#001H                           ;cursor in 1st visible col
     MOVX  @R0,A
     INC   R0
     MOV   A,#000H                           ;cursor on first row
     MOVX  @R0,A
     INC   R0
     MOV   A,#001H                           ;set FAT bit
     MOVX  @R0,A
     INC   R0
     MOV   A,#' '                            ;fill char is blank
     MOVX  @R0,A
     MOV   A,R0                              ;scanline count for top visible
     ADD   A,#5
     MOV   R0,A
     MOV   A,#034H
     MOVX  @R0,A
     MOV   R0,#BgdMDB1.AN.OFST+MDB_RowPag    ;next main def 1st row page
     DJNZ  R1,InitMDB
```

```
;  Initialize Window Definition blocks

   MOV    P2,#WndWDB0.SR.PAGE              ;P2 = window def page
   MOV    R2,#2                            ;R2 = window def count
InitWndDefBlk:                       ;window def loop point
   MOV    R0,#WndWDB0.AN.OFST              ;scroll window flag
   MOV    A,#080H
   MOVX   @R0,A
   INC    R0
   INC    R0
   MOV    A,#WndRCB7.SR.PAGE               ;page of first row
   MOVX   @R0,A
   INC    R0
   MOV    A,#WndRCB7.AN.OFST               ;offset of first row
   MOVX   @R0,A
   INC    R0
   INC    R0
   INC    R0
   MOV    A,#TrmWDB.SR.PAGE                ;page of term wind def
   MOVX   @R0,A
   INC    R0
   MOV    A,#TrmWDB.AN.OFST                ;offset of term wind def
   MOVX   @R0,A
   INC    R0
   MOV    A,#6                             ;window begins in row 6
   MOVX   @R0,A
   INC    R0
   MOV    A,#12                            ;window ends in row 12
   MOVX   @R0,A
   INC    R0
   MOV    A,#29                            ;window begins in column 29
   MOVX   @R0,A
   INC    R0
   MOV    A,#68                            ;window ends in column 68
   MOVX   @R0,A
   MOV    R0,#WndWDB1.AN.OFST              ;ready for next def block
   MOV    P2,#WndWDB1.SR.PAGE
   DJNZ   R2,InitWndDefBlk
```

```
; Initialize the Message Window Definition Block

   MOV    P2,#MsgWDB.SR.PAGE               ;P2 is page of msg wnd block
   MOV    R0,#MsgWDB.AN.OFST+WDB_RowPag    ;Set row page (offset is 0)
   MOV    A,#MsgRCB.SR.PAGE
   MOVX   @R0,A
   INC    R0
   INC    R0
   INC    R0
   INC    R0
   MOV    A,#TrmWDB.SR.PAGE                ;next window is term wind
   MOVX   @R0,A
   INC    R0
   MOV    A,#TrmWDB.AN.OFST                ;also set term offset
   MOVX   @R0,A
   INC    R0
   MOV    A,#24                            ;msg begins at row 24
   MOVX   @R0,A
   INC    R0
   MOV    A,#24                            ;msg ends at row 24
   MOVX   @R0,A
   INC    R0
   CLR    A                                ;msg starts in col 0
   MOVX   @R0,A
   INC    R0
   MOV    A,#128                           ;msg ends in column 80
   MOVX   @R0,A

; Initialize Termination Window Definition Block

   MOV    P2,#TrmWDB.SR.PAGE               ;P2 = page of term wind block
   MOV    R0,#TrmWDB.AN.OFST+WDB_RowPag    ;Its row is the term row
   MOV    A,TrmRow
   MOVX   @R0,A
   INC    R0
   MOV    A,TrmOff
   MOVX   @R0,A
   MOV    R0,#TrmWDB.AN.OFST+WDB_BgnRow
   MOV    A,#24                            ; Start and end on bottom row
   MOVX   @R0,A
   INC    R0
   MOV    A,#24
   MOVX   @R0,A
   INC    R0
   MOV    A,#0
   MOVX   @R0,A
   INC    R0
   MOV    A,#131
   MOVX   @R0,A

; Initialize the Row Redefinition blocks (one normal, 15 for font loading)

   MOV    P2,#NrmRRB.SR.PAGE            ;start with the normal one
   MOV    R0,#NrmRRB.AN.OFST
   MOV    R2,#00FH                      ;cursor start, end
   MOV    R1,#16                        ;16 redef blocks total
InitRdfBlk:
   MOV    A,#034H                       ;scan line, char start and end
   MOVX   @R0,A
   INC    R0
   MOV    A,#04DH
   MOVX   @R0,A
   INC    R0
   MOV    A,#000H                       ;row attr, super start and end
   MOVX   @R0,A
   INC    R0
   MOV    A,#00DH
   MOVX   @R0,A
   INC    R0
   MOV    A,#000H                       ;row attr, sub start and end
   MOVX   @R0,A
   INC    R0
   MOV    A,#08DH
   MOVX   @R0,A
   INC    R0
   MOV    A,R2                          ;cursor start, end (5 bits ea.)
   ANL    A,#0F8H
   RR     A
   RR     A
   RR     A
```

```
   MOVX   @R0,A
   INC    R0
   MOV    A,R2
   MOV    R3,A
   ANL    A,#07H
   SWAP   A
   RL     A
   ORL    A,R3
   MOVX   @R0,A
   INC    R0
   MOV    A,#001H                       ;double height and underline
   MOVX   @R0,A
   INC    R0
   MOV    A,#086H
   MOVX   @R0,A
   INC    P2
   INC    R2
   CJNE   R1,#16,IRB1

   MOV    P2,#FntRRB0.SR.PAGE           ;switch to font redefs
   MOV    R2,#0                         ;no cursor
IRB1:
   MOV    R0,#FntRRB0.AN.OFST           ;offset of font redef
   DJNZ   R1,InitRdfBlk                 ;continue with font redefs

; Initialize 8052 Registers

   MOV    R1,#TOPHrdLoInd               ;Top of Page Hard points to
   MOV    R2,#BgdMDB0.SR.PAGE           ;    main definition
   MOV    R3,#BgdMDB0.AN.OFST
   LCALL  WrAm8052Reg
   MOV    R1,#TOWHrdLoInd               ;Top of Window Hard points to
   MOV    R2,#TrmWDB.SR.PAGE            ;    termination window
   MOV    R3,#TrmWDB.AN.OFST
   LCALL  WrAm8052Reg
   MOV    R1,#ModReg1Ind                ;Mode register 1
   MOV    R2,#0C8H
   MOV    R3,#001H
   LCALL  WrAm8052Reg
   MOV    R1,#ModReg2Ind                ;Mode register 2
   MOV   R2,#016H
   MOV   R3,#0D2H
   LCALL WrAm8052Reg

; Now ready to enable interrupts and load font

   MOV   IE,#097H                     ;enable interrupts
   MOV   DPTR,#Fnt_5x7                ;point to 5x7 font
   MOV   DisStt,#1                    ;    and set up to load it
IF0:
   CLR   A                            ; initialize font ram
   MOVC  A,@A+DPTR
   MOV   R0,#PrmBuf                   ; each character font in turn is
   MOV   @R0,A                        ;  loaded into the char gen
   CLR   A
   INC   DPTR
   INC   R0
   MOVC  A,@A+DPTR
   MOV   @R0,A
   CLR   A
   INC   DPTR
   INC   R0
   MOVC  A,@A+DPTR
   JZ    IF2

   INC   DPTR
   MOV   R2,A
IF1:
   CLR   A
   MOVC  A,@A+DPTR
   INC   DPTR
   MOV   @R0,A
   INC   R0
   DJNZ  R2,IF1

   MOV   A,R0
   CLR   C
   SUBB  A,#PrmBuf
   MOV   PrmCnt,A
   PUSH  DPH
```

```
   PUSH  DPL
   MOV   A,DisStt                           ; Indicate font type being loaded
   LCALL WrFntCel                           ;    and write to one cell
   POP   DPL
   POP   DPH
   SJMP  IF0

IF2:                                     ; Finished loading a font
   MOV   A,DisStt                           ; Check font that was just loaded
   JZ    C_Int1                             ; Jump if just finished 7x9

   MOV   DPTR,#Fnt_7x9                      ; Point to 7x9 font and
   MOV   DisStt,#0                          ;    set up to load it
   SJMP  IF0                                ; Go load font

C_Int1:
   LCALL ShwCsr                             ;make cursor visible
   CLR   HstRcvBsyFlg                       ;ready for host data
   LJMP  DisCon                             ;wait for host data

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; end of C_Init
```

```
"8051"
   TITLE "    CALEB 0.00     Dispatch Control"
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; C_Switch                                CALEB 0.00
;
;       Copyright 1985 Advanced Micro Devices, Inc.
;
;
; This file contains the central input stream decoder and control dispatcher.
; It is a simple state machine which parses single characters (graphics and
; controls), escape sequences and control sequences.  These types of controls
; are defined in ANSI X3.4-1977, ANSI X3.41-1974 and ANSI X3.64-1979 documents.
; The parameters included in control sequences are also decoded and stored as
; a sequence of 8-bit unsigned binary values.

   NAME  "Dispatch Control"
   PROG

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   GLB   DisCon                         ; Dispatch control procedure
   GLB   UnImpCtl                       ; Unimplemented control (common)
   GLB   Escape                         ; Escape
   GLB   CtlSeqIntro                    ; Control Sequence Introducer
   GLB   PutMap0                        ; Checks for font remaping of lower 32
   GLB   PutMap1                        ; Checks for font remaping of 3FH & 0BFH
   GLB   PutChr                         ; Write cell address and attribute.

;-----------------------------------------------------------------------------

   EXT   LoDirChrTbl,HiDirChrTbl,DirEscSeqTbl,X3_64DirSeqTbl   ; in C_Tables
   EXT   ScrollLeft,ScrollRight                                ; in C_Work
   EXT   PlcCsr                                                ; in C_Util

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
   INCLUDE C_MemMap

   SKIP
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
DirChrSttHdl:

; Handles all direct character graphics and controls.  It uses two, 128-entry
; tables, indexed by the received character, to dispatch control quickly.
;
; NOTE:  THIS PART OF THE PROCEDURE MUST BE LOCATED DIRECTLY BEFORE "DisCon".

   MOV   A,R2                            ; Use the character as an index
   MOV   PrmBuf,A                        ; Save for use in repeat case
   JBC   ACC.7,DCSH1                     ; High bit selects table (and is
                                         ;    cleared in the process)
   MOV   DPTR,#LoDirChrTbl               ; Select low table (00-7F) and
   SJMP  DCSH2                           ;    go use it

DCSH1:
   MOV   DPTR,#HiDirChrTbl               ; Select high table (80-FF)
DCSH2:
   LCALL DoWrk                           ; General address table handler

; NOTE:  Instead of jumping back to "DisCon", this part of the procedure is
;        located directly before it; therefore, we can simply fall through.

;-----------------------------------------------------------------------------

DisCon:                                ; Dispatch control procedure

; Waits for a character to be available in the host reception buffer then
; extracts it and processes it according to the current state.

   MOV   DPH,#HstRcvBuf.SR.PAGE          ; Address (page and
   MOV   DPL,HstRcvExtOff                ;    offset) of next character
DC1:                                   ; Idle while waiting for a character
   MOV   A,HstRcvCnt                     ; Check number of chars in buffer
   JZ    DC1                             ; Loop if none
   MOVX  A,@DPTR                         ; Get character from buffer
   MOV   R2,A                            ;    and keep it safe
   DEC   HstRcvCnt                       ; Reduce buffer contents count
   MOV   A,#NEAR_EMPTY_CNT               ; Check for
   SUBB  A,HstRcvCnt                     ;    nearly empty buffer
   JC    DC2                             ; Jump if still plenty to do
   CLR   HstRcvBsyFlg                    ; Ready to accept more
```

```
DC2:
   MOV   A,HstRcvExtOff               ; Pointer to next character
   INC   A                            ;    advanced
   JNZ   DC3                          ; Jump if still in buffer
   MOV   A,#HstRcvBuf.AN.OFST         ; Reset to start if went past end
DC3:
   MOV   HstRcvExtOff,A               ; Keep new next char pointer
   MOV   A,DisStt                     ; Get current state
   JZ    DirChrSttHdl                 ;    and jump directly if direct
   CLR   C                            ; Clear carry for other parts
   MOV   DPTR,#SttJmpTbl              ; Use jump table to continue with
   JMP   @A+DPTR                      ;    correct part of procedure

;-----------------------------------------------------------------------
SttJmpTbl:

; This jump table and the state constants defined in "C_MemMap" must
; correspond.  The state constants represent offsets into this table
; rather than indices (i.e. they increase by three's, not by one's).
;
; NOTE:  The first entry in the table is for direct character state,
;        as it must be to ensure proper offsets for the other jumps,
;        but direct state is always handled specially rather than
;        through this table.

   LJMP  DirChrSttHdl              ; Direct character state
   LJMP  BgnEscSttHdl              ; Beginning escape state
   LJMP  ExtEscSttHdl              ; Extended escape state
   LJMP  BgnCSISttHdl              ; Beginning control sequence state
   LJMP  PrmCSISttHdl              ; Parameter string (in ctl seq) state
   LJMP  ExtCSISttHdl              ; Extended control sequence state
   LJMP  UnImpCSISttHdl            ; Unimplemented control sequence state

;--- ------------------------------------------------------------------
BgnEscSttHdl:

; Processes the character immediately following an ESC.  It may be a final
; character, in which case the corresponding control routine is executed
; using the direct escape sequence table.  If an intermediate character is
; encountered then the state changes to handle extended escape sequences.
; An invalid character ends the escape sequence and causes both characters
; (this one and the ESC) to be disregarded; the state is set back to handle
; direct characters and controls.
```

```
   MOV   A,R2                                   ; Get character and check
   SUBB  A,#' '                                 ;    for a C0 control character
   JNC   BESH2                                  ; Jump if not a control char
BESH1:                                        ; Invalid escape sequence
   CLR   A                                      ; Clear
   MOV   CtlPtrHi,A                             ;    control routine address
   MOV   CtlPtrLo,A                             ;    (makes it unrepeatable)
   SJMP  BESH4                                  ; Finish escape sequence
BESH2:                                        ; Check for intermediate character
   SUBB  A,#('0'-' ')                           ; Reduce by intermediate range
   JNC   BESH3                                  ; Jump if not an intermediate
   MOV   DisStt,#EXT_ESC_STT                    ; Set state for extended escape
   LJMP  DisCon                                 ;    sequences and continue
BESH3:                                        ; Check for final character
   MOV   R7,A                                   ; Save index temporarily
   SUBB  A,#(DEL-'0')                           ; Check for invalid character
   JNC   BESH1                                  ; Jump if invalid sequence
   MOV   A,R7                                   ; Restore control routine index
   MOV   DPTR,#DirEscSeqTbl                     ; Use direct escape sequence
   LCALL DoWrk                                  ;    table and do control routine
BESH4:                                        ; Completed escape sequence
   MOV   DisStt,#DIR_CHR_STT                    ; Set state for single, direct
   LJMP  DisCon                                 ;    characters and continue

;-----------------------------------------------------------------------
ExtEscSttHdl:

; Processes the characters in an extended escape sequence.  Currently, no
; such controls are implemented, so this part only passes over intermediates
; until either a final character or an invalid character is encountered.  At
; that time the state is set back to handle direct characters and controls.
;
; NOTE:  Further implementations could be accomplished with the addition of
;        other tables of control routine addresses.  When a final character
;        is found, the corresponding control routine would be executed using
;        the appropriate table.  Which table is appropriate would depend on
;        the sequence of intermediate characters, which could be interpreted
;        by changing to additional states, or using another state variable.

   MOV   A,R2                                   ; Get character and check
   SUBB  A,#' '                                 ;    for a C0 control character
   JC    EESH1                                  ; Jump if it is a control char
```

```
    SUBB  A,#('0'-' ')               ; Reduce by intermediate range
    JC    EESH2                      ; Jump if it is an intermediate
EESH1:                             ; Completed escape sequence
    CLR   A                          ; Clear
    MOV   CtlPtrHi,A                 ;    control routine address
    MOV   CtlPtrLo,A                 ;    (makes it unrepeatable)
    MOV   DisStt,#DIR_CHR_STT        ; Set state for direct chars
EESH2:
    LJMP  DisCon                     ; Continue

;---------------------------------------------------------------------------
BgnCSISttHdl:

; Processes the character immediately following a Control Sequence Introducer,
; whether the CSI is a single, 8-bit character or an "ESC [" escape sequence.
; It may be a final character, in which case the corresponding control routine
; is executed using the X3.64 direct sequence table, with the parameter state
; indicating a null parameter string.  If an intermediate is encountered then
; the state is changed to handle extended control sequences, if it is a space,
; and unimplemented control sequences for any other intermediate.  This case
; also indicates a null parameter.  Any parameter character is decoded and
; changes state to decode the rest of the parameter string after initializing
; parameter accumulation.  An invalid character ends the sequence and discards
; the CSI as well.

    MOV   A,R2                       ; Get character and check
    SUBB  A,#' '                     ;    for a C0 control character
    JNC   BCSH2                      ; Jump if not a control character

BCSH1:                             ; Invalid sequence
    CLR   A                          ; Clear
    MOV   CtlPtrHi,A                 ;    control routine address
    MOV   CtlPtrLo,A                 ;    (makes it unrepeatable)
    LJMP  BCSH13                     ; Finish sequence
BCSH2:                             ; Check for intermediate character
    SUBB  A,#('0'-' ')               ; Reduce by intermediate range
    JNC   BCSH4                      ; Jump if not an intermediate
    CJNE  R2,#' ',BCSH3              ; Jump if unimplemented
    CLR   PrmBadFlg                  ; Indicate no error
    MOV   PrmPvt,#0                  ;    not a private parameter,
    MOV   PrmCnt,#0                  ;    null parameter string, and
    CLR   PrmMaxFlg                  ;    not too many
    MOV   DisStt,#EXT_CSI_STT        ; Change state for extended CSI
    LJMP  DisCon                     ;    sequences and continue
```

```
BCSH3:                             ; Unimplemented intermediate characters
    MOV   DisStt,#UNIMP_CSI_STT         ; Set state for unimplemented
    LJMP  DisCon                        ;    CSI sequences and continue
BCSH4:                             ; Check for parameter character
    SUBB  A,#('@'-'0')                  ; Reduce by parameter range
    JNC   BCSH11                        ; Jump if not a parameter
    ADD   A,#('?'-'9')                  ; Check for special param char
    JC    BCSH6                         ; Jump if not a digit parameter
    MOV   PrmPvt,#0                     ; Indicate not private params
    ADD   A,#10                         ; Readjust decoded param digit
    MOV   PrmAcc,A                      ;    and start accumulator
    SETB  PrmBgnFlg                     ; Indicate start of param string
BCSH5:                             ; Peform parameter decoding
    MOV   DisStt,#PRM_CSI_STT           ; Change state to decode CSI
    LJMP  DisCon                        ;    parameters and continue
BCSH6:                             ; Special parameters
    CJNE  R2,#';',BCSH9                 ; Jump if not good separator
    CLR   PrmBadFlg                     ; Indicate no errors if good
BCSH7:                             ; Initial default parameter
    MOV   PrmPvt,#0                     ; Indicate not private params
    MOV   PrmCnt,#1                     ; One parameter so far and
    MOV   PrmBuf,#0                     ;    it is zero (default)
BCSH8:                             ; Set up for parameter accumulation
    MOV   PrmAcc,#0                     ; Clear accumulator
    CLR   PrmBgnFlg                     ; Indicate start of parameter
    CLR   PrmMaxFlg                     ;    string and not too many
    SJMP  BCSH5                         ; Continue with new state
BCSH9:                             ; Special parameters (not semi-colon)
    CJNE  R2,#':',BCSH10                ; Jump if not unused separator
    SETB  PrmBadFlg                     ; Indicate an error if found
    SJMP  BCSH7                         ; Treat as initial default
BCSH10:                            ; Special private parameters
    CLR   PrmBadFlg                     ; Indicate no error and
    MOV   PrmPvt,R2                     ;    save special parameter
    MOV   PrmCnt,#0                     ; Indicate empty param buffer
    SJMP  BCSH8                         ; Get ready to accumulate params
BCSH11:                            ; Check for final character
    MOV   R7,A                          ; Save index temporarily
    SUBB  A,#(DEL-'@')                  ; Check for invalid character
    JNC   BCSH1                         ; Jump if invalid sequence
    CJNE  R2,#'b',BCSH12                ; Jump if not REP sequence
```

```
   MOV   PrmRep,#1                       ; Set default parameter and do
   LCALL Repeat                          ;     special repeat (if possible)
   SJMP  BCSH13                          ; Finish sequence
BCSH12:                             ; Normal final character
   CLR   PrmBadFlg                       ; Indicate no error
   MOV   PrmPvt,#0                       ;     not a private parameter,
   MOV   PrmCnt,#0                       ;     null parameter string, and
   CLR   PrmMaxFlg                       ;     not too many
   MOV   A,R7                            ; Restore control routine index
   MOV   DPTR,#X3_64DirSeqTbl            ; Use CSI direct sequence table
   LCALL DoWrk                           ;     and do the control routine
BCSH13:                             ; Completed CSI sequence
   MOV   DisStt,#DIR_CHR_STT             ; Set state for single, direct
   LJMP  DisCon                          ;     characters and continue


;-----------------------------------------------------------------------
PrmCSISttHdl:

; Decodes the parameters in a control sequence until a non-parameter character
; is encountered.  If it is a final character then the corresponding control
; routine is executed using the X3.64 direct sequence table.  An intermediate
; changes state to handle extended control sequences, if it is a space, and to
; unimplemented control sequences for any other intermediate.  An invalid
; character ends the sequence and discards the entire control sequence.

   MOV   A,R2                            ; Get character and check
   SUBB  A,#' '                          ;     for a C0 control character
   JNC   PCSH2                           ; Jump if not a control character

PCSH1:                              ; Invalid control sequence
   CLR   A                               ; Clear
   MOV   CtlPtrHi,A                      ;     control routine address
   MOV   CtlPtrLo,A                      ;     (makes it unrepeatable)
   SJMP  PCSH12                          ; Finish sequence
PCSH2:                              ; Check for intermediate character
   SUBB  A,#('0'-' ')                    ; Reduce by intermediate range
   JNC   PCSH4                           ; Jump if not an intermediate
   CJNE  R2,#' ',PCSH3                   ; Jump if unimplemented
   MOV   DisStt,#EXT_CSI_STT             ; Set state for extended CSI seqs
   JNB   PrmBgnFlg,PCSH5                 ; Jump if not first parameter
   MOV   PrmCnt,#0                       ; initialize param cnt
   SJMP  PCSH5                           ; Go handle parameter
```

```
PCSH3:                              ; Unimplemented intermediate characters
   MOV   DisStt,#UNIMP_CSI_STT           ; Set state for unimplemented
   SJMP  PCSH5                           ;     CSI sequences and continue

PCSH4:                              ; Check for parameter character
   SUBB  A,#('@'-'0')                    ; Reduce by parameter range
   JNC   PCSH9                           ; Jump if not a parameter
   ADD   A,#('?'-'9')                    ; Check for special param char
   JNC   PCSH7                           ; Jump if a digit parameter
   CJNE  R2,#';',PCSH6                   ; Jump if not a valid separator
PCSH5:                              ; Parameter separator
   LCALL SavPrm                          ; Save latest parameter
   LJMP  DisCon                          ;     and continue
PCSH6:                              ; Invalid special parameter character
   SETB  PrmBadFlg                       ; Signal bad parameters
   SJMP  PCSH5                           ; Treat as a separator & continue
PCSH7:                              ; Parameter digit
   ADD   A,#10                           ; Readjust decoded param digit
   MOV   R7,A                            ;     and save it temporarily
   MOV   B,#10                           ; Multiply (by 10)
   MOV   A,PrmAcc                        ;     current parameter value
   MUL   AB                              ;     to account for another digit
   JB    OV,PCSH8                        ; Jump if param greater than 255
   ADD   A,R7                            ; Accumulate latest digit
   JC    PCSH8                           ; Jump if param greater than 255
   MOV   PrmAcc,A                        ; Save accumulated param value
   LJMP  DisCon                          ;     and continue
PCSH8:                              ; Parameter too large
   MOV   PrmAcc,#255                     ; Save largest possible value
   LJMP  DisCon                          ;     and continue
PCSH9:                              ; Check for final character
   MOV   R7,A                            ; Save index temporarily
   SUBB  A,#(DEL-'@')                    ; Check for invalid character
   JNC   PCSH1                           ; Jump if invalid sequence
   CJNE  R2,#'b',PCSH10                  ; Jump if not REP sequence
   MOV   PrmRep,PrmAcc
   LCALL Repeat                          ; Do special repeat (if possible)
   SJMP  PCSH12                          ; Finish sequence
PCSH10:                             ; Normal final character
   LCALL SavPrm                          ; Save latest parameter
   MOV   A,R7                            ; Restore control routine index
   MOV   DPTR,#X3_64DirSeqTbl            ; Use CSI direct sequence table
   LCALL DoWrk                           ;     and do the control routine
```

```
PCSH12:                                     ; Completed CSI sequence
   MOV    DisStt,#DIR_CHR_STT                     ; Set state for single, direct
   LJMP   DisCon                                  ;    characters and continue

;---------------------------------------------------------------------------------
ExtCSISttHdl:

; Processes the character immediately following the first space intermediate
; in a control sequence; no other intermediates are implemented.  It does a
; special check for the two acceptable final characters which are implemented
; and executes their control routines directly if found.  Any other valid final
; character or an invalid character ends the sequence with the entire sequence
; being discarded.  If an intermediate character is encountered then the state
; is changed to handle unimplemented control sequences.
;
; NOTE:  Further implementations could be accomplished with the addition of
;        other tables of control routine addresses.  When a final character
;        is found, the corresponding control routine would be executed using
;        the appropriate table.  Which table is appropriate would depend on
;        the sequence of intermediate characters, which could be interpreted
;        by changing to additional states, or using another state variable.

   CJNE   R2,#'@',ECSH1                           ; Jump if not SL final character
   LCALL  ScrollLeft                              ; Do scroll left control then
   SJMP   ECSH4                                   ;    continue with direct state
ECSH1:
   CJNE   R2,#'A',ECSH2                           ; Jump if not SR final character
   LCALL  ScrollRight                             ; Do scroll right control then
   SJMP   ECSH4                                   ;    continue with direct state
ECSH2:                                      ; Unimplemented or invalid character
   MOV    A,R2                                    ; Get character and check
   SUBB   A,#' '                                  ;    for a C0 control character
   JC     ECSH3                                   ; Jump if it is a control char
   SUBB   A,#('0'-' ')                            ; Reduce by intermediate range
   JNC    ECSH3                                   ; Jump if not an intermediate
   MOV    DisStt,#UNIMP_CSI_STT                   ; Change state for unimplemented
   LJMP   DisCon                                  ;    CSI sequences and continue
ECSH3:                                      ; Invalid CSI sequence
   CLR    A                                       ; Clear
   MOV    CtlPtrHi,A                              ;    control routine address
   MOV    CtlPtrLo,A                              ;    (makes it unrepeatable)
```

```
ECSH4:                                      ; Completed extended CSI sequence
   MOV    DisStt,#DIR_CHR_STT                     ; Set state for single, direct
   LJMP   DisCon                                  ;    characters and continue

;---------------------------------------------------------------------------------
UnImpCSISttHdl:

; Processes unimplemented CSI sequences with intermediate characters by passing
; over intermediates until either a final character or an invalid character is
; encountered.  It then changes the state back to handle direct characters.
;
; NOTE:  Further implementations could be accomplished with the addition of
;        other tables of control routine addresses.  When a final character
;        is found, the corresponding control routine would be executed using
;        the appropriate table.  Which table is appropriate would depend on
;        the sequence of intermediate characters, which could be interpreted
;        by changing to additional states, or using another state variable.

   MOV    A,R2                                    ; Get character and check
   SUBB   A,#' '                                  ;    for a C0 control character
   JC     UCSH1                                   ; Jump if it is a control char
   SUBB   A,#('0'-' ')                            ; Reduce by intermediate range
   JC     UCSH2                                   ; Jump if it is an intermediate

UCSH1:                                      ; Completed CSI sequence
   CLR    A                                       ; Clear
   MOV    CtlPtrHi,A                              ;    control routine address
   MOV    CtlPtrLo,A                              ;    (makes it unrepeatable)
   MOV    DisStt,#DIR_CHR_STT                     ; Set state for direct characters
UCSH2:
   LJMP   DisCon                                  ; Continue

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SavPrm:

; Saves the current contents of the parameter accumulator in the parameter
; buffer and increments the parameter count, provided the parameter buffer
; is not full.  If this is a first parameter then the parameter accumulator
; is saved as the special repeat parameter; otherwise, the special repeat
; parameter is checked and, if present (i.e. this is the second parameter),
; it is saved before the parameter accumulator and then cleared.  Finally,
; the parameter buffer is checked to see if it has become full.
; Bad:   A,R0

   JB    PrmMaxFlg,SP3                        ; Jump if param buffer is full
   JNB   PrmBgnFlg,SP1
   MOV   PrmCnt,#0
SP1:
   CLR   PrmBgnFlg
   MOV   A,PrmCnt                             ; Check count and
   CJNE  A,#PRM_CNT_MAX,SP3                   ;    jump if maximum not reached
   SETB  PrmMaxFlg                            ; Indicate full if max is reached
   SJMP  SP4                                  ;    and discard parameter
SP3:                                    ; Reset for more parameters
   MOV   A,#PrmBuf                            ; Point into parameter buffer
   ADD   A,PrmCnt                             ;    at location where next
   MOV   R0,A                                 ;    parameter is to be stored
   MOV   @R0,PrmAcc                           ; Store latest parameter
   INC   PrmCnt                               ; Account for latest parameter
SP4:                                    ; Get ready for next parameter
   MOV   PrmAcc,#0                            ; Clear parameter accumulator
   RET                                        ;    and exit

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DoWrk:

; Transfers control to the subroutine indicated by the index into the given
; address table.  The address is also saved for possible repetition.
;
; In:    A                              control routine index
;        DPTR                           base of control routine address table

   MOV   R0,CtlPtrHi                          ; Save previous control routine
   MOV   R1,CtlPtrLo                          ;    so ESC and CSI can restore
   RL    A                                    ; Turn index into offset into tbl
   MOV   R7,A                                 ;    and save it temporarily
   MOVC  A,@A+DPTR                            ; Get high byte of address and
   MOV   CtlPtrHi,A                           ;    save it
   MOV   A,R7                                 ; Restore offset and
   INC   A                                    ;    adjust for next location
   MOVC  A,@A+DPTR                            ; Get low byte of address and
   MOV   CtlPtrLo,A                           ;    save it
   MOV   DPH,CtlPtrHi                         ; Set indirect pointer's high and
   MOV   DPL,A                                ;    low bytes
; NOTE:  This routine falls through to the next
```

```
DoIndRtn:

; Provides an entrypoint for indirect subroutine calls.
;
; In:    DPTR                           address of indirect subroutine
;        (and whatever the indirect subroutine needs)
; Out:   (whatever the indirect subroutine returns)
; Bad:   A    (and whatever the indirect subroutine affects)
;
; NOTE:  This may not be used to call a subroutine which requires either
;        the accumulator (A) or the data pointer (DPTR) as input.

   CLR   A                                    ; Clear indirect offset and
   JMP   @A+DPTR                              ;    transfer control

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
UnImpCtl:

; Catch all for unimplemented controls.

   CLR   A                                    ; Clear
   MOV   CtlPtrHi,A                           ;    control routine address
   MOV   CtlPtrLo,A                           ;    (makes it unrepeatable)
   RET                                        ;    and exit

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Escape:

; Control routine for ESC control character--changes state to handle escape
; sequences.

   MOV   CtlPtrHi,R0                          ; Restore previous
   MOV   CtlPtrLo,R1                          ;    control routine pointer
   MOV   DisStt,#BGN_ESC_STT                  ; Set state for escape sequences
   RET                                        ;    and exit

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CtlSeqIntro:

; Control routine for CSI control character or escape sequence--changes state
; to handle control sequences.
```

```
   MOV   CtlPtrHi,R0                    ; Restore previous
   MOV   CtlPtrLo,R1                    ;    control routine pointer
   MOV   DisStt,#BGN_CSI_STT            ; Set state for CSI sequences
   DEC   SP                             ; Remove return address
   DEC   SP                             ;    from stack and
   LJMP  DisCon                         ;    continue

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
PutMap1:
;
   JNB   FntMapFlg,PutChr
   MOV   A,PrmBuf
   ADD   A,#040H
   SJMP  PC0

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
PutMap0:
;
   JNB   FntMapFlg,PutChr
   MOV   A,PrmBuf
   CLR   C
   SUBB  A,#040H
   SJMP  PC0

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
PutChr:

; Writes a character generator cell address and the current attribute to the
; appropriate locations in display memory indicated by the active position.
; It advances the active position provided it is not at the rightmost column.
; The cursor position is also updated using the cursor zone information.
;
; In:    PrmBuf                   cell address (i.e. character code)

   MOV   A,PrmBuf                       ; Get the unmapped character code

PC0:                              ; Common character placement entrypoint
   MOV   R2,A                           ; Save cell address
   MOV   P2,CurRow                      ; Page of active row and
   MOV   A,ChrOff                       ;    offset to base of characters
   ADD   A,ActCol                       ; Add active column to determine
   MOV   R0,A                           ;    offset for location to write
```

```
   MOV   A,R2                           ; Get cell address and
   MOVX  @R0,A                          ;    write it to display memory
   MOV   A,AtrOff                       ; Offset to base of attributes
   ADD   A,ActCol                       ; Add active column
   ADD   A,ActCol                       ;    twice and
   INC   A                              ;    adjust for attribute byte to
   MOV   R0,A                           ;    get offset for loc to write
   JNB   WndActFlg,PC1                  ; Jump if window disp not active
   SETB  P2.4                           ; Adjust page for window display
   SJMP  PC2                            ;    and go on
PC1:
   SETB  P2.5                           ; Adjust page for bgd/msg display
PC2:                              ; Write current attribute
   MOV   A,CurAtr                       ; Get current attribute and
   MOVX  @R0,A                          ;    write it
   MOV   P2,#MsgActCnt.SR.PAGE          ; Page containing active counts
   JNB   MsgActFlg,PC3                  ; Jump if message disp not active
   MOV   A,#MsgActCnt.AN.OFST           ; Offset of message active count
   SJMP  PC5                            ;    and go use it
PC3:
   JNB   WndActFlg,PC4                  ; Jump if window disp not active
   MOV   A,CurRow                       ; Page of current row converted
   ANL   A,#00FH                        ;    to physical number and
   ADD   A,#WndActCntBuf.AN.OFST        ;    added to base of window
   SJMP  PC5                            ;    active count buffer
PC4:
   MOV   A,CurRow                       ; Page of current row converted
   ANL   A,#01FH                        ;    to physical number and added
   ADD   A,#BgdActCntBuf.AN.OFST        ;    to base of bgd act cnt buf
PC5:                              ; Update active count
   MOV   R0,A                           ; Offset of this row's active cnt
   MOVX  A,@R0                          ; Get current active count,
   INC   ActCol                         ;    new active count and
   SUBB  A,ActCol                       ;    compare them
   MOV   A,ActCol                       ; Get new active column for later
   JNC   PC6                            ; Jump if old active cnt is OK
   MOVX  @R0,A                          ; Write new active cnt if greater
PC6:                              ; Check for end of row (rightmost col)
   JNB   WndActFlg,PC7                  ; Jump if window disp not active
   CJNE  A,#40,PC8                      ; Jump if not at right of window
   DEC   ActCol                         ; Restore active column if at end
   RET                                  ;    and exit
```

```
PC7:
   CJNE  A,#128,PC8                         ; Jump if not at right of bgd/msg
   DEC   ActCol                             ; Restore active column if at end
   RET                                      ;    and exit
PC8:                                  ; Advance cursor location
   DJNZ  CsrZonCnt,PC9                      ; Jump if still in same zone
   LCALL PlcCsr                             ; Place cursor in new zone
   RET                                      ;    and exit
PC9:                                  ; Speedy update of cursor location
   JNB   CsrZonFlg,PC10                     ; Jump if cursor is invisible
   MOV   P2,#BgdMDB0.SR.PAGE                ; Page of MDBs and
   MOV   R0,#BgdMDB0.AN.OFST+MDB_Cux        ;    offset to cursor location
   MOVX  A,@R0                              ; Current location (both MDBs)
   INC   A                                  ;    advanced rightward and
   MOVX  @R0,A                              ;    put back then
   MOV   R0,#BgdMDB1.AN.OFST+MDB_Cux        ;    other MDB
   MOVX  @R0,A                              ;    gets same location
PC10:
   RET                                      ; Exit

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Repeat:

; Repeats the previous control routine if it is repeatable.  The parameter
; decoding part of the state machine is careful to preserve the previous
; parameter buffer and provides a special repeat parameter for this control
; routine, which is checked for and executed directly.  This is necessary to
; prevent this control's sequence from interfering with the previous control's
; parameters.  If the special repeat parameter is zero then the previous
; information has been lost and this sequence is ignored.

   JNB   PrmBgnFlg,Rp2
   MOV   A,CtlPtrHi                         ; Check previous
   ORL   A,CtlPtrLo                         ;    control routine address
   JZ    Rp3                                ; Jump if not repeatable
   MOV   A,PrmRep
   JNZ   Rp1
   MOV   PrmRep,#1
Rp1:                                  ; For each repetition
   MOV   DPH,CtlPtrHi                       ; Get previous control routine
   MOV   DPL,CtlPtrLo                       ;    address into indirect ptr
   LCALL DoIndRtn                           ; Execute the control routine
   DJNZ  PrmRep,Rp1                         ; Loop specified number of times
```

```
Rp2:
   CLR   A                                  ; Clear
   MOV   CtlPtrHi,A                         ;    control routine address
   MOV   CtlPtrLo,A                         ;    (may only be REP'd once)
Rp3:
   RET                                      ; Exit

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  End of C_Switch
```

```
"8051"
   TITLE "     CALEB 0.00     Control Tables"
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; C_Tables                                CALEB 0.00
;
;       Copyright 1985 Advanced Micro Devices, Inc.
;
;
; This file contains the address tables used by the state machine to dispatch
; control to the various control routines.

   NAME  "Control Tables"
   PROG

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   GLB   LoDirChrTbl                       ; First 128 entries for direct state
   GLB   HiDirChrTbl                       ; Second 128 entries for direct state
   GLB   DirEscSeqTbl                      ; Non-intermediate escape sequences
   GLB   X3_64DirSeqTbl                    ; Non-intermediate control sequences

;-----------------------------------------------------------------------------
   EXT   UnImpCtl,Escape,CtlSeqIntro,PutMap0,PutMap1,PutChr     ; in C_Switch
   EXT   Backspace,CarriageReturn,NewLine                       ; in C_Work
   EXT   ResetInitState                                         ; in C_Work
   EXT   CursorBackward,CursorDown,CursorForward                ; in C_Work
   EXT   CursorPosition,CursorUp,DeleteLine,EraseInDisplay      ; in C_Work
   EXT   EraseInLine,InsertLine,ResetMode,ScrollDown            ; in C_Work
   EXT   SelGrfRendition,SetMode,ScrollUp                       ; in C_Work
   EXT   CharBlinkRate,LoadFontCell,SelActiveDisp               ; in C_Work
   EXT   SelCursorAppear,SmoothScrlRate,SelWindowVis            ; in C_Work
   EXT   SelMessageVis                                          ; in C_Work

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

LoDirChrTbl:                               ; First 128 entries for direct state
```

```
   DW    UnImpCtl         ; 00H  NUL    Null
   DW    UnImpCtl         ; 01H  SOH    Start of Heading
   DW    UnImpCtl         ; 02H  STX    Start of Text
   DW    UnImpCtl         ; 03H  ETX    End of Text
   DW    UnImpCtl         ; 04H  EOT    End of Transmission
   DW    UnImpCtl         ; 05H  ENQ    Enquiry
   DW    UnImpCtl         ; 06H  ACK    Acknowledge
   DW    UnImpCtl         ; 07H  BEL    Bell
   DW    Backspace        ; 08H  BS     Backspace
   DW    UnImpCtl         ; 09H  HT     Horizontal Tabulation
   DW    NewLine          ; 0AH  LF/NL  Line Feed (New Line)
   DW    UnImpCtl         ; 0BH  VT     Vertical Tabulation
   DW    UnImpCtl         ; 0CH  FF     Form Feed
   DW    CarriageReturn   ; 0DH  CR     Carriage Return
   DW    UnImpCtl         ; 0EH  SO     Shift Out
   DW    UnImpCtl         ; 0FH  SI     Shift In
   DW    UnImpCtl         ; 10H  DLE    Data Link Escape
   DW    UnImpCtl         ; 11H  DC1    Device Control 1
   DW    UnImpCtl         ; 12H  DC2    Device Control 2
   DW    UnImpCtl         ; 13H  DC3    Device Control 3
   DW    UnImpCtl         ; 14H  DC4    Device Control 4
   DW    UnImpCtl         ; 15H  NAK    Negative Acknowledge
   DW    UnImpCtl         ; 16H  SYN    Synchronous Idle
   DW    UnImpCtl         ; 17H  ETB    End of Transmission Block
   DW    UnImpCtl         ; 18H  CAN    Cancel
   DW    UnImpCtl         ; 19H  EM     End of Medium
   DW    UnImpCtl         ; 1AH  SUB    Substitute
   DW    Escape           ; 1BH  ESC    Escape
   DW    UnImpCtl         ; 1CH  FS     File Separator
   DW    UnImpCtl         ; 1DH  GS     Group Separator
   DW    UnImpCtl         ; 1EH  RS     Record Separator
   DW    UnImpCtl         ; 1FH  US     Unit Separtor
   DW    PutChr           ; 20H         Space
   DW    PutChr           ; 21H  !      Start of G0 Characters
   DW    PutChr           ; 22H  "
   DW    PutChr           ; 23H  #
   DW    PutChr           ; 24H  $
   DW    PutChr           ; 25H  %
   DW    PutChr           ; 26H  &
   DW    PutChr           ; 27H  '
   DW    PutChr           ; 28H  (
   DW    PutChr           ; 29H  )
```

```
DW      PutChr          ; 2AH  *
DW      PutChr          ; 2BH  +
DW      PutChr          ; 2CH  ,
DW      PutChr          ; 2DH  -
DW      PutChr          ; 2EH  .
DW      PutChr          ; 2FH  /
DW      PutChr          ; 30H  0
DW      PutChr          ; 31H  1
DW      PutChr          ; 32H  2
DW      PutChr          ; 33H  3
DW      PutChr          ; 34H  4
DW      PutChr          ; 35H  5
DW      PutChr          ; 36H  6
DW      PutChr          ; 37H  7
DW      PutChr          ; 38H  8
DW      PutChr          ; 39H  9
DW      PutChr          ; 3AH  :
DW      PutChr          ; 3BH  ;
DW      PutChr          ; 3CH  <
DW      PutChr          ; 3DH  =
DW      PutChr          ; 3EH  >
DW      PutMap1         ; 3FH  ?
DW      PutMap0         ; 40H  @
DW      PutMap0         ; 41H  A
DW      PutMap0         ; 42H  B
DW      PutMap0         ; 43H  C
DW      PutMap0         ; 44H  D
DW      PutMap0         ; 45H  E
DW      PutMap0         ; 46H  F
DW      PutMap0         ; 47H  G
DW      PutMap0         ; 48H  H
DW      PutMap0         ; 49H  I
DW      PutMap0         ; 4AH  J
DW      PutMap0         ; 4BH  K
DW      PutMap0         ; 4CH  L
DW      PutMap0         ; 4DH  M
DW      PutMap0         ; 4EH  N
DW      PutMap0         ; 4FH  O
DW      PutMap0         ; 50H  P
DW      PutMap0         ; 51H  Q
DW      PutMap0         ; 52H  R
DW      PutMap0         ; 53H  S
DW      PutMap0         ; 54H  T
```

```
DW      PutMap0         ; 55H  U
DW      PutMap0         ; 56H  V
DW      PutMap0         ; 57H  W
DW      PutMap0         ; 58H  X
DW      PutMap0         ; 59H  Y
DW      PutMap0         ; 5AH  Z
DW      PutMap0         ; 5BH  [
DW      PutMap0         ; 5CH  \
DW      PutMap0         ; 5DH  ]
DW      PutMap0         ; 5EH  ^
DW      PutMap0         ; 5FH  _
DW      PutChr          ; 60H  '
DW      PutChr          ; 61H  a
DW      PutChr          ; 62H  b
DW      PutChr          ; 63H  c
DW      PutChr          ; 64H  d
DW      PutChr          ; 65H  e
DW      PutChr          ; 66H  f
DW      PutChr          ; 67H  g
DW      PutChr          ; 68H  h
DW      PutChr          ; 69H  i
DW      PutChr          ; 6AH  j
DW      PutChr          ; 6BH  k
DW      PutChr          ; 6CH  l
DW      PutChr          ; 6DH  m
DW      PutChr          ; 6EH  n
DW      PutChr          ; 6FH  o
DW      PutChr          ; 70H  p
DW      PutChr          ; 71H  q
DW      PutChr          ; 72H  r
DW      PutChr          ; 73H  s
DW      PutChr          ; 74H  t
DW      PutChr          ; 75H  u
DW      PutChr          ; 76H  v
DW      PutChr          ; 77H  w
DW      PutChr          ; 78H  x
DW      PutChr          ; 79H  y
DW      PutChr          ; 7AH  z
DW      PutChr          ; 7BH  {
DW      PutChr          ; 7CH  |
DW      PutChr          ; 7DH  }
DW      PutChr          ; 7EH  ~    End of G0 Characters
DW      UnImpCtl        ; 7FH  DEL  Delete
```

```
;---------------------------------------------------------------------------
  SKIP
;---------------------------------------------------------------------------
HiDirChrTbl:                              ; Second 128 entries for direct state

  DW    UnImpCtl          ; 80H         fut. std.
  DW    UnImpCtl          ; 81H         fut. std.
  DW    UnImpCtl          ; 82H         fut. std.
  DW    UnImpCtl          ; 83H         fut. std.
  DW    UnImpCtl          ; 84H  IND    Index
  DW    UnImpCtl          ; 85H  NEL    Next Line
  DW    UnImpCtl          ; 86H  SSA    Start of Select Area
  DW    UnImpCtl          ; 87H  ESA    End of Selected Area
  DW    UnImpCtl          ; 88H  HTS    Horizontal Tabulation Set
  DW    UnImpCtl          ; 89H  HTJ    Horizontal Tab with Justify
  DW    UnImpCtl          ; 8AH  VTS    Vertical Tabulation Set
  DW    UnImpCtl          ; 8BH  PLD    Partial Line Down
  DW    UnImpCtl          ; 8CH  PLU    Partial Line Up
  DW    UnImpCtl          ; 8DH  RI     Reverse Index
  DW    UnImpCtl          ; 8EH  SS2    Single Shift Two
  DW    UnImpCtl          ; 8FH  SS3    Single Shift Three
  DW    UnImpCtl          ; 90H  DCS    Device Control String
  DW    UnImpCtl          ; 91H  PU1    Private Use One
  DW    UnImpCtl          ; 92H  PU2    Private Use Two
  DW    UnImpCtl          ; 93H  STS    Set Transmit State
  DW    UnImpCtl          ; 94H  CCH    Cancel Character
  DW    UnImpCtl          ; 95H  MW     Message Waiting
  DW    UnImpCtl          ; 96H  SPA    Start of Protected Area
  DW    UnImpCtl          ; 97H  EPA    End of Protected Area
  DW    UnImpCtl          ; 98H         fut. std.
  DW    UnImpCtl          ; 99H         fut. std.
  DW    UnImpCtl          ; 9AH         fut. std.
  DW    CtlSeqIntro       ; 9BH  CSI    Control Sequence Introducer
  DW    UnImpCtl          ; 9CH  ST     String Terminator
  DW    UnImpCtl          ; 9DH  OSC    Operating System Command
  DW    UnImpCtl          ; 9EH  PM     Privacy Message
  DW    UnImpCtl          ; 9FH  APC    Application Program Command
  DW    PutChr            ; A0H
  DW    PutChr            ; A1H         Start of G1 Characters
  DW    PutChr            ; A2H
  DW    PutChr            ; A3H
  DW    PutChr            ; A4H
  DW    PutChr            ; A5H
```

```
  DW    PutChr            ; A6H
  DW    PutChr            ; A7H
  DW    PutChr            ; A8H
  DW    PutChr            ; A9H
  DW    PutChr            ; AAH
  DW    PutChr            ; ABH
  DW    PutChr            ; ACH
  DW    PutChr            ; ADH
  DW    PutChr            ; AEH
  DW    PutChr            ; AFH
  DW    PutChr            ; B0H
  DW    PutChr            ; B1H
  DW    PutChr            ; B2H
  DW    PutChr            ; B3H
  DW    PutChr            ; B4H
  DW    PutChr            ; B5H
  DW    PutChr            ; B6H
  DW    PutChr            ; B7H
  DW    PutChr            ; B8H
  DW    PutChr            ; B9H
  DW    PutChr            ; BAH
  DW    PutChr            ; BBH
  DW    PutChr            ; BCH
  DW    PutChr            ; BDH
  DW    PutChr            ; BEH
  DW    PutMap1           ; BFH
  DW    PutMap0           ; C0H
  DW    PutMap0           ; C1H
  DW    PutMap0           ; C2H
  DW    PutMap0           ; C3H
  DW    PutMap0           ; C4H
  DW    PutMap0           ; C5H
  DW    PutMap0           ; C6H
  DW    PutMap0           ; C7H
  DW    PutMap0           ; C8H
  DW    PutMap0           ; C9H
  DW    PutMap0           ; CAH
  DW    PutMap0           ; CBH
  DW    PutMap0           ; CCH
  DW    PutMap0           ; CDH
  DW    PutMap0           ; CEH
  DW    PutMap0           ; CFH
```

```
        DW      PutMap0             ; D0H
        DW      PutMap0             ; D1H
        DW      PutMap0             ; D2H
        DW      PutMap0             ; D3H
        DW      PutMap0             ; D4H
        DW      PutMap0             ; D5H
        DW      PutMap0             ; D6H
        DW      PutMap0             ; D7H
        DW      PutMap0             ; D8H
        DW      PutMap0             ; D9H
        DW      PutMap0             ; DAH
        DW      PutMap0             ; DBH
        DW      PutMap0             ; DCH
        DW      PutMap0             ; DDH
        DW      PutMap0             ; DEH
        DW      PutMap0             ; DFH
        DW      PutChr              ; E0H
        DW      PutChr              ; E1H
        DW      PutChr              ; E2H
        DW      PutChr              ; E3H
        DW      PutChr              ; E4H
        DW      PutChr              ; E5H
        DW      PutChr              ; E6H
        DW      PutChr              ; E7H
        DW      PutChr              ; E8H
        DW      PutChr              ; E9H
        DW      PutChr              ; EAH
        DW      PutChr              ; EBH
        DW      PutChr              ; ECH
        DW      PutChr              ; EDH
        DW      PutChr              ; EEH
        DW      PutChr              ; EFH
        DW      PutChr              ; F0H
        DW      PutChr              ; F1H
        DW      PutChr              ; F2H
        DW      PutChr              ; F3H
        DW      PutChr              ; F4H
        DW      PutChr              ; F5H
        DW      PutChr              ; F6H
        DW      PutChr              ; F7H
        DW      PutChr              ; F8H
        DW      PutChr              ; F9H
```

```
        DW      PutChr              ; FAH
        DW      PutChr              ; FBH
        DW      PutChr              ; FCH
        DW      PutChr              ; FDH
        DW      PutChr              ; FEH          End of G1 Characters
        DW      UnImpCtl            ; FFH
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        SKIP
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DirEscSeqTbl:                                  ; Non-intermediate escape sequences
        DW      UnImpCtl            ; 30H          priv. use
        DW      UnImpCtl            ; 31H          priv. use
        DW      UnImpCtl            ; 32H          priv. use
        DW      UnImpCtl            ; 33H          priv. use
        DW      UnImpCtl            ; 34H          priv. use
        DW      UnImpCtl            ; 35H          priv. use
        DW      UnImpCtl            ; 36H          priv. use
        DW      UnImpCtl            ; 37H          priv. use
        DW      UnImpCtl            ; 38H          priv. use
        DW      UnImpCtl            ; 39H          priv. use
        DW      UnImpCtl            ; 3AH          priv. use
        DW      UnImpCtl            ; 3BH          priv. use
        DW      UnImpCtl            ; 3CH          priv. use
        DW      UnImpCtl            ; 3DH          priv. use
        DW      UnImpCtl            ; 3EH          priv. use
        DW      UnImpCtl            ; 3FH          priv. use
        DW      UnImpCtl            ; 40H          fut. std.
        DW      UnImpCtl            ; 41H          fut. std.
        DW      UnImpCtl            ; 42H          fut. std.
        DW      UnImpCtl            ; 43H          fut. std.
        DW      UnImpCtl            ; 44H   IND    Index
        DW      UnImpCtl            ; 45H   NEL    Next Line
        DW      UnImpCtl            ; 46H   SSA    Start of Select Area
        DW      UnImpCtl            ; 47H   ESA    End of Selected Area
        DW      UnImpCtl            ; 48H   HTS    Horizontal Tabulation Set
        DW      UnImpCtl            ; 49H   HTJ    Horizontal Tab with Justify
        DW      UnImpCtl            ; 4AH   VTS    Vertical Tabulation Set
        DW      UnImpCtl            ; 4BH   PLD    Partial Line Down
        DW      UnImpCtl            ; 4CH   PLU    Partial Line Up
        DW      UnImpCtl            ; 4DH   RI     Reverse Index
        DW      UnImpCtl            ; 4EH   SS2    Single Shift Two
        DW      UnImpCtl            ; 4FH   SS3    Single Shift Three
        DW      UnImpCtl            ; 50H  DCS   Device Control String
        DW      UnImpCtl            ; 51H  PU1   Private Use One
        DW      UnImpCtl            ; 52H  PU2   Private Use Two
        DW      UnImpCtl            ; 53H  STS   Set Transmit State
        DW      UnImpCtl            ; 54H  CCH   Cancel Character
        DW      UnImpCtl            ; 55H  MW    Message Waiting
        DW      UnImpCtl            ; 56H  SPA   Start of Protected Area
        DW      UnImpCtl            ; 57H  EPA   End of Protected Area
        DW      UnImpCtl            ; 58H        fut. std.
        DW      UnImpCtl            ; 59H        fut. std.
        DW      UnImpCtl            ; 5AH        fut. std.
        DW      CtlSeqIntro         ; 5BH  CSI   Control Sequence Introducer
        DW      UnImpCtl            ; 5CH  ST    String Terminator
        DW      UnImpCtl            ; 5DH  OSC   Operating System Command
        DW      UnImpCtl            ; 5EH  PM    Privacy Message
        DW      UnImpCtl            ; 5FH  APC   Application Program Command
        DW      UnImpCtl            ; 60H  DMI   Disable Manual Input
        DW      UnImpCtl            ; 61H  INT   Interrupt
        DW      UnImpCtl            ; 62H  EMI   Enable Manual Input
        DW      ResetInitState      ; 63H  RIS   Reset to Initial State
        DW      UnImpCtl            ; 64H        fut. std.
        DW      UnImpCtl            ; 65H        fut. std.
        DW      UnImpCtl            ; 66H        fut. std.
        DW      UnImpCtl            ; 67H        fut. std.
        DW      UnImpCtl            ; 68H        fut. std.
        DW      UnImpCtl            ; 69H        fut. std.
        DW      UnImpCtl            ; 6AH        fut. std.
        DW      UnImpCtl            ; 6BH        fut. std.
        DW      UnImpCtl            ; 6CH        fut. std.
        DW      UnImpCtl            ; 6DH        fut. std.
        DW      UnImpCtl            ; 6EH        fut. std.
        DW      UnImpCtl            ; 6FH        fut. std.
        DW      UnImpCtl            ; 70H        fut. std.
        DW      UnImpCtl            ; 71H        fut. std.
        DW      UnImpCtl            ; 72H        fut. std.
        DW      UnImpCtl            ; 73H        fut. std.
        DW      UnImpCtl            ; 74H        fut. std.
        DW      UnImpCtl            ; 75H        fut. std.
        DW      UnImpCtl            ; 76H        fut. std.
        DW      UnImpCtl            ; 77H        fut. std.
        DW      UnImpCtl            ; 78H        fut. std.
        DW      UnImpCtl            ; 79H        fut. std.
        DW      UnImpCtl            ; 7AH        fut. std.
```

```
        DW      UnImpCtl            ; 7BH        fut. std.
        DW      UnImpCtl            ; 7CH        fut. std.
        DW      UnImpCtl            ; 7DH        fut. std.
        DW      UnImpCtl            ; 7EH        fut. std.

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        SKIP
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

X3_64DirSeqTbl:                              ; Non-intermediate control sequences

        DW      UnImpCtl            ; 40H  ICH   Insert Character
        DW      CursorUp            ; 41H  CUU   Cursor Up
        DW      CursorDown          ; 42H  CUD   Cursor Down
        DW      CursorForward       ; 43H  CUF   Cursor Forward
        DW      CursorBackward      ; 44H  CUB   Cursor Backward
        DW      UnImpCtl            ; 45H  CNL   Cursor Next Line
        DW      UnImpCtl            ; 46H  CPL   Cursor Preceding Line
        DW      UnImpCtl            ; 47H  CHA   Cursor Horizontal Absolute
        DW      CursorPosition      ; 48H  CUP   Cursor Position
        DW      UnImpCtl            ; 49H  CHT   Cursor Horizontal Tabulation
        DW      EraseInDisplay      ; 4AH  ED    Erase in Display
        DW      EraseInLine         ; 4BH  EL    Erase in Line
        DW      InsertLine          ; 4CH  IL    Insert Line
        DW      DeleteLine          ; 4DH  DL    Delete Line
        DW      UnImpCtl            ; 4EH  EF    Erase in Field
        DW      UnImpCtl            ; 4FH  EA    Erase in Area
        DW      UnImpCtl            ; 50H  DCH   Delete Character
        DW      UnImpCtl            ; 51H  SEM   Select Editing Extend Mode
        DW      UnImpCtl            ; 52H  CPR   Cursor Position Report
        DW      ScrollUp            ; 53H  SU    Scroll Up
        DW      ScrollDown          ; 54H  SD    Scroll Down
        DW      UnImpCtl            ; 55H  NP    Next Page
        DW      UnImpCtl            ; 56H  PP    Preceding Page
        DW      UnImpCtl            ; 57H  CTC   Cursor Tabulation Control
        DW      UnImpCtl            ; 58H  ECH   Erase Character
        DW      UnImpCtl            ; 59H  CVT   Cursor Vertical Tabulation
        DW      UnImpCtl            ; 5AH  CBT   Cursor Backward Tabulation
        DW      UnImpCtl            ; 5BH        fut. std.
        DW      UnImpCtl            ; 5CH        fut. std.
        DW      UnImpCtl            ; 5DH        fut. std.
        DW      UnImpCtl            ; 5EH        fut. std.
        DW      UnImpCtl            ; 5FH        fut. std.
```

```
    DW    UnImpCtl          ; 60H  HPA   Horizontal Position Absolute
    DW    UnImpCtl          ; 61H  HPR   Horizontal Position Relative
    DW    UnImpCtl          ; 62H  REP   Repeat
    DW    UnImpCtl          ; 63H  DA    Device Attributes
    DW    UnImpCtl          ; 64H  VPA   Vertical Position Absolute
    DW    UnImpCtl          ; 65H  VPR   Vertical Position Relative
    DW    UnImpCtl          ; 66H  HVP   Horizontal and Vertical Position
    DW    UnImpCtl          ; 67H  TBC   Tabulation Clear
    DW    SetMode           ; 68H  SM    Set Mode
    DW    UnImpCtl          ; 69H  MC    Media Copy
    DW    UnImpCtl          ; 6AH        fut. std.
    DW    UnImpCtl          ; 6BH        fut. std.
    DW    ResetMode         ; 6CH  RM    Reset Mode
    DW    SelGrfRendition   ; 6DH  SGR   Select Graphic Rendition
    DW    UnImpCtl          ; 6EH  DSR   Device Status Report
    DW    UnImpCtl          ; 6FH  DAQ   Define Area Qualification
    DW    SelActiveDisp     ; 70H  AmSAD Select Active Display
    DW    SelMessageVis     ; 71H  AmSMV Select Message Visibility
    DW    SelWindowVis      ; 72H  AmSWV Select Window Visibility
    DW    UnImpCtl          ; 73H        priv. use
    DW    SmoothScrlRate    ; 74H  AmSSR Smooth Scroll Rate
    DW    CharBlinkRate     ; 75H  AmCBR Character Blink Rate
    DW    SelCursorAppear   ; 76H  AmSCA Select Cursor Appearance
    DW    UnImpCtl          ; 77H        priv. use
    DW    UnImpCtl          ; 78H        priv. use
    DW    UnImpCtl          ; 79H        priv. use
    DW    UnImpCtl          ; 7AH        priv. use
    DW    UnImpCtl          ; 7BH        priv. use
    DW    UnImpCtl          ; 7CH        priv. use
    DW    UnImpCtl          ; 7DH        priv. use
    DW    LoadFontCell      ; 7EH  AmLFC Load Font Cell

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; end of C_Tables
```

```
"8051"
   TITLE "     CALEB 0.00     Control Routines"
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; C_Work                                CALEB 0.00
;
;       Copyright 1985 Advanced Micro Devices, Inc.
;
;
; This file contains all of the control routines supported by CALEB.  Both
; ANSI standard and AMD private controls are included.

   NAME  "Control Routines"
   PROG

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; ANSI Standard Control Routines

   GLB   Backspace                      ; Backspace
   GLB   CarriageReturn                 ; Carriage Return
   GLB   NewLine                        ; New Line
   GLB   ResetInitState                 ; Reset to Initial State
   GLB   CursorBackward                 ; Cursor Backward
   GLB   CursorDown                     ; Cursor Down
   GLB   CursorForward                  ; Cursor Forward
   GLB   CursorPosition                 ; Cursor Position
   GLB   CursorUp                       ; Cursor Up
   GLB   DeleteLine                     ; Delete Line
   GLB   EraseInDisplay                 ; Erase in Display
   GLB   EraseInLine                    ; Erase in Line
   GLB   InsertLine                     ; Insert Line
   GLB   ResetMode                      ; Reset Mode
   GLB   ScrollDown                     ; Scroll Down
   GLB   SelGrfRendition                ; Select Graphic Rendition
   GLB   ScrollLeft                     ; Scroll Left
   GLB   SetMode                        ; Set Mode
   GLB   ScrollRight                    ; Scroll Right
   GLB   ScrollUp                       ; Scroll Up

; AMD Private Control Routines

   GLB   CharBlinkRate                  ; Character Blink Rate
   GLB   LoadFontCell                   ; Load Font Cell
```

```
   GLB   SelActiveDisp                  ; Select Active Display
   GLB   SelCursorAppear                ; Select Cursor Appearance
   GLB   SmoothScrlRate                 ; Smooth Scroll Rate
   GLB   SelWindowVis                   ; Select Window Visibility
   GLB   SelMessageVis                  ; Select Message Visibility

;--------------------------------------------------------------------------------
   EXT   Reset                                                  ; in C_Init
   EXT   EraActEnd,EraBgnAct,ChgBlnkSpd,SwpVar,ChgCsrSiz,ChgCsrTyp
   EXT   HidCsr,NewCsr,PlcCsr,WrAm8052Reg,BldTrmRcb
   EXT   FrcEraRow,EraRow,DelRow_MovDn,InsRow_MovDn,DelRow_MovUp
   EXT   InsRow_MovUp,ScrlUpNewRow,ScrlUpDsp,ScrlDnDsp,ScrlRtDsp,ScrlLtDsp
   EXT   DlyTilEndFrm,HidWnd,ShwWnd
   EXT   ShwCsr,SetCelWid,WrFntCel,SetWndPos

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
   INCLUDE C_MemMap

   SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Backspace:
;................................................................................
; Moves the active position left one position on the screen. Backspace does
; not support auto wrap therefore the active position can be moved left only
; until it reaches the first memory location of the active row.
;
;  inp   none
;  out   ActCol                        updated
;  bad   A
;................................................................................

   MOV   A,ActCol                       ; Get the current active col
   JZ    BS1                            ;    decrement its value and
   DEC   ActCol                         ;    test for 0, if 0 do nothing
BS1:                                    ;    else decrement ActCol
   LCALL PlcCsr
   RET
```

```
CarriageReturn:
;............................................................................
;  Forces a movement of the active position to the first location on the
;  current row.
;
;  Inp   ActCol
;  Out   ActCol                         loaded to 0
;  bad   none
;............................................................................

   MOV   ActCol,#00H
   LCALL PlcCsr
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
NewLine:
;............................................................................
;  Moves the active position to move down one row. If the current row is the
;  at the bottom of the screen then a scroll of the screen is done.
;
;  Inp   CurRow
;        BtmRow
;  Out   ActRow                         incremented to next row page
;        BtmRow                         changed if a scroll has occurred
;  bad   A,R0,P2
;............................................................................

   JB    MsgActFlg,NL4                       ; Newline has no action in msg
   MOV   ActCol,#0                           ; In all cases ActCol goes to 0
   MOV   A,CurRow                            ; If we are not at the end of the
   CJNE  A,EndRow,NL1                        ;    linked list just move to
                                             ;    next row
   MOV   CurRow,ExtRow                       ; else make the extra row our
   LCALL ScrlUpNewRow                        ;    current row and scroll
   RET
NL1:
   INC   ActRow                              ; Inc ActRow and test which row
   CJNE  A,BtmRow,NL2                        ;    next row pointer to use
   MOV   A,RemRow                            ; if bottom of screen use RemRow
   SJMP  NL3
NL2:
   MOV   P2,CurRow                           ; else use next row in list
   MOV   A,#RCB_RowPag
```

```
   ADD   A,RcbOff
   MOV   R0,A
   MOVX  A,@R0                                   ; Acc now has next row page ptr
NL3:
   MOV   CurRow,A                                ; Update CurRow and cursor pos
   LCALL PlcCsr
NL4:
   RET                                           ; and leave

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ResetInitState:
;............................................................................
; Blanks the Am8052 (Mode Register 1--VB=1) without disabling it and waits
; until vertical retrace time.  Then it jumps to the power-up procedure.
; inp    none
; out    none
; bad    A,R0,R1,R2,R3,
;............................................................................

   LCALL DlyTilEndFrm                            ; Wait until near end of frame
   MOV   R1,#ModReg1Ind                          ; In Mode Register 1
   MOV   R2,#0CCH                                ;    set normal bits plus VB
   MOV   R3,#001H                                ;    and leave Am8052 enabled
   LCALL WrAm8052Reg
   MOV   R0,#4                                   ; Wait for approximately
   CLR   A                                       ;    two milliseconds
RIS1:
   DJNZ  ACC,RIS1
   DJNZ  R0,RIS1
   LJMP  Reset                                   ; Go do power-up procedure

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CursorBackward:
;............................................................................
;  Moves the active position backward on the screen the indicated number
;  of positions. If no count is suppplied then one position is moved. Also
;  if the amount moved is beyond 0 then movement stops at 0.
;
;  Inputs:   PrmCnt
;            PrmBuf
;            ActCol
;  Outputs:  ActCol                   altered by the appropriate amount
;  bad   A
```

```
   JB    PrmBadFlg,CBW2            ; If a bad parameter buffer is present
                                   ;     get with an error return
   MOV   A,PrmCnt                  ; Test if no parameters
   JNZ   CBW_00                    ;     if none then the default is move
                                   ;     one_coloumn left
CBW_99:
   MOV   A,#1
   MOV   PrmBuf,A
   SJMP  CBW_01
CBW_00:
   DEC   A                         ; Then test for only one parameter
   JNZ   CBW2                      ;     any more parameters is considered
                                   ;     an error return
   MOV   A,PrmBuf
   JZ    CBW_99
CBW_01:
   CLR   C                         ; We must subtract the requested
   XCH   A,ActCol                  ;     amount from ActCol and then test
   SUBB  A,ActCol                  ;     that we have not moved the
   JNC   CBW1                      ;     cursor below 0
   MOV   A,#00H                    ; If so make ActCol 0
CBW1:
   MOV   ActCol,A                  ; Otherwise restore adjusted ActCol
   SJMP  CBW3
CBW2:
   MOV   A,#00H                    ; On an error return remove all traces
   MOV   CtlPtrHi,A                ;     of this control
   MOV   CtlPtrLo,A
CBW3:
   LCALL PlcCsr                    ; Set new cursor position and zone
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CursorDown:
;.............................................................................
;  Moves the active position down on the screen the indicated number
;  of rows. If no count is supplied then one row is moved. Also
;  if the amount moved is beyond the bottom row then movement stops.
;
;  Inputs:   PrmCnt
;            PrmBuf
;            ActRow
;  Outputs:  ActRow                    altered by the appropriate amount
```

```
;  bad   A,R2,R3

   JB    PrmBadFlg,CD6             ; If a bad parameter buffer is indicate
                                   ;     error return
   MOV   A,PrmCnt                  ; Test for zero parameters indicating
   JNZ   CD1                       ;     a default value of 1
CD0:
   MOV   PrmBuf,#1
   SJMP  CD2
CD1:
   DEC   A                         ; If more then 1 parameter this is an
   JNZ   CD6                       ;     error return
   MOV   A,PrmBuf
   JZ    CD0
CD2:
   JNB   WndActFlg,CD3
   MOV   A,#13                          ; If window is active limit of
   SJMP  CD4                            ;     movement is 14
CD3:
   MOV   A,#29                          ; If background is active limit
                                        ;     of movement is 30
CD4:
   CLR   C
   SUBB  A,ActRow
   MOV   R2,A
   SUBB  A,PrmBuf
   JC    CD5
   MOV   R2,PrmBuf
CD5:
   MOV   A,ActRow
   ADD   A,R2
   MOV   R2,A
   MOV   R3,ActCol                      ; Set input values for NewCsr
   LCALL NewCsr                         ; Setup new cursor variables
   SJMP  CD7                            ;     (CurRow,ActRow,ActCol)
                                        ;     and place cursor
CD6:
   MOV   A,#00H                         ; On error remove all traces
   MOV   CtlPtrHi,A                     ;     of control
   MOV   CtlPtrLo,A
CD7:
   RET
```

```
CursorForward:
;.......................................................................
; Moves the active position forward on the screen the indicated number
; of positions. If no count is suppplied then one position is moved. Also
; if the amount moved is beyond the last column then movement stops.
;
; Inputs:   PrmCnt
;           PrmBuf
;           ActCol
; Outputs:  ActCol                   altered by the appropriate amount
; bad    A,R3
;.......................................................................

   JB     PrmBadFlg,CFW6                 ; Indicates a bad parameter buffer
                                         ;    error return

   MOV    A,PrmCnt
   JNZ    CFW1
CFW0:
   MOV    PrmBuf,#1                      ; No parameters indicate a
   SJMP   CFW2                           ;    movement of 1
CFW1:
   DEC    A                              ; If more than 1 parameter
   JNZ    CFW6                           ;    error return
   MOV    A,PrmBuf
   JZ     CFW0
CFW2:
   JNB    WndActFlg,CFW3                 ; If window is currently active
   MOV    A,#39                          ;    limit is 40 character pos.
   SJMP   CFW4
CFW3:
   MOV    A,#127                         ; Else if either Bgd. or Msg
CFW4:                                    ;    is active limit is 128
   CLR    C
   SUBB   A,ActCol                       ; The maximum amount we may move
   MOV    R3,A                           ;    is Limit-ActCol = MAx
   SUBB   A,PrmBuf                       ; To determine whether to use Max
   JC     CFW5                           ;    or requested is Max-Req, if
                                         ;    Max > Req then move Req
   MOV    R3,PrmBuf
CFW5:
   MOV    A,R3                           ;    else move Max
   ADD    A,ActCol                       ; Add our relative movement to
   MOV    ActCol,A                       ;    our current position
```

```
   SJMP  CFW7                                ;    and we're done
CFW6:
   MOV   A,#00H                              ; If an error is discovered
   MOV   CtlPtrHi,A                          ;    remove all traces of this
   MOV   CtlPtrLo,A                          ;    control
CFW7:
   LCALL PlcCsr                              ; Relocate our cursor before we
   RET                                       ;    leave

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CursorPosition:
;...........................................................................
; Moves the active position to the position on the screen as specified
; If no valus are suppplied then the active position is moved to the home
; position. Also if either of the parameters are lacking hte the value
; of 0 is defaulted to.
;
; Inputs:   PrmCnt
;           PrmBuf
;           ActCol
;           ActRow
; Outputs:  ActCol                   altered by the appropriate amount
;           ActRow
; bad    A,R2,R3,R4
;...........................................................................

   JB    PrmBadFlg,CP9                       ; Indicates a bad param buffer
                                             ;    error return
   CLR   A                                   ; Establish default values
   MOV   R2,A
   MOV   R3,A
   MOV   A,PrmCnt                            ; Determine default case
   JZ    CP8                                 ;    default if jump taken
                                             ; Set buffer pointer for next prm
   DEC   A                                   ; Test if first param is default
   JZ    CP4                                 ;    jump if true
CP1:
   MOV   A,PrmCnt                            ; Last test for only 2 parameters
   CJNE  A,#02H,CP9                          ;    error if jump taken
   MOV   A,PrmBuf+1
   JZ    CP91
   DEC   A
```

```
CP91:
   MOV   R3,A
   JB    WndActFlg,CP2                          ; Limit for window is 40 cols.
   MOV   A,#127                                 ; Limit for bgd and msg is
   SJMP  CP3                                    ;     128 columns
CP2:
   MOV   A,#39
CP3:
   CLR   C
   MOV   R4,A                                   ; Decide if maximum value or
   SUBB  A,R3                                   ;    requested value is used
   JNC   CP4                                    ;    for the new cursor column
   MOV   A,R4
   MOV   R3,A
CP4:
   MOV   A,PrmBuf                               ; Calculate new row position
   JZ    CP92
   DEC   A
CP92:
   MOV   R2,A
   JB    WndActFlg,CP5                          ; Limit for Window is 15 rows
   JB    MsgActFlg,CP6                          ; Limit for Message is 1 row
   MOV   A,#29                                  ; Limit for Background is 30 rows
   SJMP  CP7
CP5:
   MOV   A,#13
   SJMP  CP7
CP6:
   MOV   A,#0
CP7:
   CLR   C
   MOV   R4,A                                   ; Decide if maximum value or
   SUBB  A,R2                                   ;    requested value is used
   JNC   CP8                                    ;    for the new cursor row
   MOV   A,R4
   MOV   R2,A
CP8:
   LCALL NewCsr                                 ; Establish new cursor variables
   SJMP  CP10                                   ;    and we're finished
CP9:
   CLR   A                                      ; If an error has been detected
   MOV   CtlPtrHi,A                             ;    remove all traces of this
```

```
   MOV   CtlPtrLo,A                                  ;    control
CP10:
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CursorUp:
;............................................................................
;  Moves the active position up on the screen the indicated number
;  of rows. If no count is suppplied then one row is moved. Also
;  if the amount moved is beyond the top row then movement stops.
;
;  Inputs:   PrmCnt
;            PrmBuf
;            ActRow
;  Outputs:  ActRow                        altered by the appropriate amount
;  bad    A,R2,R3
;............................................................................

   JB    PrmBadFlg,CU4                               ; Indicates bad param buffer
                                                     ;    error return
   MOV   A,PrmCnt
   JNZ   CU1                                   ; If not zero test if more then 1 param
CU0:
   MOV   PrmBuf,#1                                   ; Default (no Parameters)
   SJMP  CU2                                         ;    Move cursor up 1 row
CU1:
   DEC   A
   JNZ   CU4                                   ; If not zero too many parameters error
   MOV   A,PrmBuf
   JZ    CU0
CU2:
   MOV   A,ActRow                                    ; Insure that requested cursor
   CLR   C
   SUBB  A,PrmBuf                                    ;    movement doesn't move cusor
   JNC   CU3                                         ;    below 0
   CLR   A                                           ; Absolute minimum cursor vert.
CU3:                                                 ;    position
   MOV   R2,A                                        ; Set new cursor vert. position
   MOV   R3,ActCol                                   ; Maintain current horz. position
   LCALL NewCsr                                      ; Establish new cursor variables
   SJMP  CU5
```

```
CU4:
   CLR   A                               ; If an error occurs remove
   MOV   CtlPtrHi,A                      ;    all traces of this control
   MOV   CtlPtrLo,A
CU5:
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DeleteLine:
;............................................................................
; Deletes the number of rows specified by the single allowed parameter.  The
; Vertical Editing Mode (VEM) determines whether blank rows are shifted into
; the bottom or the top of the display.  If more rows are specified than can
; be deleted then the maximum amount is deleted.  After ensuring parameter
; validity this routine waits for vertical smooth scrolling to finish before
; beginning its work.  This control is not allowed when the message display
; is active.
;
;  inp   none
;  out   Display dependent variables may change
;  bad   A,R2
;............................................................................


   JB    MsgActFlg,DL9
   JB    PrmBadFlg,DL9
   MOV   A,PrmCnt
   JNZ   DL2
DL1:
   MOV   PrmBuf,#1
   SJMP  DL3
DL2:
   DEC   A
   JNZ   DL9
   MOV   A,PrmBuf
   JZ    DL1
DL3:
   JB    VrtScrlFlg,$
   JB    VEMBit,DL7
   JNB   WndActFlg,DL4
   MOV   A,#14
   SJMP  DL5
```

```
DL4:
   MOV   A,#30
DL5:
   CLR   C
   SUBB  A,ActRow
   MOV   R2,A
   SUBB  A,PrmBuf
   JC    DL6
   MOV   R2,PrmBuf
DL6:
   LCALL DelRow_MovUp
   DJNZ  R2,DL6
   RET
DL7:
   CLR   C
   MOV   A,ActRow
   INC   A
   MOV   R2,A
   SUBB  A,PrmBuf
   JC    DL8
   MOV   R2,PrmBuf
DL8:
   LCALL DelRow_MovDn
   DJNZ  R2,DL8
   RET
DL9:
   CLR   A
   MOV   CtlPtrHi,A
   MOV   CtlPtrLo,A
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
EraseInDisplay:
;............................................................................
;  Depending on the parameter sent this control erases from the top of the
;  display to the active postion, the active postion to the bottom of the
;  dispplay, or the entire display.
;
;  inp   PrmCnt                          the count of parameters
;        PrmBuf                          buffer containing parameters
;
;  out   none
;  bad   A,R0,R1,R5,P2
```

```
    JB     PrmBadFlg,EID17              ; Indicates a bad param buffer
                                        ;    error return
    JB     PrmMaxFlg,EID17              ; Indicates too many parameters
                                        ;    error return
    MOV    R1,#PrmBuf                   ;    parameter buffer
    MOV    A,PrmCnt                     ; Prepare for progression thru
    JNZ    EID0
    MOV    @R1,A
    INC    A
EID0:
    MOV    R2,A
EID1:
    CJNE   @R1,#00H,EID6                ; If 0 (default) then erase from
                                        ;    active pos to last position
                                        ;    in display
    LCALL  EraActEnd                    ; First erasethe remainder of
    MOV    A,CurRow                     ;    this row and get pointer
EID2:
    CJNE   A,EndRow,EID3                ; If ptr is last row quit
    LJMP   EID16a
EID3:
    CJNE   A,BtmRow,EID4                ; If ptr is last row in visible
                                        ;    dsp start erasing rows below
    MOV    A,RemRow
    SJMP   EID5
EID4:
    MOV    P2,A                         ; Otherwise get next row ptr to
    MOV    A,RcbOff                     ;    erase
    ADD    A,#RCB_RowPag
    MOV    R0,A
    MOVX   A,@R0
EID5:
    MOV    R5,A                         ; Save row pointer
    LCALL  FrcEraRow                    ; Erase row
    MOV    A,R5                         ; Restore pointer
    SJMP   EID2                         ; Prepare for next row
EID6:
    CJNE   @R1,#01H,EID11               ; If 1 then erase from beginning
                                        ;    of display thru active pos
    MOV    A,BgnRow                     ; Start at the beginning of the
```

```
EID7:                                   ;    linked list
    CJNE   A,CurRow,EID8                ; If not at top get erase first
                                        ;    first row
    LCALL  EraBgnAct                    ; Finally erase current row to
    SJMP   EID16a                       ;    active pos. and get next prm
EID8:
    MOV    R5,A                         ; Preserve erased page ptr
    LCALL  FrcEraRow                    ;    erase this row
    MOV    A,R5
    CJNE   A,BtmRow,EID9                ; Test for bottom of display
    MOV    A,RemRow                     ;    if true, next row is RemRow
    SJMP   EID7
EID9:
    MOV    P2,A                         ; Otherwise get next row in list
    MOV    A,RcbOff
    ADD    A,#RCB_RowPag
    MOV    R0,A
    MOVX   A,@R0
    SJMP   EID7                         ; Proceed to erase it
EID11:
    CJNE   @R1,#02H,EID16               ; If 2 then erase from top to
                                        ;    bottom
    MOV    A,BgnRow                     ; Start at the beginning
EID12:
    MOV    R5,A
    LCALL  FrcEraRow                    ; Erase this row then proceed
    MOV    A,R5                         ;    to the next appropriate
    CJNE   A,EndRow,EID13               ; Continue til last row is done
    SJMP   EID16a                       ;    then procedd with next param
EID13:
    CJNE   A,BtmRow,EID14               ; When we reach the bottom of the
                                        ;    dsp start with RemRow and
    MOV    A,RemRow                     ;    continue
    SJMP   EID12
EID14:
    MOV    P2,A                         ; Otherwise just continue with
    MOV    A,RcbOff                     ;    the next row
    ADD    A,#RCB_RowPag
    MOV    R0,A
    MOVX   A,@R0
    SJMP   EID12
```

```
EID16a:
   MOV    A,ExtRow
   LCALL  FrcEraRow
EID16:                                          ;    done with this parameter
   INC    R1                                    ; Point to next parameter
   DJNZ   R2,EID1                               ; If more parameters proceed
   LCALL  PlcCsr                                ;    else return
   RET
EID17:
   CLR    A
   MOV    CtlPtrHi,A
   MOV    CtlPtrLo,A
EID18:
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
EraseInLine:
;...........................................................................
; Denpending on the parameter sent this control erases from the beginning
; of the row to the active position, the active position to the end of
; the row, or the entire row.
;
; inp   PrmCnt                                  the count of parameters
;       Prmbuf                                  buffer containing control params
; out   none
; bad   A,R1,R2
;...........................................................................


   JB     PrmBadFlg,EIL5                        ; Indicates a bad param buffer
                                                ;    error return
   JB     PrmMaxFlg,EIL5                        ; Indicates too many parameters
                                                ;    error return
   MOV    R1,#PrmBuf                            ; Point to first parameter value
   MOV    A,PrmCnt
   JNZ    EIL0
   MOV    @R1,A                                 ;    and function to 0
   INC    A                                     ; If default, set count to 1
EIL0:
   MOV    R2,A
EIL1:                                      ; Test for each of the allowed params
   CJNE   @R1,#00H,EIL2                         ; Each in turn
```

```
   LCALL  EraActEnd                             ; If 0 (default) then erase from
   SJMP   EIL4                                  ;    active pos to end of row

EIL2:
   CJNE   @R1,#01H,EIL3                         ; If 1 then erase from beginning
                                                ;    of row until the active pos
   LCALL  EraBgnAct
   SJMP   EIL4
EIL3:
   CJNE   @R1,#02H,EIL4                         ; If 2 then erase the whole line
   MOV    A,CurRow
   LCALL  FrcEraRow
EIL4:
   INC    R1                                    ; Update our pointer into PrmBuf
   DJNZ   R2,EIL1                               ;    and get all the parameters
   SJMP   EIL6
EIL5:
   CLR    A                                     ; If an error was detected remove
   MOV    CtlPtrHi,A                            ;    all traces of this control
   MOV    CtlPtrLo,A
EIL6:
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
InsertLine:
;...........................................................................
; Inserts the number of rows specified by the single allowed parameter.  The
; Vertical Editing Mode (VEM) determines whether blank rows are shifted into
; the bottom or the top of the display.  If more rows are specified than can
; be inserted then the maximum amount is inserted.  After ensuring parameter
; validity this routine waits for vertical smooth scrolling to finish before
; beginning its work.  This control is not allowed when the message display
; is active.
;
; inp   PrmCnt                                  parameter count
;       PrmBuf                                  buffer containing parameter(s)
; out   none
; bad   A,R2
;...........................................................................
```

```
    JB     MsgActFlg,IL9                  ; Insert line is not functional
                                          ;    in message window
    JB     PrmBadFlg,IL9                  ; Bad parameter buffer
    MOV    A,PrmCnt                       ; Test for default parameter
    JNZ    IL2                            ;    jump if not default
IL1:
    MOV    PrmBuf,#1                      ; Else setup variables for
    SJMP   IL3                            ;    default
IL2:
    DEC    A                              ; Test for only one parameter
    JNZ    IL9                            ;    if not zero too many prms
    MOV    A,PrmBuf                       ; 0 is handled as a prm of 1
    JZ     IL1
IL3:
    JB     VrtScrlFlg,$                   ; If a scroll is in progress
                                          ;    wait til finished to cont.
    JB     VEMBit,IL7                     ; Decide which way to move rows
    JNB    WndActFlg,IL4                  ; Bgd is active if taken
    MOV    A,#14                          ; Limit of insert in window
    SJMP   IL5                            ;    is fourteen
IL4:
    MOV    A,#30                          ; Limit for background is thirty
IL5:
    CLR    C                              ; Maximum amount able to move
    SUBB   A,ActRow                       ; Max=Limit-Current
    MOV    R2,A                           ; Preserve maximum
    SUBB   A,PrmBuf
    JC     IL6                            ; If taken move maximum
    MOV    R2,PrmBuf                      ; else move requested
IL6:
    LCALL  InsRow_MovDn                   ; Insert rows
    DJNZ   R2,IL6                         ; Count times
    RET
IL7:
    CLR    C
    MOV    A,ActRow                       ; With VEM bit set we just check
    INC    A                              ;    how far it is to the top
    MOV    R2,A                           ; and use the smaller value
    SUBB   A,PrmBuf
    JC     IL8
    MOV    R2,PrmBuf
```

```
IL8:
    LCALL  InsRow_MovUp                   ; Insert count rows
    DJNZ   R2,IL8
    RET
IL9:
    CLR    A                              ; If an error was indicated
    MOV    CtlPtrHi,A                     ;    remove all traces of this
    MOV    CtlPtrLo,A                     ;    controll
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ResetMode:
;.............................................................................
; Reset the modes indicated by the selective parameters to their initial
; states.
;
; Parameters                              Meaning
; ----------                              -------
; 7                                       VEM (insert/delete above active row)
; ?3                                      AMDDWM (compressed)
; ?4                                      AMDSCM (smooth scrolling)
; ?5                                      AMDSPM (reversed screen)
;
; inp   PrmCnt                            count of parameters sent
;       PrmBuf                            buffer containing parameters
; out   none
; bad   A,R0,R1,R2,R3,R6,DPTR
;.............................................................................

    JB     PrmBadFlg,RSTMD7               ; Indicates a bad param buffer
                                          ;    error return
    JB     PrmMaxFlg,RSTMD7               ; Indicates too many parameters
                                          ;    error return
    MOV    A,PrmCnt
    JZ     RSTMD8
    MOV    R6,A
    MOV    R0,#PrmBuf
RSTMD1:
    MOV    A,PrmPvt
    JZ     RSTMD5
```

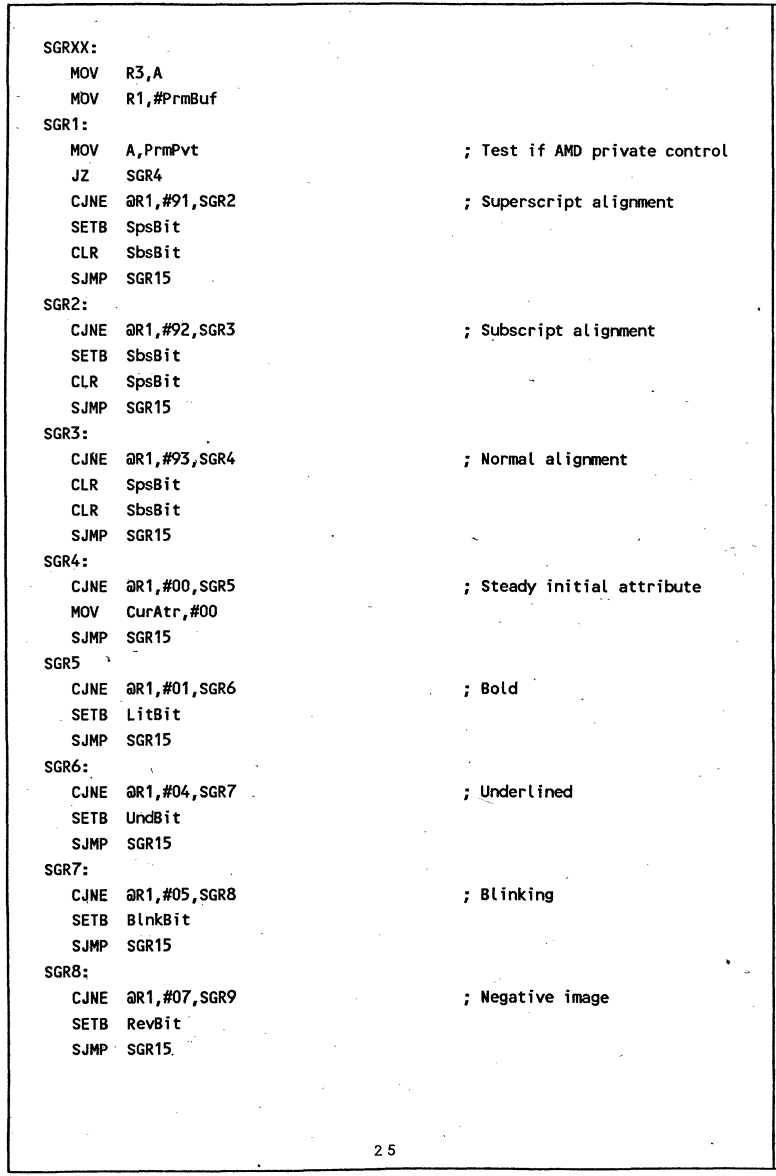
```
RSTMD2:
   CJNE  @R0,#03H,RSTMD3              ; AMDDWM (normal mode)
   JNB   AMDDWMBit,RSTMD6
   LCALL RMdSup
   SJMP  RSTMD6
RSTMD3:
   CJNE  @R0,#04,RSTMD4               ; AMDSCM (jump scrolling)
   CLR   AMDSCMBit
   SJMP  RSTMD6
RSTMD4:
   CJNE  @R0,#05H,RSTMD6              ; AMDSPM (normal screen)
   CLR   AMDSPMBit
   SJMP  RSTMD6
RSTMD5:
   CJNE  @R0,#07H,RSTMD6              ; VEM (insert/delete below active
                                      ;    row)
   CLR   VEMBit
RSTMD6:
   INC   R0
   DJNZ  R6,RSTMD1
   SJMP  RSTMD8
RSTMD7:
   CLR   A                            ; If an error is indicated
   MOV   CtlPtrHi,A                   ;    remove all traces of
   MOV   CtlPtrLo,A                   ;    this control
RSTMD8:
   RET

;-----------------------------------------------------------------------
RMdSup:

   JB    VrtScrlFlg,$
   JB    HrzScrlFlg,$
   MOV   HrzFrmSet,R0
   MOV   HrzPxlShf,R6
   LCALL DlyTilEndFrm
   MOV   R1,#ModReg1Ind
   MOV   R2,#0CCH
   MOV   R3,#001H
   LCALL WrAm8052Reg
   CLR   AMDDWMBit
   MOV   A,#007H
```

```
   LCALL SetCelWid
   MOV   VisCol,#0
   LCALL SetWndPos
   JB    MsgActFlg,RMd0
   JB    WndActFlg,RMd0
   MOV   A,BgnRow
   SJMP  RMd1
RMd0:
   MOV   DPTR,#BgdVarBuf+(BgnRow-CurAtr)
   MOVX  A,@DPTR
RMd1:
   MOV   DPH,A
   MOV   DPL,#BgdRCB0.AN.OFST+RCB_RowPag
   MOV   R1,#6
RMd2:
   MOVX  A,@DPTR
   MOV   DPH,A
   DJNZ  R1,RMd2
   MOV   DPTR,#BgdVarBuf+(TopRow-CurAtr)
   CLR   EX0
   MOVX  @DPTR,A
   MOV   DPTR,#BgdMDB0+MDB_RowPag
   MOVX  @DPTR,A
   MOV   DPTR,#BgdMDB1+MDB_RowPag
   MOVX  @DPTR,A
   JB    MsgActFlg,RMd3
   JB    WndActFlg,RMd4
   MOV   TopRow,A
   MOV   VisRow,#6
   MOV   DspHgt,#24
RMd3:
   MOV   DspWid,#80
RMd4:
   MOV   DPTR,#BgdVarBuf+(VisCol-CurAtr)
   CLR   A
   MOVX  @DPTR,A
   INC   DPL
   MOV   A,#6
   MOVX  @DPTR,A
   MOV   DPTR,#MsgVarBuf+(VisCol-CurAtr)
   CLR   A
```

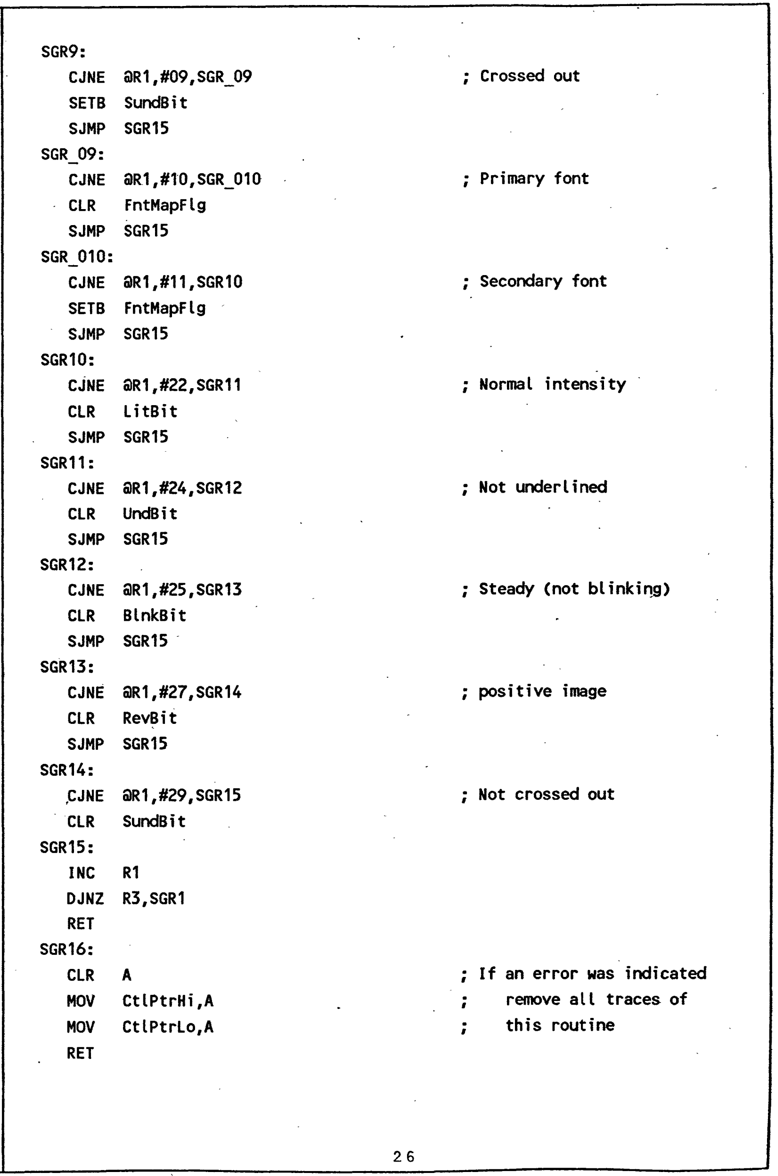
```
    MOVX  @DPTR,A
    MOV   A,#034H
    MOV   DPTR,#BgdMDB0+MDB_Tslc
    MOVX  @DPTR,A
    MOV   DPTR,#BgdMDB1+MDB_Tslc
    MOVX  @DPTR,A
    MOV   DPTR,#NrmRRB+RRB_Tslc_NcsHi
    MOVX  @DPTR,A
    INC   DPL
    MOV   A,#04DH
    MOVX  @DPTR,A
    INC   DPL
    CLR   A
    MOVX  @DPTR,A
    INC   DPL
    MOV   A,#00DH
    MOVX  @DPTR,A
    INC   DPL
    CLR   A
    MOVX  @DPTR,A
    INC   DPL
    MOV   A,#08DH
    MOVX  @DPTR,A
    INC   DPL
    INC   DPL
    INC   DPL
    MOV   A,#001H
    MOVX  @DPTR,A
    INC   DPL
    MOV   A,#086H
    MOVX  @DPTR,A
    JNB   MsgVisFlg,RMd9
    MOV   A,#26
    SJMP  RMd10
RMd9:
    MOV   A,#24
RMd10:
    MOV   DPTR,#TrmWDB+WDB_BgnRow
    MOVX  @DPTR,A
    INC   DPL
    MOVX  @DPTR,A
    MOV   A,#24
```

```
    MOV   DPTR,#MsgWDB+WDB_BgnRow
    MOVX  @DPTR,A
    INC   DPL
    MOVX  @DPTR,A
    SETB  EX0
    JNB   MsgActFlg,RMd11
    MOV   RowAdd,#24
RMd11:
    MOV   A,CsrSiz
    CJNE  A,#09AH,RMd5
    MOV   CsrSiz,#0BCH
    SJMP  RMd8
RMd5:
    CJNE  A,#0AAH,RMd6
    MOV   CsrSiz,#0CCH
    SJMP  RMd8
RMd6:
    CJNE  A,#058H,RMd7
    MOV   CsrSiz,#06AH
    SJMP  RMd8
RMd7:
    MOV   CsrSiz,#00DH
RMd8:
    LCALL ChgCsrSiz
    MOV   R1,#ModReg1Ind
    MOV   R2,#0C8H
    MOV   R3,#001H
    LCALL WrAm8052Reg
    LCALL PlcCsr
    MOV   R0,HrzFrmSet
    MOV   R6,HrzPxlShf
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrollDown:
;.............................................................................
; Scrolls number of rows specified by the single allowed parameter.
; If more rows are specified than can be scrolled then the maximum amount is
; scrolled.
;
;  inp   PrmCnt                                   parameter count
;        PrmBuf                                   buffer containing parameter(s)
; out   none
;
; bad   A,R2
;.............................................................................

   JB    PrmBadFlg,SD5                  ; Indicates a bad param buffer
                                        ;    error return
   CLR   C
   JB    MsgActFlg,SD6                  ; Message window cannot scroll
                                        ;    vertically
   MOV   A,PrmCnt
   JNZ   SD1
   MOV   PrmBuf,#1                      ; If count = 0 default to 1 row
   SJMP  SD2
SD1:
   DEC   A                              ; If more then one parameter
   JNZ   SD5                            ;     error return
SD2:
   MOV   A,PrmBuf                       ; Amount to scroll is the smaller of
   SUBB  A,VisRow                       ;     requested rows Vs. VisRow
   JC    SD3
   MOV   A,VisRow
   JZ    SD6
   SJMP  SD4
SD3:
   MOV   A,PrmBuf
SD4:
   LCALL ScrlDnDsp                      ; Scroll in progress
   SJMP  SD6                            ;     and we're finished
SD5:
   CLR   A                              ; On an error remove all traces
   MOV   CtlPtrHi,A                     ;     of this control
   MOV   CtlPtrLo,A
SD6:
   RET
```

```
SelGrfRendition:
;...............................................................................
; After checking parameter validity tests this control changes the following
; character attributes depending on the selective parameter(s) sent.
;
; Parameters                            meaning
; ----------                            -------
; 0                                     Steady, initial attributes
; 1                                     Bold, hi intensity
; 4                                     Underlined
; 5                                     Blinking
; 7                                     Negative image
; 9                                     Crossed out
; 10                                    Primary Font
; 11                                    Secondary Font
; 22                                    Normal intensity
; 24                                    Not underlined
; 25                                    Steady (not blinking)
; 27                                    Positive image
; 29                                    Not crossed out
; ?91                                   Superscript alignment
; ?92                                   Subscript alignment
; ?93                                   Normal alignment
; any other parameter is ignored
;
; inp   PrmCnt                          number of parameters to work on
;       PrmBuf                          buffer containing the parameter(s)
; out   none
; bad   A,R1,R3
;...............................................................................

   JNB   PrmBadFlg,SGR01                ; Indicates a bad param buffer
                                        ;    error return
   LJMP  SGR16
SGR01:
   JNB   PrmMaxFlg,SGR02                ; Indicates too many parameters
                                        ;    error return
   LJMP  SGR16
SGR02:
   MOV   A,PrmCnt
   JNZ   SGRXX
   LJMP  SGR16
```



```
SGRXX:
   MOV    R3,A
   MOV    R1,#PrmBuf
SGR1:
   MOV    A,PrmPvt                        ; Test if AMD private control
   JZ     SGR4
   CJNE   @R1,#91,SGR2                    ; Superscript alignment
   SETB   SpsBit
   CLR    SbsBit
   SJMP   SGR15
SGR2:
   CJNE   @R1,#92,SGR3                    ; Subscript alignment
   SETB   SbsBit
   CLR    SpsBit
   SJMP   SGR15
SGR3:
   CJNE   @R1,#93,SGR4                    ; Normal alignment
   CLR    SpsBit
   CLR    SbsBit
   SJMP   SGR15
SGR4:
   CJNE   @R1,#00,SGR5                    ; Steady initial attribute
   MOV    CurAtr,#00
   SJMP   SGR15
SGR5
   CJNE   @R1,#01,SGR6                    ; Bold
   SETB   LitBit
   SJMP   SGR15
SGR6:
   CJNE   @R1,#04,SGR7                    ; Underlined
   SETB   UndBit
   SJMP   SGR15
SGR7:
   CJNE   @R1,#05,SGR8                    ; Blinking
   SETB   BlnkBit
   SJMP   SGR15
SGR8:
   CJNE   @R1,#07,SGR9                    ; Negative image
   SETB   RevBit
   SJMP   SGR15
```



```
SGR9:
   CJNE   @R1,#09,SGR_09                  ; Crossed out
   SETB   SundBit
   SJMP   SGR15
SGR_09:
   CJNE   @R1,#10,SGR_010                 ; Primary font
   CLR    FntMapFlg
   SJMP   SGR15
SGR_010:
   CJNE   @R1,#11,SGR10                   ; Secondary font
   SETB   FntMapFlg
   SJMP   SGR15
SGR10:
   CJNE   @R1,#22,SGR11                   ; Normal intensity
   CLR    LitBit
   SJMP   SGR15
SGR11:
   CJNE   @R1,#24,SGR12                   ; Not underlined
   CLR    UndBit
   SJMP   SGR15
SGR12:
   CJNE   @R1,#25,SGR13                   ; Steady (not blinking)
   CLR    BlnkBit
   SJMP   SGR15
SGR13:
   CJNE   @R1,#27,SGR14                   ; positive image
   CLR    RevBit
   SJMP   SGR15
SGR14:
   CJNE   @R1,#29,SGR15                   ; Not crossed out
   CLR    SundBit
SGR15:
   INC    R1
   DJNZ   R3,SGR1
   RET
SGR16:
   CLR    A                               ; If an error was indicated
   MOV    CtlPtrHi,A                      ;    remove all traces of
   MOV    CtlPtrLo,A                      ;    this routine
   RET
```

```
ScrollLeft:
;...........................................................................
;  Scrolls the display leftward the number of columns specified by the single
;  numeric parameter. An attempt to scroll the rightmost column of the display
;  leftward beyond the rightmost column on the monitor leaves it at the right-
;  most column.
;
;  inp   PrmCnt                          count of parameters
;        PrmBuf                          buffer containing the parameters
;  out   none
;  bad   A,R1,R2
;...........................................................................

   JB    PrmBadFlg,SL7                   ; Indicates a bad param buffer
                                         ;    error return
   JB    WndActFlg,SL8               ; If Window Horz. scrolling is not
                                     ;    allowed
   MOV   A,PrmCnt
   JNZ   SL1
   MOV   PrmBuf,#1                   ; If no parameters default to 1 column
   SJMP  SL2
SL1:
   DEC   A                           ; If more then one parameter error rtn
   JNZ   SL7
SL2:
   JNB    AMDDWMBit,SL3              ; If compressed mode maximum number
                                     ;    of columns to be scrolled is 48
   MOV   R2,#8                       ; Else columns = 8
   SJMP  SL4
SL3:
   MOV   R2,#48
SL4:
   CLR   C
   MOV   A,VisCol                    ; Number of columns available for
   XCH   A,R2                        ;    scrolling = Maximum - VisCol
   SUBB  A,R2                        ;
   JZ    SL8
   MOV   R7,A                        ;    smaller of Available Vs. requested
   MOV   A,PrmBuf
   CLR   C
   SUBB  A,R7
   JNC   SL5
   MOV   R7,PrmBuf
```

```
SL5:
   MOV   A,R7
   LCALL ScrlLtDsp
   RET
SL7:
   CLR   A                                   ; If error remove all traces of
   MOV   CtlPtrHi,A                          ;    control routine
   MOV   CtlPtrLo,A
SL8:
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SetMode:
;...........................................................................
;  Set the modes indicated by the selective parameters to their alternate
;  states.
;
;  Parameters                                Meaning
;  ----------                                -------
;  7                                         VEM (insert/delete above active row)
;  ?3                                        AMDDWM (compressed)
;  ?4                                        AMDSCM (smooth scrolling)
;  ?5                                        AMDSPM (reversed screen)
;
;  inp   PrmCnt                              count of parameters sent
;        PrmBuf                              buffer containing parameters
;
;  out   none
;
;  bad   A,R0,R1,R2,R3,R6,DPTR
;...........................................................................

   JB    PrmBadFlg,STMD7                       ; Indicates a bad param buffer
                                               ;    error return
   JB    PrmMaxFlg,STMD7                       ; Indicates too many parameters
                                               ;    error return
   MOV   A,PrmCnt                              ; If zero no action just return
   JZ    STMD8
   MOV   R6,A                                  ; Establish loop count from PrmCnt
   MOV   R0,#PrmBuf                            ; Establish pointer for param
STMD1:                                         ;    comparisons
   MOV   A,PrmPvt                              ; Test if private selective
   JZ    STMD5                                 ;    parameter
```

```
STMD2:
   CJNE  @R0,#03H,STMD3                          ; AMDDWM Compressed mode
   JB    AMDDWMBit,STMD6
   LCALL SMdSup
   SJMP  STMD6
STMD3:
   CJNE  @R0,#04H,STMD4                          ; AMDSCM Smooth scrolling
   SETB  AMDSCMBit
   SJMP  STMD6
STMD4:
   CJNE  @R0,#05H,STMD6                          ; AMDSPM reversed screen
   SETB  AMDSPMBit
   SJMP  STMD6
STMD5:
   CJNE  @R0,#07H,STMD6                          ; VEM mode
   SETB  VEMBit
STMD6:
   INC   R0
   DJNZ  R6,STMD1
   SJMP  STMD8
STMD7:
   CLR   A                                       ; If an error is indicated
   MOV   CtlPtrHi,A                              ;    remove all traces of
   MOV   CtlPtrLo,A                              ;    this control
STMD8:
   RET

;----------------------------------------------------------------------------

SMdSup:

   JB    VrtScrlFlg,$
   JB    HrzScrlFlg,$
   MOV   HrzFrmSet,R0
   MOV   HrzPxlShf,R6
   LCALL DlyTilEndFrm
   MOV   R1,#ModReg1Ind
   MOV   R2,#0CCH
   MOV   R3,#001H
   LCALL WrAm8052Reg
   SETB  AMDDWMBit
   MOV   A,#004H
```

```
   LCALL SetCelWid
   MOV   VisCol,#0
   LCALL SetWndPos
   JB    MsgActFlg,SMd1
   JB    WndActFlg,SMd1
   MOV   R4,BgnRow
   MOV   R1,BtmRow
   MOV   R2,RemRow
   MOV   R3,EndRow
   SJMP  SMd2
SMd1:
   MOV   DPTR,#BgdVarBuf+(BgnRow-CurAtr)
   CLR   EX0
   MOVX  A,@DPTR
   MOV   R4,A
   INC   DPL
   INC   DPL
   MOVX  A,@DPTR
   INC   DPL
   MOV   R1,A
   MOVX  A,@DPTR
   INC   DPL
   MOV   R2,A
   MOVX  A,@DPTR
   MOV   R3,A
SMd2:
   MOV   DPH,R1
   MOV   DPL,#BgdRCB0.AN.OFST+RCB_RowPag
   MOV   A,R2
   MOVX  @DPTR,A
   INC   DPL
   MOV   A,#BgdRCB0.AN.OFST
   MOVX  @DPTR,A
   MOV   DPH,R3
   MOV   A,TrmOff
   MOVX  @DPTR,A
   DEC   DPL
   MOV   A,TrmRow
   MOVX  @DPTR,A
   MOV   DPTR,#BgdVarBuf+(TopRow-CurAtr)
   MOV   A,R4
   MOVX  @DPTR,A
```

```
    MOV    DPTR,#BgdMDB0+MDB_RowPag
    MOVX   @DPTR,A
    MOV    DPTR,#BgdMDB1+MDB_RowPag
    MOVX   @DPTR,A
    JB     MsgActFlg,SMd3
    JB     WndActFlg,SMd4
    MOV    TopRow,A
    MOV    VisRow,#0
    MOV    BtmRow,R3
    MOV    RemRow,R3
    MOV    DspHgt,#30
SMd3:
    MOV    DspWid,#120
SMd4:
    MOV    DPTR,#BgdVarBuf+(VisCol-CurAtr)
    CLR    A
    MOVX   @DPTR,A
    INC    DPL
    MOVX   @DPTR,A
    INC    DPL
    INC    DPL
    MOV    A,R4
    MOVX   @DPTR,A
    INC    DPL
    MOV    A,R3
    MOVX   @DPTR,A
    INC    DPL
    MOVX   @DPTR,A
    MOV    DPTR,#MsgVarBuf+(VisCol-CurAtr)
    CLR    A
    MOVX   @DPTR,A
    MOV    A,#028H
    MOV    DPTR,#BgdMDB0+MDB_Tslc
    MOVX   @DPTR,A
    MOV    DPTR,#BgdMDB1+MDB_Tslc
    MOVX   @DPTR,A
    MOV    DPTR,#NrmRRB+RRB_Tslc_NcsHi
    MOVX   @DPTR,A
    INC    DPL
    MOV    A,#04AH
    MOVX   @DPTR,A
    INC    DPL
```

```
    CLR    A
    MOVX   @DPTR,A
    INC    DPL
    MOV    A,#00AH
    MOVX   @DPTR,A
    INC    DPL
    CLR    A
    MOVX   @DPTR,A
    INC    DPL
    MOV    A,#08AH
    MOVX   @DPTR,A
    INC    DPL
    INC    DPL
    INC    DPL
    MOV    A,#001H
    MOVX   @DPTR,A
    INC    DPL
    MOV    A,#045H
    MOVX   @DPTR,A
    JNB    MsgVisFlg,SMd9
    MOV    A,#32
    SJMP   SMd10
SMd9:
    MOV    A,#30
SMd10:
    MOV    DPTR,#TrmWDB+WDB_BgnRow
    MOVX   @DPTR,A
    INC    DPL
    MOVX   @DPTR,A
    MOV    A,#30
    MOV    DPTR,#MsgWDB+WDB_BgnRow
    MOVX   @DPTR,A
    INC    DPL
    MOVX   @DPTR,A
    SETB   EX0
    JNB    MsgActFlg,SMd11
    MOV    RowAdd,#30
SMd11:
    MOV    A,CsrSiz
    CJNE   A,#0BCH,SMd5
    MOV    CsrSiz,#09AH
    SJMP   SMd8
```

```
SMd5:
   CJNE  A,#0CCH,SMd6
   MOV   CsrSiz,#0AAH
   SJMP  SMd8
SMd6:
   CJNE  A,#06AH,SMd7
   MOV   CsrSiz,#058H
   SJMP  SMd8
SMd7:
   MOV   CsrSiz,#00AH
SMd8:
   LCALL ChgCsrSiz
   MOV   R1,#ModReg1Ind
   MOV   R2,#0C8H
   MOV   R3,#001H
   LCALL WrAm8052Reg
   LCALL PlcCsr
   MOV   R0,HrzFrmSet
   MOV   R6,HrzPxlShf
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrollRight:
;.............................................................................
;  Scrolls the display rightward the number of columns specified by the single
;  numeric parameter. An attempt to scroll the leftmost column of the display
;  rightward beyond the leftmost column on the monitor leaves it at the left-
;  most column.
;
;  inp   PrmCnt                         count of parameters
;        PrmBuf                         buffer containing the parameters
;  out   none
;  bad   A
;.............................................................................

   JB    PrmBadFlg,SR5                  ; Indicates a bad param buffer
                                        ;    error return
   CLR   C
   JB    WndActFlg,SR6                  ;Window cannot scroll horz.

   MOV   A,PrmCnt                       ;Zero Parameters default to
   JNZ   SR1
```

```
   MOV   PrmBuf,#1                      ;one row
   SJMP  SR2
SR1:
   DEC   A                              ;If more then one parameter this
   JNZ   SR5                            ;    is an error return
SR2:
   MOV   A,PrmBuf                       ;Amount scrolled is equal to the
   SUBB  A,VisCol                       ;    small of requested columns
   JC    SR3                            ;    Vs.Vis.Col
   MOV   A,VisCol
   JZ    SR6
   MOV   R7,A
   SJMP  SR4
SR3:
   MOV   R7,PrmBuf
SR4:
   MOV   A,R7
   LCALL ScrlRtDsp                      ;Scroll in Progress
   RET
SR5:
   CLR   A                              ;If error remove all traces of
   MOV   CtlPtrHi,A                     ;    of control
   MOV   CtlPtrLo,A
SR6:
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrollUp:
;.............................................................................
;  Scrolls the display upward the number of columns specified by the single
;  numeric parameter. An attempt to scroll the bottom row of the display
;  upward beyond the bottom row on the monitor leaves it at the bottom of the
;  display.
;
;  inp   PrmCnt                         count of parameters
;        PrmBuf                         buffer containing the parameters
;  out   none
;  bad   A,R1,R2
;.............................................................................

   JB    PrmBadFlg,SU6                  ; Indicates a bad param buffer
                                        ;    error return
```

```
    JB    MsgActFlg,SU7
    MOV   A,PrmCnt                 ;If message active vert scroll
    JNZ   SU1                      ;    not allowed
    MOV   PrmBuf,#1                ;If no parameter, default the
    SJMP  SU01                     ;    one row
SU1:
    DEC   A                        ;If more than one parameter this
    JNZ   SU6                      ;    is an error return
SU01:
    JB    WndActFlg,SU2            ;If window is active maximum
                                   ;    scroll value is 7
    MOV   R2,#6                    ;    or background max is 6
    JB    AMDDWMBit,SU7            ;If in compressed mode scroll is
    SJMP  SU3                      ;    not allowed
SU2:
    MOV   R2,#7
SU3:
    CLR   C
    MOV   A,VisRow                 ;The current allowed is maximum
    XCH   A,R2                     ;    _VisCut
    SUBB  A,R2
    JZ    SU7
    MOV   R1,A                     ;    save max to move for later
    MOV   A,PrmBuf
    CLR   C                        ;Request amount to scroll
    SUBB  A,R1                     ;Move either requested amount or
    JC    SU4                      ;    maximum allowed
    MOV   A,R1
    SJMP  SU5
SU4:
    MOV   A,PrmBuf                 ;If requested is less then
                                   ;    allowed do that many

SU5:
    LCALL ScrlUpDsp                ; Scroll in progress
    SJMP  SU7                      ;    we're done
SU6:
    CLR   A                        ;If an error clear history ptr
    MOV   CtlPtrHi,A
    MOV   CtlPtrLo,A
SU7:
    RET                            ;Done
```

```
CharBlinkRate:

;..............................................................................
; Selects the rate and duty cycle for characters dispalyed with the blink
; attribute
;
; Parameters                                Meaning
; ----------                                -------
; 0                                         Initial blink, fastest,25/75 cycle
; 11                                        Blink 50/50 cycle
; 12                                        Blink 25/75 cycle
; 20                                        Fastest blink rate
; 21                                        Fast blink rate
; 22                                        Slow blink rate
; 23                                        Slowest blink rate
;
; inp   PrmCnt                                  count of parameters
;       PrmBuf                                  buffer containing the parameters
; out   none
; bad   A,R1,R2
;..............................................................................

    JB    PrmBadFlg,CBR9                        ; Indicates a bad param buffer
                                                ;    error return
    JB    PrmMaxFlg,CBR9                        ; Indicates too many parameters
                                                ;    error return
    MOV   R1,#PrmBuf
    MOV   A,PrmCnt
    JNZ   CBR0
    MOV   @R1,A
    INC   A
CBR0:
    MOV   R2,A
CBR1:
    CJNE  @R1,#00,CBR2                          ; initial type
    SETB  ChdBit
    CLR   ChbBit1
    CLR   ChbBit0
    SJMP  CBR8
CBR2:
    CJNE  @R1,#11,CBR3                          ; Blink 50/50
    SETB  ChdBit
    SJMP  CBR8
```

```
CBR3:
   CJNE  @R1,#12,CBR4                        ; Blink 25/75
   CLR   ChdBit
   SJMP  CBR8
CBR4:
   CJNE  @R1,#20,CBR5                        ; Fastest rate
   CLR   ChbBit1
   CLR   ChbBit0
   SJMP  CBR8
CBR5:
   CJNE  @R1,#21,CBR6                        ; Fast rate
   CLR   ChbBit1
   SETB  ChbBit0
   SJMP  CBR8
CBR6:
   CJNE  @R1,#22,CBR7                        ; Slow rate
   SETB  ChbBit1
   CLR   ChbBit0
   SJMP  CBR8
CBR7:
   CJNE  @R1,#23,CBR8                        ; Slowest rate
   SETB  ChbBit1
   SETB  ChbBit0
CBR8:
   INC   R1
   DJNZ  R2,CBR1
   LCALL ChgBlnkSpd
   SJMP  CBR10
CBR9:
   CLR   A                                   ; If an error is detected remove
   MOV   CtlPtrHi,A                          ;    all traces of this control
   MOV   CtlPtrLo,A
CBR10:
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
LoadFontCell:
;............................................................................
; Loads a cell of the character generator RAM at the location, and with the
; pattern, specified in the parameters.  The first parameter is the cell
; address (0-255), the second is the starting slice (counting downward from
; zero) and the remaining parameters are the patterns for each slice working
```

```
; downward.  Unspecified slices are loaded with zeroes.  After checking fo
; parameter validity this routine waits until all smooth scrolling is finished
; before beginning its work.  The Display Width Mode (AMDDWM) determines which
; type of font (normal or compressed) is to be loaded.
;............................................................................

   JB    PrmBadFlg,LFC5                          ; Indicates a bad param buffer
                                                 ;    error return
   MOV   A,PrmCnt
   JNZ   LFC1
   MOV   PrmBuf,#0
   SJMP  LFC2
LFC1:
   DEC   A
   JNZ   LFC3
LFC2:
   MOV   PrmBuf+1,#0
   MOV   PrmCnt,#2
   SJMP  LFC4
LFC3:
   CLR   C
   SUBB  A,#17
   JNC   LFC5
LFC4:
   CJNE  A,#' ',LFC6
LFC5:
   CLR   A
   MOV   CtlPtrHi,A
   MOV   CtlPtrLo,A
   RET
LFC6:
   MOV   A,PrmBuf+1
   ADD   A,PrmCnt
   CLR   C
   SUBB  A,#18
   JNC   LFC5
   LCALL HidCsr
   JB    VrtScrlFlg,$
   JB    HrzScrlFlg,$
   CLR   A
   JNB   AMDDWMBit,LFC7
   INC   A
```

```
LFC7:
   LCALL WrFntCel
   LCALL ShwCsr
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SelActiveDisp:
;..............................................................................
;  Selects the currently active display, background, window, or message.
;  attribute
;
;  Parameters                              Meaning
;  ----------                              -------
;  0                                       makes the background display active
;  1                                       makes the message display active
;  2                                       makes the window display active
;
;  inp   PrmCnt                                 count of parameters
;        PrmBuf                                 buffer containing the parameters
;  out   none
;  bad   A,R1,R2,R4,R5
;..............................................................................

   JNB   PrmBadFlg,SAD1                     ; Indicates a bad param buffer
   LJMP  SAD19                              ;     error return
SAD1:
   JNB   PrmMaxFlg,SAD2                    ; Indicates too many parameters
   LJMP  SAD19                              ;     error return
SAD2:
   JB    VrtScrlFlg,$
   JB    HrzScrlFlg,$
   MOV   R1,#PrmBuf
   MOV   A,PrmCnt
   JNZ   SAD2a
   MOV   @R1,A
   INC   A
SAD2a:
   MOV   R2,A
SAD3:
   CJNE  @R1,#00H,SAD3a
   SJMP  SAD3b
SAD3a:
   LJMP  SAD9
```

```
SAD3b:
   MOV   R5,#BgdVarBuf.SR.PAGE                    ; Make the background display
   MOV   R6,#BgdVarBuf.AN.OFST                    ;    active
   JB    MsgActFlg,SAD5
   JB   WndActFlg,SAD4                            ; If Background is already active
                                                  ;    do nothing further
   LJMP  SAD18
SAD4:
   MOV   R3,#WndVarBuf.SR.PAGE                    ; If the wnd window was active
   MOV   R4,#WndVarBuf.AN.OFST                    ;    move its dsp. vars. out
   LCALL BldTrmRcb
   SJMP  SAD6
SAD5:
   MOV   R3,#MsgVarBuf.SR.PAGE                    ; If the msg window was active
   MOV   R4,#MsgVarBuf.AN.OFST                    ;    move its dsp. vars. out
SAD6:
   LCALL SwpVar
   CLR   WndActFlg                                ; Indicate current active state
   CLR   MsgActFlg                                ;    with internal flags
   JB    AMDDWMBit,SAD7
   MOV   DspWid,#80                               ; Update non-moving display vars
   MOV   DspHgt,#24
   SJMP  SAD8
SAD7:
   MOV   DspWid,#120
   MOV   DspHgt,#30
SAD8:
   MOV   ColAdd,#01
   MOV   RowAdd,#00
   MOV   RcbOff,#BgdRCB0.AN.OFST
   MOV   ChrOff,#BgdChrBuf0.AN.OFST
   MOV   AtrOff,#BgdAtrBuf0.AN.OFST
   MOV   P2,ExtRow                                ; Set page address to extra row
   MOV   A,RcbOff                                 ; Build offset into RCB at
   ADD   A,#RCB_RowPag                            ;    next row pointer
   MOV   R0,A                                     ; Use R0 as index pointer
   MOV   A,ExtRow                                 ; Next row pointer = ExtRow
   MOVX  @R0,A                                    ; Store it in RCB
   INC   R0                                       ; Get index to offset
   MOV   A,RcbOff                                 ; Move current rcb offset
   MOVX  @R0,A                                    ; Store it
```

```
        INC   R0                          ; Set hidden count
        CLR   A
        MOVX  @R0,A                       ; Store hidden count
        INC   R0                          ; Index to visible count
        INC   A                           ; Set visible count to 1
        MOVX  @R0,A                       ; Store it
        INC   R0
        MOV   A,#080H                     ; Continue bit set
        MOVX  @R0,A
        INC   R0
        CLR   A
        MOVX  @R0,A                       ; Store always zero byte
        INC   R0                          ; Index to chr ptr page
        MOV   A,#BgdFncChr0.SR.PAGE       ; Set to current function char
        MOVX  @R0,A                       ; Store it in RCB
        INC   R0                          ; Index to chr ptr offset
        MOV   A,#BgdFncChr0.AN.OFST       ; Offset of function character
        MOVX  @R0,A                       ; Stored
        INC   R0                          ;
        CLR   A
        MOVX  @R0,A                       ; Store empty word in RCB
        INC   R0
        MOVX  @R0,A
        INC   R0                          ; Index to atr page
        MOV   A,#BgdFncAtr0.SR.PAGE       ; Build Attribute page
        MOVX  @R0,A                       ; Store page to atr
        INC   R0                          ; Index to atr offset
        MOV   A,#BgdFncAtr0.AN.OFST
        MOVX  @R0,A                       ; Store it
        INC   R0                          ;
        MOV   A,VisCol                    ; Length of hidden 2nd seg=VisCol
        MOVX  @R0,A                       ; Store it
        INC   R0                          ; Index to visible 2nd segment
        SETB  C
        SUBB  A,WndCol                    ; Visible count = WndCol-VisCol
        CPL   A                           ; We get negative so complement
        MOVX  @R0,A                       ; Store it
        INC   R0                          ; Index to continue bit
        MOV   A,#80H                      ; Set continue bit 0 rest of byte
        MOVX  @R0,A                       ; Store it
        INC   R0                          ; and 1 empty byte
```

```
        CLR   A
        MOVX  @R0,A                                  ; Extra RCB is now rebuilt
        LJMP  SAD18

SAD9:
        CJNE  @R1,#02,SAD13
        MOV   R5,#WndVarBuf.SR.PAGE                  ; Make the wnd window active
        MOV   R6,#WndVarBuf.AN.OFST
        JB    MsgActFlg,SAD11
        JNB    WndActFlg,SAD10                       ; If wnd window is already active
                                                     ;    do nothing further
        LJMP  SAD18
SAD10:
        MOV   R3,#BgdVarBuf.SR.PAGE                  ; If the background was active
        MOV   R4,#BgdVarBuf.AN.OFST                  ;    move its dsp. vars. out
        SJMP  SAD12
SAD11:
        MOV   R3,#MsgVarBuf.SR.PAGE                  ; If the msg window was active
        MOV   R4,#MsgVarBuf.AN.OFST                  ;    move its dsp. vars. out
SAD12:
        LCALL BldTrmRcb
        MOV   DspWid,#40                             ; Update non-moving display vars
        MOV   DspHgt,#7
        MOV   A,WndCol
        SUBB  A,VisCol
        INC   A
        MOV   ColAdd,A
        MOV   RowAdd,#6
        MOV   RcbOff,#WndRCB0.AN.OFST
        MOV   ChrOff,#WndChrBuf0.AN.OFST
        MOV   AtrOff,#WndAtrBuf0.AN.OFST
        LCALL SwpVar
        CLR   MsgActFlg                              ; Indicate current active state
        SETB  WndActFlg                              ;    by internal flags
        MOV   P2,ExtRow                              ; Set page address to extra row
        MOV   A,RcbOff                               ; Build offset into RCB at
        ADD   A,#RCB_RowPag                          ;    next row pointer
        MOV   R0,A                                   ; Use R0 as index pointer
        MOV   A,ExtRow                               ; Next row pointer = ExtRow
        MOVX  @R0,A                                  ; Store it in RCB
        INC   R0                                     ; Get index to offset
        MOV   A,RcbOff                               ; Move current rcb offset
```

```
    MOVX  @R0,A                     ; Store it
    INC   R0                        ; Set hidden count
    CLR   A
    MOVX  @R0,A                     ; Store hidden count
    INC   R0                        ; Index to visible count
    MOV   A,#40                     ; Visible count is wnd width
    MOVX  @R0,A                     ; Store it
    INC   R0
    CLR   A                         ; No continue bit
    MOVX  @R0,A
    INC   R0
    CLR   A
    MOVX  @R0,A                     ; Store always zero byte
    INC   R0                        ; Index to chr ptr page
    MOV   A,ExtRow                  ; Set to char buffer
    MOVX  @R0,A                     ; Store it in RCB
    INC   R0                        ; Index to chr ptr offset
    MOV   A,#WndChrBuf0.AN.OFST     ; Offset of character buffer
    MOVX  @R0,A                     ; Stored
    INC   R0                        ;
    CLR   A
    MOVX  @R0,A                     ; Store empty word in RCB
    INC   R0
    MOVX  @R0,A
    INC   R0                        ; Index to atr page
    MOV   A,ExtRow                  ; Build Attribute page
    SETB  ACC.4
    MOVX  @R0,A                     ; Store page to atr
    INC   R0                        ; Index to atr offset
    MOV   A,#WndAtrBuf0.AN.OFST
    MOVX  @R0,A                     ; Store it
    INC   R0                        ;
    CLR   A
    MOVX  @R0,A                     ; Store empty word in RCB
    INC   R0
    MOVX  @R0,A
    INC   R0                        ; Index to atr page
    MOV   A,#NrmRRB.SR.PAGE         ; Page of normal RRB
    MOVX  @R0,A                     ; Store it
    INC   R0
    MOV   A,#NrmRRB.AN.OFST         ; Offset of normal RRB
    MOVX  @R0,A                     ; Extra RCB is now rebuilt
    SJMP  SAD18
```

```
SAD13:
    CJNE  @R1,#01H,SAD18
    MOV   R5,#MsgVarBuf.SR.PAGE          ; Make the Msg window active
    MOV   R6,#MsgVarBuf.AN.OFST
    JB    WndActFlg,SAD14
    JB    MsgActFlg,SAD18                ; If Msg window already active
                                         ;    do nothing further
    MOV   R3,#BgdVarBuf.SR.PAGE          ; If background was active
    MOV   R4,#BgdVarBuf.AN.OFST          ;    move its dsp. vars. out
    SJMP  SAD15
SAD14:
    MOV   R3,#WndVarBuf.SR.PAGE          ; If Wnd window was active
    MOV   R4,#WndVarBuf.AN.OFST          ;    move its dsp. vars. out
    LCALL BldTrmRcb
SAD15:                                   ;    will be updated
    JB    AMDDWMBit,SAD16
    MOV   DspWid,#80                     ; Update non-moving display vars
    SJMP  SAD17
SAD16:
    MOV   DspWid,#120
SAD17:
    MOV   DspHgt,#01
    MOV   ColAdd,#01
    JNB   AMDDWMBit,SAD17a
    MOV   RowAdd,#30
    SJMP  SAD17b
SAD17a:
    MOV   RowAdd,#24
SAD17b:
    MOV   RcbOff,#BgdRCB0.AN.OFST
    MOV   ChrOff,#BgdChrBuf0.AN.OFST
    MOV   AtrOff,#BgdAtrBuf0.AN.OFST
    LCALL SwpVar
    CLR   WndActFlg                      ; Indicate current active state
    SETB  MsgActFlg                      ;    with internal flags
SAD18:
    INC   R1                             ; Test if we are at the end
    DEC   R2                             ;    of our parameters
    MOV   A,R2
    JZ    SAD20                          ; If true get out
    LJMP  SAD3                           ; Else proceed with the next
```

```
SAD19:
   CLR   A                                 ; If an error was detected
   MOV   CtlPtrHi,A                        ;    remove all traces of this
   MOV   CtlPtrLo,A                        ;    control
SAD20:
   LCALL PlcCsr                            ; Relocate our cursor
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SelCursorAppear:
;..............................................................................
;  Selects the type and appearance of the cursor.
;
;
;  Parameters                        Meaning
;  ----------                        -------
;  0                                 Steady reversed full block, initial
;  1                                 Reversed full block
;  2                                 Reversed block half of character cell
;  3                                 Solid block half character cell
;  4                                 Underscore
;  5                                 Thick underscore
;  10                                Steady, non-blinking
;  11                                Blink 50/50 cycle
;  12                                Blink 25/74 cycle
;  20                                Fastest blink
;  21                                Fast blink
;  22                                Slow blink
;  23                                Slowest blink
;
;  inp   PrmCnt                            count of parameters
;        PrmBuf                            buffer containing the parameters
;  out   none
;  bad   A,R0,R2,R4,R5,R6
;..............................................................................

   JNB   PrmBadFlg,SCA1                    ; Indicates a bad param buffer
   LJMP  SCA20                             ;    error return

SCA1:
   JNB   PrmMaxFlg,SCA2                    ; Indicates too many parameters
   LJMP  SCA20                             ;    error return
```

```
SCA2:
   MOV   R0,#PrmBuf
   MOV   A,PrmCnt
   JNZ   SCA2a
   MOV   @R0,A
   INC   A
SCA2a:
   MOV   R6,A
SCA3:
   CJNE  @R0,#00,SCA4                      ; Initial cursor
   CLR   CxybeBit
   LCALL ChgBlnkSpd
   MOV   R5,#06H
   MOV   CsrSiz,#00DH
   JNB   AMDDWMBit,SCA9
   MOV   CsrSiz,#00AH
   SJMP  SCA9
SCA4:
   CJNE  @R0,#01,SCA5                      ; Reversed full block
   MOV   R5,#006H
   MOV   CsrSiz,#00DH
   JNB   AMDDWMBit,SCA9
   MOV   CsrSiz,#00AH
   SJMP  SCA9
SCA5:
   CJNE  @R0,#02,SCA6                      ; Reversed half block
   MOV   R5,#06H
   MOV   CsrSiz,#06AH
   JNB   AMDDWMBit,SCA9
   MOV   CsrSiz,#058H
   SJMP  SCA9
SCA6:
   CJNE  @R0,#03,SCA7                      ; Solid half block
   MOV   R5,#04H
   MOV   CsrSiz,#06AH
   JNB   AMDDWMBit,SCA9
   MOV   CsrSiz,#058H
   SJMP  SCA9
SCA7:
   CJNE  @R0,#04,SCA8                      ; Underscore
   MOV   R5,#04H
   MOV   CsrSiz,#0CCH
   JNB   AMDDWMBit,SCA9
```

```
    MOV   CsrSiz,#0AAH
    SJMP  SCA9

SCA8:
    CJNE  @R0,#05,SCA10                  ; Thick underscore
    MOV   R5,#04H
    MOV   CsrSiz,#0BCH
    JNB   AMDDWMBit,SCA9
    MOV   CsrSiz,#09AH
SCA9:
    LCALL ChgCsrSiz
    LCALL ChgCsrTyp
    SJMP  SCA18
SCA10:
    CJNE  @R0,#10,SCA11                  ; Steady non-blinking
    CLR   CxybeBit
    SJMP  SCA17
SCA11:
    CJNE  @R0,#11,SCA12                  ; Blink 50/50 cycle
    SETB  CxybeBit
    SETB  CudBit
    SJMP  SCA17
SCA12:
    CJNE  @R0,#12,SCA13                  ; Blink 25/75 cycle
    SETB  CxybeBit
    CLR   CudBit
    SJMP  SCA17
SCA13:
    CJNE  @R0,#20,SCA14                  ; Fastest rate
    CLR   CubBit1
    CLR   CubBit0
    SETB  CxybeBit
    SJMP  SCA17
SCA14:
    CJNE  @R0,#21,SCA15                  ; Fast rate
    CLR   CubBit1
    SETB  CubBit0
    SETB  CxybeBit
    SJMP  SCA17
SCA15:
    CJNE  @R0,#22,SCA16                  ; Slow rate
    SETB  CubBit1
```

```
    CLR   CubBit0
    SETB  CxybeBit
    SJMP  SCA17
SCA16:
    CJNE  @R0,#23,SCA18                  ; Slowest rate
    SETB  CubBit1
    SETB  CubBit0
    SETB  CxybeBit
SCA17:
    LCALL ChgBlnkSpd
SCA18:
    INC   R0
    DEC   R6
    MOV   A,R6
    JZ    SCA21
    LJMP  SCA3
SCA20:
    CLR   A
    MOV   CtlPtrHi,A
    MOV   CtlPtrLo,A
SCA21:
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SmoothScrlRate:
;-----------------------------------------------------------------------------
;  Selects the rate at which smooth scrolling occurs.
;
;
;  Parameters                          Meaning
;  ----------                          -------
;  0                                   1 scan line / pixel / frame
;  1                                   1 scan line / pixel / frame
;  2                                   2 scan line / pixel / frame
;  3                                   3 scan line / pixel / frame
;  4                                   4 scan line / pixel / frame
;  5                                   5 scan line / pixel / frame
;  6                                   6 scan line / pixel / frame
;  7                                   7 scan line / pixel / frame
;  8                                   8 scan line / pixel / frame
;  12                                  1 scan line / pixel / 2 frames
;  13                                  1 scan line / pixel / 3 frames
;  14                                   1 scan line / pixel / 4 frames
;  15                                   1 scan line / pixel / 5 frames
;  16                                   1 scan line / pixel / 6 frames
;  17                                   1 scan line / pixel / 7 frames
;  18                                   1 scan line / pixel / 8 frames
;
;  inp   PrmCnt                             count of parameters
;        PrmBuf                             buffer containing the parameters
;  out   none
;  bad   A,R1,R2,R3
;.............................................................................

   JB    PrmBadFlg,SSR6                    ; Indicates a bad param buffer
                                           ;    error return
   JB    PrmMaxFlg,SSR6                    ; Indicates too many parameters
                                           ;    error return
   MOV   R1,#PrmBuf
   MOV   A,PrmCnt
   JNZ   SSR0
   MOV   @R1,A
   INC   A
SSR0:
   MOV   R2,A
SSR1:
   MOV   A,@R1
   JZ    SSR4
   MOV   R3,A                              ; work on current parameter
   CLR   C
   SUBB  A,#09                             ; test if in first group of
   JNC   SSR3                              ;    parameters
   MOV   A,R3                              ; If true adjust for calculation
   DEC   A
   SJMP  SSR4
SSR3:
   MOV   A,R3                              ; Test if between groups 9-11
   CLR   C
   SUBB  A,#12
   JC    SSR5                              ; If true exit
   MOV   A,R3                              ; else adjust for calculation
   SUBB  A,#3
SSR4:
   SWAP  A                                 ; work on hi nibble first
   RR    A                                 ; isolate hi byte
```

```
   MOV   R3,A                                 ; Store for future use
   ANL   A,#.NT.SCRL_RAT_MASK                 ; Mask off unused bits
   JNZ   SSR5
   MOV   A,ScrlByt                            ; bring in scroll byte
   ANL   A,#.NT.SCRL_RAT_MASK                 ; Mask balance of byte to write'
   ORL   A,R3                                 ; generate combine Scrlbyt parts
   MOV   ScrlByt,A                            ; return new value
SSR5:
   INC   R1
   DJNZ  R2,SSR1                              ; Continue until last parameter
   SJMP  SSR7
SSR6:
   CLR   A                                    ; If an error was indicated
   MOV   CtlPtrHi,A                           ;    remove all traces of
   MOV   CtlPtrLo,A                           ;    control
SSR7:
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SelWindowVis:
;.............................................................................
;  Selects window visibliity.
;
;
;  Parameters                            Meaning
;  ----------                            -------
;  0                                     make window invisible
;  1                                     make window visible
;
;  inp   PrmCnt                                  count of parameters
;        PrmBuf                                  buffer containing the parameters
;  out   none
;  bad   R1,R4
;.............................................................................

   JB    PrmBadFlg,SWV8                          ; Indicates a bad param buffer
                                                 ;    error return
   JB    PrmMaxFlg,SWV8                          ; Indicates too many parameters
                                                 ;    error return
   JB    VrtScrlFlg,$
   JB    HrzScrlFlg,$
   MOV   R0,#PrmBuf
   MOV   A,PrmCnt
   JNZ   SWV0
   MOV   @R0,A
   INC   A
SWV0:
   MOV   R4,A
SWV1:
   CJNE  @R0,#00,SWV3              ; Make window invisible if not taken
   JNB   WndVisFlg,SWV7
   LCALL HidWnd
   CLR   WndVisFlg
   SJMP  SWV7
SWV3:
   CJNE  @R0,#01,SWV7                     ; Make window visible
   JB    WndVisFlg,SWV7
   LCALL ShwWnd
   SETB  WndVisFlg
SWV7:
   INC   R0
   DJNZ  R4,SWV1
   LCALL PlcCsr
   RET
SWV8:
   CLR   A                                ; if an error was indicated
   MOV   CtlPtrHi,A                       ;    remove all traces of
   MOV   CtlPtrLo,A                       ;    this control
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SelMessageVis:
;.............................................................................
; Selects message window visibliity.
;
;
; Parameters                    Meaning
; ----------                    -------
; 0                             make message window invisible
; 1                             make message window visible
;
; inp   PrmCnt                         count of parameters
;       PrmBuf                         buffer containing the parameters
; out   none
; bad   A,R1,R2,R3,R4
```

```
   JB     PrmBadFlg,SMV11                  ; Indicates a bad param buffer
   JB     PrmMaxFlg,SMV11                  ; Indicates too many parameters
                                           ;    error return
   JB    VrtScrlFlg,$
   JB    HrzScrlFlg,$
   MOV   R1,#PrmBuf
   MOV   A,PrmCnt
   JNZ   SMV1
   MOV   @R1,A
   INC   A
SMV1:
   MOV   R4,A
SMV2:
   CLR   EX0
   CJNE  @R1,#00,SMV5                      ; Make message window invisible
                                           ;    if not taken
   JNB   MsgVisFlg,SMV10
   MOV   DPTR,#TrmWDB+WDB_BgnRow
   JB    AMDDWMBit,SMV3                    ; Adjust Termination start and
   MOV   A,#24                             ;    end row count
   SJMP  SMV4
SMV3:
   MOV   A,#30                             ; if compressed mode
SMV4:
   MOVX  @DPTR,A
   INC   DPTR
   MOVX  @DPTR,A
   CLR   MsgVisFlg
   MOV   R2,#TrmWDB.SR.PAGE                ; Termination Def. Blk. Ptr
   JB    WndVisFlg,SMV9                    ; Window is visible if taken
   MOV   R3,#TrmWDB.AN.OFST
   SJMP  SMV8
SMV5:
   CJNE  @R1,#01,SMV10                     ; Make message window visible
   JB    MsgVisFlg,SMV10                   ; If msg window is already
                                           ;    showing just return
   MOV   DPTR,#TrmWDB+WDB_BgnRow
   JB    AMDDWMBit,SMV6
   MOV   A,#26                             ; In both normal and compressed
   SJMP  SMV7                              ;    mode rows are just after
SMV6:
   MOV   A,#32                             ; Msg row in display
```

```
SMV7:
   MOVX  @DPTR,A
   INC   DPTR
   MOVX  @DPTR,A
   SETB  MsgVisFlg
   MOV   R2,#MsgWDB.SR.PAGE
   JB    WndVisFlg,SMV9
   MOV   R3,#MsgWDB.AN.OFST
SMV8:
   MOV   R1,#TOWHrdLoInd                    ; Write new TOWHrdLo Ptr
   LCALL WrAm8052Reg
   SJMP  SMV10
SMV9:
   MOV   DPTR,#WndWDB0+WDB_NxtPag
   MOV   A,R2
   MOVX  @DPTR,A
   INC   DPH
   MOVX  @DPTR,A
SMV10:
   SETB  EX0
   INC   R1
   DJNZ  R4,SMV2
   LCALL PlcCsr
   RET
SMV11:
   CLR   A                                  ; If an error was indicated
   MOV   CtlPtrHi,A                         ;    remove all traces of this
   MOV   CtlPtrLo,A                         ;    control
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; end of C_Work
```

```
"8051"
  TITLE "     CALEB 0.00     System Utilities"
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; C_Util                          CALEB 0.00
;
;       Copyright 1985 Advanced Micro Devices, Inc.
;
;
; This file contains the various system utilities used by the control routines.

  NAME  "System Utilities"
  PROG

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  GLB   DlyTilEndFrm,PlcCsr,EraActEnd,EraBgnAct,ChgBlnkSpd,SwpVar,ChgCsrSiz
  GLB   SetCelWid,ChgCsrTyp
  GLB   DelRow_MovDn,DelRow_MovUp,HidCsr,ShwCsr
  GLB   InsRow_MovDn,InsRow_MovUp
  GLB   WrAm8052Reg,RdAm8052Reg,WrFntCel
  GLB   EraRow,ScrlUpDsp,ScrlDnDsp,ScrlUpNewRow,ScrlRtDsp,ScrlLtDsp
  GLB   SetForScrlDn,SetForScrlUp,ScrlLtOne,ScrlRtOne,FrcEraRow
  GLB   SetAftScrlDn
  GLB   SetWndPos,NewCsr,HalfSwap,BldTrmRcb,HidWnd,ShwWnd
;------------------------------------------------------------------------------
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  SKIP
  INCLUDE C_MemMap
  SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

DlyTilEndFrm:                          ; Delay until end-of-frame time starts

; Ensures two character row times of nearly unimpeded processing time.  This
; routine works with the timer 0 interrupt to wait until near the end of the
; frame (28 scan lines from the bottom).  During this end-of-frame time the
; Am8052 is displaying information it has already fetched and needs the bus
; only twice, each time to fetch only the termination row control block with
; its single character and single latched attribute.  Changes accomplished
; during this time will not be visible until the next frame starts (at blank
; time at the bottom of the screen).  Thus, there will be no distracting
; interference with the Am8052.
```

```
; In:    none
; Out:   none

   JB    EndFrmFlg,$                     ; Ensure we're in middle of frame
   JNB   EndFrmFlg,$                     ; Wait for end-of-frame interrupt
   RET                                   ; Exit

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
FndCsrZon:
;..............................................................................
; Determines the type of zone (visible or invisible) containing the active
; position.  It also calculates the number of columns to the first column
; of the next zone to the right.  This value is used to speed advancing the
; cursor following a simple character input.
;
; In:    ActCol          active position's column within display
;        ActRow          active position's row within display
; Out:   CsrZonFlg       set if cursor is visible, cleared if invisible
;        CsrZonCnt       distance to next zone rightward
; Bad:   A,R0,PSW
;..............................................................................

   JB    MsgActFlg,FCZ3                  ;skip if in message
   CLR   C
   MOV   A,ActRow                        ;A = # rows down from top
   SUBB  A,VisRow                        ;    of screen
   JC    FCZ5                            ;skip if "above" top of screen
   JB    WndActFlg,FCZ2                  ;skip if in window
   JNB   WndVisFlg,FCZ2                  ;skip if window not visible
                                 ;in background, window visible
   MOV   R0,A                            ;R0 = # screen row
   SUBB  A,#WND_TOP_MRG                  ;A = # rows down in window
   JC    FCZ3                            ;skip if above top of window
   SUBB  A,#WND_VIS_HGT                  ;A = # rows down below window
   JNC   FCZ1                            ;skip if below window
                                 ;in background, in window row range
   CLR   C
   MOV   R0,ActCol                       ;R0 = current column
   MOV   A,R0                            ;A = # columns right of visible
   SUBB  A,VisCol                        ;    left side
   JC    FCZ8                            ;done if left of screen
```

```
    XCH    A,R0                        ;hold visible col in R0
    SUBB   A,WndCol                    ;A = # cols into window
    JC     FCZ9                        ;done if left of window
    SUBB   A,#WND_VIS_WID              ;A = # cols right of window
    JC     FCZ8                        ;done if beneath window
    MOV    A,R0                        ;A = visible column
    SJMP   FCZ4                        ;skip to check vs screen right
FCZ1:                            ;reset A for linkage
    MOV    A,R0                        ;A = screen row
FCZ2:                            ;check if beneath screen
    SUBB   A,DspHgt                    ;A = # rows beneath screen
    JNC    FCZ6                        ;skip if beneath screen
                                 ;row is visible background or message
FCZ3:                            ;check if left of screen
    CLR    C
    MOV    A,ActCol                    ;A = visible column
    SUBB   A,VisCol
    JC     FCZ8                        ;done if left of screen
FCZ4:                            ;check if right of screen
    SUBB   A,DspWid                    ;A = # cols right of screen
    JC     FCZ9                        ;done if visible on screen
FCZ5:                            ;not in visible screen row
    CLR    C                           ;buffer widths bound zones
FCZ6:
    MOV    A,ActCol                    ;A = current column
    JNB    WndActFlg,FCZ7              ;skip if window is not active
                                 ;window is active
    SUBB   A,#WND_BUF_WID              ;zone extends to window end
    SJMP   FCZ8
FCZ7:                            ;window is not active
    SUBB   A,#BGD_BUF_WID              ;zone extends to buffer end
FCZ8:
    CLR    CsrZonFlg                   ;cursor is not visible
    SJMP   FCZ12
FCZ9:
    JNB    MsgActFlg,FCZ10             ;if msg is active check if
    JNB    MsgVisFlg,FCZ8              ;    visible, adjust CsrZonFlg
                                       ;    accordingly
FCZ10:
    JNB    WndActFlg,FCZ11             ; do the same for the window
    JNB    WndVisFlg,FCZ8
```

```
FCZ11:
    SETB   CsrZonFlg                        ;cursor is visible
FCZ12:
    CPL    A                                ;-A is zone remaining count
    INC    A
    MOV    CsrZonCnt,A
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
NewCsr:
;.............................................................................
; Assigns the new active position from the given location and updates the
; current row page address.
;
; In:    R2                          new active row position
;        R3                          new active column position
; Out:   ActCol                      active column position
;        ActRow                      active row position
;        CurRow                      active row page address
; Bad:   A,P2,R0,R1,PSW
;
; NOTE:  This routine must be located immediately before "PlcCsr".
;.............................................................................

    CLR    C                           ; Ready for comparison below
    MOV    A,R2                        ; Check new active row
    JNZ    NC1                         ; Jump if not at first row
    MOV    A,BgnRow                    ; Get page address of first row
    SJMP   NC7                         ;    and go assign new position
NC1:                                 ; Determine direction of movement
    SUBB   A,ActRow                    ; Compare new row to old
    JZ     NC8                         ; Jump if they are the same
    JNC    NC2                         ; Jump if new is below old
    MOV    P2,BgnRow                   ; Start at first row if new is
    MOV    A,R2                        ;    above old and count down to
    SJMP   NC3                         ;    new row
NC2:
    MOV    P2,CurRow                   ; Count difference from old row
NC3:                                 ; Set up for search
    MOV    R1,A                        ; Save number of rows to skip
    MOV    A,RcbOff                    ; Get offset into active RCBs
    ADD    A,#RCB_RowPag               ;    of next RCB's page address
    MOV    R0,A                        ;    ready for search
```

```
NC4:                                 ; For each row skipped
    CJNE  A,BtmRow,NC5                    ; Jump if row is not bottom vis
    MOV   A,RemRow                        ; Set for remaining rows
    SJMP  NC6                             ;     and continue search
NC5:
    MOVX  A,@R0                           ; Get next row page address
NC6:
    MOV   P2,A                            ; Point to row
    DJNZ  R1,NC4                          ; Loop if more to skip
NC7:                                 ; Assign new position
    MOV   CurRow,A                        ; New current row page address
    MOV   ActRow,R2                       ; New active row position
NC8:                                 ; New row same as old
    MOV   ActCol,R3                       ; New active column position

; NOTE:  This routine falls through to "PlcCsr" below.
;-------------------------------------------------------------------------------
PlcCsr:
;...............................................................................
; Sets the cursor in the main definition block.  The cursor is shown (enabled)
; or hidden (disabled) depending on the type of zone (visible or invisible)
; containing the active position.  However, nothing is done if a smooth scroll
; operation is in progress.
;
; In:    ActCol          active position's column within display
;        ActRow          active position's row within display
; Out:   BgdMDB0         main definition blocks modified
;        BgdMDB1
;        (see also FndCsrZon)
; Bad:   A,DPTR,R0,R1,R2,R3,PSW
;
; NOTE:  This routine must immediately follow "NewCsr" and immediately
;        precede "ShwCsr", with "HidCsr" immediately after that.
;...............................................................................

    JB    VrtScrlFlg,PC1                  ;exit if vert smooth scroll
    JNB   HrzScrlFlg,PC2                  ;skip if not horz smooth scroll
PC1:
    RET
PC2:
    LCALL FndCsrZon                       ;need to recalculate zone
    JNB   CsrZonFlg,HidCsr                ;jump if cursor not visible
```

```
    MOV   DPH,#BgdMDB0.SR.PAGE                  ;set page for main blocks
    MOV   R0,#BgdMDB0.AN.OFST+MDB_Cux           ;R0 -> cursor x, block 0
    MOV   R1,#BgdMDB1.AN.OFST+MDB_Cux           ;R1 -> cursor x, block 1
    CLR   C
    MOV   A,ActCol                              ;A = # columns right of visible
    SUBB  A,VisCol                              ;    left margin
    ADD   A,ColAdd                              ;A = screen column
    MOV   DPL,R0
    MOVX  @DPTR,A                               ;set the x position of cursor
    MOV   DPL,R1
    MOVX  @DPTR,A
    INC   R0                                    ;advance ptrs to cursor y
    INC   R1
    MOV   A,ActRow                              ;A = # rows down from top
    SUBB  A,VisRow                              ;    of screen
    ADD   A,RowAdd
    MOV   DPL,R0
    MOVX  @DPTR,A                               ;set the y position of cursor
    MOV   DPL,R1
    MOVX  @DPTR,A

; NOTE:  This routine falls through to "ShwCsr" below.

;-------------------------------------------------------------------------------
ShwCsr:
;...............................................................................
; Sets up for, but defers, enabling the Am8052 X-Y cursor.
;
; In:   (none)
; Out:  (none)
; Bad:  (none)
;
; NOTE:  This routine must immediately follow "PlcCsr".
;...............................................................................

    JB    VrtScrlFlg,SC1                        ; Exit if vertical or
    JB    HrzScrlFlg,SC1                        ;    horz smooth scroll going on
    SETB  CsrShwFlg                             ; Defer until vertical retrace
SC1:
    RET                                         ; Exit
;-------------------------------------------------------------------------------
```

```
HidCsr:                              ; Remove cursor for hidden positions

;.......................................................................
; Disables the Am8052 X-Y cursor so that the active position is not marked.
;
; In:     (none)
; Out:    (none)
; Bad:    A,DPTR,R1,R2,R3
;
; NOTE:  This routine must immediately follow "ShwCsr".
;.......................................................................

   CLR    CsrShwFlg                     ; Ensure no cursor
   CLR    CsrSetFlg
   MOV    R1,#ModReg2Ind                ; Need Mode Register 2
   LCALL  RdAm8052Reg                   ;    read from Am8052
   MOV    A,R2                          ; Get high byte and
   CLR    ACC.7                         ;    reset CUE bit to disable the
   MOV    R2,A                          ;    X-Y cursor then put it back
   LCALL  WrAm8052Reg                   ;    and write it to Am8052
   RET                                  ; Exit

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
EraActEnd:
;.......................................................................
; Erases from, and including, the active position through the end of the
; active row.  The erased positions will contain spaces with the current
; attribute.
;.......................................................................

   MOV    A,CurRow
   MOV    P2,A
   JNB    WndActFlg,EAE1                ; Test if in window or Background
   SETB   ACC.4                         ; Build window attribute page ptr
   MOV    R5,A
   MOV    A,#40                         ; max count for window row
   SJMP   EAE2
EAE1:
   SETB   ACC.5                         ; Build Bgd attribute page ptr
   MOV    R5,A
   MOV    A,#128                        ; Max count for background row
EAE2:
   CLR    C
```

```
   SUBB   A,ActCol
   MOV    R6,A
   MOV    R7,A
   MOV    A,ChrOff
   ADD    A,ActCol
   MOV    R0,A
   MOV    A,#' '
EAE3:
   MOVX   @R0,A
   INC    R0
   DJNZ   R6,EAE3
   MOV    P2,R5
   MOV    A,AtrOff
   ADD    A,ActCol
   ADD    A,ActCol
   INC    A
   MOV    R0,A
   MOV    A,CurAtr
EAE4:
   MOVX   @R0,A
   INC    R0
   INC    R0
   DJNZ   R7,EAE4
   MOV    P2,#BgdActCntBuf.SR.PAGE
   MOV    A,R5
   JNB    MsgActFlg,EAE5
   MOV    R0,#MsgActCnt.AN.OFST
   SJMP   EAE8
EAE5:
   JNB    WndActFlg,EAE6
   MOV    R0,#WndActCntBuf.AN.OFST
   ANL    A,#00FH
   SJMP   EAE7
EAE6:
   MOV    R0,#BgdActCntBuf.AN.OFST
   ANL    A,#01FH
EAE7:
   ADD    A,R0
   MOV    R0,A
EAE8:
   MOVX   A,@R0
   CLR    C
   SUBB   A,ActCol
   JC    EAE9
   MOV   A,ActCol
   MOVX  @R0,A
EAE9:
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
EraBgnAct:
;..............................................................................
; Erase from, and including, the first position in the active row through the
; active position.  The erased positions will contain spaces with the current
; attribute.
;..............................................................................

   MOV   A,CurRow
   MOV   P2,A
   JNB   WndActFlg,EBA1
   SETB  ACC.4
   SJMP  EBA2
EBA1:
   SETB  ACC.5
EBA2:
   MOV   R5,A
   MOV   A,ActCol
   INC   A
   MOV   R6,A
   MOV   R7,A
   MOV   R0,ChrOff
   MOV   A,#' '
EBA3:
   MOVX  @R0,A
   INC   R0
   DJNZ  R6,EBA3

   MOV   P2,R5
   MOV   R0,AtrOff
   INC   R0
   MOV   A,CurAtr
EBE4:
   MOVX  @R0,A
   INC   R0
   INC   R0
   DJNZ  R7,EBE4
```

```
   MOV   P2,#BgdActCntBuf.SR.PAGE
   MOV   A,R5
   JNB   MsgActFlg,EBE5
   MOV   R0,#MsgActCnt.AN.OFST
   SJMP  EBE8
EBE5:
   JNB   WndActFlg,EBE6
   MOV   R0,#WndActCntBuf.AN.OFST
   ANL   A,#00FH
   SJMP  EBE7
EBE6:
   MOV   R0,#BgdActCntBuf.AN.OFST
   ANL   A,#01FH
EBE7:
   ADD   A,R0
   MOV   R0,A
EBE8:
   MOVX  A,@R0
   SETB  C
   SUBB  A,ActCol
   JNC   EBE9
   CLR   A
   MOVX  @R0,A
EBE9:
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
FrcEraRow:
;..............................................................................
; Forces an entire erasure of a row for Erase In Display or Erase In Line.
;
; NOTE:  This routine must immediately precede "EraRow".
;..............................................................................

   MOV   R6,A
   MOV   DPH,#BgdActCntBuf.SR.PAGE
   JNB   MsgActFlg,FER1
   MOV   DPL,#MsgActCnt.AN.OFST
   MOV   R7,#128
   SJMP  FER4
FER1:
   JNB   WndActFlg,FER2
   MOV   DPL,#WndActCntBuf.AN.OFST
```

```
    MOV   R7,#40
    SJMP  FER3
FER2:
    MOV   DPL,#BgdActCntBuf.AN.OFST
    MOV   R7,#128
FER3:
    ANL   A,#01FH
    ADD   A,DPL
    MOV   DPL,A
FER4:
    MOV   A,R7
    MOVX  @DPTR,A
    MOV   A,R6

; NOTE:  This routine falls through to "EraRow" below.

;------------------------------------------------------------------------
EraRow:
;........................................................................
; Erases the given row to a blank condition (i.e. all spaces with the current
; attributes).
;
; In:    A                          page address of row
; Out:                              active count update
; Bad:   A,DPTR,P2,R0,R4,R6,R7
;........................................................................

    MOV   P2,A                            ;put page address in ptr
    JNB   MsgActFlg,ER1                   ;skip if not msg row
                                    ;message row
    SETB  ACC.5                           ;R2 = attribute page
    MOV   R4,A
    MOV   DPTR,#MsgActCnt                 ;ptr to active char count
    SJMP  ER4                             ;do the erase
ER1:                                ;check for window
    MOV   R4,A                            ;put character page in R2
    JNB   WndActFlg,ER2                   ;skip if not in window
                                    ;window row
    SETB  ACC.4                           ;A = attribute page
    MOV   DPTR,#WndActCntBuf              ;ptr to active character counts
    SJMP  ER3
```

```
ER2:                                      ;must be background
    SETB  ACC.5                                 ;A = attribute page
    MOV   DPTR,#BgdActCntBuf                    ;ptr to active character counts
ER3:
    XCH   A,R4                                  ;put attrib page in R2
    CLR   ACC.7                                 ;get row number in A
    ADD   A,DPL                                 ;index DPTR to correct count
    MOV   DPL,A                                 ;    for this row
ER4:
    MOVX  A,@DPTR                               ;fetch the active character cnt
    JZ    ER7                                   ;skip if none
    MOV   R6,A                                  ;R6 = R7 = Active count
    MOV   R7,A                                  ;(one for char and one for attr)
    CLR   A                                     ;Active count set to 0
    MOVX  @DPTR,A
    MOV   R0,ChrOff                             ;R0 = offset of first char
    MOV   A,#" "                                ;A = blank character
ER5:                                      ;blank characters loop
    MOVX  @R0,A                                 ;blank one character
    INC   R0                                    ;next character
    DJNZ  R6,ER5
                                          ;done with character blanking
    MOV   P2,R4                                 ;attribute page selected
    MOV   R0,AtrOff                             ;attribute offset in R0
    INC   R0                                    ;select lower attribute byte
    MOV   A,CurAtr                              ;current attributes
ER6:
    MOVX  @R0,A                                 ;set lower attribute byte
    INC   R0                                    ;next attribute
    INC   R0
    DJNZ  R7,ER6
                                          ;done with attribute clear
ER7:
    RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
HidWnd:
;........................................................................
; Hides the window if window is visible, if the message window is visible
; it maintains its visibliity.
;
; Bad:   R1,R2,R3
;........................................................................
```

```
    JB     MsgVisFlg,HW1
    MOV    R2,#TrmWDB.SR.PAGE
    MOV    R3,#TrmWDB.AN.OFST
    SJMP   HW2
HW1:
    MOV    R2,#MsgWDB.SR.PAGE
    MOV    R3,#MsgWDB.AN.OFST
HW2:
    MOV    R1,#TOWHrdLoInd
    LCALL  WrAm8052Reg
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ShwWnd:
;...............................................................................
; Makes the window visible, if the message window is visible it is maintained.
;
; Bad:   A,DPTR,R1,R2,R3
;...............................................................................

    JB     MsgVisFlg,SW1
    MOV    R2,#TrmWDB.SR.PAGE
    MOV    R3,#TrmWDB.AN.OFST
    SJMP   SW2
SW1:
    MOV    R2,#MsgWDB.SR.PAGE
    MOV    R3,#MsgWDB.AN.OFST
SW2:
    MOV    DPTR,#WndWDB0+WDB_NxtPag
    MOV    A,R2
    CLR    EX0
    MOVX   @DPTR,A
    INC    DPTR
    MOV    A,R3
    MOVX   @DPTR,A
    INC    DPH
    MOVX   @DPTR,A
    DEC    DPL
    MOV    A,R2
    MOVX   @DPTR,A
    SETB   EX0
    JB     CurWDBFlg,SW3
    MOV    R2,#WndWDB0.SR.PAGE
```

```
    MOV    R3,#WndWDB0.AN.OFST
    SJMP   SW4
SW3:
    MOV    R2,#WndWDB1.SR.PAGE
    MOV    R3,#WndWDB1.AN.OFST
SW4:
    MOV    R1,#TOWHrdLoInd
    LCALL  WrAm8052Reg
SW5:
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
BldTrmRcb:
;...............................................................................
; Writes a new termination row control block when activating a different
; display.
;
; Bad:   P2,A,R0
;...............................................................................

    JNB    MsgActFlg,BTR1
    MOV    DPTR,#BgdVarBuf+(ExtRow-CurAtr)
    MOVX   A,@DPTR
    MOV    TrmRow,A
    SJMP   BTR2
BTR1:
    MOV    TrmRow,ExtRow
BTR2:
    MOV    TrmOff,RcbOff
    MOV    P2,TrmRow                        ; When the background is to be
    MOV    R0,RcbOff                        ;    active it must have a
    MOV    A,#80H
    MOVX   @R0,A                            ;    properly initialized Term.
    INC    R0                               ;    this will be the Window dsp
    INC    R0                               ;    ExtRow.
    MOV    A,TrmRow                         ;    Termination RCBs point to
    MOVX   @R0,A                            ;    themselves, with a segment
                                            ;    count of one, a hidden
    INC    R0                               ;    count of zero, and a visible
    MOV    A,TrmOff                         ;    count of one.
    MOVX   @R0,A
    INC    R0
    CLR    A
    MOVX  @R0,A
    INC   R0
    INC   A
    MOVX  @R0,A
    INC   R0
    CLR   A
    MOVX  @R0,A
    INC   R0
    INC   R0
    MOVX  @R0,A
    INC   R0
    MOVX  @R0,A
    INC   R0
    INC   R0
    INC   R0
    MOV   A,#TrmAtr.SR.PAGE
    MOVX  @R0,A
    INC   R0
    MOV   A,#TrmAtr.AN.OFST
    MOVX  @R0,A
    INC   R0
    CLR   A
    MOVX  @R0,A
    INC   R0
    MOVX  @R0,A
    INC   R0
    MOV   A,#NrmRRB.SR.PAGE
    MOVX  @R0,A
    INC   R0
    MOV   A,#NrmRRB.AN.OFST
    MOVX  @R0,A
    MOV   DPTR,#TrmWDB+WDB_RowPag
    MOV   A,TrmRow
    CLR   EX0
    MOVX  @DPTR,A
    INC   DPTR
    MOV   A,TrmOff
    MOVX  @DPTR,A
    SETB  EX0
    MOV   DPTR,#MsgRCB+RCB_RowOff
    CLR   EX0
    MOVX  @DPTR,A
    DEC   DPL
```

```
    MOV   A,TrmRow
    MOVX  @DPTR,A
    SETB  EX0
    JB    WndActFlg,BTR3
    JB    MsgActFlg,BTR3
    MOV   A,BtmRow
    SJMP  BTR4
BTR3:
    MOV   DPTR,#BgdVarBuf+(BtmRow-CurAtr)
    MOVX  A,@DPTR
BTR4:
    MOV   DPH,A
    MOV   DPL,#BgdRCB0.AN.OFST+RCB_RowPag
    MOV   A,TrmRow
    CLR   EX0
    MOVX  @DPTR,A
    INC   DPTR
    MOV   A,TrmOff
    MOVX  @DPTR,A
    SETB  EX0
    JB    MsgActFlg,BTR5
    JNB   WndActFlg,BTR5
    MOV   A,BtmRow
    SJMP  BTR6
BTR5:
    MOV   DPTR,#WndVarBuf+(BtmRow-CurAtr)
    MOVX  A,@DPTR
BTR6:
    MOV   DPH,A
    MOV   DPL,#WndRCB0.AN.OFST+RCB_RowPag
    MOV   A,TrmRow
    CLR   EX0
    MOVX  @DPTR,A
    INC   DPTR
    MOV   A,TrmOff
    MOVX  @DPTR,A
    SETB  EX0
    RET
```

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
HalfSwap:
; Copies display dependent variables to external memory
;
; In:    R3                         Out going pointer page
;        R4                         Out going pointer offset
; Out:   external memory at R3:R4
; Bad:   P2,A,R0,R1,R2

   MOV   P2,R3                          ;set page register
   MOV   A,R4                           ;set external offset
   MOV   R0,A
   MOV   R1,#CurAtr                     ;set internal pointer
   MOV   R2,#(DspWid-CurAtr)            ;count of dependent var
HS1:
   MOV   A,@R1                          ;move one byte
   MOVX  @R0,A
   INC   R0                             ;next byte
   INC   R1
   DJNZ  R2,HS1
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SwpVar:
; Moves a set of display dependent variables to external storage
; then moves in a new set of dependent variables from a another
; external location.
;
; In:    R3                         Out going pointer hi
;        R4                         Out going pointer lo
;        R5                         In coming pointer hi
;        R6                         In coming pointer lo
; Out:   internal display dependent variables
;        external memory at R3:R4
; Bad:   P2,R0,R7
;
; NOTE:  R1 is preserved

   MOV   A,R1                           ;save R1
   PUSH  ACC
   MOV   P2,R3                          ;set ouput page
   MOV   A,R4                           ;set output offset
   MOV   R0,A
```

```
   MOV   R1,#CurAtr                     ;set internal address
   MOV   R7,#(DspWid-CurAtr)            ;count of variables
SV1:                                 ;move out loop
   MOV   A,@R1                          ;move one byte
   MOVX  @R0,A
   INC   R0                             ;next byte
   INC   R1
   DJNZ  R7,SV1
                                        ;done with move out
   MOV   P2,R5                          ;set input page
   MOV   A,R6                           ;set input offset
   MOV   R0,A
   MOV   R1,#CurAtr                     ;set internal address
   MOV   R7,#(DspWid-CurAtr)            ;count of variables
SV2:                                 ;move in loop
   MOVX  A,@R0                          ;move one byte
   MOV   @R1,A
   INC   R0                             ;next byte
   INC   R1
   DJNZ  R7,SV2
                                     ;done
   POP   ACC                            ;restore R1
   MOV   R1,A
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SetCelWid:
;.............................................................................
; Sets the upper attribute byte for all positions thus changing the character
; widths uniformly.
;
; In:    A                           upper attribute byte
; Out:   all attributes              (upper byte only)
;        R1                          set to this byte
; Bad:   P2,R0,R2,R3

   MOV   R1,A                           ;set R1 to attribute byte
   MOV   P2,#BgdAtrBuf0.SR.PAGE         ;bacground start page
   MOV   R2,#32                         ;31 backgrd + msg rows
SCW1:                                ;bgrd and msg row loop point
   MOV   R0,#BgdAtrBuf0.AN.OFST         ;attribute offset
   MOV   R3,#128                        ;character count
```

```
SCW4:                                   ;bgrd and msg char loop point
    MOVX  @R0,A                               ;set attribute byte
    INC   R0                                  ;next attribute
    INC   R0
    DJNZ  R3,SCW4
                                        ;done with row
    INC   P2
    DJNZ  R2,SCW1                             ;next row
                                        ;done with bgrd and msg
    MOV   P2,#WndAtrBuf0.SR.PAGE              ;window start page
    MOV   R2,#15                              ;window row count
SCW2:                                   ;window row loop point
    MOV   R0,#WndAtrBuf0.AN.OFST              ;attribute offset
    MOV   R3,#40                              ;character count
SCW3:                                   ;window character loop point
    MOVX  @R0,A                               ;set attribute byte
    INC   R0                                  ;next attribute
    INC   R0
    DJNZ  R3,SCW3
                                        ;done with row
    INC   P2
    DJNZ  R2,SCW2                             ;next row
                                        ;done with window
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ChgBlnkSpd:
;...........................................................................
; Changes the blink rates for the cursor and blinking character attribute.
;
; In:   BlnkByt                         new blink control byte
; Out:  (none)
; Bad:  P2,R0,R1,A
;...........................................................................

    MOV   A,BlnkByt                           ;replace blink control
    MOV   DPTR,#BgdMDB0+MDB_Blnk
    MOVX  @DPTR,A
    MOV   DPTR,#BgdMDB1+MDB_Blnk
    MOVX  @DPTR,A
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
ChgCsrSiz:
;...........................................................................
; Translates the internal cursor size representation (in the form of 2
; nibbles) to the row redefinition block representation of two five-bit
; fields stored in a 16-bit word;
;
; In:   CsrSiz                          variable defining new size
; Out:  normal row redefinition block cursor bytes
; Bad:  A,P2,R0,R1,R2
;...........................................................................

    MOV   DPTR,#NrmRRB+RRB_CursHi               ;set rwo redef location
    MOV   A,CsrSiz                              ;R2 = cursor end
    ANL   A,#0FH
    MOV   R2,A
    MOV   A,CsrSiz                              ;R1 = CsrSiz rotated left 1
    RL    A
    MOV   R1,A
    ANL   A,#001H                               ;most sig cursor start bit
    MOVX  @DPTR,A                               ;written in high byte
    INC   DPL                                   ;next byte
    MOV   A,R1                                  ;upper three bits of start in A
    ANL   A,#0E0H
    ORL   A,R2                                  ;cursor end joined in
    MOVX  @DPTR,A                               ;write lower byte
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ChgCsrTyp:
;...........................................................................
; Changes the cursor type bits in mode register 2
;
; In:   R5                              cursor type bits (bits 1 and 2)
; Out:  Mode Register 2                 bits 9 and 10 modified
; Bad:  A,R1,R2,R3
;...........................................................................

    MOV   R1,#ModReg2Ind                        ;mode register 2 index
    LCALL RdAm8052Reg
    MOV   A,R2                                  ;high byte of mode register 2
    ANL   A,#0F9H                               ;keep all but bits 1 and 2
    ORL   A,R5                                  ;get these from R5
    MOV   R2,A                                  ;write it back
```

```
        LCALL WrAm8052Reg
        RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DelRow_MovUp:

        MOV   A,ExtRow
        LCALL FrcEraRow                      ; Erase extra row
        MOV   A,RcbOff
        ADD   A,#RCB_RowPag
        MOV   DPL,A
        MOV   A,ExtRow
        MOV   DPH,A                          ; Make extra row point to itself
        CLR   EX0
        MOVX  @DPTR,A
        INC   DPL
        MOV   A,RcbOff
        MOVX  @DPTR,A
        MOV   DPH,EndRow                     ; Make end row point to extra row
        MOVX  @DPTR,A                        ;     thereby adding extra row to
        DEC   DPL                            ;     end of display
        MOV   A,ExtRow
        MOVX  @DPTR,A
        SETB  EX0
        MOV   EndRow,A                       ; Extra row becomes new end row
        MOV   A,DPH                          ; Compare old end row to bottom
        CJNE  A,BtmRow,DRMU1                 ;     visible row, jump if differ
        MOV   A,EndRow                       ; New end row
        MOV   RemRow,A                       ;     is also new remaining rows
DRMU1:                                 ; Bottom row not at end of display
        MOV   A,BgnRow                       ; Start at first row of display
        CJNE  A,CurRow,DRMU2                 ; Jump if not currently at begin
        MOV   ExtRow,A                       ; New extra row is old begin row
        MOV   DPH,A                          ; Point to it
        MOVX  A,@DPTR                        ;     and get row following it
        MOV   BgnRow,A                       ;     as new first row of display
        MOV   CurRow,A                       ;     and new current row
        MOV   A,ExtRow                       ; Compare old begin row to top
        CJNE  A,TopRow,DRMU5                 ;     visible row, jump if differ
        SJMP  DRMU7                          ; Else handle non-critical cases
DRMU2:                                 ; Current row not at top of display
        MOV   DPH,A                          ; Point to row
        MOVX  A,@DPTR                        ;     and find row following it
```

```
        CJNE  A,CurRow,DRMU4                 ; Jump if not current row

        MOV   ExtRow,A                       ; Old curr row is new extra row
        CJNE  A,BtmRow,DRMU3                 ; Jump if not at btm visible row
        MOV   A,RemRow                       ; Special case at bottom, old
        MOV   BtmRow,A                       ; remaining row to new bottom
        MOV   CurRow,A                       ;     and new current rows
        CLR   EX0
        MOVX  @DPTR,A                        ; Make row before bottom point
        MOV   DPH,A                          ;     to new bottom (i.e. old
        MOVX  A,@DPTR                        ;     remaining row) and following
        MOV   RemRow,A                       ;     row becomes new rem row
        MOV   A,TrmRow                       ; Make new bottom row point
        MOVX  @DPTR,A                        ;     to termination row
        INC   DPL
        MOV   A,TrmOff
        MOVX  @DPTR,A
        SETB  EX0
        RET                                  ; Exit

DRMU3:                                 ; Current row found, not at bottom
        MOV   R6,DPH
        MOV   DPH,A                          ; Following row is
        MOVX  A,@DPTR                        ;     new current row
        MOV   DPH,R6
        MOVX  @DPTR,A                        ; Change linked list to delete
        MOV   CurRow,A                       ; Assign new current row
        MOV   A,ExtRow                       ; Set up to
        SJMP  DRMU6                          ;     scan rest of list
DRMU4:                                 ; Current row not found yet
        CJNE  A,BtmRow,DRMU2                 ; Jump if not at bottom visible
        MOV   A,RemRow                       ; Compare old remaining row
        CJNE  A,CurRow,DRMU2                 ;     to current, jump if differ
        MOV   ExtRow,A                       ; Old curr row is new extra row
        MOV   DPH,A                          ; Point to it
        MOVX  A,@DPTR                        ;     and following row
        MOV   RemRow,A                       ;     is new remaining row
        RET                                  ; Exit after special case
DRMU5:                                 ; Adjust rest of linked list
        MOVX  A,@DPTR                        ; Get following row
DRMU6:
        MOV   DPH,A                          ; Point to following row
        CJNE  A,TopRow,DRMU10                ; Jump if not top visible row
```

```
DRMU7:
    MOV   R7,DPL                          ; Adjust new top visible row
    MOVX  A,@DPTR                         ;    and make appropriate block
    MOV   TopRow,A                        ;    (MDB or WDB) point to it
    JNB   WndActFlg,DRMU8
    MOV   DPTR,#WndWDB0+WDB_RowPag
    MOVX  @DPTR,A
    MOV   DPTR,#WndWDB1+WDB_RowPag
    MOVX  @DPTR,A
    SJMP  DRMU9
DRMU8:
    MOV   DPTR,#BgdMDB0+MDB_RowPag
    MOVX  @DPTR,A
    MOV   DPTR,#BgdMDB1+MDB_RowPag
    MOVX  @DPTR,A
DRMU9:
    MOV   DPH,A                           ; Set up to
    MOV   DPL,R7                          ;    scan through and
    SJMP  DRMU5                           ;    adjust rest of linked list
DRMU10:                                 ; Scanning, not at top
    CJNE  A,BtmRow,DRMU11                 ; Jump if not bottom visible row
    MOV   A,RemRow                        ; Old remaining row is
    MOV   BtmRow,A                        ;    new bottom visible row
    CLR   EX0
    MOVX  @DPTR,A                         ; Make old bottom row point to
    INC   DPL                             ;    old remaining row
    MOV   A,RcbOff
    MOVX  @DPTR,A
    MOV   DPH,RemRow                      ; Make new bottom row point to
    MOV   A,TrmOff                        ;    termination row
    MOVX  @DPTR,A
    DEC   DPL
    MOVX  A,@DPTR                         ; Row following old remaining row
    MOV   RemRow,A                        ;    is new remaining row
    MOV   A,TrmRow
    MOVX  @DPTR,A
    SETB  EX0
    RET                                   ; Exit

DRMU11:                                 ; Scanning, not at top or bottom
    CJNE  A,EndRow,DRMU5                  ; Jump if not at end row

    RET                                   ; Exit when we get to the end
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DelRow_MovDn:

; Deletes the current active row from the display and moves rows above it
; downward.  An erased row is inserted at the top of the display.
;
; In:     CurRow, BgnRow, TopRow,         row variables
;         BtmRow, RemRow, EndRow,
;         ExtRow
; Out:                                    updated row variables
;
; Bad:    A,DPTR,R6,R7,PSW

    MOV   A,ExtRow                        ; Extra row is
    LCALL FrcEraRow                       ;    erased and
    MOV   DPH,ExtRow                      ;    then its
    MOV   A,RcbOff                        ;    RCB next
    ADD   A,#RCB_RowPag                   ;    row field
    MOV   DPL,A                           ;    is set so the
    MOV   A,BgnRow                        ;    old beginning row
    MOVX  @DPTR,A                         ;    follows it
    INC   DPL
    MOV   A,RcbOff
    MOVX  @DPTR,A
    DEC   DPL
    MOV   A,DPH                           ; Old extra row
    MOV   BgnRow,A                        ;    becomes new beginning row
DRMD1:                                  ; For each row above top visible row
    MOV   DPH,A                           ; Point to the row
    MOVX  A,@DPTR                         ; Get page of next row
    CJNE  A,TopRow,DRMD4                  ; Jump if next is not top row
    MOV   TopRow,DPH                      ; New top row is preceding row
    MOV   R6,A                            ; Save next row page address and
    MOV   R7,DPL                          ;    display's offset to next row
    MOV   A,DPH                           ; This row is new top row
    JNB   WndActFlg,DRMD2                 ; Jump if not in window
    MOV   DPTR,#WndWDB0+WDB_RowPag        ; Point into first window block
    MOVX  @DPTR,A                         ;    and set new top row
    MOV   DPTR,#WndWDB1+WDB_RowPag        ; Point into second window block
    MOVX  @DPTR,A                         ;    and set new top row
    SJMP  DRMD3                           ; Continue
```

```
DRMD2:
   MOV   DPTR,#BgdMDB0+MDB_RowPag             ; Point into first bgd block
   MOVX  @DPTR,A                              ;     and set new top row
   MOV   DPTR,#BgdMDB1+MDB_RowPag             ; Point into second bgd block
   MOVX  @DPTR,A                              ;     and set new top row
DRMD3:
   MOV   DPH,A                                ; Point to this row's next
   MOV   DPL,R7                               ;     row pointer again
   MOV   A,R6                                 ; Restore page of next row
   SJMP  DRMD8                                ; Go check for row to delete
DRMD4:                                  ; Still above top visible row
   CJNE  A,CurRow,DRMD1                       ; Loop if not row to delete
   SJMP  DRMD11                               ; Go delete row
DRMD5:                                  ; For each row between top and btm vis
   MOV   DPH,A                                ; Point to the row
   MOVX  A,@DPTR                              ; Get page of next row
   CJNE  A,BtmRow,DRMD8                       ; Jump if next is not bottom row
   MOV   BtmRow,DPH                           ; New bottom row is preceding row
   CJNE  A,CurRow,DRMD7                       ; Jump if next is not row to del
   MOV   CurRow,DPH                           ; New current row is preceding
   MOV   ExtRow,A                             ; New extra row is one to delete
   CJNE  A,EndRow,DRMD6                       ; Jump if next is not end row
   MOV   EndRow,DPH                           ; New end row is preceding row
   MOV   RemRow,DPH                           ; New remaining row is set same
DRMD6:                                  ; Delete old bottom row
   MOV   A,TrmRow                             ; Make new
   CLR   EX0
   MOVX  @DPTR,A                              ;     bottom row
   INC   DPL                                  ;     point to
   MOV   A,TrmOff                             ;     display's
   MOVX  @DPTR,A                              ;     termination row
   SETB  EX0
   RET                                        ; Exit

DRMD7:                                  ; New btm row (haven't found del row)
   MOV   DPH,A                                ; Make
   MOV   A,RemRow                             ;     old bottom
   CLR   EX0
   MOVX  @DPTR,A                              ;     row point
   INC   DPL                                  ;     to old
   MOV   A,RcbOff                             ;     remaining
   MOVX  @DPTR,A                              ;     row
   MOV   RemRow,DPH                           ; New rem row follows new btm row
```

```
   MOV   DPH,BtmRow                           ; Make new
   MOV   A,TrmOff                             ;     bottom row
   MOVX  @DPTR,A                              ;     point to
   DEC   DPL                                  ;     display's
   MOV   A,TrmRow                             ;     termination
   MOVX  @DPTR,A                              ;     row
   SETB  EX0
   MOV   A,RemRow                             ; Resume with new remaining row
   SJMP  DRMD9                                ;     and go check for row to del
DRMD8:                                  ; Still between top and btm vis rows
   CJNE  A,CurRow,DRMD5                       ; Loop if not row to delete
   SJMP  DRMD11                               ; Go delete row
DRMD9:                                  ; Below bottom visible row
   MOV   DPH,A                                ; Point to the row
   MOVX  A,@DPTR                              ; Get page of next row
   CJNE  A,EndRow,DRMD10                      ; Jump if next is not end row
   MOV   CurRow,DPH                           ; New current row is preceding
   MOV   EndRow,DPH                           ; New end row is preceding row
   MOV   ExtRow,A                             ; New extra row is one to delete
   MOV   A,DPH                                ; Make end row
   MOVX  @DPTR,A                              ;     point to itself
   RET                                        ; Exit

DRMD10:                                 ; Still not to end row
   CJNE  A,CurRow,DRMD9                       ; Loop if not row to delete
DRMD11:                                 ; Delete row (no special updates)
   MOV   CurRow,DPH                           ; New current row is preceding
   MOV   ExtRow,A                             ; New extra row is one to delete
   MOV   DPH,A                                ; Get
   MOVX  A,@DPTR                              ;     page of following row
   MOV   DPH,CurRow                           ; New current row points to row
   MOVX  @DPTR,A                              ;     after old current (deleted)
   RET                                        ; Exit

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
InsRow_MovUp:

   MOV   A,ExtRow
   LCALL FrcEraRow
   MOV   DPH,CurRow
   MOV   A,RcbOff
   ADD   A,#RCB_RowPag
   MOV   DPL,A
```

```
    MOVX  A,@DPTR
    MOV   R6,A
    INC   DPL
    MOVX  A,@DPTR
    MOV   DPH,ExtRow
    MOVX  @DPTR,A
    DEC   DPL
    MOV   A,R6
    MOVX  @DPTR,A
    MOV   R5,ExtRow
    MOV   A,BgnRow
    MOV   ExtRow,A
    CJNE  A,CurRow,IRMU3
    MOV   CurRow,R5
    MOV   BgnRow,R5
    CJNE  A,TopRow,IRMU2
    MOV   TopRow,R5
    MOV   A,R5
    JNB   WndActFlg,IRMU1
    MOV   DPTR,#WndWDB0+WDB_RowPag
    MOVX  @DPTR,A
    MOV   DPTR,#WndWDB1+WDB_RowPag
    MOVX  @DPTR,A
    RET

IRMU1:
    MOV   DPTR,#BgdMDB0+MDB_RowPag
    MOVX  @DPTR,A
    MOV   DPTR,#BgdMDB1+MDB_RowPag
    MOVX  @DPTR,A
IRMU2:
    RET
IRMU3:
    MOV   DPH,A
    MOVX  A,@DPTR
    MOV   BgnRow,A
    MOV   A,DPH
    CJNE  A,TopRow,IRMU4
    MOV   R6,A
    MOV   R7,DPL
    SJMP  IRMU5
IRMU4:
    MOVX  A,@DPTR
```

```
    MOV   DPH,A
    CJNE  A,TopRow,IRMU9
    MOV   R6,A
    MOV   R7,DPL
    CJNE  A,CurRow,IRMU5
    MOV   A,R5
    SJMP  IRMU6
IRMU5:
    MOVX  A,@DPTR
IRMU6:
    MOV   TopRow,A
    JNB   WndActFlg,IRMU7
    MOV   DPTR,#WndWDB0+WDB_RowPag
    MOVX  @DPTR,A
    MOV   DPTR,#WndWDB1+WDB_RowPag
    MOVX  @DPTR,A
    SJMP  IRMU8
IRMU7:
    MOV   DPTR,#BgdMDB0+MDB_RowPag
    MOVX  @DPTR,A
    MOV   DPTR,#BgdMDB1+MDB_RowPag
    MOVX  @DPTR,A
IRMU8:
    MOV   DPH,R6
    MOV   DPL,R7
    MOV   A,R6
    SJMP  IRMU13
IRMU9:
    CJNE  A,CurRow,IRMU4
    SJMP  IRMU17
IRMU10:
    MOVX  A,@DPTR
    MOV   DPH,A
    CJNE  A,BtmRow,IRMU13
    CJNE  A,CurRow,IRMU11
    MOV   BtmRow,R5
    CJNE  A,EndRow,IRMU17
    MOV   RemRow,R5
    SJMP  IRMU15a
IRMU11:
    MOV   A,RemRow
    MOV   BtmRow,A
    CLR   EX0
```

```
     MOVX  @DPTR,A
     INC   DPL
     MOV   A,RcbOff
     MOVX  @DPTR,A
     MOV   DPH,RemRow
     MOV   A,TrmOff
     MOVX  @DPTR,A
     DEC   DPL
     MOVX  A,@DPTR
     MOV   R6,A
     MOV   A,TrmRow
     MOVX  @DPTR,A
     SETB  EX0
     MOV   A,RemRow
     CJNE  A,CurRow,IRMU12
     MOV   RemRow,R5
     MOV   CurRow,R5
     RET

IRMU12:
     MOV   RemRow,R6
     MOV   DPH,R6
     MOV   A,R6
     SJMP  IRMU15
IRMU13:
     CJNE  A,CurRow,IRMU10
     SJMP  IRMU17
IRMU14:
     MOVX  A,@DPTR
     MOV   DPH,A
IRMU15:
     CJNE  A,EndRow,IRMU16
IRMU15a:
     MOV   EndRow,R5
     SJMP  IRMU17
IRMU16:
     CJNE  A,CurRow,IRMU14
IRMU17:
     MOV   A,R5
     MOV   CurRow,A
     CLR   EX0
     MOVX  @DPTR,A
     INC   DPTR
```

```
     MOV   A,RcbOff
     MOVX  @DPTR,A
     SETB  EX0
     RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
InsRow_MovDn:

     MOV   A,ExtRow
     LCALL FrcEraRow
     MOV   R5,ExtRow
     MOV   DPH,R5
     MOV   A,RcbOff
     MOV   R7,A
     ADD   A,#RCB_RowPag
     MOV   DPL,A
     MOV   A,CurRow
     MOVX  @DPTR,A
     INC   DPL
     MOV   A,R7
     MOVX  @DPTR,A
     DEC   DPL
     MOV   A,BgnRow
     CJNE  A,CurRow,IRMD1
     MOV   BgnRow,R5
     SJMP  IRMD4
IRMD1:
     MOV   DPH,A
     MOVX  A,@DPTR
     CJNE  A,CurRow,IRMD3
     MOV   R6,A
     MOV   A,R5
     MOVX  @DPTR,A
     SJMP  IRMD4
IRMD3:
     CJNE  A,BtmRow,IRMD1
     MOV   A,RemRow
     CJNE  A,CurRow,IRMD1
     MOV   RemRow,R5
IRMD4:
     MOV   CurRow,R5
     MOV   A,R5
```

```
IRMD5:
   MOV    DPH,A
   MOVX   A,@DPTR
   CJNE   A,TopRow,IRMD8
   MOV    TopRow,DPH
   MOV    R6,A
   MOV    R7,DPL
   MOV    A,DPH
   JNB    WndActFlg,IRMD6
   MOV    DPTR,#WndWDB0+WDB_RowPag
   MOVX   @DPTR,A
   MOV    DPTR,#WndWDB1+WDB_RowPag
   MOVX   @DPTR,A
   SJMP   IRMD7
IRMD6:
   MOV    DPTR,#BgdMDB0+MDB_RowPag
   MOVX   @DPTR,A
   MOV    DPTR,#BgdMDB1+MDB_RowPag
   MOVX   @DPTR,A
IRMD7:
   MOV    DPH,A
   MOV    DPL,R7
   MOV    A,R6
   SJMP   IRMD5
IRMD8:
   CJNE   A,BtmRow,IRMD12
IRMD9:
   MOV    BtmRow,DPH
   CJNE   A,EndRow,IRMD11
IRMD10:
   MOV    RemRow,DPH
   MOV    EndRow,DPH
   MOV    ExtRow,A
   MOV    A,TrmRow
   CLR    EX0
   MOVX   @DPTR,A
   INC    DPL
   MOV    A,TrmOff
   MOVX   @DPTR,A
   SETB   EX0
   RET
```

```
IRMD11:
   MOV    DPH,A
   MOV    A,RemRow
   CLR    EX0
   MOVX   @DPTR,A
   INC    DPL
   MOV    A,RcbOff
   MOVX   @DPTR,A
   MOV    RemRow,DPH
   MOV    DPH,BtmRow
   MOV    A,TrmOff
   MOVX   @DPTR,A
   DEC    DPL
   MOV    A,TrmRow
   MOVX   @DPTR,A
   SETB   EX0
   MOV    DPH,RemRow
   MOVX   A,@DPTR
IRMD12:
   CJNE   A,EndRow,IRMD5
IRMD13:
   MOV    EndRow,DPH
   MOV    ExtRow,A
   MOV    A,DPH
   MOVX   @DPTR,A
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrlUpDsp:
;.............................................................................
; Scroll the display upward the given number of rows.
;
; In:    A                           number of rows to scroll
; Out:   VrtScrlFlg                  vertical scroll flag
;        SwbBit                      window/bgrd scroll flag
;        SudBit                      up/down scroll flag
;        VrtScrlCnt                  smooth scroll row count
;        main and window def blocks  top row page and smooth scroll ctrl
; Bad:   DPTR,P2,A,R0,R4,R7
;.............................................................................

   JNB    MsgActFlg,SUD1                   ;message area does not scroll
   RET
```

```
SUD1:
   MOV   R7,A                             ;save scroll count
   JB    AMDSCMBit,SUD4                   ;skip if smooth scrool
                                    ;jump scroll
   JB    VrtScrlFlg,$                     ;wait for scroll in progress
   LCALL HidCsr                           ;cursor hidden while scrolling
SUD2:                               ;call SetForScrlUp R7 times
   LCALL SetForScrlUp
   DJNZ  R7,SUD2
   LCALL PlcCsr                           ;put the cursor back
   MOV   A,R4                             ;A = top visible row
   JNB   WndActFlg,SUD3                   ;skip if not window
                                    ;scrolling in window
   MOV   DPTR,#WndWDB0+WDB_RowPag         ;set DPTR to point to one WDB
   MOVX  @DPTR,A                          ;row page is top visible
   INC   DPH                              ;now the other WDB
   MOVX  @DPTR,A
   RET
SUD3:                               ;scrolling in background
   MOV   DPTR,#BgdMDB0+MDB_RowPag         ;set DPTR to main def first row
   MOVX  @DPTR,A                          ;set this to top visible page
   MOV   DPL,#BgdMDB1.AN.OFST+MDB_RowPag  ;repeat for second main def
   MOVX  @DPTR,A
   RET
SUD4:                               ;smooth scrolling
   JNB   WndActFlg,SUD5                   ;skip if not in window
   JB    SwbBit,SUD6                      ;skip if scrolling in window now
   JB    VrtScrlFlg,$                     ;wait for scroll in progress
   SETB  SwbBit                           ;set flag for scroll in wnd
   SJMP  SUD7                             ;initiate scroll
SUD5:                               ;smooth scrolling in background
   JNB   SwbBit,SUD6                      ;skip if scrolling in bgrd
   JB    VrtScrlFlg,$                     ;wait for scroll in progress
   CLR   SwbBit                           ;clear flag for scroll in bgrd
   SJMP  SUD7                             ;initiate scroll
SUD6:                               ;scroll in progress
   JB    SudBit,SUD8                      ;skip if scrolling up in prog
   JB    VrtScrlFlg,$                     ;wait for scroll down in prog
SUD7:                               ;initiate scroll
   SETB  SudBit                           ;indicate scrolling up
SUD8:                               ;add to scroll count
   LCALL HidCsr                           ;cursor hidden while scrolling
   MOV   A,R7                             ;restore requested scroll count
```

```
   ADD   A,VrtScrlCnt                          ;get new total vert scrl count
   MOV   VrtScrlCnt,A
   JB    VrtScrlFlg,SUD11                      ;skip if scroll in progress
   JNB   CurMDBFlg,SUD9                        ;skip to select current MDB
   MOV   R0,#BgdMDB1.AN.OFST+MDB_Scrl          ;MDB1 if flag was set
   SJMP  SUD10
SUD9:
   MOV   R0,#BgdMDB0.AN.OFST+MDB_Scrl          ;MDB0 if flag was clear
SUD10:
   MOV   P2,#BgdMDB0.SR.PAGE                   ;background MDB page in P2
   MOV   A,ScrlByt                             ;set the scroll byte in MDB
   SETB  ACC.0
   MOVX  @R0,A
   JNB   VrtScrlFlg,$                          ;wait here for scroll to start
SUD11:                                   ;exit
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SetForScrlUp:
;.............................................................................
;  Sets the vertical scroll row variables for a scroll up.  This routine may
;  be called from an interrupt handler.
;
; In:    (none)
; Out:   R4                              top visible row
;        VisRow                          incremented
;        row control blocks              threading changed
;        TopRow                          advanced via thread
;        BtmRow                          changed to old RemRow
;        RemRow                          advanced via thread
; Bad:   DPTR,A
;.............................................................................

   INC   VisRow                                ;move the top visible down
   MOV   A,RcbOff                              ;DPL = offset of the field in the
   ADD   A,#RCB_RowOff                         ;    row control block which
   MOV   DPL,A                                 ;    points to offset of next RCB
   MOV   DPH,BtmRow                            ;DPH = bottom row page
   MOV   A,RcbOff                              ;A = offset of row control block
   CLR   EX0                                   ;no 8052 access for a moment
   MOVX  @DPTR,A                               ;set offset of next RCB
   DEC   DPL                                   ;now point to page of next RCB
   MOV   A,RemRow                              ;set page to rows remaining
   MOVX  @DPTR,A                           ;    beneath bottom
   MOV   BtmRow,A                          ;first of old rem is new bot
   MOV   DPH,A                             ;set DPTR to new bottom
   CJNE  A,EndRow,SFSU1
   SJMP  SFSU2
SFSU1:
   MOVX  A,@DPTR                           ;fetch page of following row
SFSU2:
   MOV   RemRow,A                          ;this is new remaining row start
   MOV   A,TrmRow                          ;set bottom RCB ptr to
   MOVX  @DPTR,A                           ;    termination RCB
   INC   DPL
   MOV   A,TrmOff
   MOVX  @DPTR,A
   SETB  EX0                               ;can allow 8052 access now
   DEC   DPL                               ;set DPTR to top row RCB
   MOV   DPH,TopRow
   MOVX  A,@DPTR                           ;old next row is new top row
   MOV   TopRow,A
   MOV   R4,A                              ;return new top row
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrlUpNewRow:
;..............................................................................
;  Scrolls the entire display up one row, inserting a blank row at the bottom
;  and deleting the row at the top.  Either a jump scroll or a smooth scroll
;  is done, depending on the Scroll Mode.
;..............................................................................

   JNB   MsgActFlg,SUNR1                   ;no scrolling in message row
   RET

SUNR1:
   MOV   A,ExtRow                          ;erase the extra row
   LCALL EraRow
   MOV   A,RcbOff                          ;R0 = offset of next RCB offset
   ADD   A,#RCB_RowPag
   MOV   R0,A
   MOV   A,EndRow                          ;check if last RCB is bottom
   CJNE  A,BtmRow,SUNR2                    ;skip if not
   MOV   A,ExtRow                          ;if so, make the extra
   MOV   EndRow,A                          ;    the new last row
```

```
   MOV   RemRow,A                          ;    and thus a remaining row
   SJMP  SUNR3
SUNR2:                               ;last RCB is not bottom
   MOV   P2,EndRow                         ;P2 is current end row
   MOV   A,RcbOff                          ;A = row control block offset
   INC   R0
   CLR   EX0                               ;no 8052 access for a moment
   MOVX  @R0,A                             ;set offset in old end row
   DEC   R0                                ;set page in old end row
   MOV   A,ExtRow                          ;    to point to extra row
   MOVX  @R0,A
   SETB  EX0                               ;8052 access OK now
   MOV   EndRow,A                          ;extra row is new end row
SUNR3:
   DEC   VisRow
   JB    AMDSCMBit,SUNR5                   ;skip if smooth scroll
   JB    VrtScrlFlg,$                      ;wait for scroll in progress
   LCALL HidCsr                            ;cursor hidden while scrolling
   LCALL SetForScrlUp                      ;can now set for scroll up
   LCALL PlcCsr                            ;and replace cursor
   MOV   A,R4                              ;A = top row page
   JNB   WndActFlg,SUNR4                   ;skip if in background
                                     ;jump scrolling in window
   MOV   DPTR,#WndWDB0+WDB_RowPag          ;set top row in one window
   MOVX  @DPTR,A                           ;    definition block 0
   INC   DPH                               ;now other WDB
   MOVX  @DPTR,A
   SJMP  SUNR12                            ;make new extra row
SUNR4:                               ;jump scrolling in background
   MOV   DPTR,#BgdMDB0+MDB_RowPag          ;set top row in main
   MOVX  @DPTR,A                           ;    definition block 0
   MOV   DPL,#BgdMDB1.AN.OFST+MDB_RowPag   ;repeat for main definition
   MOVX  @DPTR,A                           ;    block 1
   SJMP  SUNR12                            ;make new extra row
SUNR5:
   JNB   WndActFlg,SUNR6                   ;skip if scrolling in background
                                     ;smooth scrolling in window
   JB    SwbBit,SUNR7                      ;skip if window scroll in prog
   JB    VrtScrlFlg,$                      ;wait for scroll in progress
   SETB  SwbBit                            ;set scrolling in window flag
   SJMP  SUNR8
SUNR6:                               ;smooth scrolling in background
   JNB   SwbBit,SUNR7                      ;skip if bgrd scroll in prog
```

```
    JB    VrtScrlFlg,$                        ;wait for scroll in prog
    CLR   SwbBit                              ;clear to indicate bgrd scroll
    SJMP  SUNR8
SUNR7:                                  ;same area scroll in progress
    JB    SudBit,SUNR9                        ;skip if same type of scroll
    JB    VrtScrlFlg,$                        ;wait for scroll in progress
SUNR8:                                  ;initiate scroll
    SETB  SudBit                              ;mark scroll up in progress
SUNR9:
    LCALL HidCsr                              ;cursor hidden while scrolling
    INC   VrtScrlCnt                          ;one more row to scroll
    JNB   CurMDBFlg,SUNR10                    ;skip to correct main def
    MOV   R0,#BgdMDB1.AN.OFST+MDB_Scrl        ;R0 = main def offset
    SJMP  SUNR11
SUNR10:
    MOV   R0,#BgdMDB0.AN.OFST+MDB_Scrl        ;R0 = main def offset
SUNR11:
    MOV   P2,#BgdMDB0.SR.PAGE                 ;P2 = main def page
    MOV   A,ScrlByt                           ;set scroll byte in main def
    SETB  ACC.0
    MOVX  @R0,A
    JNB   VrtScrlFlg,$                        ;wait for scroll to start
    JB    VrtScrlNewFlg,$                     ;wait for beginning row free
    SETB  VrtScrlNewFlg                       ;mark beginning row not free
SUNR12:
    MOV   A,RcbOff                            ;R0 = offset of nex row page
    ADD   A,#RCB_RowPag
    MOV   R0,A
    MOV   A,EndRow
    CJNE  A,BtmRow,SUNR13
    SJMP  SUNR14
SUNR13:
    MOV   P2,A
    MOVX  @R0,A
    INC   R0
    MOV   A,RcbOff
    MOVX  @R0,A
    DEC   R0
SUNR14:
    MOV   A,BgnRow                            ;old beginning row becomes
    MOV   ExtRow,A                            ;    the extra row
    MOV   P2,A                                ;P2 = new extra row
    MOVX  A,@R0                               ;following row becomes new
```

```
    MOV   BgnRow,A                            ;    beginning row
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrlDnDsp:
;.............................................................................
; Scrolls the display downward the given number of rows.
;
; In:   A                              number of rows to scroll
; Out:  VrtScrlFlg                     vertical scroll flag
;       SwbBit                         window/bgrd scroll flag
;       SudBit                         up/down scroll flag
;       VrtScrlCnt                     smooth scroll row count
;       main and window def blocks     top row page and smooth scroll ctrl
; Bad:  DPTR,P2,A,R0,R4,R7
;.............................................................................
    JNB   MsgActFlg,SDD1                      ;message area does not scroll
    RET
SDD1:
    MOV   R7,A                                ;save scroll count
    JB    AMDSCMBit,SDD4                      ;skip if smooth scroll
                                        ;jump scroll
    JB    VrtScrlFlg,$                        ;wait for scroll in progress
    LCALL HidCsr                              ;cursor hidden while scrolling
SDD2:                                   ;call SetForScrDn R7 times
    LCALL SetForScrlDn
    LCALL SetAftScrlDn
    DJNZ  R7,SDD2
    LCALL PlcCsr                              ;put the cursor back
    MOV   A,R4                                ;A = top visible row
    JNB   WndActFlg,SDD3                      ;skip if not in window
                                        ;jump scrolling in window
    MOV   DPTR,#WndWDB0+WDB_RowPag            ;set DPTR to point to one WDB
    MOVX  @DPTR,A                             ;row page is top visible
    INC   DPH                                 ;now other WDB
    MOVX  @DPTR,A
    RET
SDD3:                                   ;jump scrolling in background
    MOV   DPTR,#BgdMDB0+MDB_RowPag            ;set DPTR to main def first row
    MOVX  @DPTR,A                             ;set this to top visible
    MOV   DPL,#BgdMDB1.AN.OFST+MDB_RowPag     ;repeat for second main def
    MOVX  @DPTR,A
    RET
```

```
SDD4:                                          ;smooth scrolling
   JNB    WndActFlg,SDD5                              ;skip if not in window
                                               ;smooth scrolling in window
   JB     SwbBit,SDD6                                 ;skip if scrolling in window now
   JB     VrtScrlFlg,$                                ;wait for scroll in progress
   SETB   SwbBit                                      ;set flag fro scroll in wnd
   SJMP   SDD7                                        ;initiate scroll
SDD5:                                          ;smooth scrolling in background
   JNB    SwbBit,SDD6                                 ;skip if scrolling in bgrd now
   JB     VrtScrlFlg,$                                ;wait for scroll in progress
   CLR    SwbBit                                      ;set flag for scroll in bgrd
   SJMP   SDD7                                        ;initiate scroll
SDD6:                                          ;scroll in progress
   JNB    SudBit,SDD8                                 ;skip if scrolling down in prog
   JB     VrtScrlFlg,$                                ;wait for scroll in progress
SDD7:                                          ;initiate scroll
   CLR    SudBit                                      ;indicate scrolling down
SDD8:                                          ;add to scroll count
   LCALL  HidCsr                                      ;cursor hidden while scrolling
   MOV    A,R7                                        ;restore requested scroll count
   ADD    A,VrtScrlCnt                                ;get new total vert scrl count
   MOV    VrtScrlCnt,A
   JB     VrtScrlFlg,SDD13                            ;skip if scroll in progress
   LCALL  SetForScrlDn                                ;prepare new top row
   JNB    WndActFlg,SDD10                             ;jump if not in window
   MOV    R1,#TOWSftLoInd                             ;setup for write to Am8052 reg
   MOV    R3,#WndWDB0.AN.OFST+WDB_RowPag              ;offset into WDB top row ptr
   JB     CurWDBFlg,SDD9                              ;select alternate WDB page
   SETB   CurWDBFlg
   MOV    R2,#WndWDB1.SR.PAGE
   SJMP   SDD12
SDD9:
   CLR    CurWDBFlg
   MOV    R2,#WndWDB0.SR.PAGE
   SJMP   SDD12
SDD10:
   MOV    R1,#TOPSftLoInd                             ;setup for write to Am8052
   MOV    R2,#BgdMDB0.SR.PAGE                         ;backgrd MDB page in P2
   JB     CurMDBFlg,SDD11                             ;select alternate MDB top row off
   SETB   CurMDBFlg
   MOV    R3,#BgdMDB1.AN.OFST+MDB_RowPag              ;MDB1 if flag was set
   SJMP   SDD12
```

```
SDD11:
   CLR    CurMDBFlg
   MOV    R3,#BgdMDB0.AN.OFST+MDB_RowPag              ;MDB2 if flag was clear
SDD12:
   MOV    A,R4                                        ;new top visible row
   MOV    DPH,R2
   MOV    DPL,R3
   MOVX   @DPTR,A
   DEC    R3
   DEC    R3
   LCALL  WrAm8052Reg
   MOV    P2,#BgdMDB0.SR.PAGE                         ;MDB page in P2
   MOV    A,ScrlByt                                   ;update scroll byte in both
   SETB   ACC.0                                       ;    MDB's
   MOV    R0,#BgdMDB0.AN.OFST+MDB_Scrl
   MOV    R1,#BgdMDB1.AN.OFST+MDB_Scrl
   CLR    EX0                                         ;no 8052 access while doing this
   MOVX   @R0,A
   MOVX   @R1,A
   SETB   EX0
   JNB    VrtScrlFlg,$
SDD13:                                              ;exit
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SetForScrlDn:
;............................................................................
;  Sets the vertical scroll row variables for a scroll down.  This routine may
;  be called from an interrupt handler.
;
; In:    (none)
; Out:   R4                             top visible row
;        VisRow                         decremented
;        row control blocks             threading changed
;        TopRow                         moved up via thread
;        BtmRow                         moved up via thread
;        RemRow                         changed to old reamining row
; Bad:   DPTR,A
;............................................................................

   DEC    VisRow                                      ;move the top visible up
   MOV    A,RcbOff                                    ;DPL = offset of the field in the
   ADD    A,#RCB_RowPag                               ;    row control block which
   MOV   DPL,A                       ;    points to offset of next RCB
   MOV   A,BgnRow                    ;A = beginning row page
SFSD1:
   MOV   DPH,A                       ;DPH = row page
   MOVX  A,@DPTR                     ;fetch the next row page
   CJNE  A,TopRow,SFSD1              ;cont until the top row is next
   MOV   R4,DPH                      ;make row before top
   MOV   TopRow,R4                   ;    the new top row
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SetAftScrlDn:
;..............................................................................
; Sets the vertical scroll variables after a scroll down.  This routine may
; be called from an interrupt routine.
;..............................................................................

   MOV   A,RcbOff
   ADD   A,#RCB_RowPag
   MOV   DPL,A
   MOV   A,TopRow
SASD1:
   MOV   DPH,A
   MOVX  A,@DPTR
   CJNE  A,BtmRow,SASD1
   MOV   BtmRow,DPH
   XCH   A,TrmRow
   CLR   EX0
   MOVX  @DPTR,A
   INC   DPL
   XCH   A,TrmOff
   MOVX  @DPTR,A
   SETB  EX0
   XCH   A,TrmOff
   XCH   A,TrmRow
   MOV   DPH,A
   MOV   A,RcbOff
   MOVX  @DPTR,A
   DEC   DPL
   MOV   A,RemRow
   MOVX  @DPTR,A
   MOV   RemRow,DPH
   RET
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrlLtDsp:
;..............................................................................
; Scrolls the active display (background or message) left the given number of
; columns.
;
; In:    A                               number of columns to scroll
; Out:   HrzScrlFlg
;        HrzDirFlg
;        HrzDspFlg
;        HrzPxlShf
;        HrzFrmCnt
;        HrzFrmSet
;        HrzScrlCnt
;        (see also ScrlLtOne)
; Bad:   A,R0,R1,R2,R3,R4,R5,R7
;..............................................................................

   JNB   WndActFlg,SLD1                     ;can't scroll horz in window
   RET
SLD1:
   MOV   R7,A                               ;save scroll count in R7
   JB    AMDSCMBit,SLD3                     ;skip if smooth scroll
   JB    HrzScrlFlg,$                       ;wait for scroll in progress
   LCALL HidCsr                             ;hide cursor while scrolling
   JB    MsgActFlg,SLD2
   JNB   WndVisFlg,SLD2
   LCALL HidWnd
   LCALL DlyTilEndFrm
SLD2:                                 ;call ScrlLtOne R7 times
   LCALL ScrlLtOne
   DJNZ  R7,SLD2
   JB    MsgActFlg,SLD2a
   LCALL SetWndPos
   JNB   WndVisFlg,SLD2a
   LCALL ShwWnd
SLD2a:
   LCALL PlcCsr                             ;replace the cursor
   RET
SLD3:                                 ;smooth scroll
   JNB   MsgActFlg,SLD4                     ;skip if not message area
                                      ;message area is active
   JB    HrzDspFlg,SLD5                     ;skip if scrolling message area
```

```
    JB    HrzScrlFlg,$                ;wait for scroll in progress
    SETB  HrzDspFlg                   ;mark scrolling in message
    SJMP  SLD6                        ;set scroll rates
SLD4:                             ;background is active
    JNB   HrzDspFlg,SLD5              ;skip if scrolling in bgrd
    JB    HrzScrlFlg,$                ;wait for scroll in progress
    CLR   HrzDspFlg                   ;mark now scrollin in bgrd
    SJMP  SLD6                        ;set scroll rates
SLD5:
    JNB   HrzDirFlg,SLD7              ;skip if now scrolling left
    JB    HrzScrlFlg,$                ;wait for scroll in progress
SLD6:
    CLR   HrzDirFlg                   ;mark scrolling left now
SLD7:
    LCALL HidCsr
    JB    MsgActFlg,SLD7a
    JNB   WndVisFlg,SLD7a
    LCALL HidWnd
SLD7a:
    MOV   A,ScrlByt                   ;A = old scroll byte
    ANL   A,#SCRL_RAT_MASK            ;extract scroll rate bits
    RL    A                           ;move rate to upper nibble
    SWAP  A                           ;move rate to lower nibble
    JBC   ACC.3,SLD10                 ;skip if pixel every n frames
                                  ;scrolling n pixels per frame
    MOV   HrzFrmSet,#1                ;mark num frames to next move
    INC   A                           ;convert to number per frame
    JNB   AMDDWMBit,SLD9              ;skip if normal width
                                  ;compressed display
    CLR   C                           ;check for 7 or 8 per frame
    SUBB  A,#7
    JC    SLD8                        ;skip if 6 or fewer
    MOV   A,#-1                       ;limit to 6 for frame
SLD8:
    ADD   A,#7                        ;convert back to pixels per frame
SLD9:
    MOV   HrzPxlShf,A                 ;set this in the variable
    SJMP  SLD11                       ;initiate the scroll
SLD10:                            ;scrolling 1 pixel every n frames
    INC   A                           ;A = number of frames
    MOV   HrzFrmSet,A                 ;mark num frames to next move
    MOV   HrzPxlShf,#1                ;mark single pixel shift
```

```
SLD11:
    CLR   ET0                                  ;ensure no interruptions
    JB    HrzScrlFlg,SLD12                     ;skip if scroll in progress
                                           ;now starting a scroll
    MOV   HrzFrmCnt,#1                         ;initiate on next frame
    SETB  HrzScrlFlg                           ;mark scroll in progress
SLD12:
    MOV   A,R7                                 ;add new request to old count
    ADD   A,HrzScrlCnt
    MOV   HrzScrlCnt,A
    SETB  ET0                                  ;allow horz smooth scroll intr
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrlLtOne:
;...............................................................................
; Scrolls the active display (background or message) left one character
; position.  This routine may be called from an interrupt handler.
;
; In:   (none)
; Out:  VisCol                                 incremented
;       row control blocks
;       attribute of old leftmost visibles
; Bad:  DPTR,A,R0,R1,R2,R3,R4,R5
;...............................................................................

    JNB   MsgActFlg,SLO1                       ;skip if scrolling bgrd
                                           ;scroll the message
    MOV   DPTR,#MsgRCB+RCB_2nd+SEG_NumHid      ;A = 2nd seg, number hidden
    MOVX  A,@DPTR                              ;    in message area
    MOV   R1,A                                 ;save old number hidden in R1
    RL    A                                    ;double old number hidden
    XCH   A,R1                                 ;old number back in A
    INC   A                                    ;one more hidden column
    CLR   EX0                                  ;no 8052 access while changing
    MOVX  @DPTR,A                              ;    to new hidden col count
    INC   DPL                                  ;now decrement number visible
    MOVX  A,@DPTR                              ;    in this segment
    DEC   A
    MOVX  @DPTR,A
    MOV   DPH,#MsgAtrBuf.SR.PAGE               ;now set the ignore bit
    MOV   DPL,R1                               ;    in the attribute of the
    MOVX  A,@DPTR                              ;    previously leftmost visible
    SETB  ACC.5                         ;    character
    MOVX  @DPTR,A
    SETB  EX0                           ;now allow 8052 access
    INC   VisCol                        ;update horz scroll position
    RET
SLO1:                             ;scroll the background
    MOV   DPH,CurRow                    ;use current row
    MOV   DPL,#BgdRCB0.AN.OFST+RCB_2nd+SEG_NumVis
    MOVX  A,@DPTR                       ;get number visible in 2nd seg
    JNZ   SLO2                          ;skip if not zero
    MOV   DPL,#BgdRCB0.AN.OFST+RCB_3rd+SEG_NumVis
    MOVX  A,@DPTR                       ;get number visible in 3rd seg
SLO2:
    DEC   A                             ;reduce number visible
    MOV   R5,A                          ;keep number visible in R5
    DEC   DPL                           ;point back to number hidden
    MOV   R0,DPL                        ;save this ptr in R0
    MOVX  A,@DPTR                       ;A = old number hidden
    MOV   R4,A                          ;R4 = old number hidden
    INC   R4                            ;R4 = new number hidden
    MOV   A,VisCol                      ;horz scroll position
    RL    A                             ;A = double above for attr offset
    MOV   R1,A                          ;save old attrib offset in R1
    MOV   R2,#BgdRCB0.SR.PAGE           ;R2 is ptr to first RCB
    MOV   R3,#BgdAtrBuf0.SR.PAGE        ;R3 is ptr to first attribute
SLO3:                             ;row loop point
    MOV   DPH,R2                        ;DPTR points to number hidden
    MOV   DPL,R0                        ;    in this row
    MOV   A,R4                          ;A = new number hidden
    CLR   EX0                           ;no 8052 access while changing
    MOVX  @DPTR,A                       ;set new number hidden
    INC   DPL                           ;point to number visible
    MOV   A,R5                          ;set new number visible
    MOVX  @DPTR,A
    MOV   DPH,R3                        ;DPTR points to attribute of
    MOV   DPL,R1                        ;    old leftmost visible
    MOVX  A,@DPTR                       ;change to ignore this character
    SETB  ACC.5
    MOVX  @DPTR,A
    SETB  EX0                           ;OK for 8052 access now
    INC   R3                            ;next row control block
    INC   R2                            ;next block of attributes
    CJNE  R2,#BgdRCB30.SR.PAGE+1,SLO3   ;continue until al 31 are done
```

```
    INC   VisCol                             ;update horz scroll position
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrlRtDsp:
;.............................................................................
; Scrolls the active display (background or message) right the given number of
; columns.
;
; In:    A                          number of columns to scroll
; Out:   HrzScrlFlg
;        HrzDirFlg
;        HrzDspFlg
;        HrzPxlShf
;        HrzFrmCnt
;        HrzFrmSet
;        HrzScrlCnt
;        (see also ScrlRtOne)
; Bad:   A,R0,R1,R2,R3,R4,R5,R7
;.............................................................................

    JNB   WndActFlg,SRD1                     ;can't scroll horz in window
    RET
SRD1:
    MOV   R7,A                               ;save scroll count in R7
    JB    AMDSCMBit,SRD3                     ;skip if smooth scroll
    JB    HrzScrlFlg,$                       ;wait for scroll in progress
    LCALL HidCsr
    JB    MsgActFlg,SRD2
    JNB   WndVisFlg,SRD2
    LCALL HidWnd
    LCALL DlyTilEndFrm
SRD2:                                   ;call ScrlRtOne R7 times
    LCALL ScrlRtOne
    DJNZ  R7,SRD2
    JB    MsgActFlg,SRD2a
    LCALL SetWndPos
    JNB   WndVisFlg,SRD2a
    LCALL ShwWnd
SRD2a:
    LCALL PlcCsr                             ;replace the currsor
    RET
```

```
SRD3:
   JNB   MsgActFlg,SRD4               ;skip if background active
                                ;scrolling in message
   JB    HrzDspFlg,SRD5               ;skip if scrolling in msg
   JB    HrzScrlFlg,$                 ;wait for scroll in progress
   SETB  HrzDspFlg                    ;mark scrolling in msg
   SJMP  SRD6                         ;set scroll rates
SRD4:                           ;scrolling in background
   JNB   HrzDspFlg,SRD5               ;skip if scrolling in background
   JB    HrzScrlFlg,$                 ;wait for scroll in progress
   CLR   HrzDspFlg                    ;mark scrolling in bgrd
   SJMP  SRD6                         ;set scroll rates
SRD5:                           ;now scrolling
   JB    HrzDirFlg,SRD7               ;skip if now scrolling right
   JB    HrzScrlFlg,$                 ;wait for scroll in progress
SRD6:                           ;initiate scrolling
   SETB  HrzDirFlg                    ;mark scrolling right
SRD7:
   LCALL HidCsr
   JB    MsgActFlg,SRD7a
   JNB   WndVisFlg,SRD7a
   LCALL HidWnd
SRD7a:
   MOV   A,ScrlByt                    ;fetch scroll byte
   ANL   A,#SCRL_RAT_MASK             ;get rate in lower nibble
   RL    A
   SWAP  A
   JBC   ACC.3,SRD10                  ;skip if 1 pixel per n frames
                                ;scrolling n pixels per frame
   MOV   HrzFrmSet,#1                 ;1 frame per scroll
   INC   A                            ;A = number of pixels per frame
   JNB   AMDDWMBit,SRD9               ;skip if normal
                                ;compressed
   CLR   C                            ;check for rate of 7 or 8
   SUBB  A,#7
   JC    SRD8
   MOV   A,#-1                        ;limit rate to 6
SRD8:
   ADD   A,#7                         ;convert back to rate
SRD9:
   MOV   HrzPxlShf,A                  ;set pixels per frame
   SJMP  SRD11                        ;initiate scroll
```

```
SRD10:                                         ;scrolling 1 pixel per n frames
   INC   A                                           ;A = frames per pixel
   MOV   HrzFrmSet,A                                 ;set number of frames per scrl
   MOV   HrzPxlShf,#1                                ;always one pixel shifted
SRD11:                                         ;start scrolling
   CLR   ET0                                         ;ensure no interruptions
   JB    HrzScrlFlg,SRD12                            ;skip if scroll in progress
                                               ;now starting a scroll
   MOV   HrzFrmCnt,#1                                ;initiate on next frame
   SETB  HrzScrlFlg                                  ;mark scroll in progress
SRD12:
   MOV   A,R7                                        ;add new request to old count
   ADD   A,HrzScrlCnt
   MOV   HrzScrlCnt,A
   SETB  ET0                                         ;allow horz smooth scroll intr
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ScrlRtOne:
;.............................................................................
; Scrolls the active display (background or message) right one character
; position.  This routine may be called from an interrupt handler.
;
; In:   (none)
; Out:  VisCol                                 decremented
;       row control blocks
;       attribute of old rightmost ignored
; Bad:  DPTR,A,R0,R1,R2,R3,R4,R5
;.............................................................................

   DEC   VisCol                                      ;visible column decremented
   JNB   MsgActFlg,SR01                              ;skip if not in msg
                                               ;scrolling message
   MOV   DPTR,#MsgRCB+RCB_2nd+SEG_NumHid             ;ptr to number hidden, 2nd seg
   MOVX  A,@DPTR                                     ;A = old number hidden
   DEC   A                                           ;reduce number hidden
   MOV   R1,A                                        ;R1 = old number hidden
   RL    A                                           ;double for attr offset
   XCH   A,R1                                        ;save attribute offset in R1
   CLR   EX0                                         ;no 8052 access while changing
   MOVX  @DPTR,A
   INC   DPL                                         ;increment number visible
   MOVX  A,@DPTR
```

```
    INC    A
    MOVX   @DPTR,A
    MOV    DPH,#MsgAtrBuf.SR.PAGE         ;now point to attribute
    MOV    DPL,R1                         ;    of old rightmost hidden
    MOVX   A,@DPTR
    CLR    ACC.5                          ;make it visible
    MOVX   @DPTR,A
    SETB   EX0                            ;OK for 8052 access now
    RET
SR01:                                  ;scrolling in background
    MOV    DPH,CurRow                     ;use current row (any would do)
    MOV    DPL,#BgdRCB0.AN.OFST+RCB_3rd+SEG_NumHid
    MOVX   A,@DPTR                        ;check for hidden in 3rd seg
    JNZ    SR02                           ;skip if some hidden there
    MOV    DPL,#BgdRCB0.AN.OFST+RCB_2nd+SEG_NumHid
    MOVX   A,@DPTR                        ;else use 2nd segment
SR02:
    MOV    R0,DPL                         ;save the pointer to hidden
    DEC    A                              ;decrement the number hidden
    MOV    R4,A                           ;save number in R4
    MOV    A,VisCol                       ;horz scroll position
    RL     A                              ;R1 = offset of attribute
    MOV    R1,A                           ;    for new first visible
    INC    DPL                            ;point to number visible
    MOVX   A,@DPTR
    INC    A                              ;R5 = new number visible
    MOV    R5,A
    MOV    R2,#BgdRCB0.SR.PAGE            ;R2 = first RCB
    MOV    R3,#BgdAtrBuf0.SR.PAGE         ;R3 = first attribute vlock
SR03:                                  ;scroll row loop
    MOV    DPH,R2                         ;DPTR->number hidden in RCB
    MOV    DPL,R0
    MOV    A,R4                           ;A = new number hidden
    CLR    EX0                            ;no 8052 access while changing
    MOVX   @DPTR,A                        ;update number hidden
    INC    DPL                            ;point to number visible
    MOV    A,R5                           ;set that from R5
    MOVX   @DPTR,A
    MOV    DPH,R3                         ;point to attribute of new 1st
    MOV    DPL,R1                         ;    visible
    MOVX   A,@DPTR                        ;mark it visible
    CLR    ACC.5
    MOVX   @DPTR,A
```

```
    SETB   EX0                                            ;OK for 8052 access now
    INC    R3                                             ;next attribute block
    INC    R2                                             ;next RCB
    CJNE   R2,#BgdRCB30.SR.PAGE+1,SR03                    ;continue through 31st row
    RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SetWndPos:                                          ; Set new window position

; Determines the current window position and sets the background's row control
; block segments accordingly.
;
; In:     VisCol                              background horizontal scroll position
; Out:    BgdRCB0-BgdRCB30                    segments updated
;         WndCol                              window position relative to background
;         ColOff                              updated when window is active
; Bad:    A,DPTR,R0,R1,R2,R3,R4,R5,R6,R7,PSW

    MOV    DPTR,#BgdVarBuf+(VisCol-CurAtr)
    MOVX   A,@DPTR
    JB     MsgActFlg,SWP0
    JB     WndActFlg,SWP0
    MOV    A,VisCol
SWP0:
    MOV    R0,A
    JNB    AMDDWMBit,SWP1                                 ; Jump if normal mode
    MOV    A,#68                                          ; Compressed window position
    SJMP   SWP2                                           ;    and continue
SWP1:
    MOV    A,#28                                          ; Normal window position
SWP2:
    ADD    A,R0                                           ; Compute actual total offset
    CLR    ACC.0                                          ;    aligned on word boundary
    MOV    WndCol,A                                       ;    and keep it
    SUBB   A,R0                                           ; Compute actual visible offset
    MOV    R1,A                                           ;    and keep it
    INC    A                                              ; Add one for invisible function
    MOV    DPTR,#WndWDB0+WDB_BgnCol
    MOVX   @DPTR,A
    MOV    DPTR,#WndWDB1+WDB_BgnCol
    MOVX   @DPTR,A
    JNB    WndActFlg,SWP3                                 ; Jump if window not active
    MOV    ColAdd,A                                       ;    else save offset
```

```
SWP3:
    MOV   R2,#40                        ; Set visible width of window
    ADD   A,R2
    DEC   A
    MOV   DPTR,#WndWDB0+WDB_EndCol
    MOVX  @DPTR,A
    MOV   DPTR,#WndWDB1+WDB_EndCol
    MOVX  @DPTR,A
    MOV   A,#BgdChrBuf0.AN.OFST         ; Start of bgd chr buffer
    ADD   A,WndCol                      ;    plus total offset is
    MOV   R4,A                          ;    3rd seg chr ptr offset;
    ADD   A,R2                          ;    plus 3rd seg width is
    MOV   R6,A                          ;    4th seg chr ptr offset
    MOV   A,#BgdAtrBuf0.AN.OFST         ; Start of bgd atr buffer
    ADD   A,WndCol                      ;    plus twice
    ADD   A,WndCol                      ;    total offset is
    MOV   R5,A                          ;    2nd seg atr ptr offset;
    ADD   A,R2                          ;    plus twice
    ADD   A,R2                          ;    3rd seg width is
    MOV   R7,A                          ;    4th seg atr ptr offset
    CLR   C                             ; Clear for below
    MOV   A,#128                        ; Width of background buffer
    SUBB  A,WndCol                      ;    minus total offset to window
    SUBB  A,R2                          ;    minus width of window is
    MOV   R3,A                          ;    width of 4th segment
    MOV   DPH,#BgdRCB0.SR.PAGE          ; Start at first RCB in memory
SWP4:                                 ; For each background row control block
    CLR   EX0                           ; No interference from Am8052
    MOV   DPL,#BgdRCB0.AN.OFST+RCB_2nd+SEG_NumHid
    MOV   A,R0                          ; Horizontal scroll offset into
    MOVX  @DPTR,A                       ;    2nd seg hidden count
    INC   DPL
    MOV   A,R1                          ; Offset to window boundary into
    MOVX  @DPTR,A                       ;    2nd seg visible count

; The second segment's character and attribute pointers never change.

    MOV   DPL,#BgdRCB0.AN.OFST+RCB_3rd+SEG_NumHid
    CLR   A                             ; Zero into
    MOVX  @DPTR,A                       ;    3rd seg hidden count
    INC   DPL
    MOV   A,R2                          ; Width of window into
    MOVX  @DPTR,A                       ;    3rd seg visible count
```

```
    MOV   DPL,#BgdRCB0.AN.OFST+RCB_3rd+SEG_ChrOff
    MOV   A,R4                          ; Even boundary offset into
    MOVX  @DPTR,A                       ;    3rd seg character pointer
    MOV   DPL,#BgdRCB0.AN.OFST+RCB_3rd+SEG_AtrOff
    MOV   A,R5                          ; Corresponding offset into
    MOVX  @DPTR,A                       ;    3rd seg attribute pointer
    MOV   DPL,#BgdRCB0.AN.OFST+RCB_4th+SEG_NumVis
    MOV   A,R3                          ; Remaining character count into
    MOVX  @DPTR,A                       ;    4th seg visible count

; The fourth segment's hidden count is zero and never changed.

    MOV   DPL,#BgdRCB0.AN.OFST+RCB_4th+SEG_ChrOff
    MOV   A,R6                          ; Next boundary offset into
    MOVX  @DPTR,A                       ;    4th seg character pointer
    MOV   DPL,#BgdRCB0.AN.OFST+RCB_4th+SEG_AtrOff
    MOV   A,R7                          ; Corresponding offset into
    MOVX  @DPTR,A                       ;    4th seg attribute pointer
    SETB  EX0                           ; Allow Am8052 bus requests
    INC   DPH                           ; Next row control block
    MOV   A,DPH                         ; Check it and
    CJNE  A,#BgdRCB30.SR.PAGE+1,SWP4    ;    jump if not finished

    RET                                 ; Exit

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
RdAm8052Reg:
;............................................................................
; Reads from the specified register in the Am8052.
;
; In:    R1                       Am8052 register number
; Out:   R2                       high byte of value read
;        R3                       low byte of value read
; Bad:   A,DPTR
;............................................................................

    CLR   EX1                     ;ensure no Am8052 interruptions
    CLR   EX0
    CLR   Am8052XfrFlg            ;give Am8052 address strobe
    MOV   DPTR,#Am8052Ptr         ;point to Am8052 control reg
    MOV   A,R1                    ;indicate register to be read
    MOVX  @DPTR,A
    MOV   DPTR,#Am8052RegLo       ;point to low data byte
```

```
        MOVX  A,@DPTR                         ;read low data byte
        MOV   R3,A
        DEC   DPL                             ;point to high data byte
        MOVX  A,@DPTR                         ;read high data byte
        MOV   R2,A
        SETB  Am8052XfrFlg                    ;remove Am8052 address strobe
        SETB  EX0                             ;allow Am8052 interrupts
        SETB  EX1
        RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
WrAm8052Reg:
;...............................................................................
; Writes the given value to the specified register in the Am8052.
;
; In:     R1                          Am8052 register number
;         R2                          high byte of value to be written
;         R3                          low byte of value to be written
; Out:    (none)
; Bad:    A,DPTR
;...............................................................................

        CLR   EX1                             ;ensure no Am8052 interruptions
        CLR   EX0
        CLR   Am8052XfrFlg                    ;give address strobe to 8052
        MOV   DPTR,#Am8052Ptr                 ;set pointer to 8052 control
        MOV   A,R1                            ;select register
        MOVX  @DPTR,A
        MOV   DPTR,#Am8052RegHi               ;set ptr to 8052 data
        MOV   A,R2                            ;set high byte
        MOVX  @DPTR,A
        INC   DPL                             ;set ptr to low data
        MOV   A,R3                            ;set low byte
        MOVX  @DPTR,A
        SETB  Am8052XfrFlg                    ;remove 8052 address strobe
        SETB  EX0                             ;allow Am8052 interrupts
        SETB  EX1
        RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
SetRowFntRdfPtr:
;...............................................................................
;  Sets the first 15 visible row redefinition block pointers to the
;  font loading redifinition blocks
;
;  inputs   none
;  outputs  none
;...............................................................................

    MOV   P2,TopRow
    MOV   R2,#FntRRB0.AN.OFST
    MOV   R3,#FntRRB0.SR.PAGE
    MOV   R0,#BgdRCB0.AN.OFST+RCB_BgdRdfPag
    MOV   R1,#BgdRCB0.AN.OFST+RCB_RowPag
    MOV   R4,#15
SRFRP1:
    MOV   A,R3
    MOVX  @R0,A                                   ; Change page pointer in RCB
    INC   R0
    INC   R3
    MOV   A,R2
    MOVX  @R0,A                                   ; Change offset of pointer in RCB
    DEC   R0
    MOVX  A,@R1
    MOV   P2,A
    DJNZ  R4,SRFRP1
    RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SetRowNmlRdfPtr:
;...............................................................................
;  Sets the first 15 visible row redefinition block pointers to the
;  normal redifinition blocks
;
;  inputs   none
;  outputs  none
;...............................................................................

    MOV   P2,TopRow                               ;
    MOV   R0,#BgdRCB0.AN.OFST+RCB_BgdRdfPag
    MOV   R1,#BgdRCB0.AN.OFST+RCB_RowPag
    MOV   R2,#15                                  ; Number of rows to update
```

```
SRNRP1:
   MOV   A,#NrmRRB.SR.PAGE
   MOVX  @R0,A                          ; Change page pointer in RCB
   INC   R0
   MOV   A,#NrmRRB.AN.OFST
   MOVX  @R0,A                          ; Change offset of pointer in RCB
   DEC   R0
   MOVX  A,@R1
   MOV   P2,A
   DJNZ  R2,SRNRP1
   RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
WrFntCel:
;.............................................................................
; Writes to a single character generator cell the pattern specified in the
; parameter buffer.
;
; In:    A                        font select (=0 normal, <>0 compressed)
;        PrmCnt                   parameter count
;        PrmBuf                   list of parameters
;.............................................................................


   MOV   R4,A
   MOV   R6,CsrSiz
   MOV   CsrSiz,#0FFH
   LCALL ChgCsrSiz
   MOV   CsrSiz,R6
   MOV   P2,#FntRRB0.SR.PAGE
   MOV   R6,#15
   MOV   R1,#PrmBuf
   MOV   R2,PrmCnt
   MOV   A,@R1
   INC   R1
   DEC   R2
   MOV   R3,A
   MOV   A,@R1
   INC   R1
   DEC   R2
   JZ    WFC2
   MOV   R5,A
WFC1:
   MOV   R0,#FntRRB0.AN.OFST+RRB_ApHi_SpcsHi
```

```
   CLR   A
   MOVX  @R0,A
   MOV   R0,#FntRRB0.AN.OFST+RRB_ApLo_SbcsHi
   CLR   A
   MOVX  @R0,A
   INC   P2
   DEC   R6
   DJNZ  R5,WFC1
WFC2:
   MOV   A,R2
   JZ    WFC4
WFC3:
   MOV   A,@R1
   INC   R1
   MOV   R7,A
   ANL   A,#0F8H
   RR    A
   MOV   R0,#FntRRB0.AN.OFST+RRB_ApHi_SpcsHi
   MOVX  @R0,A
   MOV   A,R7
   ANL   A,#07H
   SWAP  A
   MOV   R0,#FntRRB0.AN.OFST+RRB_ApLo_SbcsHi
   MOVX  @R0,A
   INC   P2
   DEC   R6
   DJNZ  R2,WFC3
WFC4:
   CLR   A
   MOV   R0,#FntRRB0.AN.OFST+RRB_ApHi_SpcsHi
   MOVX  @R0,A
   CLR   A
   MOV   R0,#FntRRB0.AN.OFST+RRB_ApLo_SbcsHi
   MOVX  @R0,A
   INC   P2
   DJNZ  R6,WFC4
   MOV   DPTR,#BgdFncChr0                ; Set character cell value in
   MOV   A,R3                            ;    dummy character
   MOVX  @DPTR,A
   MOV   A,R4
   JZ    WFC5
   MOV   R6,#044H
   MOV   R7,#010H
   SJMP  WFC6
```

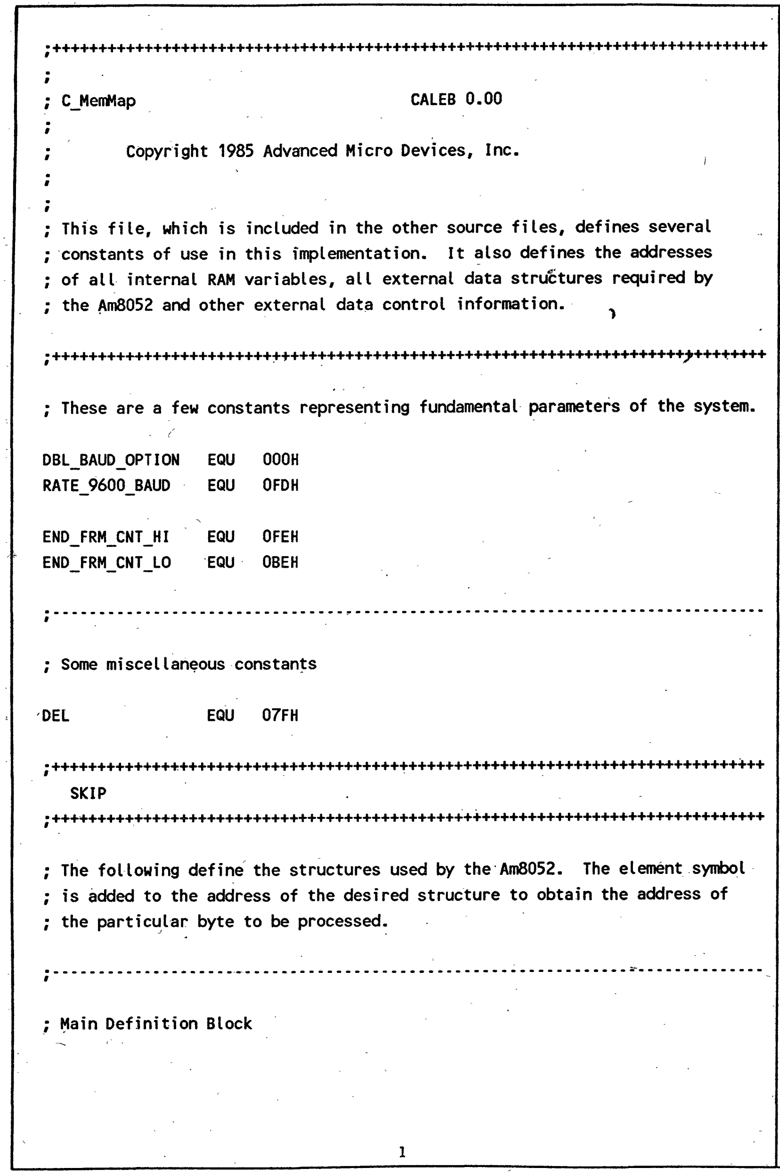
```
WFC5:
   MOV   R6,#042H
   MOV   R7,#090H
WFC6:
   MOV   DPTR,#BgdFncAtr0
   MOVX  A,@DPTR
   MOV   R5,A
   LCALL DlyTilEndFrm                      ; Wait until ready
   LCALL SetRowFntRdfPtr                   ; Reset RDFptrs to font RDF's
   MOV   DPTR,#BgdFncAtr0
   MOV   A,R6
   CLR   EX0
   MOVX  @DPTR,A
   INC   DPL
   MOV   A,R7
   MOVX  @DPTR,A
   SETB  EX0
   LCALL DlyTilEndFrm                      ;    it's thing, when known to
   CJNE  R5,#004H,WFC7
   MOV   A,#010H
   SJMP  WFC8
WFC7:
   MOV   A,#090H
WFC8:
   CLR   EX0
   MOVX  @DPTR,A
   DEC   DPL
   MOV   A,R5
   MOVX  @DPTR,A
   SETB  EX0
   LCALL SetRowNmlRdfPtr                   ; Clean up after ourselves
   LCALL ChgCsrSiz
   RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; end of C_Util
```

```
"8051"
   TITLE "     CALEB 0.00     Initial Font"
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; C_Font                                   CALEB 0.00
;
;        Copyright 1985 Advanced Micro Devices, Inc.
;
;
; This is the compact, binary representation for the default font to be loaded
; during initialization.

   NAME  "Initial Font"
   PROG
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   GLB   Fnt_5x7                          ; Initial compressed mode font
   GLB   Fnt_7x9                          ; Initial normal mode font

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Fnt_5x7:                                  ; Initial compressed mode font

   DB    041H,000H,007H,070H,088H,088H,088H,0F8H,088H,088H              ; A
   DB    042H,000H,007H,0F0H,088H,088H,0F0H,088H,088H,0F0H              ; B
   DB    000H,000H,000H                                                 ; end

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Fnt_7x9:                                  ; Initial normal mode font

   DB    021H,000H,009H,010H,010H,010H,010H,010H,000H,000H,010H,010H    ; !
   DB    022H,000H,003H,048H,048H,048H                                  ; "
   DB    023H,000H,009H,038H,044H,040H,040H,0E0H,040H,040H,042H,0FCH    ; #
   DB    024H,000H,009H,010H,07EH,090H,090H,07CH,012H,012H,0FCH,010H    ; $
   DB    025H,000H,009H,040H,0A2H,044H,008H,010H,020H,044H,08AH,004H    ; %
   DB    026H,000H,009H,070H,088H,088H,050H,020H,052H,08CH,08CH,072H    ; &
```

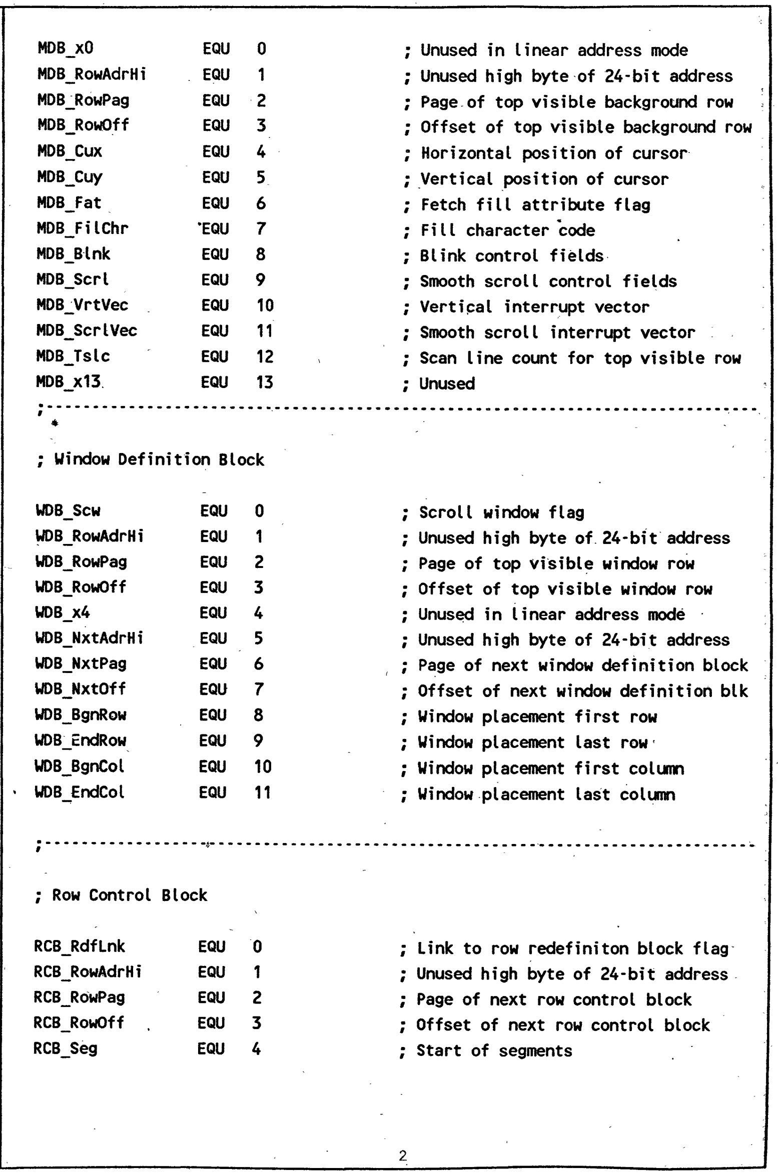
```
DB     027H,000H,004H,018H,018H,010H,020H                             ; '
DB     028H,000H,009H,008H,010H,020H,020H,020H,020H,020H,010H,008H    ; (
DB     029H,000H,009H,020H,010H,008H,008H,008H,008H,008H,010H,020H    ; )
DB     02AH,001H,007H,010H,092H,054H,038H,054H,092H,010H              ; *
DB     02BH,001H,007H,010H,010H,010H,0FEH,010H,010H,010H              ; +
DB     02CH,004H,004H,030H,030H,020H,040H                             ; ,
DB     02DH,004H,001H,0FEH                                            ; -
DB     02EH,006H,002H,030H,030H                                       ; .
DB     02FH,001H,007H,002H,004H,008H,010H,020H,040H,080H              ; /

DB     030H,000H,009H,07CH,082H,086H,08AH,092H,0A2H,0C2H,082H,07CH    ; 0
DB     031H,000H,009H,010H,030H,050H,010H,010H,010H,010H,010H,07CH    ; 1
DB     032H,000H,009H,07CH,082H,082H,004H,038H,040H,080H,080H,0FEH    ; 2
DB     033H,000H,009H,07CH,082H,002H,002H,03CH,002H,002H,082H,07CH    ; 3
DB     034H,000H,009H,004H,00CH,014H,024H,044H,084H,0FEH,004H,004H    ; 4
DB     035H,000H,009H,0FEH,080H,080H,0F8H,004H,002H,002H,084H,078H    ; 5
DB     036H,000H,009H,03CH,040H,080H,080H,0FCH,082H,082H,082H,07CH    ; 6
DB     037H,000H,009H,0FEH,082H,004H,008H,010H,020H,020H,020H,020H    ; 7
DB     038H,000H,009H,07CH,082H,082H,082H,07CH,082H,082H,082H,07CH    ; 8
DB     039H,000H,009H,07CH,082H,082H,082H,07EH,002H,002H,004H,078H    ; 9
DB     03AH,003H,006H,030H,030H,000H,000H,030H,030H                   ; :
DB     03BH,000H,008H,030H,030H,000H,000H,030H,030H,020H,040H         ; ;
DB     03CH,000H,009H,008H,010H,020H,040H,080H,040H,020H,010H,008H    ; <
DB     03DH,003H,003H,07CH,000H,07CH                                  ; =
DB     03EH,000H,009H,020H,010H,008H,004H,002H,004H,008H,010H,020H    ; >
DB     03FH,000H,009H,03CH,042H,042H,042H,002H,00CH,010H,000H,010H    ; ?


DB     040H,000H,009H,03CH,042H,09AH,0AAH,0AAH,0BCH,080H,040H,03CH    ; @
DB     041H,000H,009H,038H,044H,082H,082H,082H,0FEH,082H,082H,082H    ; A
DB     042H,000H,009H,0FCH,042H,042H,042H,07CH,042H,042H,042H,0FCH    ; B
DB     043H,000H,009H,03CH,042H,080H,080H,080H,080H,080H,042H,03CH    ; C
DB     044H,000H,009H,0F8H,044H,042H,042H,042H,042H,042H,044H,0F8H    ; D
DB     045H,000H,009H,0FEH,080H,080H,080H,0F0H,080H,080H,080H,0FEH    ; E
DB     046H,000H,009H,0FEH,080H,080H,080H,0F0H,080H,080H,080H,080H    ; F
DB     047H,000H,009H,03CH,042H,080H,080H,080H,09EH,082H,042H,03CH    ; G
DB     048H,000H,009H,082H,082H,082H,082H,0FEH,082H,082H,082H,082H    ; H
DB     049H,000H,009H,07CH,010H,010H,010H,010H,010H,010H,010H,07CH    ; I
DB     04AH,000H,009H,03EH,008H,008H,008H,008H,008H,008H,088H,070H    ; J
DB     04BH,000H,009H,082H,084H,088H,090H,0A0H,0D0H,088H,084H,082H    ; K
DB      04CH,000H,009H,080H,080H,080H,080H,080H,080H,080H,080H,0FEH     ; L
DB      04DH,000H,009H,082H,0C6H,0AAH,092H,092H,082H,082H,082H,082H     ; M
DB      04EH,000H,009H,082H,0C2H,0A2H,092H,08AH,086H,082H,082H,082H     ; N
DB      04FH,000H,009H,038H,044H,082H,082H,082H,082H,082H,044H,038H     ; O

DB      050H,000H,009H,0FCH,082H,082H,082H,0FCH,080H,080H,080H,080H     ; P
DB      051H,000H,009H,038H,044H,082H,082H,082H,092H,08AH,044H,03AH     ; Q
DB      052H,000H,009H,0FCH,082H,082H,082H,0FCH,090H,088H,084H,082H     ; R
DB      053H,000H,009H,07CH,082H,080H,080H,07CH,002H,002H,082H,07CH     ; S
DB      054H,000H,009H,0FEH,010H,010H,010H,010H,010H,010H,010H,010H     ; T
DB      055H,000H,009H,082H,082H,082H,082H,082H,082H,082H,082H,07CH     ; U
DB      056H,000H,009H,082H,082H,082H,044H,044H,028H,028H,010H,010H     ; V
DB      057H,000H,009H,082H,082H,082H,082H,092H,092H,0AAH,0C6H,082H     ; W
DB      058H,000H,009H,082H,082H,044H,028H,010H,028H,044H,082H,082H     ; X
DB      059H,000H,009H,082H,082H,044H,028H,010H,010H,010H,010H,010H     ; Y
DB      05AH,000H,009H,0FEH,002H,004H,008H,010H,020H,040H,080H,0FEH     ; Z
DB      05BH,000H,009H,078H,040H,040H,040H,040H,040H,040H,040H,078H     ; [
DB      05CH,001H,007H,080H,040H,020H,010H,008H,004H,002H               ; \
DB      05DH,000H,009H,078H,008H,008H,008H,008H,008H,008H,008H,078H     ; ]
DB      05EH,000H,003H,010H,028H,044H                                   ; ^
DB      05FH,008H,001H,0FEH                                             ; _

DB      060H,000H,004H,030H,030H,010H,008H                              ; '
DB      061H,003H,006H,078H,004H,07CH,084H,084H,07AH                    ; a
DB      062H,000H,009H,080H,080H,080H,0B8H,0C4H,084H,084H,0C4H,0B8H     ; b
DB      063H,003H,006H,078H,084H,080H,080H,084H,078H                    ; c
DB      064H,000H,009H,004H,004H,004H,074H,08CH,084H,084H,08CH,074H     ; d
DB      065H,003H,006H,078H,084H,0FCH,080H,080H,078H                    ; e
DB      066H,000H,009H,018H,024H,020H,020H,0F8H,020H,020H,020H,020H     ; f
DB      067H,003H,009H,074H,08CH,084H,08CH,074H,004H,004H,084H,078H     ; g
DB      068H,000H,009H,080H,080H,080H,0B8H,0C4H,084H,084H,084H,084H     ; h
DB      069H,001H,008H,010H,000H,030H,010H,010H,010H,010H,038H          ; i
DB      06AH,003H,009H,00CH,004H,004H,004H,004H,004H,004H,044H,038H     ; j
DB      06BH,000H,009H,080H,080H,080H,088H,090H,0A0H,0D0H,088H,084H     ; k
DB      06CH,000H,009H,030H,010H,010H,010H,010H,010H,010H,010H,038H     ; l
DB      06DH,003H,006H,0ECH,092H,092H,092H,092H,092H                    ; m
DB      06EH,003H,006H,0B8H,0C4H,084H,084H,084H,084H                    ; n
DB      06FH,003H,006H,078H,084H,084H,084H,084H,078H                    ; o
```

```
DB      070H,003H,009H,0B8H,0C4H,084H,084H,0C4H,0B8H,080H,080H,080H     ; p
DB      071H,003H,009H,074H,08CH,084H,084H,08CH,074H,004H,004H,004H     ; q
DB      072H,003H,006H,0B8H,0C4H,080H,080H,080H,080H                    ; r
DB      073H,003H,006H,078H,084H,060H,018H,084H,078H                    ; s
DB      074H,001H,008H,020H,020H,0F8H,020H,020H,020H,024H,018H          ; t
DB      075H,003H,006H,084H,084H,084H,084H,08CH,074H                    ; u
DB      076H,003H,006H,082H,082H,082H,044H,028H,010H                    ; v
DB      077H,003H,006H,082H,092H,092H,092H,092H,06CH                    ; w
DB      078H,003H,006H,084H,048H,030H,030H,048H,084H                    ; x
DB      079H,003H,009H,084H,084H,084H,084H,08CH,074H,004H,084H,078H     ; y
DB      07AH,003H,006H,0FCH,008H,010H,020H,040H,0FCH                    ; z
DB      07BH,000H,009H,018H,020H,020H,020H,040H,020H,020H,020H,018H     ; {
DB      07CH,000H,008H,010H,010H,010H,000H,000H,010H,010H,010H          ; :
DB      07DH,000H,009H,030H,008H,008H,008H,004H,008H,008H,008H,030H     ; }
DB      07EH,001H,003H,060H,092H,00CH                                   ; ~
DB      07FH,001H,007H,0FEH,07EH,006H,046H,0C6H,0F6H,0E2H               ; Logo
DB      000H,000H,000H                                                  ; end

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; end of C_Font
```

```
"8051"
   TITLE "     CALEB 0.00     Configuration"
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; C_Config                              CALEB 0.00
;
;        Copyright 1985 Advanced Micro Devices, Inc.
;
; This file contains the extra EEPROM copyright claim as well as the
; serial port configuration data.  The locations defined in this module
; currently assume that the extra EEPROM is a 27128 (i.e. 16 Kbytes).

   NAME  "Configuration"

;-----------------------------------------------------------------------------

   GLB   ExtraCpyRghtMsg                ; Resident claim in extra EEPROM

   GLB   DblBaudOpt                     ; PCON contents
   GLB   BaudRatCnt                     ; Timer one value

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   ORG   03FC0H

ExtraCpyRghtMsg:                        ; Resident claim in extra EEPROM

   DB    " Copyright 1985 Advanced Micro Devices, Inc. "

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   ORG   03FF0H

DblBaudOpt:     DB   000H
BaudRatCnt:     DB   0FDH

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; end of C_Config
```



```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; C_MemMap                             CALEB 0.00
;
;       Copyright 1985 Advanced Micro Devices, Inc.
;
;
; This file, which is included in the other source files, defines several
; constants of use in this implementation.  It also defines the addresses
; of all internal RAM variables, all external data structures required by
; the Am8052 and other external data control information.

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; These are a few constants representing fundamental parameters of the system.

DBL_BAUD_OPTION   EQU   000H
RATE_9600_BAUD    EQU   0FDH

END_FRM_CNT_HI    EQU   0FEH
END_FRM_CNT_LO    EQU   0BEH

;-------------------------------------------------------------------------------

; Some miscellaneous constants

DEL               EQU   07FH

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; The following define the structures used by the Am8052.  The element symbol
; is added to the address of the desired structure to obtain the address of
; the particular byte to be processed.

;-------------------------------------------------------------------------------

; Main Definition Block
```



```
MDB_x0          EQU   0           ; Unused in linear address mode
MDB_RowAdrHi    EQU   1           ; Unused high byte of 24-bit address
MDB_RowPag      EQU   2           ; Page of top visible background row
MDB_RowOff      EQU   3           ; Offset of top visible background row
MDB_Cux         EQU   4           ; Horizontal position of cursor
MDB_Cuy         EQU   5           ; Vertical position of cursor
MDB_Fat         EQU   6           ; Fetch fill attribute flag
MDB_FilChr      EQU   7           ; Fill character code
MDB_Blnk        EQU   8           ; Blink control fields
MDB_Scrl        EQU   9           ; Smooth scroll control fields
MDB_VrtVec      EQU   10          ; Vertical interrupt vector
MDB_ScrlVec     EQU   11          ; Smooth scroll interrupt vector
MDB_Tslc        EQU   12          ; Scan line count for top visible row
MDB_x13         EQU   13          ; Unused
;-------------------------------------------------------------------------------

; Window Definition Block

WDB_Scw         EQU   0           ; Scroll window flag
WDB_RowAdrHi    EQU   1           ; Unused high byte of 24-bit address
WDB_RowPag      EQU   2           ; Page of top visible window row
WDB_RowOff      EQU   3           ; Offset of top visible window row
WDB_x4          EQU   4           ; Unused in linear address mode
WDB_NxtAdrHi    EQU   5           ; Unused high byte of 24-bit address
WDB_NxtPag      EQU   6           ; Page of next window definition block
WDB_NxtOff      EQU   7           ; Offset of next window definition blk
WDB_BgnRow      EQU   8           ; Window placement first row
WDB_EndRow      EQU   9           ; Window placement last row
WDB_BgnCol      EQU   10          ; Window placement first column
WDB_EndCol      EQU   11          ; Window placement last column

;-------------------------------------------------------------------------------

; Row Control Block

RCB_RdfLnk      EQU   0           ; Link to row redefiniton block flag
RCB_RowAdrHi    EQU   1           ; Unused high byte of 24-bit address
RCB_RowPag      EQU   2           ; Page of next row control block
RCB_RowOff      EQU   3           ; Offset of next row control block
RCB_Seg         EQU   4           ; Start of segments
```

```
RCB_1st          EQU   4            ; Start of first segment (= RCB_Seg)
RCB_2nd          EQU   14           ; Start of second segment (if present)
RCB_3rd          EQU   24           ; Start of third segment (if present)
RCB_4th          EQU   34           ; Start of fourth segment (if present)

; NOTE:  The segment element symbol (defined below) is added to the element
;        symbol defining the start of the desired segment (defined above).

SEG_NumHid       EQU   0            ; Number of hidden chars in this seg
SEG_NumVis       EQU   1            ; Number of visible chars in this seg
SEG_Cont         EQU   2            ; Continue flag (set if a seg follows)
SEG_ChrAdrHi     EQU   3            ; Unused high byte of 24-bit address
SEG_ChrPag       EQU   4            ; Page of this seg's character buffer
SEG_ChrOff       EQU   5            ; Offset of this seg's character buffer
SEG_x6           EQU   6            ; Unused in linear address mode
SEG_AtrAdrHi     EQU   7            ; Unused high byte of 24-bit address
SEG_AtrPag       EQU   8            ; Page of this seg's attribute buffer
SEG_AtrOff       EQU   9            ; Offset of this seg's attribute buffer

; NOTE:  The element symbol for the row redefinition block pointer (at the
;        end of each row control block) depends on the type of row control
;        block.  Each display has a different size row control block (i.e.
;        they have different numbers of segments).

RCB_x44          EQU   44           ; Unused in linear address mode
RCB_BgdRdfAdrHi  EQU   45           ; Unused high byte of 24-bit address
RCB_BgdRdfPag    EQU   46           ; Page of redef block for bgd RCBs
RCB_BgdRdfOff    EQU   47           ; Offset of redef block for bgd RCBs

RCB_x26          EQU   24           ; Unused in linear address mode
RCB_MsgRdfAdrHi  EQU   25           ; Unused high byte of 24-bit address
RCB_MsgRdfPag    EQU   26           ; Page of redef block for message RCB
RCB_MsgRdfOff    EQU   27           ; Offset of redef block for message RCB

RCB_x16          EQU   14           ; Unused in linear address mode
RCB_WndRdfAdrHi  EQU   15           ; Unused high byte of 24-bit address
RCB_WndRdfPag    EQU   16           ; Page of redef block for window RCBs
RCB_WndRdfOff    EQU   17           ; Offset of redef block for window RCBs
```

```
;RCB_x16         (already defined)    ; Unused in linear address mode
RCB_ClrRdfAdrHi  EQU   15             ; Unused high byte of 24-bit address
RCB_ClrRdfPag    EQU   16             ; Page of redef blk for clr font RCBs
RCB_ClrRdfOff    EQU   17             ; Offset of redef blk for clr font RCBs

;--------------------------------------------------------------------------

; Row Redefinition Block

RRB_Tslc_NcsHi   EQU   0        ; Scan line count/part of normal char start
RRB_NcsLo_Nce    EQU   1        ; Rest of normal char start/normal char end
RRB_ApHi_SpcsHi  EQU   2        ; Part of row attrs/part of superscript start
RRB_SpcsLo_Spce  EQU   3        ; Rest of superscript start/superscript end
RRB_ApLo_SbcsHi  EQU   4        ; Rest of row attrs/part of subscript start
RRB_SbcsLo_Sbce  EQU   5        ; Rest of subscript start/subscript end
RRB_CursHi       EQU   6        ; Part of cursor start
RRB_CursLo_Cure  EQU   7        ; Rest of cursor start/cursor end
RRB_Dr_UndHi     EQU   8        ; Double height flags/part of underline
RRB_UndLo_Sund   EQU   9        ; Rest of underline/shifted underline

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; Internal RAM Variables

; The definitions of the internal RAM variables are given below.  These are
; used for all control values during normal operations on the active display
; and also for all system wide controls.

;--------------------------------------------------------------------------

; Variables used for fundamental system operations are defined here.

StkBas           DATA  067H           ; Base of stack

EndFrmFlg        BIT   00CH           ; Set by timer 0 (end-of-frame) intr

Am8052BusReqFlg  BIT   P3.2           ; Low when Am8052 wants bus (INT0*)
Am8052BusAckFlg  BIT   P1.2           ; Cleared to give bus to Am8052
```

```
MemTstTmp       DATA  010H          ; Used only during memory tests

;-------------------------------------------------------------------------

; The variables that are used for dispatching control to the various control
; routines and special purpose routines (e.g. graphic character placement)
; are defined below.  The dispatcher is also responsible for parsing control
; sequences and decoding parameters.

DisStt          DATA  010H          ; Current state of dispatcher
                                    ;    (the states are defined below)
DIR_CHR_STT     EQU   000H            ; Direct (single-char level)
BGN_ESC_STT     EQU   003H            ; Escape sequence (after ESC)
EXT_ESC_STT     EQU   006H            ; Extend ESC seq (w/intermediate)
BGN_CSI_STT     EQU   009H            ; Control Sequence (after CSI)
PRM_CSI_STT     EQU   00CH            ; Sequence (params in CSI seq)
EXT_CSI_STT     EQU   00FH            ; Extend CSI seq (w/intermediate)
UNIMP_CSI_STT   EQU   012H            ; Unimplemented (but valid) seq

PrmAcc          DATA  011H          ; Temporary parameter accumulator
PrmPvt          DATA  012H          ; Private parameter string introducer
PrmRep          DATA  013H          ; Special repeat (first) parameter
PrmCnt          DATA  014H          ; Number of parameters in sequence
PrmMaxFlg       BIT   00FH          ; Set when parameter buffer overflows
PrmBadFlg       BIT   00EH          ; Set when a bad parameter is decoded
PrmBgnFlg       BIT   00DH          ; Set when beginning parameter string
PrmBuf          DATA  04EH          ; Decoded parameters
PRM_CNT_MAX     EQU   18            ; Maximum number of parameters allowed
CtlPtrHi        DATA  016H          ; Address of control or special
CtlPtrLo        DATA  017H          ;    routine last executed (for REP)

;-------------------------------------------------------------------------
   SKIP
;-------------------------------------------------------------------------

; These variables maintain the communications ring buffers.  Three buffers
; are defined: a host reception buffer for characters from the host, a host
; transmission buffer for characters being sent to the host, and a keyboard
; reception buffer for characters from the keyboard.
```

```
HstRcvCnt       DATA  04BH          ; Number of chars received from host
HstRcvInsOff    DATA  04CH          ; Place to insert next char into ring
HstRcvExtOff    DATA  04DH          ; Place to extract next char from ring

NEAR_FULL_CNT   EQU   3             ; Stop if less space remaining
NEAR_EMPTY_CNT  EQU   12            ; Start if fewer characters available

; NOTE:  The actual host reception buffer is too large to place in internal
;        RAM, so it is defined (later in this file) in external data memory.

HstRcvBsyFlg    BIT   P1.6          ; Set if too busy to rcv, clear if rdy
                                    ; NOTE:    This signal is inverted by
                                    ;          the RS232 drivers so that
                                    ;          a positive level indicates
                                    ;          ready, negative level means
                                    ;          don't send chars from host

; NOTE:  There is currently no software support for the following variables.
;        They have only been defined for possible extensions.  The affect
;        these new capabilities would have on existing operations, and any
;        necessary restrictions on their use, will need to be considered.

HstXmtFlg       BIT   017H          ; Semaphore to lock out keyboard source
                                    ;    characters while a software source
                                    ;    sequence is being transmitted

HstXmtCnt       DATA  025H          ; Number of chars to send to host
HstXmtInsOff    DATA  026H          ; Place to insert next char into ring
HstXmtExtOff    DATA  027H          ; Place to extract next char from ring
HstXmtBuf       DATA  064H          ; Host transmission ring buffer

HstXmtBsyFlg    BIT   P1.7          ; Set when host is too busy to receive
                                    ; NOTE:    This signal is inverted by
                                    ;          the RS232 drivers so that
                                    ;          a positive level indicates
                                    ;          ready, negative level means
                                    ;          don't send chars to host
```

```
KbdRcvCnt       DATA  028H          ; Number of chars received from keybrd
KbdRcvInsOff    DATA  029H          ; Place to insert next char into ring
KbdRcvExtOff    DATA  02AH          ; Place to extract next char from ring
KbdRcvBuf       DATA  060H          ; Keyboard reception ring buffer
KbdRcvRdyFlg    BIT   P1.5          ; Set when char ready from keyboard

;-----------------------------------------------------------------------
  SKIP
;-----------------------------------------------------------------------

; These are the display dependent variables.  The first twelve are those which
; must be copied out to and in from external data memory with each change of
; the active display.  An index variable is one which represents a zero origin
; count from the beginning of something, a page variable contains a page
; address, an offset variable contains an offset into a page and a count
; variable represents a quantity (counting from one).

CurAtr          DATA  02FH          ; Attribute byte written to memory
                                    ;    (composed of the following bits)
LitBit          BIT   07EH              ; Highlight
RevBit          BIT   07DH              ; Reverse
SpsBit          BIT   07CH              ; Superscript
SbsBit          BIT   07BH              ; Subscript
SundBit         BIT   07AH              ; Strike-out (shifted underline)
UndBit          BIT   079H              ; Underscore
BlnkBit         BIT   078H              ; Blink

ActCol          DATA  030H          ; Active position horizontal (index)
ActRow          DATA  031H          ; Active position vertical (index)
CurRow          DATA  032H          ; Active row control block (page)
VisCol          DATA  033H          ; Horizontal scroll position (index)
VisRow          DATA  034H          ; Vertical scroll position (index)
BgnRow          DATA  035H          ; First RCB in display (page)
TopRow          DATA  036H          ; First visible RCB (page)
BtmRow          DATA  037H          ; Last visible RCB (page)
RemRow          DATA  038H          ; Remaining RCBs below BtmRow (page)
EndRow          DATA  039H          ; Last RCB in display (page)
ExtRow          DATA  03AH          ; Extra row (page)
```

```
; The remaining display dependent variables are not copied.  They are set
; after each change of active display.

DspWid          DATA  03BH          ; Visible width of display (count)
DspHgt          DATA  03CH          ; Visible height of display (count)

ColAdd          DATA  03DH          ; Aids horz. cursor placement (index)
RowAdd          DATA  03EH          ; Aids vert. cursor placement (index)

RcbOff          DATA  03FH          ; Offset of display's RCBs (offset)
ChrOff          DATA  040H          ; Offset of character buffer (offset)
AtrOff          DATA  041H          ; Offset of attribute buffer (offset)

TrmRow          DATA  042H          ; Termination RCB (page)
TrmOff          DATA  043H          ; Termination RCB (offset)

;-----------------------------------------------------------------------
  SKIP
;-----------------------------------------------------------------------

; The following variables are used to control various special features.  The
; first two are used to switch between two definition blocks in support of
; the Am8052 vertical smooth scroll feature.

CurMDBFlg       BIT   000H          ; Set when alternate MDB is current
CurWDBFlg       BIT   001H          ; Set when alternate WDB is current

; This group supports the message and window displays.

MsgActFlg       BIT   004H          ; Set when message display is active
MsgVisFlg       BIT   005H          ; Set when message display is visible

WndActFlg       BIT   006H          ; Set when window display is active
WndVisFlg       BIT   007H          ; Set when window display is visible

WndCol          DATA  044H          ; Current bgd col of left window bound
```

```
; The next group supports vertical and horizontal smooth scrolling.

VrtScrlCnt        DATA  045H             ; Number of rows to scroll
VrtScrlNewFlg     BIT   00BH             ; Used for new-line scrolling
ScrlByt           DATA  02DH             ; Image of byte written to the MDBs
                                         ;    (composed of the following bits)
Sr3Bit            BIT   06EH                   ; Four bit field holding current
Sr2Bit            BIT   06DH                   ;    smooth scroll rate (normally
Sr1Bit            BIT   06CH                   ;    the rate is changed by mask
Sr0Bit            BIT   06BH                   ;    and these names are unused)
SwbBit            BIT   06AH                   ; Wnd/bgd vert smooth scroll
SudBit            BIT   069H                   ; Up/down vertical smooth scroll
VrtScrlFlg        BIT   068H                   ; Set during vert smooth scroll
SCRL_RAT_MASK     EQU   078H             ; Mask for manipulating scroll rate
HrzScrlCnt        DATA  046H             ; Number of characters to scroll
HrzFrmSet         DATA  047H             ; Number of frames per scroll
HrzFrmCnt         DATA  048H             ; Number of frames until next scroll
HrzPxlShf         DATA  049H             ; Number of pixels each scroll
HrzCurPxl         DATA  04AH             ; Current pixel shift
HrzDspFlg         BIT   00AH             ; Set when scrolling message display
HrzDirFlg         BIT   009H             ; Set when scrolling right
HrzScrlFlg        BIT   008H             ; Set while doing horz smooth scroll

; The following flag is used to indicate the current font selection for
; remapping character codes to character font cell addresses.

FntMapFlg         BIT   014H             ; Set when alternate font selected

   SKIP

; The next two variables support the alterable cursor appearance and character
; blink features.

CsrSiz            DATA  02BH             ; Cursor start/end lines (in nibbles)
BlnkByt           DATA  02EH             ; Image of byte written to the MDBs
                                         ;    (composed of the following bits)
ChdBit            BIT   077H                   ; Character blink duty cycle
ChbBit1           BIT   076H                   ; Character blink rate high and
ChbBit0           BIT   075H                   ;    low bits (two-bit field)
```

```
CatbeBit          BIT   074H                   ; Attribute cursor blink enable
CxybeBit          BIT   073H                   ; X-Y cursor blink enable
CudBit            BIT   072H                   ; Cursor blink duty cycle
CubBit1           BIT   071H                   ; Cursor blink rate high and
CubBit0           BIT   070H                   ;    low bits (two-bit field)

; These aid in cursor placement in the special advance cursor code used
; after placing a character/attribute in display memory.

CsrZonFlg         BIT   010H             ; Set when cursor is in a visible zone
CsrZonCnt         DATA  015H             ; Amount cursor may be advanced until
                                         ;    it moves into the next zone
CsrShwFlg         BIT   011H             ; Defers showing the cursor until
CsrSetFlg         BIT   012H             ;    second vertical retrace time.

; The following support the modes which are software selectable.

ModByt            DATA  02CH             ; Provides byte access to modes

VEMBit            BIT   067H             ; Vertical editing (downward/upward)
AMDDWMBit         BIT   066H             ; Display width (normal/compressed)
AMDSCMBit         BIT   065H             ; Scroll (normal jump/smooth)
AMDSPMBit         BIT   P1.1             ; Screen polarity (normal/reversed)
                                         ;   (not part of regular mode byte)
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; The following address is used when accessing the keyboard.  It is only
; possible to read from the keyboard.

Keybrd            XDATA 00001H           ; Read character from keyboard

; The keyboard is enabled by holding a high level on a port 1 pin.  When
; there is a character available from the keyboard, this fact is signalled
; by a high level on another port 1 pin (configured for input).  The two
; pins are defined below.

KeybrdEnbFlg      BIT   P1.4             ; High level enables the keyboard
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; The following addresses are used in accessing the Am8052 registers.  The
; order of operations is critical.  First, regardless of access type, the
; register number should be written to the Am8052 register pointer.  Then
; to write to a register, the high byte of the 16-bit register value must
; be written first followed by the low byte.  To read from a register, the
; low byte must be read first followed by the high byte.

Am8052Ptr           XDATA 04003H          ; Address of pointer register
Am8052RegHi         XDATA 04000H          ; Address of high byte of data register
Am8052RegLo         XDATA 04001H          ; Address of low byte of data register

; When reading or writing the Am8052, its address strobe (AS*) must be held
; low.  This is accomplished by clearing the port 1 pin which is connected
; to it before beginning the access and setting this pin when finished.

Am8052XfrFlg        BIT   P1.3            ; Connected to Am8052 pin AS*

; The register numbers which are written to the Am8052 pointer register are
; defined below.

ModReg1Ind          EQU   000H            ; Mode Register 1
ModReg2Ind          EQU   001H            ; Mode Register 2
AtrEnbInd           EQU   002H            ; Attribute Port Enable
AtrRdfInd           EQU   003H            ; Attribute Redefinition
TOPSftHiInd         EQU   004H            ; Top of Page Soft Pointer (hi word)
TOPSftLoInd         EQU   005H            ; Top of Page Soft Pointer (lo word)
TOWSftHiInd         EQU   006H            ; Top of Window Soft Pointer (hi word)
TOWSftLoInd         EQU   007H            ; Top of Window Soft Pointer (lo word)
AtrFlgInd           EQU   008H            ; Attribute Flag
TOPHrdHiInd         EQU   009H            ; Top of Page Hard Pointer (hi word)
TOPHrdLoInd         EQU   00AH            ; Top of Page Hard Pointer (lo word)
TOWHrdHiInd         EQU   00BH            ; Top of Window Hard Pointer (hi word)
TOWHrdLoInd         EQU   00CH            ; Top of Window Hard Pointer (lo word)
DMABstInd           EQU   010H            ; DMA Burst and Space
VrtWthInd           EQU   011H            ; Vert Sync Width/Vert Scan Delay Reg
VrtActLneInd        EQU   012H            ; Vertical Active Lines
```

```
VrtTotLneInd        EQU   013H            ; Vertical Total Lines
HsyncVIntInd        EQU   014H            ; Horz Sync Width/Vertical Event Row
HDrvInd             EQU   015H            ; Horizontal Drive
HScnDlyInd          EQU   016H            ; Horizontal Scan Delay
HTotCntInd          EQU   017H            ; Horizontal Total Count
HTotDspInd          EQU   018H            ; Horizontal Total Display

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; The following are the locations of the structures used during normal Am8052
; operations.  There are three displays: background, message and window.  The
; background display is implemented as the Am8052 background and the others
; are implemented as Am8052 windows.  The latter can be enabled (made visible)
; or disabled (made invisible).  Structures to support these displays, as well
; as others to support vertical smooth scrolling, horizontal smooth scrolling
; and a loadable character font are all allocated at fixed locations and
; initialized following the reset/self-test procedure and after the character
; generator RAM has been initially cleared.
;
; The background display contains 30 rows of 128 characters each.  In normal
; mode, only 24 rows of 80 characters each are displayed.  In compressed mode
; all 30 rows, but only 120 characters, are shown.  The undisplayed characters
; are stored in display memory and can be viewed by scrolling.  The background
; display can be scrolled both vertically and horizontally.  There is also an
; extra row to support vertical smooth scrolling.
;
; The message display has a single row of 128 characters.  Like the background
; display, 80 characters are shown in normal mode (provided the message display
; is enabled) and 120 characters are visible in compressed mode.  The message
; display can be scrolled horizontally to view all of its characters.  Since
; it is not vertically scrollable it has no need for an extra row.  The message
; display is implemented as an Am8052 window placed at the lowest row on the
; monitor screen.
;
```

```
; The window display has 14 rows of 40 characters each.  Regardless of mode,
; only 7 rows and all 40 characters are shown.  It can be scrolled vertically
; to view all of its rows.  It cannot be scrolled horizontally.  When enabled,
; it is shown near the upper right corner of the monitor screen with portions
; of the background display surrounding it.
;
   SKIP
;
; Memory allocation is shown diagramatically in the figures below.
;
;        +-----------+--------------------------------+-------------------+
; 8000-> |           |                                |                   |
;        :           :                                :                   :
;        |           |                                |                   |
;        |   BRCB0   |             BCHR0              |                   |
;        |     :     |                :               | (see next figure) |
;        |   BRCB30  |             BCHR30             |                   |
;        |           |                                |                   |
;        :           :                                :                   :
;        |           |                                |                   |
; 9E00-> |           |                                |                   |
;        +------+-+--+--------------------------------+                   |
; 9F00-> | MRCB |*|**|              MCHR              |                   |
;        +------+-+--+--------------------------------+-------------------+
; A000-> |                                                                |
;        |                                                                |
;        |                                                                |
;        |                             BATR0                              |
;        |                               :                                |
;        |                             BATR30                             |
;        :                                                                :
;        |                                                                |
; BE00-> |                                                                |
;        +----------------------------------------------------------------+
; BF00-> |                              MATR                              |
;        +----------------------------------------------------------------+
;                 *   - message function character (1 byte),
;                       message active count (1 byte),
;                       message function attribute (2 bytes)
;                 **  - message tab table (16 bytes)
;
```

```
;        +---------+-----------------+---------- - - -------+-----------+
; 80B0-> |         |                 |                      |   WWDB0   |
;        |         |                 |                      +-----------+
; 81B0-> |         |                 |                      |   WWDB1   |
;        |         |                 |                      +-----------+
; 82B0-> |         |                 |                      |   MWDB    |
;        |         |                 |                      +-----------+
; 83B0-> |         |                 |                      |   TWDB    |
;        |         |                 |                      +---------+-+
; 84B0-> |         |                 |                      |  NRRB   |#|
;        |         |                 |                      +---------+-+
; 85B0-> |         |                 |                      |  DNRRB  |x|
;        |         |                 |                      +---------+-+
; 86B0-> |         |                 |                      |  SURRB  |x|
;        |  FRRB0  |      WRCB0      |         WCHR0        +---------+-+
; 87B0-> |    :    |        :        |           :          |  SLRRB  |x|
;        |  FRRB14 |      WRCB14     |         WCHR14       +---------+-+
; 88B0-> |         |                 |                      |  DURRB  |x|
;        |         |                 |                      +---------+-+
; 89B0-> |         |                 |                      |  DLRRB  |x|
;        |         |                 |                      +---------+-+
; 8AB0-> |         |                 |                      |    BTL    |
;        |         |                 |                      +---+--+----+
; 8BB0-> |         |                 |                      |BTR|xx| WTB|
;        |         |                 |                      +---+--+--+-+
; 8CB0-> |         |                 |                      |  BGDVARS  |
;        |         |                 |                      +-----------+
; 8DB0-> |         |                 |                      |  MSGVARS  |
;        |         |                 |                      +-----------+
; 8EB0-> |         |                 |                      |  WNDVARS  |
;        +---------+-----------------+---------- - - -------+-----------+
; 8FB0-> |                            HRCV                              |
;        +-------------------------------------- - - -------------------+
; 90B0-> |                           WATR0                              |
;        |                             :                                |
;        |                           WATR14                             |
; 9EB0-> |                                                              |
;        +-------------+-------------+-+---+---- - - ----+--------------+
; 9FB0-> |   BMDB0     |   BMDB1     |*|** |    BACT     |    WACT      |
;        +-------------+-------------+-+---+---- - - ----+--------------+
;                *   - two background function characters (2 bytes)
;                **  - two background function attributes (4 bytes)
;                #   - one termination blank attribute (2 bytes)
;                x   - unused (13 bytes total)
```

```
;-------------------------------------------------------------------------
   SKIP
;-------------------------------------------------------------------------

; These are the definitions used for all display memory.  They are related to
; the actual, physical parameters of our external data RAM (i.e. amount and
; location).  In this implmentation, display memory is organized as 64 pages
; of 256 bytes each.  This is the easiest way for the Am8751 processor to
; treat external data.  Each byte has an address consisting of two components,
; its page and its offset within that page.  By allocating similar structures
; at the same offset in different pages, and guaranteeing that none of them
; cross a page boundary, we are able to manipulate addresses one byte at a
; time.  This is important since the processor has no 16-bit arithmetic
; operations.

DspMemBas      XDATA 08000H       ; Base of external (display) memory

DSP_MEM_SIZ    EQU   04000H       ; Number of bytes of external memory
PAG_SIZ        EQU   00100H       ; Number of bytes in a page of memory

PAGE           EQU   8            ; Shift right by this value extracts a
                                  ;    page address from a 16-bit addr
OFST           EQU   000FFH       ; Mask (and) with this value extracts
                                  ;    an offset from a 16-bit address

;-------------------------------------------------------------------------

; Here we define the main definition blocks.  These control operations that
; can be changed from one frame to the next.  We need two of them to switch
; between when doing a vertical smooth scroll of the background display.

BgdMDB0        XDATA 09FB0H       ; Parameterizes the background display
BgdMDB1        XDATA 09FBEH       ;    (and supports smooth scrolling)

;-------------------------------------------------------------------------

; Next we define the four window definition blocks in the system.  These
; control Am8052 window operations.  There are two of them to switch between
; when doing a vertical smooth scroll of the window display.  Another is used
; for the message display.  The last one is used to terminate the linked list
; of window definition blocks (as required by the Am8052).
```

```
WndWDB0        XDATA 080F4H       ; Parameterizes the window display
WndWDB1        XDATA 081F4H       ;    (and supports smooth scrolling)

MsgWDB         XDATA 082F4H       ; Parameterizes the message display

TrmWDB         XDATA 083F4H       ; Terminates the list of WDBs

;-------------------------------------------------------------------------
   SKIP
;-------------------------------------------------------------------------

; These are the row control blocks for the background display.  There are 31
; of them, one for each displayable row and an extra one for use with the
; insert and delete line controls and bottom-of-display scrolling.

BgdRCB0        XDATA 08000H       ; NOTE:    Each row control block
BgdRCB1        XDATA 08100H       ;          is at the same offset in
BgdRCB2        XDATA 08200H       ;          different pages.  They are
BgdRCB3        XDATA 08300H       ;          named for their order in
BgdRCB4        XDATA 08400H       ;          memory.  Their apparent
BgdRCB5        XDATA 08500H       ;          order (i.e. as they are
BgdRCB6        XDATA 08600H       ;          shown on the monitor) will
BgdRCB7        XDATA 08700H       ;          depend on the linked list
BgdRCB8        XDATA 08800H       ;          pointers they contain.
BgdRCB9        XDATA 08900H       ;          This order will change
BgdRCB10       XDATA 08A00H       ;          during normal operations
BgdRCB11       XDATA 08B00H       ;          as a result of inserting
BgdRCB12       XDATA 08C00H       ;          and deleting rows.  The
BgdRCB13       XDATA 08D00H       ;          order will also be changed
BgdRCB14       XDATA 08E00H       ;          by bottom-of-display
BgdRCB15       XDATA 08F00H       ;          scrolling.
BgdRCB16       XDATA 09000H       ;
BgdRCB17       XDATA 09100H       ;          There is a correspondence
BgdRCB18       XDATA 09200H       ;          between a particular row
BgdRCB19       XDATA 09300H       ;          control block and the same
BgdRCB20       XDATA 09400H       ;          numbered character and
BgdRCB21       XDATA 09500H       ;          attribute buffers.  This
BgdRCB22       XDATA 09600H       ;          correspondence is kept in
BgdRCB23       XDATA 09700H       ;          spite of any logical order.
```

```
BgdRCB24        XDATA 09800H        ;         Therefore, the characters
BgdRCB25        XDATA 09900H        ;         and attributes which are
BgdRCB26        XDATA 09A00H        ;         refered to by a particular
BgdRCB27        XDATA 09B00H        ;         row control block can be
BgdRCB28        XDATA 09C00H        ;         easily determined at any
BgdRCB29        XDATA 09D00H        ;         time.
BgdRCB30        XDATA 09E00H
BGD_BUF_WID     EQU   128           ; Width of background (and message)
                                    ;    display buffers
; These are the character buffers for the background display.  There is one
; for each row control block and each contains 128 characters.

BgdChrBuf0      XDATA 08030H        ; NOTE:   Each buffer is at the same
BgdChrBuf1      XDATA 08130H        ;         offset in different pages.
BgdChrBuf2      XDATA 08230H        ;         Each is in the same page as
BgdChrBuf3      XDATA 08330H        ;         the row control block which
BgdChrBuf4      XDATA 08430H        ;         refers to it.
BgdChrBuf5      XDATA 08530H
BgdChrBuf6      XDATA 08630H
BgdChrBuf7      XDATA 08730H
BgdChrBuf8      XDATA 08830H
BgdChrBuf9      XDATA 08930H
BgdChrBuf10     XDATA 08A30H
BgdChrBuf11     XDATA 08B30H
BgdChrBuf12     XDATA 08C30H
BgdChrBuf13     XDATA 08D30H
BgdChrBuf14     XDATA 08E30H
BgdChrBuf15     XDATA 08F30H
BgdChrBuf16     XDATA 09030H
BgdChrBuf17     XDATA 09130H
BgdChrBuf18     XDATA 09230H
BgdChrBuf19     XDATA 09330H
BgdChrBuf20     XDATA 09430H
BgdChrBuf21     XDATA 09530H
BgdChrBuf22     XDATA 09630H
BgdChrBuf23     XDATA 09730H
BgdChrBuf24     XDATA 09830H
BgdChrBuf25     XDATA 09930H
BgdChrBuf26     XDATA 09A30H
BgdChrBuf27     XDATA 09B30H
BgdChrBuf28     XDATA 09C30H
BgdChrBuf29     XDATA 09D30H
BgdChrBuf30     XDATA 09E30H
```

```
; These are the attribute buffers for the background display.  There is one
; for each row control block and each contains 128 attributes.

BgdAtrBuf0      XDATA 0A000H        ; NOTE:   Each buffer is at the same
BgdAtrBuf1      XDATA 0A100H        ;         offset in different pages.
BgdAtrBuf2      XDATA 0A200H        ;         Each is in a page which is
BgdAtrBuf3      XDATA 0A300H        ;         32 pages beyond the page
BgdAtrBuf4      XDATA 0A400H        ;         containing the row control
BgdAtrBuf5      XDATA 0A500H        ;         block which refers to it.
BgdAtrBuf6      XDATA 0A600H
BgdAtrBuf7      XDATA 0A700H
BgdAtrBuf8      XDATA 0A800H
BgdAtrBuf9      XDATA 0A900H
BgdAtrBuf10     XDATA 0AA00H
BgdAtrBuf11     XDATA 0AB00H
BgdAtrBuf12     XDATA 0AC00H
BgdAtrBuf13     XDATA 0AD00H
BgdAtrBuf14     XDATA 0AE00H
BgdAtrBuf15     XDATA 0AF00H
BgdAtrBuf16     XDATA 0B000H
BgdAtrBuf17     XDATA 0B100H
BgdAtrBuf18     XDATA 0B200H
BgdAtrBuf19     XDATA 0B300H
BgdAtrBuf20     XDATA 0B400H
BgdAtrBuf21     XDATA 0B500H
BgdAtrBuf22     XDATA 0B600H
BgdAtrBuf23     XDATA 0B700H
BgdAtrBuf24     XDATA 0B800H
BgdAtrBuf25     XDATA 0B900H
BgdAtrBuf26     XDATA 0BA00H
BgdAtrBuf27     XDATA 0BB00H
BgdAtrBuf28     XDATA 0BC00H
BgdAtrBuf29     XDATA 0BD00H
BgdAtrBuf30     XDATA 0BE00H
```

```
;-----------------------------------------------------------------------
  SKIP
;-----------------------------------------------------------------------
; This is the row control block for the message display.  Only one is needed
; since the insert and delete line controls and vertical scrolling are not
; allowed in this display.

MsgRCB          XDATA 09F00H

; This is the character buffer for the message display.  It is at the same
; offset as the background display character buffers and is in the same page
; as its row control block.

MsgChrBuf       XDATA 09F30H

; This is the attribute buffer for the message display.  It bears the same
; relationship to its row control block as the background attribute buffers
; bear to their row control blocks (i.e. 32 pages beyond it).

MsgAtrBuf       XDATA 0BF00H

;-----------------------------------------------------------------------

; These are the row control blocks for the window display.  There are 15
; of them, one for each displayable row and an extra one for use with the
; insert and delete line controls and bottom-of-display scrolling.

WndRCB0         XDATA 080BAH        ; NOTE:    Each row control block
WndRCB1         XDATA 081BAH        ;          is at the same offset in
WndRCB2         XDATA 082BAH        ;          different pages.  They are
WndRCB3         XDATA 083BAH        ;          named for their order in
WndRCB4         XDATA 084BAH        ;          memory.  Their apparent
WndRCB5         XDATA 085BAH        ;          order (i.e. as they are
WndRCB6         XDATA 086BAH        ;          shown on the monitor) will
WndRCB7         XDATA 087BAH        ;          depend on the linked list
WndRCB8         XDATA 088BAH        ;          pointers they contain.
WndRCB9         XDATA 089BAH        ;          This order will change
WndRCB10        XDATA 08ABAH        ;          during normal operations
WndRCB11        XDATA 08BBAH        ;          as a result of inserting
WndRCB12        XDATA 08CBAH        ;          and deleting rows and by
WndRCB13        XDATA 08DBAH        ;          bottom-of-display scrolling.
WndRCB14        XDATA 08EBAH
```

```
WND_BUF_WID     EQU   40            ; Width of window display buffers
WND_VIS_WID     EQU   40            ; Width of visible window display
WND_VIS_HGT     EQU   7             ; Height of visible window display
WND_TOP_MRG     EQU   6             ; Background rows above window display

  SKIP

; These are the character buffers for the wnidow display.  There is one
; for each row control block and each contains 40 characters.
WndChrBuf0      XDATA 080CCH        ; NOTE:    Each buffer is at the same
WndChrBuf1      XDATA 081CCH        ;          offset in different pages.
WndChrBuf2      XDATA 082CCH        ;          Each is in the same page as
WndChrBuf3      XDATA 083CCH        ;          the row control block which
WndChrBuf4      XDATA 084CCH        ;          refers to it.
WndChrBuf5      XDATA 085CCH
WndChrBuf6      XDATA 086CCH
WndChrBuf7      XDATA 087CCH
WndChrBuf8      XDATA 088CCH
WndChrBuf9      XDATA 089CCH
WndChrBuf10     XDATA 08ACCH
WndChrBuf11     XDATA 08BCCH
WndChrBuf12     XDATA 08CCCH
WndChrBuf13     XDATA 08DCCH
WndChrBuf14     XDATA 08ECCH

; These are the attribute buffers for the window display.  There is one
; for each row control block and each contains 40 attributes.
WndAtrBuf0      XDATA 090B0H        ; NOTE:    Each buffer is at the same
WndAtrBuf1      XDATA 091B0H        ;          offset in different pages.
WndAtrBuf2      XDATA 092B0H        ;          Each is in a page which is
WndAtrBuf3      XDATA 093B0H        ;          16 pages beyond the page
WndAtrBuf4      XDATA 094B0H        ;          containing the row control
WndAtrBuf5      XDATA 095B0H        ;          block which refers to it.
WndAtrBuf6      XDATA 096B0H
WndAtrBuf7      XDATA 097B0H
WndAtrBuf8      XDATA 098B0H
WndAtrBuf9      XDATA 099B0H
WndAtrBuf10     XDATA 09AB0H
WndAtrBuf11     XDATA 09BB0H
WndAtrBuf12     XDATA 09CB0H
WndAtrBuf13     XDATA 09DB0H
WndAtrBuf14     XDATA 09EB0H
```

```
;--------------------------------------------------------------------------
  SKIP
;--------------------------------------------------------------------------

; These are the row redefinition blocks used during normal operations.  They
; control the vertical placement of characters and attributes within the
; character row.  In particular, a change of cursor appearance requires a
; change of the cursor start and end lines.

NrmRRB          XDATA 084F4H        ; Normal row redefinition block

; NOTE:  Only the preceding definition has current software support.  The
;        following definitions are for possible extensions to support rows
;        of double width and/or double height characters.  How these extra
;        capabilities would affect existing operations, and any necessary
;        restrictions on their use, will need to be considered.

DwNhRRB         XDATA 085F4H        ; Double width/normal height
SwUhRRB         XDATA 086F4H        ; Single width/upper half of dbl height
SwLhRRB         XDATA 087F4H        ; Single width/lower half of dbl height
DwUhRRB         XDATA 088F4H        ; Double width/upper half of dbl height
DwLhRRB         XDATA 089F4H        ; Double width/lower half of dbl height

;--------------------------------------------------------------------------
; These are the row redefiniton blocks used for loading the character
; generator (font) RAM during normal operations.  There is one for each
; slice of a character cell which can be programmed by a user.

FntRRB0         XDATA 080B0H        ; NOTE:   A user is only allowed to
FntRRB1         XDATA 081B0H        ;         change the first fifteen
FntRRB2         XDATA 082B0H        ;         slices of a character cell.
FntRRB3         XDATA 083B0H        ;         This is because the Am8052
FntRRB4         XDATA 084B0H        ;         requires that the last slice
FntRRB5         XDATA 085B0H        ;         be cleared for use above and
FntRRB6         XDATA 086B0H        ;         below the lines specified
FntRRB7         XDATA 087B0H        ;         in the row redefinition
FntRRB8         XDATA 088B0H        ;         block.  Actually, in this
FntRRB9         XDATA 089B0H        ;         implementation, only the
FntRRB10        XDATA 08AB0H        ;         first fourteen can be
FntRRB11        XDATA 08BB0H        ;         changed in normal mode,
FntRRB12        XDATA 08CB0H        ;         and only the first eleven
FntRRB13        XDATA 08DB0H        ;         in compressed mode.
FntRRB14        XDATA 08EB0H
```

```
;--------------------------------------------------------------------------
  SKIP
;--------------------------------------------------------------------------
; These are the special characters which support the featrues of this
; implementation.  They support horizontal smooth scrolling, the loadable
; font and ensure maximal processing time by reducing unnecessary DMA
; activity by the Am8052.
; First, the background function character which supports horizontal smooth
; scrolling in the background and the loadable font.  There are two of them,
; and two associated attributes, to allow font loading of both normal and
; compressed mode characters.
        BgdFncChr0      XDATA 09FCCH
        BgdFncChr1      XDATA 09FCDH
        BgdFncAtr0      XDATA 09FCEH
        BgdFncAtr1      XDATA 09FD0H

; Next, the message function character character which supports horizontal
; smooth scrolling in the message display and its associated attribute.
        MsgFncChr       XDATA 09F1CH
        MsgFncAtr       XDATA 09F1EH


; Next, a latched attribute for use with the termination row control block is
; defined.  This is the extra row control block in the window display when it
; is not otherwise being used.  If the window display is active then the extra
; row control block in the background is used as the termination row control
; block.  It is pointed to by the last visible row in the background display,
; the last visible row in the window display, the message display row and the
; termination window definition block.  The termination row control block is
; set to point to itself.  By setting the character pointer to zero we force
; the Am8052 to use the fill code (defined in the main definition block) for
; the entire row.  Because the FAT bit (also in the main definition block) is
; set, the termination attribute is fetched.  Since this attribute is latched
; (the only latched attribute in the system), it forces all fill characters to
; have the same, blank attribute.  By using this termination row we avoid DMA
; activity by the Am8052 almost entirely during the time that the last two
; character rows are being displayed.  (DMA occurs when the Am8052 pre-fetchs
; up to two extra rows.)  Therefore, the processor has a nearly uninterrupted
; one-and-a-half milliseconds just prior to the time when the main definition
; block is fetched to begin the next frame.  Any accesses made to display
; memory during this end-of-frame time cannot interfere with video refresh.
; The timer 0 interrupt has been set to tell us when this time begins.

      TrmAtr            084FEH
```

```
;-----------------------------------------------------------------------------
   SKIP
;-----------------------------------------------------------------------------

; These are the active counts associated with each row in the system.  They
; tell us where the farthest right, non-blank character is in each row (i.e.
; how much of the row we may need to erase).

BgdActCntBuf      XDATA 09FD2H          ; One byte for each background row
WndActCntBuf      XDATA 09FF1H          ; One byte for each window row
MsgActCnt         XDATA 09F1DH          ; One byte for the one message row

;-----------------------------------------------------------------------------

; These are the bit tables for tab position storage.  There is one table for
; each display, although the background table is actually in two parts.  Each
; bit in a table corresponds to a column in the display.  If the bit is set,
; then that column is a tab location.

; NOTE:  The following definitions are not supported by current software.
;        They are for possible extensions to support horizontal tabulation.
;        Vertical tabulation could also be supported by using an unused bit
;        in each row control block (addressed in their physical memory order
;        rather than their logical display order).  The affect of these new
;        capabilites on existing operations, and any necessary restrictions
;        on their use, will need to be considered.

BgdTabTblLt       XDATA 08AF4H          ; For left 96 columns of background
BgdTabTblRt       XDATA 08BF4H          ; For right 32 columns of background
MsgTabTbl         XDATA 09F20H          ; For all 128 columns of message
WndTabTbl         XDATA 08BFBH          ; For all 40 columns of window

;-----------------------------------------------------------------------------

; These are the display dependent variable buffers.  While a particular display
; is active its variables are kept in internal RAM.  When a different display
; becomes active the old display's variables are copied out to their external
; RAM location and the new display's variables are copied in.

BgdVarBuf         XDATA 08CF4H
MsgVarBuf         XDATA 08DF4H
WndVarBuf         XDATA 08EF4H
```

```
;-----------------------------------------------------------------------------

; This is the serial communications ring buffer for receiving characters from
; the host computer.

HstRcvBuf         XDATA 08FB0H

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   SKIP
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; These are the locations of the various structures used in the initial
; clearing of the character generator RAM.  They are in the same memory
; area where the background character buffers are located.  When these
; buffers are initialized, the font clearing information (which will no
; longer be needed) will be overwritten with spaces.

ClrFntMDB         XDATA 09030H          ; Main definition block

ClrFntRCBBas      XDATA 08030H          ; First row control block
ClrFntChrBas      XDATA 08080H          ; First character

; NOTE:  The remaining fifteen RCBs and characters are at the same
;        offsets in subsequent pages.

ClrFntAtr         XDATA 09080H          ; Common attributes

ClrFntRRB         XDATA 09130H          ; Common row redefinition block

ClrFntWDB         XDATA 09230H          ; Termination window definition block

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; end of C_MemMap
```

ADVANCED
MICRO
DEVICES, INC.
901 Thompson Place
P.O. Box 3453
Sunnyvale,
California 94088
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL FREE
(800) 538-8450

IH-RRD- ' 5M-7/86-0