intel®

APPLICATION NOTE

AP-223

October 1984

8051 Based CRT Terminal Controller

Michael A. Shafer
Microcontroller Applications

ORDER NUMBER: 270032-001

1.0 INTRODUCTION

This is the third application note that Intel has produced on CRT terminal controllers. The first Ap Note (ref. 1), written in 1977, used the 8080 as the CPU and required 41 packages including 11 LSI devices. In 1979, another application note (ref. 2) using the 8085 as the controller was produced and the chip count decreased to 20 with 11 LSI devices.

Advancing technology has integrated a complete system onto a single device that contains a CPU, program memory, data memory, serial communication, interrupt controller, and I/O. These "computer-on-a-chip" devices are known as microcontrollers. Intel's MCS®-51 microcontroller was chosen for this application because of its highly integrated functions. This CRT terminal design uses 12 packages with only 4 LSI devices.

This application note has been divided into five general sections:

1) CRT Terminal Basics
2) 8051 Description
3) 8276 Description
4) Design Background
5) System Description

2.0 CRT TERMINAL BASICS

A terminal provides a means for humans to communicate with a computer. Terminals may be as simple as a LED display and a couple of push buttons, or it may be an elaborate graphics system that contains a full function keyboard with user programmable keys, color CRT and several processors controlling its functions. This application note describes a basic low cost terminal containing a black and white CRT display, full function keyboard and a serial interface.

2.1 CRT Description

A raster scan CRT displays its images by generating a series of lines (raster) across the face of the tube. The electron beam usually starts at the top left hand corner moves left to right, back to the left of the screen, moves down one row and continues on to the right. This is repeated until the lower right hand of the screen is reached. Then the beam returns to the top left hand corner and refreshes the screen. The beam forms a zigzag pattern as shown in Figure 2.1.0.

Two independent operating circuits control this movement across the screen. The horizontal oscillator controls the left to right motion of the beam while the vertical controls the top to bottom movement. The vertical oscillator also tells the beam when to return to the upper left hand corner or "home" position.

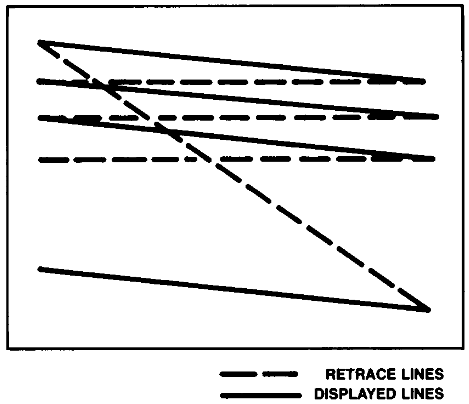


Figure 2.1.0 Raster Scan

As the electron beam moves across the screen under the control of the horizontal oscillator, a third circuit controls the current entering the electron gun. By varying the current, the image may be made as bright or as dim as the user desires. This control is also used to turn the beam off or "blank the screen".

When the beam reaches the right hand side of the screen, the beam is blanked so it does not appear on the screen as it returns to the left side. This "retrace" of the beam is at a much faster rate than it traveled across the screen to generate the image.

The time it takes to scan the whole screen and return to the home position is referred to as a "frame". In the United States, commercial television broadcast uses a horizonal sweep frequency of 15,750Hz which calculates out to 63.5 microseconds per line. The frame time is equal to 16.67 milliseconds or 60Hz vertical sweep frequency.

Although this is the commercial standard, many CRT displays operate from 18KHz to 30KHz horizonatal frequency. As the horizontal frequency increases, the number of lines per frame increases. This increase in lines or resolution is needed for graphic displays and on special text editors that display many more lines of text than the standard 24 or 25 character lines.

Since the United States operates on a 60Hz A.C. power line frequency, most CRT monitors use 60Hz as the vertical frequency. The use of 60Hz as the vertical frequency allows the magnetic and electrical variations that can modulate the electron beam to be synchronized with the display, thus they go unnoticed. If a frequency other than 60Hz is used, special shielding and power supply regulating is usually required. Very few CRTs operate on a vertical frequency other than 60Hz due to the increase in the overall system cost.

The CRT controller must generate the pulses that define the horizontal and vertical timings. On most raster scan CRTs the horizontal frequency may vary as much as 500Hz without any noticeable effect on the quality of the display. This variation can change the number of horizontal lines from 256 to 270 per frame.

The CRT controller must also shift out the information to be displayed serially to the circuit that controls the electron beam's intensity as it scans across the screen. The circuits that control the timing associated with the shifting of the information are known as the dot clock and the character clock. The character clock frequency is equal to the dot clock frequency divided by the number of dots it takes to form a character in the horizontal axis. The dot clock frequency is calculated by the following equation:

$$\text{Dot Clcok (Hz)} = (N + R)*D*L*F$$

where

- N is the number of displayed characters per row,
- R is the number of character times for the retrace,
- D is the number of dots per character in the horizontal axis,
- L is the number of horizontal lines per frame,
- F is the frame rate in Hz.

In this design $N = 80$, $R = 20$, $D = 7$, $L = 270$, and $F = 60Hz$. Plugging in the numbers results in a dot clock frequency of 11.34MHz.

The retrace number may vary on each design because it is used to set the left and right hand margins on the CRT. The number of dots per character is chosen by the designer to meet the system needs. In this design, a 5×7 dot matrix and 2 blank dots between each character (see Figure 2.1.1) makes D equal to $5 + 2 = 7$.

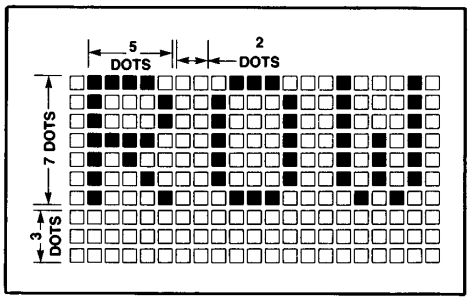


Figure 2.1.1 5 × 7 Dot Matrix

The following equation can be used to figure the number of lines per frame:

$$L = (H*Z) + V$$

where

- H is the number of horizontal lines per character,
- Z is the number of character lines per frame,
- V is the number of horizontal line times during the vertical retrace

In this design H is equal to the 7 horizontal dots per character plus 3 blank dots between each row which adds up to 10. Also 25 lines of characters are displayed, so $Z = 25$. The vertical retrace time is variable to set the top and bottom margins on the CRT and in this design is equal to 20. Plugging in the numbers gives $L = 270$ lines per frame.

2.2 Keyboard

A keyboard is the common way a human enters commands and data to a computer. A keyboard consists of a matrix of switches that are scanned every couple of milliseconds by a keyboard controller to determine if one of the keys has been pressed. Since the keyboard is made up of mechanical switches that tend to bounce or "make and break" contact everytime they are pressed, debouncing of the switches must also be a function of the keyboard controller. There are dedicated keyboard controllers available that do everything from scanning the keyboard, debouncing the keys, decoding the ASCII code for that key closure to flagging the CPU that a valid key has been depressed. The keyboard controller may present the information to the CPU in parallel form or in a serial data stream.

This Application Note integrates the function of the keyboard controller into the 8051 which is also the terminal controller. Provisions have been made to interface the 8051 to a keyboard that uses a dedicated keyboard controller. The 8051 can accept data from the keyboard controller in either parallel or serial format.

2.3 Serial Communications

Communication between a host computer and the CRT terminal can be in either parallel or serial data format. Parallel data transmission is needed in high end graphic terminals where great amounts of information must be transferred.

One can rarely type faster than 120 words per minute, which corresponds to 12 characters per second or 1 character per 83 milliseconds. The utilization of a parallel port cannot justify the cost associated with the drivers and the amount of wire needed to perform this transmission. Full duplex serial data transmission requires 3 wires and two

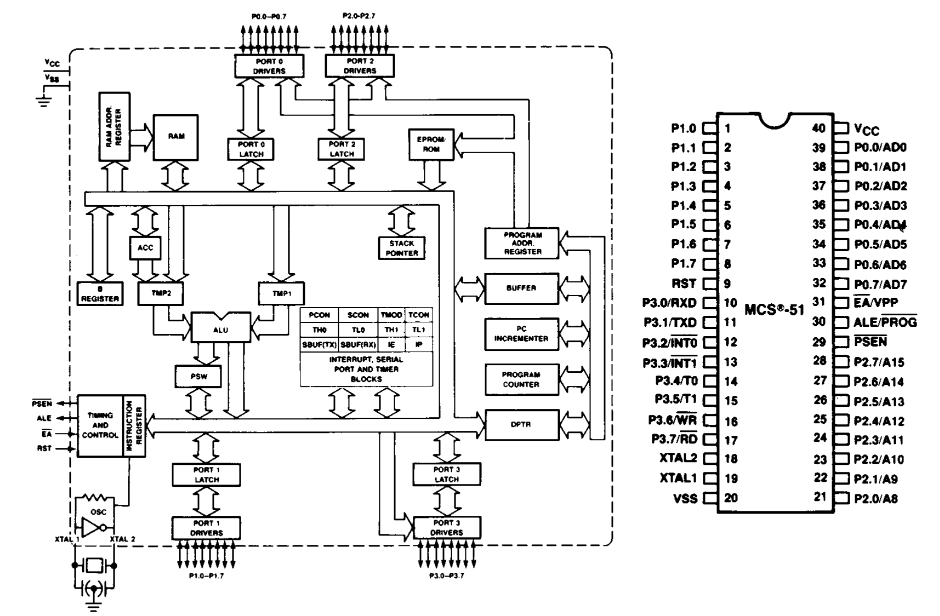


Figure 3.0.0 8051 Block Diagram

drivers to implement the communication channel between the host computer and the terminal. The data rate can be as high as 19200 BAUD in the asynchronous serial format. BAUD rate is the number of bits per second received or transmitted. In the asynchronous serial format, 10 bits of information is required to transmit one character. One character per 500 microseconds or 1,920 characters per second would then be trasmitted using 19.2 KBAUD.

This application note uses the 8051 serial port configured for full duplex asynchronous serial data transmission. The software for the 8051 has been written to support variable BAUD rates from 150 BAUD up to 9.6 KBAUD.

3.0 8051 DESCRIPTION

The 8051 is a single chip high-performance microcontroller. A block diagram is shown in figure 3.0.0. The 8051 combines CPU; Boolean processor; 4K × 8 ROM: 128 × 8 RAM; 32 I/O lines; two 16-bit timer/ event counters; a five-source, two-priority-level, nested interrupt structure; serial I/O port for either multiprocessor communications, I/O expansion, or full duplex UART; and on-chip oscillator and clock circuits.

3.1 CPU

Efficient use of program memory results from an instruction set consisting of 49 single-byte, 45 two-byte and 17 three-byte instructions. Most arithmetic, logical and branching operations can be performed using an instruction that appends either a short address or a long address. For example, branches may use either an offset that is relative to the program counter which takes two bytes or a direct 16-bit address which takes three bytes to perform. As a result, 64 instructions operate in one machine cycle, 45 in two machine cycles, and the multiply and divide instruction execute in 4 machine cycles.

The 8051 has five addressing modes for source operands: Register, Direct, Register-Indirect, Immediate, and Based-Register-plus Index-Register-Indirect Addressing.

The Boolean Processor can be thought of as a separate one-bit CPU. It has its own accumulator (the carry bit), instruction set for data moves, logic, and control transfer, and its own bit addressable RAM and I/O. The bit-manipulating instructions provide optimum code and speed efficiency for handling on chip peripherals. The

Boolean processor also provides a straight forward means of converting logic equations directly into software. Complex combinational logic functions can be resolved without extensive data movement, byte masking, and test-and-branch trees.

3.2 On-Chip Ram

The CPU manipulates operands in four memory spaces. These are the 64K-byte Program Memory, 64K-byte External Data Memory, 128-byte Internal Data Memory, and 128-byte Special Function Registers (SFRs). Four Register Banks (each with 8 registers), 128 addressable bits, and the Stack reside in the internal Data RAM. The Stack size is limited only by the available Internal Data RAM and its location is determined by the 8-bit Stack Pointer. All registers except for the Program Counter and the four 8-Register Banks reside in the SFR address space. These memory mapped registers include arithmetic registers, pointers, I/O ports, and registers for the interrupt system, timers, and serial channel.

Registers in the four 8-Register Banks can be addressed by Register, Direct, or Register-Indirect Addressing modes. The 128 bytes of internal Data Memory can be addressed by Direct or Register-Indirect modes while the SFRs are only addressed directly.

3.3 I/O Ports

The 8051 has instructions that can treat the 32 I/O lines as 32 individually addressable bits or as 4 parallel 8-bit ports addressable as Ports 0, 1, 2, and 3.

Resetting the 8051 writes a logical 1 to each pin on port 0 which places the output drivers into a high-impedance mode. Writing a logical 0 to a pin forces the pin to ground and sinks current. Re-writing the pin high will place the pin in either an open drain output or high-impedance input mode.

Ports 1, 2, and 3 are known as quasi-bidirectional I/O pins. Resetting the device writes a logical one to each pin. Writing a logical 0 to the pin will force the pin to ground and sink current. Re-writing the pin high will place the pin in an output mode with a weak depletion pullup FET or in the input mode. The weak pullup FET is easily overcome by a TTL output.

Ports 0 and 2 can also be used for off-chip peripheral expansion. Port 0 provides a multiplexed low-order address and data bus while Port 2 contains the high-order address when using external Program Memory or more than 256 byte external Data Memory.

Port 3 pins can also be used to provide external interrupt request inputs, event counter inputs, the serial port TXD and RXD pins and to generate control signals used for writing and reading external peripherals.

3.4 Interrupt System

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. A five-source, two-level, nested interrupt system ties the real time events to the normal program execution.

The 8051 has two external interrupt sources, one interrupt from each of the two timer/counters, and an interrupt from the serial port. Each interrupt vectors the program execution to its own unique memory location for servicing the interrupt. In addition, each of the five sources can be individually enabled or disabled as well as assigned to one of the two interrupt priority levels available on the 8051.

Up to two additional external interrupts can be created by configuring a timer/counter to the event counter mode. In this mode the timer/counter increments on command by either the T0 or T1 pin. An interrupt is generated when the timer/counter overflows. Thus if the timer/counter is loaded with the maximum count, the next high-to-low transition of the event counter input will cause an interrupt to be generated.

3.5 Serial Port

The 8051's serial port is useful for linking peripheral devices as well as multiple 8051s through standard asynchronous protocols with full duplex operation. The serial port also has a synchronous mode for expansion of I/O lines using shift registers. This hardware serial port saves ROM code and permits a much higher transmission rate than could be achieved through software. The processor merely needs to read or write the serial buffer in response to an interrupt. The receiver is double buffered to eliminate the possibility of overrun if the processor failed to read the buffer before the beginning of the next frame.

The full duplex asynchronous serial port provides the means of communication with standard UART devices such as CRT terminals and printers.

The reader should refer to the microcontroller handbook for a complete discussion of the 8051 and its various modes of operation.

4.0 8276 DESCRIPTION

The 8276's block diagram and pin configuration are shown in Figure 4.0.0. The following sections describe the general capabilities of the 8276.

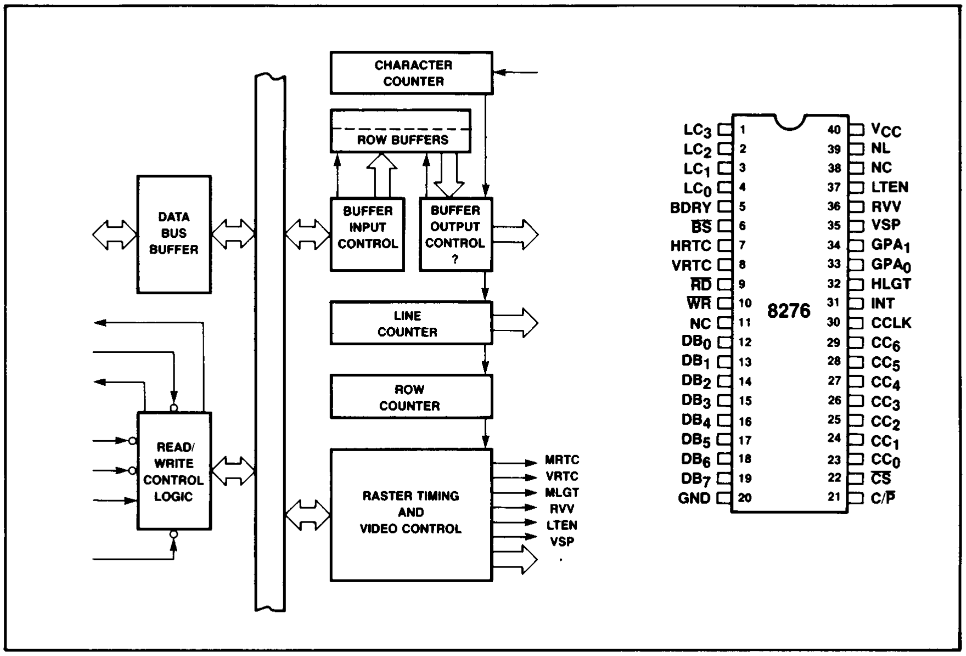


Figure 4.0.0 8276 Block Diagram

4.1 CRT Display Refreshing

The 8276, having been programmed by the system designer for a specific screen format, generates a series of Buffer Ready signals. A row of characters is then transferred by the system controller from the display memory to the 8276's row buffers. The row buffers are filled by deselecting the 8276 $\overline{CS}$ and asserting the BS and WR signals. The 8276 presents the character codes to an external character generator ROM by using outputs CC0–CC6. The parallel data from the outputs of the character generator is converted to serial information that is clocked by external dot timing logic into the video input of the CRT.

The character rows are displayed on the CRT one line at a time. Line count outputs LC0–LC3 select the current line information from the character generator ROM. The display process is illustrated in Figure 4.1.0. This process is repeated for each display character row. At the beginning of the last display row the 8276 generates an interrupt request by raising its INT output line. The interrupt request is used by the 8051 system controller to reinitialize its load buffer pointers for the next display refresh cycle.

Proper CRT refreshing requires that certain 8276 parameters be programmed at system initialization time. The 8276 has two types of internal registers; the write only Command (CREG) and Parameter (PREG) Registers, and the read only Status Register (SREG). The 8276 expects to receive a command followed by 0 to 4 parameter bytes depending on the command. A summary of the 8276's instruction set is shown in Figure 4.1.1. To access the registers, $\overline{CS}$ must be asserted along with $\overline{WR}$ or $\overline{RD}$. The status of the C/P pin determines whether the command or parameter registers are selected.

The 8276 allows the designer flexibility in the display format. The display may be from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per character row. In addition, four curser formats are available; blinking, non-blinking, underline, and reverse video. The curser position is programmable to anywhere on the screen via the Load Curser command.

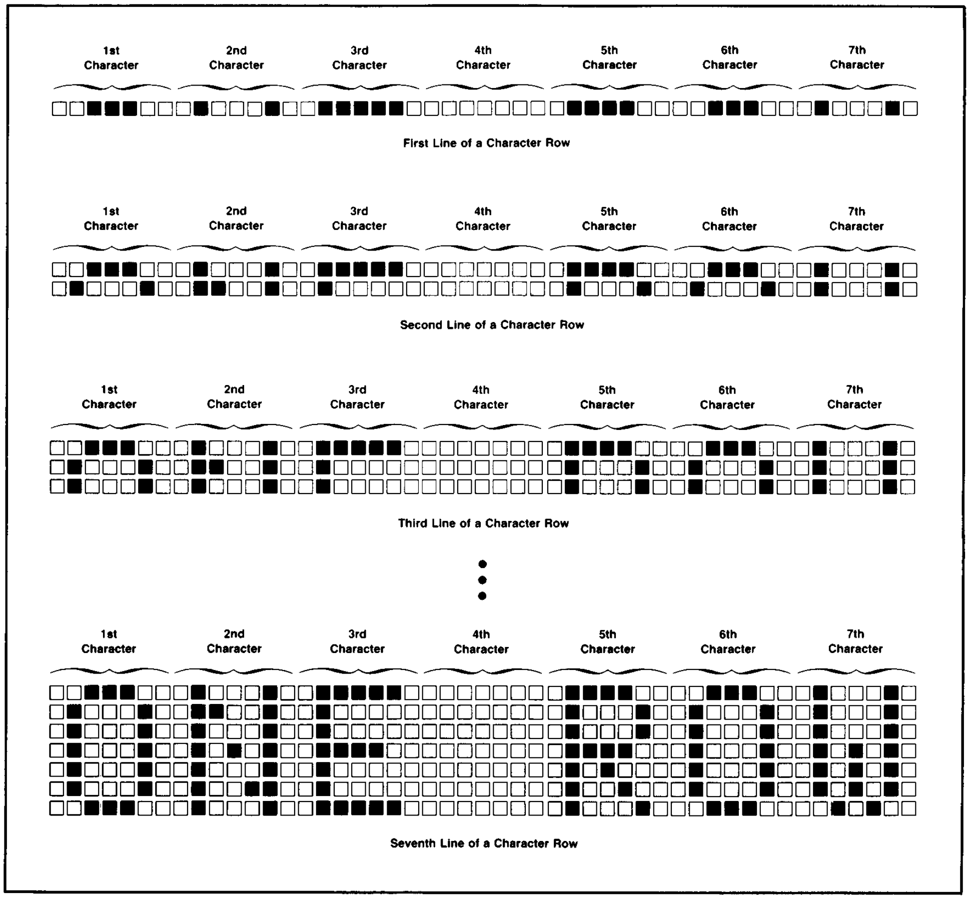


Figure 4.1.0 8276 Row Display

4.2 CRT Timing

The 8276 provides two timing outputs for controlling the CRT. The Horizontal Retrace Timing and Control (HRTC) and Vertical Retrace Timing and Control (VRTC) signals are used for synchronizing the CRT horizontal and vertical oscillators. A third output, VSP (Video Suppress), provides a signal to the dot timing logic to blank the video signal during the horizontal and vertical retraces. LTEN (Light Enable) is used to provide the ability to force the video output high regardless of the state of the VSP signal. This feature is used to place the cursor on the screen and to control attribute functions.

RVV (Reverse Video) output, if enabled, will cause the system to invert its video output. The fifth timing signal output, HLGT (highlight) allows the flexibility to increase the CRT beam intensity to a greater than normal level.

COMMAND	NO. OF PARAMETER BYTES	NOTES
RESET	4	Display format parameters required
START DISPLAY	0	DMA operation parameters included in command
STOP DISPLAY	0	—
RED LIGHT PEN	2	—
LOAD CURSOR	2	Cursor X, Y position parameters required
ENABLE INTERRUPT	0	—
DISABLE INTERRUPT	0	—
PRESET COUNTERS	0	Clears all internal counters

Figure 4.1.1 8276 Instruction Set

4.3 Special Functions

4.3.1 Special Codes

The 8276 recognizes four special codes that may be used to reduce memory, software, or system controller overhead. These characters are placed within the display memory by the system controller. The 8276 performs certain tasks when these codes are received in its row buffer memory.

1) *End of Row Code* — Activates VSP. VSP remains active until the end of the line is reached. While VSP is active the screen is blanked.

2) *End Of Row-Stop Buffer Loading Code* — Causes the Buffer Ready control logic to stop requesting buffer transfers for the rest of the row. It affects the display the same as End of Row Code.

3) *End Of Screen Code* — Activates VSP. VSP remains active until the end of the frame is reached.

4) *End Of Screen-Stop Buffer Loading Code* — Causes the Buffer Ready control logic to stop requesting buffer transfers until the end of the frame is reached. It affects the display the same way as the End of Screen code.

4.3.4 Programmable Buffer Loading Control

The 8276 can be programmed to request 1, 2, 4, or 8 characters per Buffer load. The interval between loads is also programmable. This allows the designer the flexibility to tailor the buffer transfer overhead to fit the system needs.

Each scan line requires 63.5 microseconds. A character line consists of 10 scan lines and takes 635 microseconds to form. The 8276 row buffer must be filled within the 635 microseconds or an under run condition will occur within the 8276 causing the screen to be blanked until the next vertical retrace. This blanking will be seen as a flicker in the display.

5.0 DESIGN BACKGROUND

A fully functional, microcontroller-based CRT terminal was designed and constructed using the 8051 and the 8276. The terminal has many of the functions that are found in commercially available low cost terminals. Sophisticated features such as programmable keys can be added easily with modest amounts of software.

The 8051's functions in this application note include: up to 9.6K BAUD full duplex serial transmission; decoding special messages sent from the host computer; scanning, debouncing, and decoding a full function keyboard; writing to the 8276 row buffer from the display RAM without the need for a DMA controller; and scrolling the display.

The 8276 CRT controller's functions include: presenting the data to the character generator; providing the timing signals needed for horizontal and vertical retrace; and providing blanking and video information.

5.1 Design Philosophy

Since the device count relates to costs, size, and reliability of a system, arriving at a minimum device count without degrading the performance was a driving force for this application note. LSI devices were used where possible to maintain a low chip count and to make the design cycle as short as possible.

PL/M-51 was chosen to generate the majority of the software for this application because it models the human thought process more closely than assembly language. Consequently it is easier and faster to write programs using PL/M-51 and the code is more likely to be correct because less chance exists to introduce errors.

PL/M-51 programs are easier to read and follow than assembly language programs, and thus are easier to modify and customize to the end user's application. PL/M-51 also offers lower development and maintenance costs than assembly language programming.

PL/M-51 does have a few drawbacks. It is not as efficient in code generation relative to assembly language and thus may also run slower.

This application note uses the 8051's interrupts to control the servicing of the various peripherals. The speed of the main program is less critical if interrupts are used. In the last two application notes on terminal controllers, a criterion of the system was the time required for receiving an incoming serial byte, decoding it, performing the function requested, scanning the keyboard, debouncing the keys, and transmitting the decoded ASCII code must be less than the vertical refresh time. Using the 8051 and its interrupts makes this time constraint irrelevant.

5.2 System Target Specifications

The design specifications for the CRT terminal design is as follows:

Display Format

- 80 characters/display row
- 25 display lines

Character Format

- 5 × 7 character contained within a 7 × 10 frame
- First and seventh columns blanked
- Ninth line curser position
- Programmable delay blinking underline curser

Control Characters Recognized

- Backspace
- Linefeed
- Carriage Return
- Form Feed

Escape Sequences Recognized

- ESC A, Curser up
- ESC B, Curser down
- ESC C, Curser right
- ESC D, Curser left
- ESC E, Clear screen
- ESC F, Move addressable curser
- ESC H, Home curser
- ESC J, Erase from curser to the end the screen
- ESC K, Erase the current line

Characters Displayed

- 96 ASCII Alphanumeric Characters

Characters Transmitted

- 96 ASCII Alphanumeric Characters
- ASCII Control Character Set
- ASCII Escape Sequence Set
- Auto Repeat

Display Memory

- 2K × 8 static RAM

Data Rate

- Variable rate from 150 to 9600 BAUD

CRT Monitor

- Ball Bros TV-12, 12MHZ Black and White

Keyboard

- Any standard undecoded keyboard (2 key lock-out)
- Any standard decoded keyboard with output enable pin
- Any standard decoded serial keyboard up to 150 BAUD

Scrolling Capability

Compatible With Wordstar

6.0 SYSTEM DESCRIPTION

A block diagram of the CRT terminal is shown in figure 6.0.0. The diagram shows only the essential system features. A detailed schematic of the CRT terminal is contained in the Appendix 7.1.

The "brains" of the CRT terminal is the 8051 microcontroller. The 8276 is the CRT controller in the system, and a 2716 EPROM is used as the character generator. To handle the high speed portion of the CRT, the 8276 is surrounded by a handful of TTL devices. A 2K × 8 static RAM was used as the display memory.

Following the system reset, the 8276 is initialized for curser type, number of characters per line, number of lines, and character size. The display RAM is initialized to all "spaces" (ASCII 20H). The 8051 then writes the "start display" command to the 8276. The local/line input is sampled to determine the terminal mode. If the terminal is on-line, the BAUD rate switches are read and the serial port is set up for full duplex UART mode. The processor then is put into a loop waiting to service the serial port fifo or the 8276.

The serial port is programmed to have the highest priority interrupt. If the serial port generates an interrupt, the processor reads the buffer, puts the character in a generated fifo that resides in the 8051's internal RAM, increments the fifo pointer, sets the serial interrupt flag and returns.

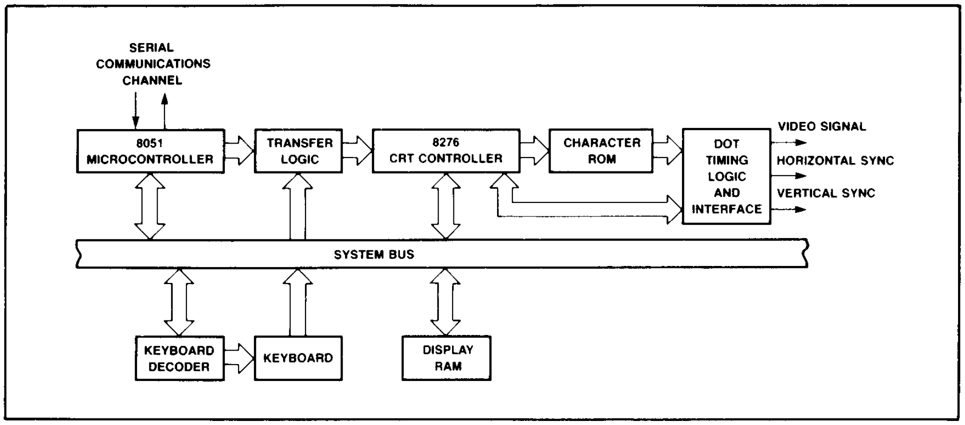


Figure 6.0.0 CRT Terminal Controller Block Diagram

The main program determines if it is a displayable character, a Control word or an ESC sequence and either puts the character in the display buffer or executes the appropriate command sent from the host computer.

If the 8276 needs servicing, the 8051 fills the row buffer for the CRT display's next line. If the 8276 generates a vertical retrace interrupt, the buffer pointers are reloaded with the display memory location that corresponds to the first character of the first display line on the CRT. The vertical retrace also signals the processor to read the keyboard for a key closure.

6.1 Hardware Description

The following section describes the unique characteristics of this design.

6.1.1 Peripheral Address Map

The display RAM, 8276 registers, and the 8276 row buffers are memory mapped into the external data RAM address area. The addresses are as follows:

Read and Write External Display RAM —	Address 1000H to 17CFH
Write to 8276 row buffers from Display RAM —	Address 1800H to 1FCFH
Write to 8276 Command Register (CREG) —	Address 0001H
Write to 8276 Parameter Register (PREG) —	Address 0000H
Read from 8276 Status Register (SREG) —	Address 0001H

Three general cases can be explored; reading and writing the display RAM, writing to the 8276 row buffers, and reading and writing the 8276's control registers.

As mentioned previously the 8051 fills the 8276 row buffer without the need of a DMA controller. This is accomplished by using a Quad 2-input multiplexor (Figure 6.1.0) as the transfer logic shown in the block diagram. The address line, P2.3, is used to select either of the two inputs. When the address line is low the $\overline{RD}$ and $\overline{WR}$ lines perform their normal functions, that is read and write the

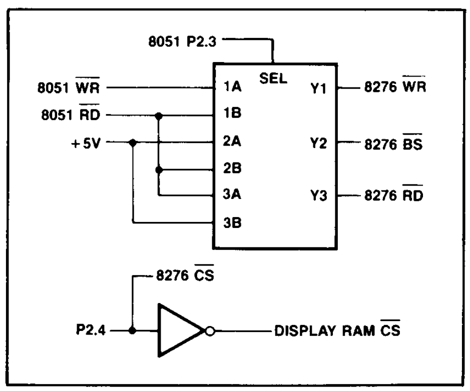


Figure 6.1.0
Simplified Version Of The Transfer Logic

8276 or the external display RAM depending on the states of their respective chip selects. If the address line is high, the 8051 $\overline{RD}$ line is transformed into $\overline{BS}$ and $\overline{WR}$ signals for the 8276. While holding the address line high, the 8051 executes an external data move (MOVX) from the display RAM to the accumulator which causes the display RAM to output the addressed byte onto the data bus. Since the multiplexor turns the same 8051 $\overline{RD}$ pulses into $\overline{BS}$ and $\overline{WR}$ pulses to the 8276, the data bus is thus read into the 8276 as a Buffer transfer. This scheme allows 80 characters to be transferred from the display RAM into the 8276 within the required character line time of 635 microseconds. The 8051 easily meets this requirement by accomplishing the task within 350 microseconds.

6.1.2 Scanning The Keyboard

Throughout this project, provision have been made to make the overall system flexible. The software has been written for various keyboards and the user simply needs to link different program modules together to suit their needs.

6.1.2.1 Undecoded Keyboard

Incorporating an undecoded keyboard controller into the other functions of the 8051 shows the flexibility and over all CPU power that is available. The keyboard in this case is a full function, non-buffered 8 × 8 matrix of switches for a total of 64 possible keys. The 8 send lines are connected to a 3-to-8 open-collector decoder as shown in Figure 6.1.1. Three high order address lines from the 8051 are the decoder inputs. The enabling of the decoder is accomplished through the use of the $\overline{PSEN}$ signal from the 8051 which makes the architecture of the separate address space for the program memory and the external data RAM work for us to eliminate the need to decode addresses externally. The move code (MOVC) instruction allows each scan line of the keyboard to be read with one instruction.

The keyboard is read by bringing one of the eight scan lines low sequentially while reading the return lines which are pulled high by an external resistor. If a switch is

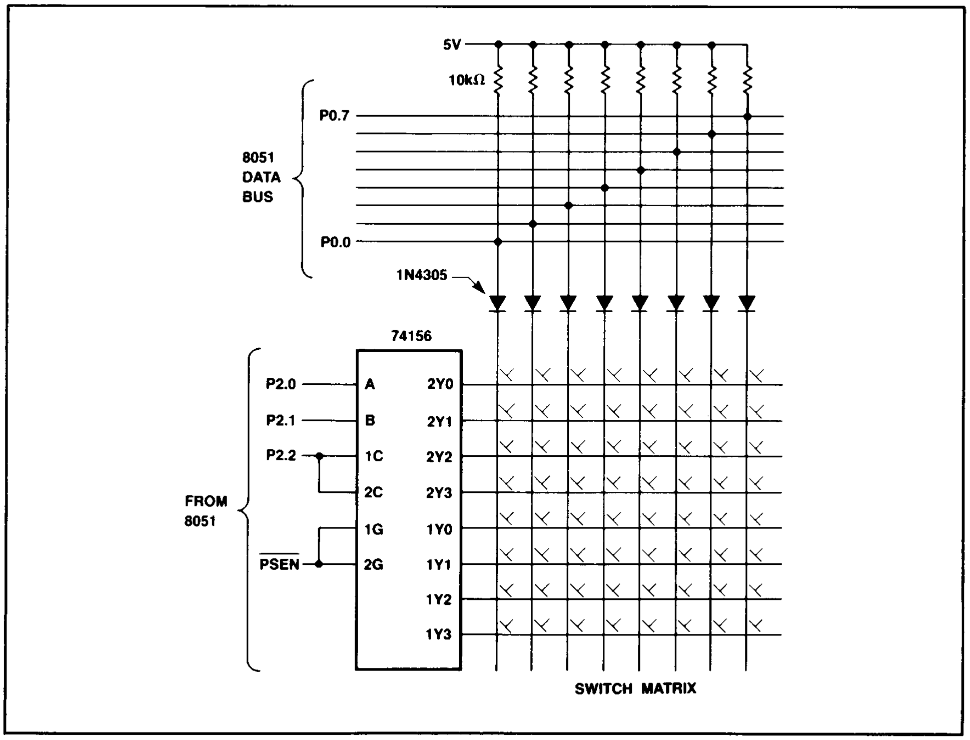


Figure 6.1.1 Keyboard

closed, the data bus line is connected through the switch to the low output of the decoder and one of the data bus lines will be read as a 0. By knowing which scan line detected a key closure and which data bus line was low, the ASCII code for that key can easily be looked up in a matrix of constants. PL/M-51 has the ability to handle arrays and structured arrays, which makes the decoding of the keyboard a trivial task.

Since the Shift, Cap Lock, and Control keys may change the ASCII code for a particular key closure, it is essential to know the status of these pins while decoding the keyboard. The Shift, Cap Lock, and Control keys are therefore not scanned but are connected to the 8051 port pins where they can be tested for closure directly.

The 8 receive lines are connected to the data bus through germanium diodes which chosen for their low forward voltage drop. The diodes keep the keyboard from interfering with the data bus during the times the keyboard is not being read. The circuit consisting of the 3-to-8 decoder and the diodes also offers some protection to the 8051 from possible Electrostatic Discharge (ESD) damage that could be transmitted through the keyboard.

6.1.2.2 Decoded Keyboard

A decoded keyboard can easily be connected to the system as shown in Figure 6.1.2. Reading the keyboard can be evoked either by interrupts or by software polling.

The software to periodically read a decoded keyboard was not written for this application note but can be accomplished with one or two PL/M-51 statements in the READER routine.

A much more interesting approach would be to have the servicing of the keyboard be interrupt driven. An additional external interrupt is created by configuring timer/counter 0 into an event counter. The event counter is initialized with the maximum count. The keyboard controller would inform the 8051 that a valid key has been depressed by pulling the input pin T0 low. This would overflow the event counter, thus causing an interrupt. The interrupt routine would simply use a MOVC ($\overline{PSEN}$ is connected to the output enable pin of the keyboard controller) to read the contents of the keyboard controller onto the data bus, reinitialize the counter to the maximum count and return from the interrupt.

6.1.2.3 Serial Decoded Keyboard

The use of detachable keyboards has become popular among the manufacturers of keyboards and personal computers. This terminal has provisions to use such a keyboard.

The keyboard controller would scan the keyboard, debounce the key and send back the ASCII code for that key closure. The message would be in an asynchronous serial format.

The flowchart for a software serial port is shown in Figure 6.1.3. An additional external interrupt is created as discussed for the decoded keyboard but the use in this case would be to detect a start bit. Once the beginning of the start bit has been detected, the timer/counter 0 is configured to become a timer. The timer is initialized to cause an interrupt one-half bit time after the beginning of the start bit. This is to validate the start bit. Once the start bit is validated, the timer is initialized with a value to cause an interrupt one bit time later to read the first data bit. This process of interrupting to read a data bit is repeated until all eight data bits have been received. After all 8 data bits are read, the software serial port is read once more to detect if a stop bit is present. If the stop bit is not present, an error flag is set, all pointers and flags are reset to their initial values, and the timer/counter is reconfigured to an event counter to detect the next start bit. If the stop bit is present, a valid flag is set and the flags and counter are reset as previously discussed.

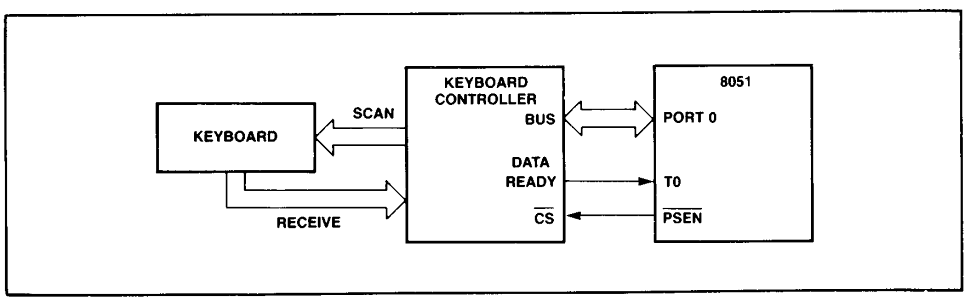


Figure 6.1.2 Using A Decoded Keyboard

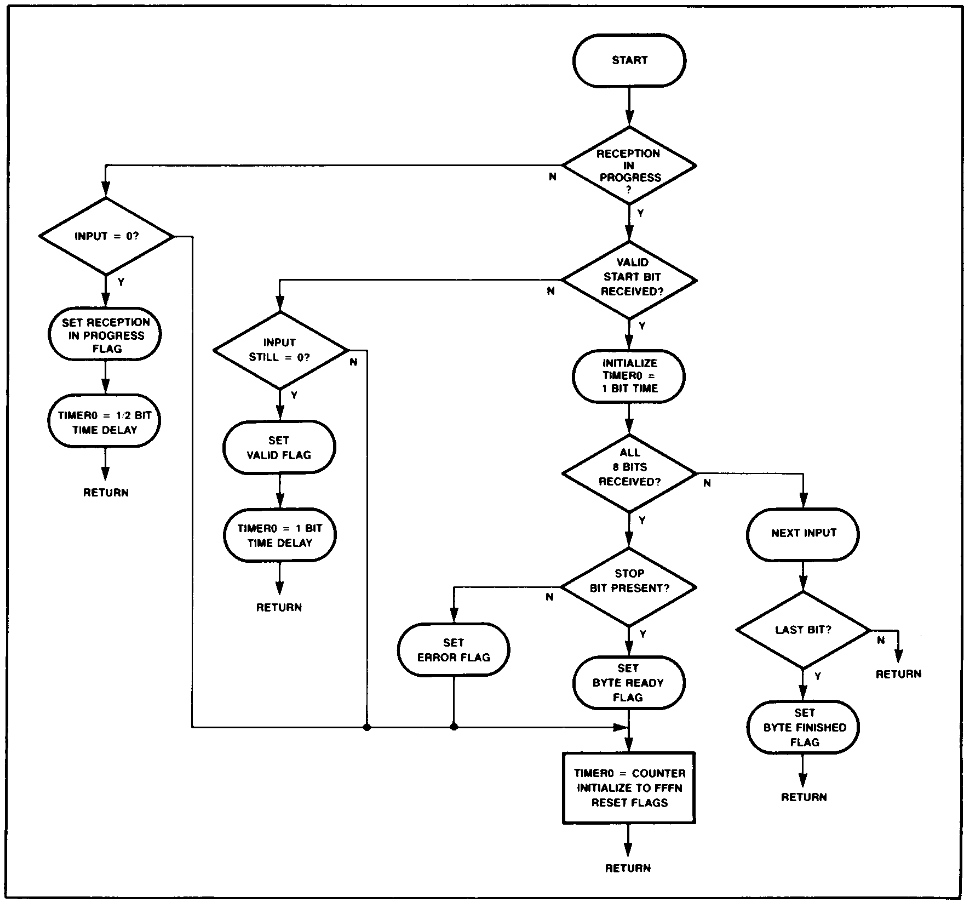


Figure 6.1.3 Flowchart for the Software Serial Port

6.1.4 System Timings

The requirements for the BALL BROTHERS. TV-12 monitor's operation is shown in table 6.1.0. From the monitor's parameters, the 8276 specifications and the system target specifications the system timing is easily calculated.

The 8276 allows the vertical retrace to be only an integer multiple of the horizontal character lines. Twenty-five display lines and a character frame of 7 × 10 are required from the target specification which will require 250 horizontal lines. If the horizontal frequency is to be within 500 Hz of 15,750 Hz, we must choose either one or two character line times for horizontal retrace. To allow for a little more margin at the top and bottom of the screen, two character line times was chosen for the vertical retrace. This choice yields 250 + 20 = 270 total character lines per frame. Assuming 60 Hz vertical retrace frequency:

60 Hz * 270 = 16,200 Hz horizontal frequency

and

1/16,200 Hz * 20 horizontal sync times = 1.2345 milliseconds

Table 6.1.0 CRT Monitor's Operational Requirements

PARAMETER	RANGE
Vertical Blanking Time (VRTC)	800 μsec nominal
Vertical Drive Pulsewidth	300 μsec ≤ PW ≤ 1.4 ms
Horizontal Blanking Time (HRTC)	11 μsec nominal
Horizontal Drive Pulsewidth	25 μsec ≤ PW ≤ 30 μsec
Horizontal Repetition Rate	15,750 ±500 pps

The 1.2345 milliseconds of retrace time meets the nominal VRTC and vertical drive pulse width time of .3mSec to 1.4mSec for the Ball monitor.

The next parameter to find is the horizontal retrace time which is wholly dependent on the monitor used. Usually it lies between 15 and 30 percent of the total horizontal line time.

Since most designs display a fixed number of characters per line it is useful to express the horizontal retrace time as a given number of character times. In this design, 80 characters are displayed, and it was experimentally found that 20 character times for the horizontal retrace gave the best results. It should be noted if too much time was given for retrace, there would be less time to display the characters and the display would not fill out the screen. Conversely, if not enough time is given for retrace, the characters would seem to run off the screen.

One hundred character times per complete horizontal line means that each character needs:

(1/16,200 Hz) /100 character times = 617.3 nanoseconds

If we multiply the 20 character times needed to retrace by 617.3 nanoseconds needed for each character, we find 12.345 microseconds are allocated for retrace. This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, a 74LS123 one-shot was used to extend the horizontal drive pulse width.

The dot clock frequency is easy to calculate now that we know the horizontal frequency. Since each character is formed by seven dots in the horizontal axis, the dot clock period would be the character clock (617.3 nanoseconds) divided by the 7 which is equal to 11.34 MHz. The basic dot timing and CRT timing are shown in the Appendix.

6.2 Software Description

6.2.1 Software Overview

The software for this application was written in a "foreground-background" format. The background programs are all interrupt driven and are written in assembly language due to time constraints. The foreground programs are for the most part written in PL/M-51 to ease the programming effort. A number of subroutines are written in assembly language due to time constraints during execution. Subroutines such as clearing display lines, clearing the screen, and scanning the keyboard require a great deal of 16 bit adds and compares and would execute much slower and would require more code space if written in PL/M-51. The background and foreground programs talk to each other through a set of flags. For example, the PL/M-51 foreground program tests "SERIAL$INT" to determine if a serial port interrupt had occurred and a character is waiting to be processed.

6.2.2 The Background Program

Two interrupt driven routines, VERT and BUFFER, (see Fig. 6.2.0) request service every 16.67 milliseconds and 617 microseconds respectively. VERT's request comes during the last character row of the display screen. This routine resets the buffer pointers to the first CRT display line in the display memory. VERT is also used as a time base for the foreground program. VERT sets the flag, SCAN, to tell the foreground program (PL/M-51) that it is time to scan the Keyboard. VERT also increments a counter used for the delay between transmitting characters in the AUTO$REPEAT routine.

The BUFFER routine is executed once per character row. BUFFER uses the multiplexor discussed earlier to fill the 8276's row buffer by executing 80 external data moves and incrementing the Data Pointer between each move.

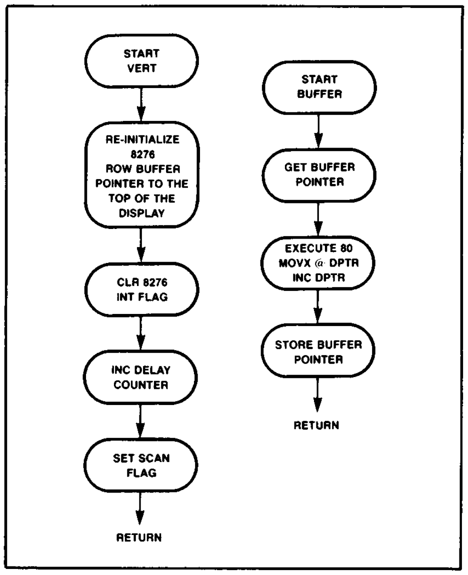


Figure 6.2.0 Flowcharts For VERT and BUFFER Routine

The MOVX reads the display RAM and writes the character into the row buffer during the same instruction.

SERBUF is an interrupt driven routine that is executed each time a character is received or transmitted through the on-chip serial port. The routine first checks if the interrupt was caused by the transmit side of the serial port, signaling that the transmitter is ready to accept another character. If the transmitter caused the interrupt, the flag "TRANSMIT$INT" is set which is checked by the foreground program before putting a character in the buffer for transmission.

If the receiver caused the interrupt, the input buffer on the serial port is read and fed into the fifo that has been manufactured in the internal RAM and increments the fifo pointer "FIFO." The flag "SERIAL$INT" is then set, telling the foreground program that there is a character in the fifo to be processed. If the read character is an ESC character, the flag "ESCSEQ" is set to tell the foreground program that an escape sequence is in the process of being received.

6.2.3 The Foreground Program

The foreground program is documented in the Appendix. The foreground program starts off by initializing the 8276 as discussed earlier. After all variables and flags are initialized, the processor is put into a loop waiting for either VERT to set SCAN so the program can scan the keyboard, or for the serial port to set SERIAL$INT so the program can process the incoming character.

The vertical retrace is used to time the delay between keyboard scans. When VERT gets set, the assembly language routine READER is called. READER scans the keyboard, writing each scan into RAM to be processed later. READER controls two flags, KEY0 and SAME. KEY0 is set when all 8 scans determine that no key is pressed. SAME is set when the same key that was pressed last time the keyboard was read is still pressed.

After READER returns execution to the main program, the flags are tested. If the KEYO flag is set the main program goes back to the loop waiting for the vertical retrace or a serial port interrupt to occur. If the SAME flag is set the main program knows that the closed key has been debounced and decoded so it sends the already known ASCII code to the AUTO$REPEAT routine which determines if that character should be transmitted or not.

If KEY0 and SAME are not set, signifying that a key is pressed but it is not the same key as before, the foreground program determines if the results from the scan are valid. First all eight scans are checked to see if only one key was closed. If only one key is closed, the ASCII code is determined, modified if necessary by the Shift, Cap Lock, or Control keys. The NEW$KEY and VALID flags are then set. The next time READER is called, if the same key is still pressed, the SAME flag will be set, causing the AUTO$REPEAT subroutine to be called as just discussed. Since the keyboard is read during the vertical retrace, 16.67 milliseconds has elapsed between the detection of the pressed key and reverifying that the key is still pressed before transmitting it, thus effectively debouncing the key.

The AUTO$REPEAT routine is written to transmit any key that the NEW$KEY flag is set for. The counter that is incremented each time the vertical refresh interrupt is serviced causes a programmable delay between the first transmission and subsequent auto repeat transmission. Once the NEW$KEY character is sent, the counter is initialized. Each time the AUTO$REPEAT routine is called, the counter is checked. Only when the counter overflows will the next character be transmitted. After the initial delay, a character will be transmitted every other time the routine is called as long as the key remains pressed.

6.2.3.1 Handling Incoming Serial Data

One of the criteria for this application note was to make the software less time dependent. By creating a fifo to store incoming characters until the 8051 has time to process them, software timing becomes less critical. This application note uses up to 8 levels of the fifo at 9.2KBAUD, and 1 level at 4.8KBAUD and lower. As discussed earlier, the interrupt service routine for the serial port uses the fifo to store incoming data, increments the fifo pointer, "FIFO", and sets SERIAL$INT to tell the main program that the fifo needs servicing. Once the main program detects that SERIAL$INT is set the routine DECIPHER is executed.

DECIPHER has three separate blocks; a block for decoding displayable characters, a block for processing Escape sequences, and a block for processing Control codes. Each block works on the fifo independently. Before exiting a block, the contents of the fifo are shifted up by the amount of characters that were processed in that particular block. The shifting of the characters insures that the beginning of the fifo contains the next character to be processed. FIFO is then decremented by the number of characters processed.

Let's look at this process more closely. Figure 6.2.1-A shows a representation of a fifo containing 5 characters. The first three characters in the fifo contain displayable characters, A, B, and C respectively with the last two characters being an ESC sequence for moving the curser up one line (ESC A) and FIFO points to the next available location to be filled by the serial port interrupt routine, in this case, 5.

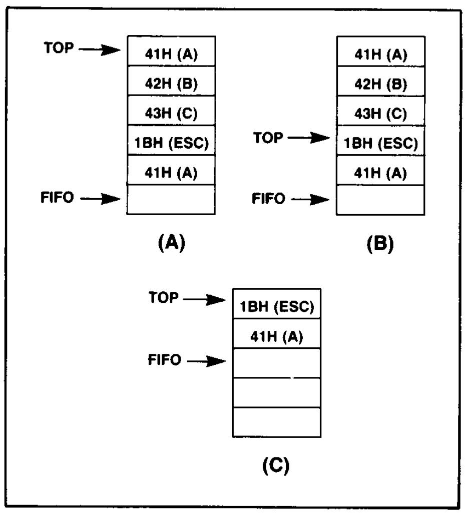


FIGURE 6.2.1 FIFO

When DECIPHER is executed, the first block begins looking at the first character of the fifo for a displayable character. If the character is displayable, it is placed into the display RAM and the software pointer "TOP" that points to the character that is being processed is incremented to the next character. The character is then looked at to see if it too is displayable and if it is, it's placed in the display RAM. The process of checking for displayable characters is continued until either the fifo is empty or a non-displayable character is detected. In our example, three characters are placed into the display RAM before a non-displayable character is detected. At this point the fifo looks like figure 6.2.1-B.

Before entering the next block, the remaining contents of the fifo between TOP, that is now pointing to 1BH and (FIFO-1) are moved up in the fifo by the amount of characters processed, in this example three. TOP is reset to 0 and FIFO is decremented by 3. The serial port interrupt is inhibited during the time the contents of the fifo and the pointers are being manipulated. The fifo now looks like figure 6.2.1-C.

The execution is now passed to the next block that processes ESC sequences. The first location of the fifo is examined to see if it is an ESC character (1BH). If not, the execution is passed to the next block of DECIPHER that processes Control codes. In this case the fifo does contain an ESC code. The flag ESC$SEQ is checked to see if the 8051 is in the process of receiving an ESC sequence thus signifying that the next byte of the sequence has not been received yet. If the ESC$SEQ is not set, the next character in the fifo is checked for a valid escape code and the proper subroutine is then called. The fifo contents are then shifted as discussed for the previous block. Due to the length of time that is needed to execute an ESC code sequence or a Control code, only one ESC code and/or Control code can be processed each time DECIPHER is executed.

If at the end of the DECIPHER routine, FIFO contains a 0, the flag SER$INT is reset. If SER$INT remains set, DECIPHER will be executed immediately after returning to the main program if SCAN had not been set during the execution of the DECIPHER routine, otherwise DECIPHER will be called after the keyboard is read.

6.2.4 Memory Pointers and Scrolling

The curser always points to the next location in display memory to be filled. Each time a character is placed in the display memory, the curser position needs to be tested to determine if the curser should be incremented to the beginning of the next line of the display or simply moved to the next position on the current display line. The curser position pointers are then updated in both the 8276 and the internal registers in the 8051.

When the 2000th character is entered into the display memory, a full display page has been reached signaling the need for the display to scroll. The memory pointer that points to the display memory that contains the first character of the first display line, LINE0, prior to scrolling contains 1800H which is the starting address of the display memory. Each scrolling operation adds 80 (50H) to LINE0 which will now point to the following row in memory as shown in figure 6.2.2-B. LINE0 is used during the vertical refresh routine to re-initialize the pointers associated with filling the 8276 row buffers.

The display memory locations that were the first line of the CRT display now becomes the last line of the CRT display. Incoming characters are now entered into the display memory starting with 1800H, which is now the first character of the last line of the display screen.

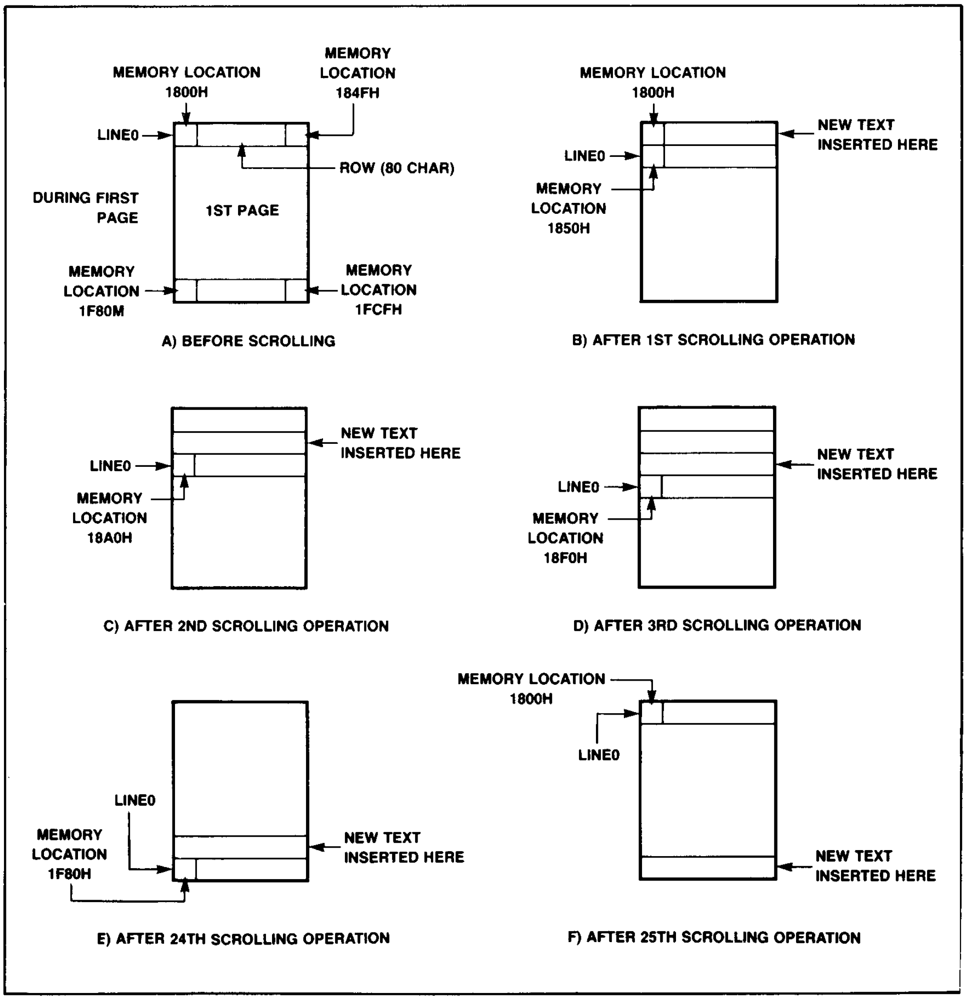


Figure 6.2.2 Pointer Manipulation During Scrolling

6.2.5 Software Timing

The use of interrupts to tie the operation of the foreground program to the real-time events of the background program has made the software timing non-critical for this system.

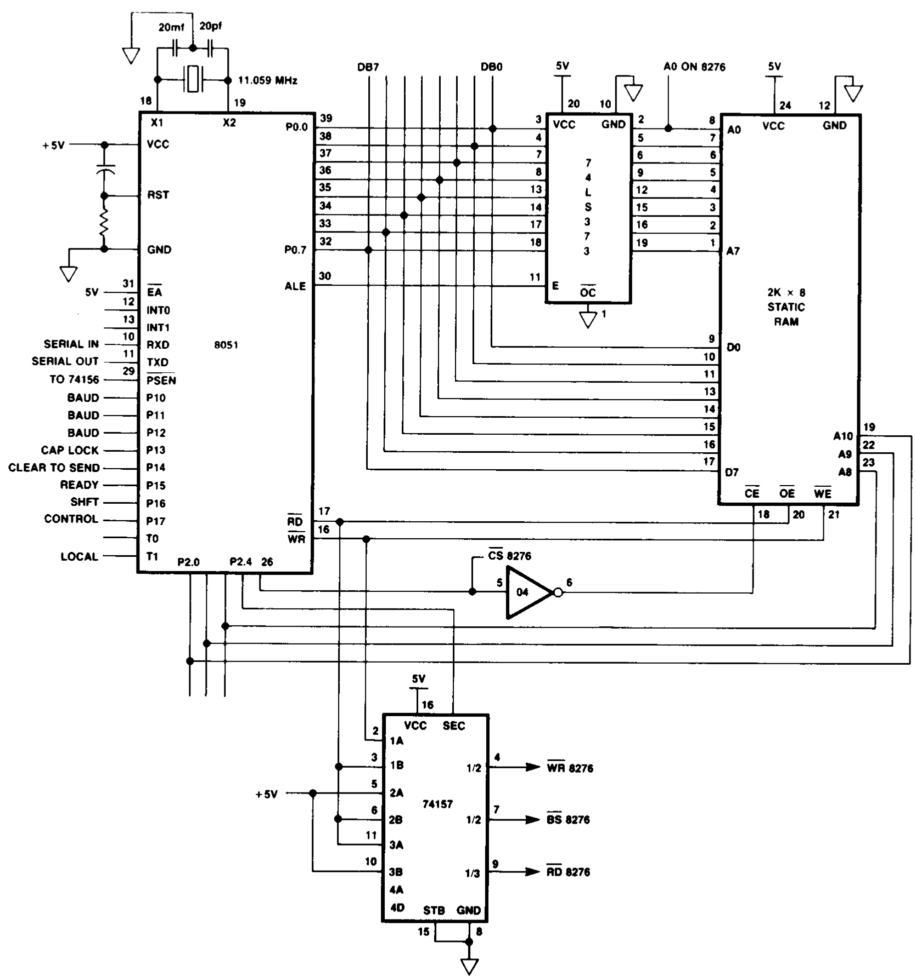
6.3 System Operation

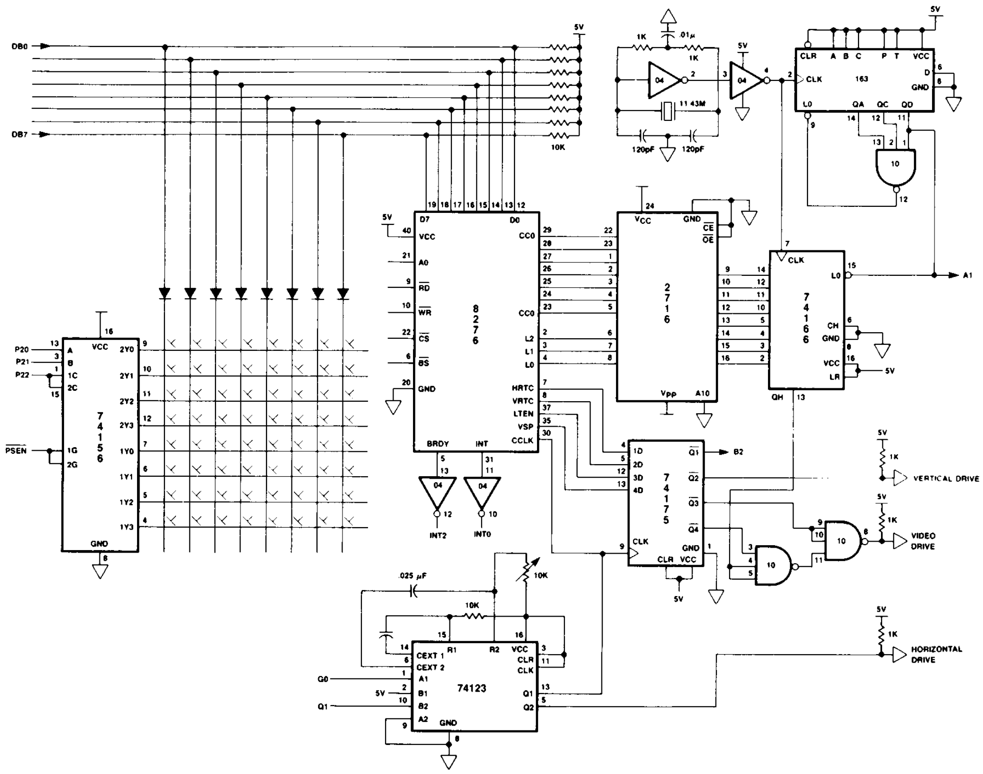
Following the system reset, the 8051 initializes all on-chip peripherals along with the 8276 and display ram. After initialization, the processor waits until the fifo has a character to process or is flagged that it is time to scan the keyboard. This foreground program is interrupted once every 617 microseconds to service the 8276 row buffers. The 8051 is also interrupted each 16.67 milliseconds to re-initialize LINE0 and to flag the foreground program to read the keyboard.

As discussed earlier, a special technique of rapidly moving the contents of the display RAM to the 8276 row buffers without the need of a DMA device was employed. The characters are then synchronously transferred to the character generator via CC0-CC6 and LC0-LC2 which are used to display one line at a time. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line is selected. This process continues until the last line of the character is transferred.

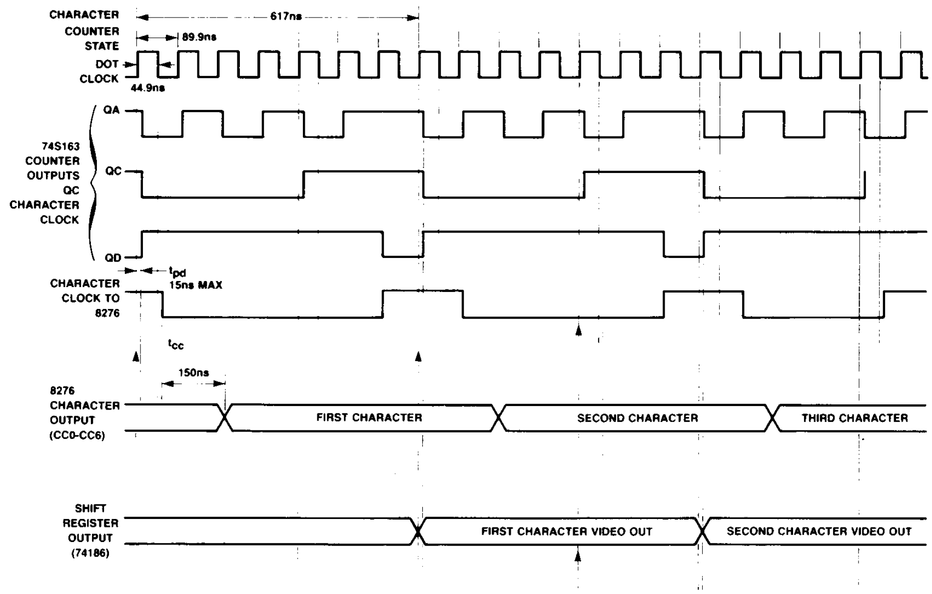
The dot timing logic latches the ouput of the character ROM in a parallel in, serial out synchronous shift register. The shift register's output constitutes the video information to the CRT.

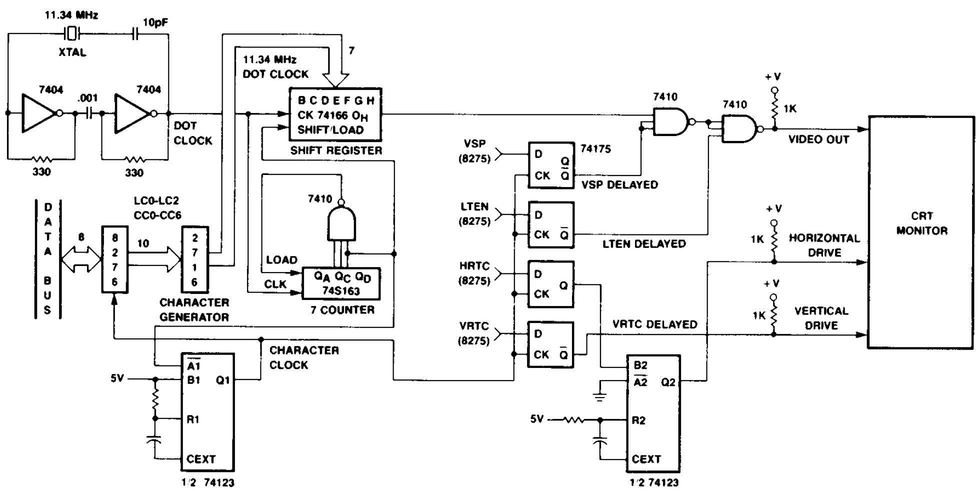
Appendix 7.1 CRT Schematics



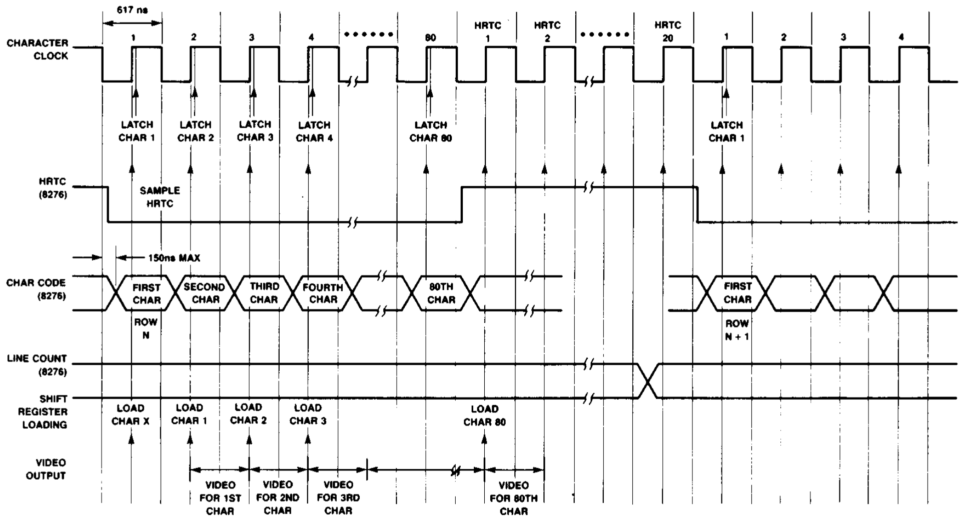


Appendix 7.2 Dot Timing





Appendix 7.3 CRT System Timing



Appendix 7.4 Escape/Control/Display Character Summary

BIT	CONTROL CHARACTERS 000	001	010	DISPLAYABLE CHARACTER 011	100	101	110	111	010	011	ESCAPE SEQUENCE 100	101	110	111
0000	NUL @	DLE P	SP	ø	@	P		P						
0001	SOH A	DC1 Q	!		A	Q	A	Q			↑ A			
0010	STX B	DC2 R	"	2	B	R	B	R			↓ B			
0011	ETX C	DC3 S	=	3	C	S	C	S			→ C			
0100	EOT D	DC4 T	$	4	D	T	D	T			← D			
0101	ENQ E	NAK U	%	5	E	U	E	U			CLR E			
0110	ACK F	SYN V	&	6	F	V	F	V						
0111	BEL G	ETB W	'	7	G	W	G	W						
1000	BS H	CAN X	(	8	H	X	H	X			HOME H			
1001	HT I	EM Y	)	9	I	Y	I	Y						
1010	LF J	SUB Z	*	:	J	Z	J	Z			EOS I			
1011	VT K	ESC !	+	;	K	[	K				EL J			
1100	FF L	FS	,		L		L							
1101	CR M	GS	–	=	M	]	M							
1110	SO N	RS \	•		N	∧	N							
1111	S1 ○	US –	/	?	O	–	O							

NOTE: Shaded blocks — functions terminal will react to. Others can be generated but are ignored upon receipt.

Appendix 7.5 Character Generator

As previously mentioned, the character generator used in this terminal is a 2716 EPROM. A 1K by 8 device would have been sufficient since a 128 character 5 by 7 dot matrix only requires 8K of memory. A custom character set could have been stored in the second 1K bytes of the 2716. Any of the free I/0 pins on the 8051 could have been used to switch between the character sets.

The three low-order line count outputs (LC0–LC2) from the 8276 are connected to the three low-order address lines of the character generator. The CC0–CC6 output lines are connected to the A3–A9 lines of the character generator.

The output of the character generator is loaded into the shift register. The serial output of the shift register is the video output to the CRT.

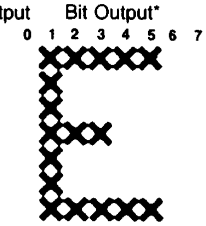
Let's assume that the letter "E" is to be displayed. The ASCII code for "E" (45H) is presented to the address lines A2–A9 of the character generator. The scan lines (LC0–LC2) will now count from 0 to seven to form the character as shown in Figure 7.5.0. The same procedure is used to form all 128 possible characters. For reference Appendix 7.6 contains the HEX dump of the character generator used in this terminal.

45H = 01000101
Address to Prom = 01000101
= 228H - 22FH
Depending on state of Scan lines.

Character generator output

Rom Address	Rom Hex Output
228H	3E
229H	02
22AH	02
22BH	0E
22CH	02
22DH	02
22EH	3E
22FH	00

Bit Output*



Bits 0, 6 and 7 are not used.
*note bit output is backward from convention.

Figure 7.5.0 Character Generator

Appendix 7.6 Hex Dump of the Character Generator

```
:1000000001000000000000000000000000000000EF
:1000100000000000000000000000000000000000E0
:1000200000000000000000000000000000000000D0
:1000300000000000000000000000000000000000C0
:1000400000000000000000000000000000000000B0
:1000500000000000000000000000000000000000A0
:100060000000000000000000000000000000000090
:100070000000000000000000000000000000000080
:100080000000000000000000000000000000000070
:10009000000000000000000082A1C031C2A03000003C
:1000A0000000000000000000000000000000000050
:1000B0000000000000000000000000000000000040
:1000C0000000000000000000000000000000000030
:1000D0000000000000000000000000000000000020
:1000E0000000000000000000000000000000000010
:1000F0000000000000000000000000000000000000
:100100000000000000000000000000000808080808000800BF
:100110001414140000000000000014143E143E141400C3
:10012000083C0A1C281E08002626100804323200 5D
:10013000040A0A042A122C000808080000000000 23
:100140000804020202040800081020202010080001
:1001500000082A1C081C2A080008083E080800009D
:10016000000000000008080400000003C0000000 3F
:100170000000000000001800002010080402000029
:100180001C22322A26221C00080C080808081C0021
:100190001C22201C02023E003E20101820221C00BF
:1001A0001018141 23E101000003E021E2020221C00C7
:1001B00038040 21E22221C003E201008040404 0001
:1001C0001C22221C22221C001C22223C20100E0079
:1001D0000000080000080000000008000008080 4F3
:1001E00010080402040810000000003E003E00000059
:1001F00004081020100804001C2220100800080021
:100200001C222A3A1A023C0008142222 3E22220012
:100210001E24241C24241E001C2202020222 1C0074
:100220001E24242424241E003E02021E02023E004C
:100230003E02021E020202003C02023A22223C005E
:10024000222222 3E222222001C08080808081C0044
:100250007020202020221C0022120A060A12220 0EE
:100260000202020202023E0022362A2A2222220032
:10027000222226 2A322222001C22222222221C0092
:100280001E22221E020202001C2222222A122C00FE
:100290001E22221E0A1222003C02021C20201E00E6
:1002A0003E08080808080800222222222222 1C00F9
:1002B00022222222221408002222222 22A3622005E
:1002C00022221408142222002222221408080800E4
:1002D0003E201008040 23E001C04040404041C0018
:1002E00000020408102000003820202020203800C0
:1002F000081C2A080808080000000000000000 7E12
:1003000000080810000000000000 3C203C223C004E
:1003100002021A2622221E0000003804040438003B
:100320002020 2C3222223C0000003824 3C0438005B
:100330003824040E040404000000 3C22223C203CAB
:1003400002021A262222220008000C0808081C0043
:10035000200020202 0A42418020222120A162200C3
:1003600008080808080 81C000000362A2A2222007F
:1003700000001A262222220000001824242418003B
:1003800000001E22221E020200001C22223C20209D
:1003900000001A260202020000003804 18201C0097
:1003A00000081C080808300000002222 22324C0095
:1003B000000022222214080000002222 2A3E1400F3
:1003C0000000221408142200000022222 23C20383F
:1003D00000003E1008043E00188897398891 90002F
:1003E00000081804080818083C090912197910D051
:1003F000000000009C2BB00100000000000000000095
```

Appendix 7.7 Composite Video

In this design it was assumed that the CRT monitor required a separate horizontal drive, vertical drive, and video input. Many monitors require a composite video signal. The schematic shown in Figure 7.7.0 illustrate how to generate a composite video from the output of the 8276.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the horizontal and vertical timing is used to center the display. The 7486 is used to mix the vertical and horizontal retrace. Q1 mix the video and retrace signals along with providing the proper D.C. levels.

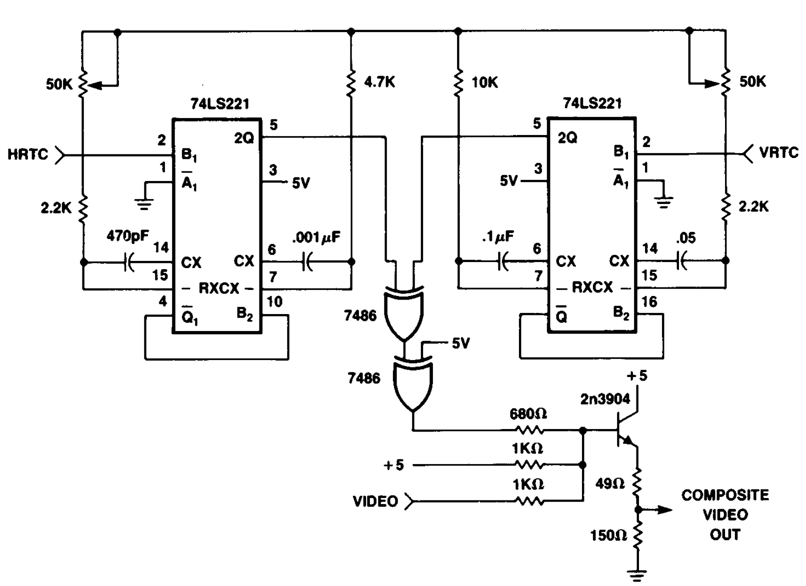


Figure 7.7.0 Composite Video

Appendix 7.8 Software Documentation

```
/*****************************************************************************
******************************************************************************
*******                                                               *******
*******         SOFTWARE DOCUMENTATION FOR THE 8051                   *******
*******         TERMINAL CONTROLLER APPLICATION NOTE                  *******
*******                                                               *******
******************************************************************************
******************************************************************************

        MEMORY MAP ASSOCIATED WITH PERIPHERAL DEVICES (USING MOVX):

        8051 WR AND READ DISPLAY RAM-    ADDRESS 1000H TO 17CFH
        8051 WR DISPLAY RAM TO THE 8276- ADDRESS 1800H TO 1FCFH
        8276 COMMAND ADDRESS-            ADDRESS 0001H
        8276 PARAMETER ADDRESS-                 ADDRESS 0000H
        8276 STATUS REGISTER-                   ADDRESS 0001H

                MEMORY MAP FOR READING THE KEYBOARD (USING MOVC):

        KEYBOARD ADDRESS-                              ADDRESS 10FFH TO 17FFH



    /******************* START MAIN PROGRAM **************************/

    /* BEGIN BY PUTTING THE ASCII CODE FOR BLANK IN THE DISPLAY RAM*/

    INIT:
    {FILL 2000 LOCATIONS IN THE DISPLAY RAM WITH SPACES (ASCII 20H)}

    /*          INITIALIZE POINTERS, RAM BITS, ETC.       */

    {INITIALIZE POINTERS AND FLAGS}
    {INITIALIZE TOP OF THE CRT DISPLAY "LINE0"=1800H}
    {INITIALIZE 8276 BUFFER POINTER "RASTER" =1800H}
    {INITIALIZE DISPLAY$RAM$POINTER=0000H}

            /* INITIALIZE THE 8276 */

    { RESET THE 8276}
    {INITIALIZE 8276 TO 80 CHARACTER/ROW }
    {INITIALIZE 8276 TO 25 ROWS PER FRAME}
    {INITIALIZE 8276 TO 10 LINES PER ROW}
    {INITIALIZE 8276 TO NON-BLINKING UNDERLINE CURSER}
    {INITIALIZE CURSER TO HOME POSITION (00,00)(UPPER LEFT HAND CORNER)}
    {START DISPLAY}
    {ENABLE 8276 INTERRUPT}

            /* SET UP 8051 INTERRUPTS AND PRIORITIES */

    {SERIAL PORT HAS HIGHEST INTERRUPT PRIORITY}
    {EXTERNAL INTERRUPTS ARE EDGE SENSITIVE}
    {ENABLE EXTERNAL INTERRUPTS}
```

```
/*PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A
  SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT
  WILL BE TRANSMITTED TO THE HOST COMPUTER.*/

SCANNER:

{ENABLE 8051 GLOBAL INTERRUPT BIT}


/*     PROGRAMMABLE DELAY FOR THE CURSER BLINK     */

IF {30 VERTICAL RETRACE INTERRUPTS HAVE OCCURRED (CURSER$COUNT=1FH)} THEN
DO;
   {COMPLEMENT CURSER$ON}
   {CLEAR CURSER$COUNT}
   IF {CURSER IS TO BE OFF (CURSER$ON=0)} THEN {MOVE CURSER OFF THE SCREEN}
   CALL LOAD$CURSER;
END;

IF {THE LOCAL$LINE SWITCH HAS CHANGED STATE} THEN
DO;
   IF {IN LOCAL MODE} THEN {DISABLE SERIAL PORT INTERRUPT}
   ELSE CALL CHECK$BAUD$RATE;
END;

DO WHILE {INBETWEEN VERTICAL REFRESHES}
   IF {THE FIFO HAS A CHARACTER TO PROCESS (SERIAL$INT=1)} THEN CALL DECIPHER;
END;

CALL READER;

IF {THE PRESENT PRESSED KEY IS EQUAL TO THE LAST KEY PRESSED AND VALID=1} THEN
   CALL AUTO$REPEAT;
ELSE
DO;
   IF {A KEY IS PRESSED BUT NOT THE SAME ONE AS THE LAST KEYBOARD SCAN} THEN
   DO;
      IF {ONLY ONE KEY IS PRESSED} THEN
           {GET THE ASCII CODE FOR IT}
           {SET NEW$KEY AND VALID FLAGS}
      ELSE {RESET VALID AND NEW$KEY FLAGS}
   END;
   ELSE {THE KEYBOARD MUST NOT HAVE A KEY PRESSED SO RESET VALID$KEY AND NEW$KEY FLAGS}
END;

GOTO SCANNER;
END;


/*    PROCEDURE AUTO$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION
      BY TRANSMITTING A CHARACTER EVERY OTHER TIME THIS ROUTINE IS CALLED.
      THE AUTO REPEAT FUNCTION IS ACTIVATED AFTER A FIXED DELAY PERIOD AFTER THE
      FIRST CHARACTER IS SENT*/


AUTO$REPEAT:

IF {THE KEY PRESSED IS NEW (NEW$KEY=1} THEN
DO;
   {CLEAR THE DIVIDE BY TWO COUNTER "TRANSMIT$TOGGLE"}
   {INITIALIZE THE DELAY COUNTER "TRANSMIT$COUNT" TO 0D0H}
   CALL TRANSMIT;                                       /* FIRST CHARACTER */
   {CLEAR NEW$KEY}
END;
```

```
ELSE
DO;
   IF {TRANSMIT$COUNT IS NOT EQUAL TO 0} THEN
   DO;
      {INCREMENT TRANSMIT$COUNT}
      IF TRANSMIT$COUNT=0FFH THEN                        /*DELAY BETWEEN FIRST CHARACTER AND THE SECOND */
      DO;
         CALL TRANSMIT;                                  /*SECOND CHARACTER */
         {CLEAR TRANSMIT$COUNT}
      END;
   END;
   ELSE
   DO;
      {TURN THE CURSER ON DURING THE AUTO REPEAT FUNCTION}
      IF TRANSMIT$TOGGLE = 1 THEN                        /* 2 VERT FRAMES BETWEEN 3RD TO NTH CHARACTER */
         CALL TRANSMIT;                                  /* 3RD THROUGH NTH CHARACTER */
      {COMPLEMENT TRANSMIT$TOGGLE}
   END;
END;

END AUTO$REPEAT;


/* PROCEDURE TRANSMIT- ONCE THE HOST COMPUTER SIGNALS THE 8051H BY BRINGING
   THE CLEAR-TO-SEND LINE LOW, THE ASCII CHARACTER IS PUT INTO THE SERIAL PORT.*/


TRANSMIT:
PROCEDURE;
IF {THE TERMINAL IS ON-LINE} THEN
DO;
   {WAIT UNTIL THE CLEAR$TO$SEND LINE IS LOW AND UNTIL THE 8051 SERIAL PORT TX IS NOT BUSY (TRANSMIT$INT=1)}
   {TRANSMIT THE ASCII CODE}
   {CLEAR THE FLAG "TRANSMIT$INT". THE SERIAL PORT SERVICE ROUTINE WILL SET THE FLAG
   {WHEN THE SERIAL PORT IS FINISHED TRANSMITTING}
END;
ELSE {THE TERMINAL IS IN THE LOCAL MODE}
DO;
   {PUT THE ASCII CODE IN THE FIFO}
   {INCREMENT THE FIFO POINTER}
   {SET SERIAL$INT}
END;
END TRANSMIT;
```

```
  /*      PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES
  WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE
  THE PROCEDURE THEN ACTS ACCORDINGLY */


DECIPHER:
START$DECIPHER:

VALID$RECEPTION=0;
DO WHILE {THE FIFO IS NOT EMPTY AND THE CHARACTER IS DISPLAYABLE}
   RECEIVE={ASCII CODE}
   CALL DISPLAY;
   {NEXT CHARACTER}
END;

IF {CHARACTERS WERE DISPLAYED} THEN
   {DISABLE SERIAL PORT INTERRUPT}
   {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
   {ENABLE SERIAL PORT INTERRUPT}
   {SET THE VALID$RECEPTION FLAG}

IF {THE FIFO IS EMPTY} THEN {CLEAR THE "SERIAL$INT FLAG AND RETURN}

IF {THE NEXT CHARACTER IS AN "ESC" CODE } THEN
DO:
   {LOOK AT THE CHARACTER IN THE FIFO AFTER THE ESC CODE AND CALL THE CORRECT SUBROUTINE}
        ;
        CALL UP$CURSER;                                  /* ESC A */
        CALL DOWN$CURSER;                                /* ESC B */
        CALL RIGHT$CURSER;                               /* ESC C */
        CALL LEFT$CURSER;                                /* ESC D */
        CALL CLEAR$SCREEN;                               /* ESC E */
        CALL MOV$CURSER;                                 /* ESC F */
        ;
        CALL HOME;                                       /* ESC H */
        ;
        CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;         /* ESC J */
        CALL BLINE;                                      /* ESC K */

   {DISABLE THE SERIAL PORT INTERRUPT}
   {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
   {ENABLE THE SERIAL PORT INTERRUPT}
   {SET THE "VALID$RECEPTION" FLAG}

   IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}
END;
```

```
IF {THE NEXT CHARACTER IS A CONTROL CODE} THEN
DO;
   {CALL THE RIGHT SUBROUTINE}

      CALL LEFT$CURSER;                                    /* CTL H */
      ;
      CALL LINE$FEED;                                      /* CTL J */
      ;
      CALL CLEAR$SCREEN;                                   /* CTL L */
      CALL CARRIAGE$RETURN;                                 /* CTL M */

   {DISABLE THE SERIAL PORT INTERRUPT}
   {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
   {ENABLE THE SERIAL PORT INTERRUPT}
   {SET THE "VALID$RECEPTION" FLAG}
END;

IF {NO VALID CODE WAS RECEIVED ("VALID$RECEPTION" IS 0)} THEN
  {THROW THE CHARACTER OUT AND MOVE THE REMAINING CONTENTS OF THE FIFO}
  {UP TO THE BEGINNING}

IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}

END DECIPHER;



/*        PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
        RECEIVE AND PUT IT INTO THE DISPLAY RAM. */


DISPLAY:

{PUT INTO THE DISPLAY RAM LOCATION POINTED TO BY "DISPLAY$RAM$POINTER
 THE CONTENTS OF RECEIVE}

IF {THE END OF THE DISPLAY MEMORY HAS BEEN REACHED} THEN
   {RESET "DISPLAY$RAM$POINTER" TO THE BEGINNING OF THE RAM}
ELSE
   {INCREMENT "DISPLAY$RAM$POINTER"}

IF {THE CURSER IS IN THE LAST COLUMN OF THE CRT DISPLAY} THEN
DO;
   {MOVE THE CURSER BACK TO THE BEGINNING OF THE LINE}
   IF {THE NEW DISPLAY RAM LOCATION HAS A END-OF-LINE CHARACTER IN IT} THEN
      CALL FILL;

   IF {THE CURSER IS ON THE LAST LINE OF THE CRT DISPLAY} THEN
      CALL SCROLL;
   ELSE
      {MOVE THE CURSER TO THE NEXT LINE}
END;
ELSE
   {INCREMENT THE CURSER TO THE NEXT LOCATION}

{TURN THE CURSER ON }
CALL LOADCURSER;
END DISPLAY;
```

```
/*        PROCEDURE LINE$FEED      */

LINE$FEED:

IF {THE CURSER IS IN THE LAST LINE OF THE CRT DISPLAY} THEN
   CALL SCROLL;
ELSE
DO;
   {MOVE THE CURSER TO THE NEXT LINE}
   {TURN THE CURSER ON}
   CALL LOAD$CURSER;
END;

IF {THE DISPLAY$RAM$POINTER IS ON THE LAST LINE IN THE DISPLAY RAM} THEN
   {MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY RAM}
ELSE
   {MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY RAM}

IF {THE FIRST CHARACTER IN THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER } THEN
   CALL FILL;

END LINE$FEED;



/*        PROCEDURE SCROLL         */

SCROLL:

CALL BLANK;
{DISABLE VERTICAL RETRACE INTERRUPT}

IF {THE FIRST LINE OF THE CRT CONTAINS THE LAST LINE OF THE DISPLAY MEMORY} THEN
   { MOVE THE POINTER "LINE0" TO THE BEGINNING OF THE DISPLAY MEMORY}
ELSE
   {MOVE "LINE0" TO THE NEXT LINE IN THE DISPLAY MEMORY}

{ENABLE VERTICAL RETRACE INTERRUPT}

END SCROLL;



/*        PROCEDURE CLEAR SCREEN      */

CLEAR$SCREEN:

CALL HOME;
CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;

END CLEAR$SCREEN;
```

```
/*        PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION */

HOME:

{MOVE THE CURSER POSITION TO THE UPPER LEFT HAND CORNER OF THE CRT}
{TURN THE CURSER ON}
CALL LOAD$CURSER;
{MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION IN THE DISPLAY RAM}

END HOME;



/*    PROCEDURE ERASE FROM CURSER TO END OF SCREEN:  */

ERASE$FROM$CURSER$TO$END$OF$SCREEN:

CALL BLINE;                                /* ERASE CURRENT LINE */

IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN
    STARTING WITH THE NEXT LINE,PUT AN END-OF-LINE CHARACTER (0F1H)
    IN THE DISPLAY RAM LOCATIONS THAT CORRESPOND TO THE BEGINNING OF
    THE CRT DISPLAY LINES UNTIL THE BOTTOM OF THE CRT SCREEN HAS BEEN REACHED}
END;

END ERASE$FROM$CURSER$TO$END$OF$SCREEN;



/*PROCEDURE MOV$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR
  IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL
  READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE
  IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */


MOV$CURSER:

{WAIT UNTIL THE FIFO HAS RECEIVED THE NEXT TWO CHARACTERS}
{MOVE THE CURSER TO THE LOCATION SPECIFIED IN THE ESCAPE SEQUENCE}
{MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION}

IF THE FIRST CHARACTER IN THE NEW LINE HAS AN END-OF-LINE CHARACTER} THEN
   CALL FILL;
END;

{DISABLE THE SERIAL PORT INTERRUPT}
{MOVE THE REMAIN CONTENTS OF THE FIFO UP TWO LOCATIONS IN MEMORY}
{DECREMENT THE FIFO BY TWO}
{ENABLE THE SERIAL PORT INTERRUPT}

END MOV$CURSER;


/*        PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
  BY SUBTRACTING 1 OF THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */


LEFT$CURSER:

IF {THE CURSER IS NOT IN THE FIRST LOCATION OF A LINE} THEN
DO;
   {MOVE THE CURSER LEFT BY ONE LOCATION}
   {TURN THE CURSER ON}
   CALL LOAD$CURSER;
   {DECREMENT THE DISPLAY$RAM$POINTER BY ONE}
END;

END LEFT$CURSER;
```

```
/*        PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
  BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

RIGHT$CURSER:

IF {THE CURSER IS NOT IN THE LAST POSITION OF THE CRT LINE} THEN
DO;
   {MOVE THE CURSER RIGHT BY ONE LOCATION}
   {TURN THE CURSER ON}
   CALL LOAD$CURSER;
   {INCREMENT THE DISPLAY$RAM$POINTER BY ONE}
END;

END RIGHT$CURSER;



/*        PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
  BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

UP$CURSER:

IF {THE CURSER IS NOT ON THE FIRST LINE OF THE CRT DISPLAY} THEN
DO;
   {MOVE THE CURSER UP ONE LINE}
   {TURN ON THE CURSER}
   CALL LOAD$CURSER;

   IF {THE DISPLAY$RAM$POINTER IS IN THE FIRST LINE OF DISPLAY MEMORY} THEN
      {MOVE THE DISPLAY$RAM$POINTER TO THE LAST LINE OF DISPLAY MEMORY}
   ELSE
      {MOVE THE DISPLAY$RAM$POINTER UP ONE LINE IN DISPLAY MEMORY}

   IF {THE FIRST LOCATION OF THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER} THEN
      CALL FILL;
END;
END  UP$CURSER;



/*        PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
  BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */


DOWN$CURSER:

IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN
DO;
   {TURN THE CURSER ON}
   {MOVE THE CURSER TO THE NEXT LINE}
   CALL LOAD$CURSER;

   IF {THE DISPLAY$RAM$POINTER IS NOT ON THE LAST LINE OF THE DISPLAY MEMORY} THEN
      {MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY MEMORY}
   ELSE
      {MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY MEMORY}

   IF {THE FIRST CHARACTER IN THE NEW LINE IS AN END-OF-LINE CHARACTER} THEN
      CALL FILL;
END;

END DOWN$CURSER;
```

```
/*      PROCEDURE CARRIAGE$RETURN       */

CARRIAGE$RETURN:

{MOVE THE DISPLAY$RAM$POINTER TO THE BEGINNING OF THE CURRENT LINE IN THE DISPLAY MEMORY}
{MOVE THE CURSER TO THE BEGINNING OF THE CURRENT LINE OF THE CRT DISPLAY}
{TURN THE CURSER ON}
CALL LOAD$CURSER;

END CARRIAGE$RETURN;


/*      PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
  LOADS IT INTO THE 8276 CURSER REGISTER. */


LOAD$CURSER:
PROCEDURE;
IF {THE CURSER IS ON} THEN
   {MOVE THE CURSER BACK ONTO THE CRT DISPLAY}
{DISABLE BUFFER INTERRUPT}
{WRITE TO THE 8276 CURSER REGISTERS THE X,Y LOCATIONS}
{ENABLE BUFFER INTERRUPT}

END LOAD$CURSER;



/*      PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP
        THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */

CHECK$BAUD$RATE:

{SET TIMER 1 TO MODE 1 AND AUTO RELOAD}
{TURN TIMER ON}
{ENABLE SERIAL PORT INTERRUPT}
{READ BAUD RATE SWITCHES AND SET UP RELOAD VALUE}

   ;                        /* 00 IS NOT ALLOWED */
   TH1=040H;                /*  150 BAUD */
   TH1=0A0H;                /*  300 BAUD */
   TH1=0D0H;                /*  600 BAUD */
   TH1=0E8H;                /* 1200 BAUD */
   TH1=0F4H;                /* 2400 BAUD */
   TH1=0FAH;                /* 4800 BAUD */
   TH1=0FDH;                /* 9600 BAUD */

END CHECK$BAUD$RATE;
```

```
/*        PROCEDURE READER: THIS PROCEDURE IS WRITTEN IS ASSEMBLY LANGUAGE. THE
  EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN
  LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL
  MEMORY ARRAY CALLED CURRENT$KEY */


READER:

{INITIALIZE FLAGS "KEY0"=0, "SAME"=1, 0 COUNTER=0}

DO UNTIL {ALL 8 KEYBOARD SCAN LINES ARE READ}
   {READ KEYBOARD SCAN}
   IF {NO KEY WAS PRESSED} THEN
      {INCREMENT 0 COUNTER}
   ELSE
   IF {THE KEY PRESSED WAS NOT THE SAME KEY THAT WAS PRESSED THE LAST TIME
       THE KEYBOARD WAS READ} THEN
      {CLEAR "SAME" AND WRITE NEW SCAN RESULT TO CURRENT$KEY RAM ARRAY}
END;

IF {ALL 8 SCANS DIDN'T HAVE A KEY PRESSED (0 COUNTER=8)} THEN
   {SET KEY0, AND CLEAR SAME}

END READER;



/*        PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 WITH SPACES (20H ASCII)
  DURING THE SCROLL ROUTINES.*/


BLANK:

DO I= {BEGINNING OF THE CRT DISPLAY (LINE0)} TO {LINE0 + 50H}
   {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
   NEXT I
END;

END BLANK;



/*        PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSER TO THE END OF
                   THE DISPLAY LINE */


BLINE:

DO I= {CURRENT CURSER POSITION ON CRT DISPLAY} TO {END OF ROW}
   {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
   NEXT I
END;

END BLINE;



/*        PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS A DISPLAY LINE WITH SPACES*/


FILL:

DO I= {BEGINNING OF THE LINE THAT THE CURSER IS ON} TO {END OF THE ROW}
   {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
   NEXT I
END;

END FILL;
```

Appendix 7.9 Software Listings

```
PL/M-51 COMPILER

ISIS-II PL/M-51 V1.1
COMPILER INVOKED BY:  PLM51 :F1:CRTPLM.SRC


            $OPTIMIZE(1)
            $NOINTVECTOR
            $ROM(LARGE)

            /*********************************************************************************
            *********************************************************************************
            *******                                                                ********
            *******          PLM51 SOFTWARE FOR THE 8051 TERMINAL                  ********
            *******          CONTROLLER APPLICATION NOTE                           ********
            *******                                                                ********
            *********************************************************************************
            *********************************************************************************

                    MEMORY MAP ASSOCIATED WITH PERIPHERAL DEVICES (USING MOVX):

                    8051 WR AND READ DISPLAY RAM-     ADDRESS 1000H TO 17CFH
                    8051 WR DISPLAY RAM TO THE 8276-  ADDRESS 1800H TO 1FCFH
                    8276 COMMAND ADDRESS-             ADDRESS 0001H
                    8276 PARAMETER ADDRESS-           ADDRESS 0000H
                    8276 STATUS REGISTER-             ADDRESS 0001H

                    MEMORY MAP FOR READING THE KEYBOARD (USING MOVC):

                    KEYBOARD ADDRESS-                 ADDRESS 10FFH TO 17FFH

                THE FOLLOWING SOFTWARE SWITCHES MUST BE SET ACCORDING TO THE TYPE OF
            KEYBOARD THAT IS GOING TO BE USED.

                SW1- SET WHEN USING AN UNDECODED KEYBOARD IS TO BE USED
                SW2- SET WHEN USING A DECODED OR A SERIAL TYPE OF KEYBOARD


                PROGRAMS TO LINK TOGETHER FOR WORKING SYSTEMS:

                UNDECODED KEYBOARD- CRTPLM.OBJ,CRTASM.OBJ,KEYBD.OBJ,PLM51.LIB
                DECODED KEYBOARD-CRTPLM.OBJ,CRTASM.OBJ,DECODE.OBJ,PLM51.LIB
                DETACHED KEYBOARD-CRTPLM.OBJ,CRTASM.OBJ,DETACH.OBJ,PLM51.LIB
            */
            $SET (SW1)
            $RESET (SW2)
```

```
PL/M-51 COMPILER

             $EJECT

             CRT$CONTROLLER:
  1   1      DO;

                /***************** DECLARE LITERALS ************************/

  2   1      DECLARE LLC LITERALLY 'LOCAL$LINE$CHANGE';
  3   1      DECLARE REG  LITERALLY 'REGISTER';
  4   1      DECLARE CURRENT$KEY LITERALLY 'CURKEY';
  5   1      DECLARE SERIAL$SERVICE LITERALLY 'SERBUF';
  6   1      DECLARE DISPLAY$RAM$POINTER LITERALLY 'POINT';
  7   1      DECLARE SERIAL$INT LITERALLY 'SERINT';
  8   1      DECLARE TRANSMIT$INT LITERALLY 'TRNINT';
  9   1      DECLARE CURSER$COLUMN LITERALLY 'CURSER';
 10   1      DECLARE LAST$KEY LITERALLY 'LSTKEY';
 11   1      DECLARE CURSER$COUNT LITERALLY 'COUNT';
 12   1      DECLARE SCAN$INT LITERALLY 'SCAN';


                /********* REGISTER DECLARATIONS FOR THE 8051 *****************/


             /********* BYTE REGISTERS ********/

 13   1      DECLARE
                P0   BYTE  AT(80H)   REG,
                P1   BYTE  AT(90H)   REG,
                P2   BYTE  AT(0A0H)  REG,
                P3   BYTE  AT(0B0H)  REG,
                PSW  BYTE  AT(0D0H)  REG,
                ACC  BYTE  AT(0E0H)  REG,
                B    BYTE  AT(0F0H)  REG,
                SP   BYTE  AT(81H)   REG,
                DPL  BYTE  AT(82H)   REG,
                DPH  BYTE  AT(83H)   REG,
                PCON BYTE  AT(87H)   REG,
                TCON BYTE  AT(88H)   REG,
                TMOD BYTE  AT(89H)   REG,
                TL0  BYTE  AT(8AH)   REG,
                TL1  BYTE  AT(8BH)   REG,
                TH0  BYTE  AT(8CH)   REG,
                TH1  BYTE  AT(8DH)   REG,
                IE   BYTE  AT(0A8H)  REG,
                IP   BYTE  AT(0B8H)  REG,
                SCON BYTE  AT(98H)   REG,
                SBUF BYTE  AT(99H)   REG;
```

```
          $EJECT
          /********* BIT REGISTERS ********/

           /********* PSW  BITS ********/
 14   1   DECLARE
             CY   BIT  AT(0D7H) REG,
             AC   BIT  AT(0D6H) REG,
             F0   BIT  AT(0D5H) REG,
             RS1  BIT  AT(0D4H) REG,
             RS0  BIT  AT(0D3H) REG,
             OV   BIT  AT(0D2H) REG,
             P    BIT  AT(0D0H) REG,

           /********* TCON BITS ********/
             TF1  BIT  AT(8FH)  REG,
             TR1  BIT  AT(8EH)  REG,
             TF0  BIT  AT(8DH)  REG,
             TR0  BIT  AT(8CH)  REG,
             IE1  BIT  AT(8BH)  REG,
             IT1  BIT  AT(8AH)  REG,
             IE0  BIT  AT(89H)  REG,
             IT0  BIT  AT(88H)  REG,

           /*********  IE  BITS ********/
             EA   BIT  AT(0AFH) REG,
             ES   BIT  AT(0ACH) REG,
             ET1  BIT  AT(0ABH) REG,
             EX1  BIT  AT(0AAH) REG,
             ET0  BIT  AT(0A9H) REG,
             EX0  BIT  AT(0A8H) REG,

           /*********  IP  BITS ********/
             PS   BIT  AT(0BCH) REG,
             PT1  BIT  AT(0BBH) REG,
             PX1  BIT  AT(0BAH) REG,
             PT0  BIT  AT(0B9H) REG,
             PX0  BIT  AT(0B8H) REG,

           /*********  P3  BITS ********/
             RD   BIT  AT(0B7H) REG,
             WR   BIT  AT(0B6H) REG,
             T1   BIT  AT(0B5H) REG,
             T0   BIT  AT(0B4H) REG,
             INT1 BIT  AT(0B3H) REG,
             INT0 BIT  AT(0B2H) REG,
             TXD  BIT  AT(0B1H) REG,
             RXD  BIT  AT(0B0H) REG,

           /********* SCON BITS ********/
             SM0  BIT  AT(9FH)  REG,
             SM1  BIT  AT(9EH)  REG,
             SM2  BIT  AT(9DH)  REG,
             REN  BIT  AT(9CH)  REG,
             TB8  BIT  AT(9BH)  REG,
             RB8  BIT  AT(9AH)  REG,
             TI   BIT  AT(99H)  REG,
             RI   BIT  AT(98H)  REG;
```

```
PL/M-51 COMPILER    CRTCONTROLLER

            $EJECT
            $IF SW1
                /*************** DECLARE CONSTANTS**********************/

 15   1     DECLARE LOW$SCAN(16) STRUCTURE
                (KEY(8) BYTE) CONSTANT

                ('890-',5CH,5EH,08H,00H,
            /* SCAN 0, SHIFT KEY =0; 8,9,0,-,\,^, BACK SPACE */

                'uiop',5BH,'@',0AH,7FH,
            /* SCAN 1, SHIFT =0; u,i,o,p,[,@, LINE FEED, DELETE */

                'jkl;:',00H,0DH,'7',
            /* SCAN 2, SHIFT =0; j,k,l,;,:, RETURN, 7 */

                'm',2CH,'.',00H,'/',00H,00H,00H,
            /* SCAN 3, SHIFT =0; m,COMMA,.,/ */

                00H,'azxcvbn',
            /* SCAN 4, SHIFT =0; a,z,x,c,v,b,n */

                'y',00H,00H,' dfgh',
            /* SCAN 5, SHIFT =0; y, SPACE, d,f,g,h */

                09H,'qwsert',00H,
            /* SCAN 6, SHIFT =0; TAB,q,w,s,e,r,t */

                1BH,'123456',00H,
            /* SCAN 7, SHIFT =0;ESC,1,2,3,4,5,6 */

                28H,29H,00H,'=',7CH,7EH,08H,00H,
            /* SCAN 0, SHIFT =1; (,),=,|,~, BACK SPACE */

                'UIOP',00H,00H,0AH,7FH,
            /* SCAN 1, SHIFT =1; U,I,O,P, LINE FEED, DELETE */

                'JKL+*',00H,0DH,27H,
            /* SCAN 2, SHIFT =1; J,K,L,+,*, RETURN, ' */

                'M<>',00H,3FH,00H,00H,00H,
            /* SCAN 3, SHIFT =1; M,<,>,? */

                00H,'AZXCVBN',
            /* SCAN 4, SHIFT =1; A,Z,X,,C,V,B,N */

                'Y',00H,00H,' DFGH',
            /* SCAN 5, SHIFT =1; Y, SPACE, D,F,G,H */

                09H,'QWSERT',00H,
            /* SCAN 6, SHIFT =1; TAB, Q,W,S,E,R,T */

                1BH,'!"#$%&',00H);
            /* SCAN 7, SHIFT =1;ESC,!,",#,$,%,& */
            $ENDIF
```

```
             $EJECT
                /******************DECLARE VARIABLES********************/


  16   1     DECLARE

             $IF SW2

INPUT              BIT AT(0B4H)    REG,
             $ENDIF
             $IF SW1
                 CAP$LOCK              BIT AT(095H)    REG,
                 SHIFT$KEY             BIT AT(096H)    REG,
                 CONTROL$KEY           BIT AT(097H)    REG,
             $ENDIF
                 LOCAL$LINE            BIT AT(0B5H)    REG,
                 CLEAR$TO$SEND         BIT AT(093H)    REG,
                 DATA$TERMINAL$READY   BIT AT(094H)    REG;

  17   1     DECLARE (

             $IF SW1
                 SAME,
                 VALID$KEY,
                 KEY0,
                 LAST$SHIFT$KEY,
                 LAST$CONTROL$KEY,
                 LAST$CAP$LOCK,
             $ENDIF

             $IF SW2
                 RCVFLG,
                 SYNC,
                 BYFIN,
                 KBDINT,
                 ERROR,
             $ENDIF

                 NEW$KEY,
                 TRANSMIT$TOGGLE,
                 CURSER$ON,
                 SERIAL$INT,
                 SCAN$INT,
                 TRANSMIT$INT,
                 ESCSEQ,
                 VALID$RECEPTION,
                 LLC,
                 ENSP)        BIT PUBLIC;
```

```
PL/M-51 COMPILER    CRTCONTROLLER

              $EJECT

 18   1       DECLARE (

                 I,
                 J,
                 K,
                 ASCII$KEY,
                 TRANSMIT$COUNT,
                 TEMP,
                 SHIFT,
                 CURSER$COL,
                 CURSER$COLUMN,
                 CURSER$ROW,
                 CURSER$COUNT,
                 FIFO,
                 RECEIVE)        BYTE PUBLIC;

              $IF SW1
 19   1       DECLARE LAST$KEY(8) BYTE PUBLIC;
              $ENDIF

              $IF SW2
              DECLARE LAST$KEY(2) BYTE PUBLIC;
              $ENDIF

 20   1       DECLARE SERIAL(16)  BYTE PUBLIC;

 21   1       DECLARE DISPLAY$RAM(7CFH) BYTE  AT(1000H)   AUXILIARY;

 22   1       DECLARE

                 PARAMETER$ADDRESS   BYTE    AT(0000H)   AUXILIARY,
                 COMMAND$ADDRESS     BYTE    AT(0001H)   AUXILIARY;

 23   1       DECLARE (

                 DISPLAY$RAM$POINTER,
                 RASTER,
                 LINE0,
                 L)          WORD PUBLIC;
```

```
             $EJECT

             /*  PROCEDURE READER: THIS PROCEDURE IS WRITTEN IS ASSEMBLY LANGUAGE. THE
                 EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN
                 LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL
                 MEMORY ARRAY CALLED CURRENT$KEY. THE PROCEEDURE CONTROLS 2 STATUS FLAGS;
                     KEY0 AND SAME. KEY0 IS SET IF ALL 8 SCANS READ NO KEY WAS PRESSED.
                     IF ALL 8 SCANS ARE THE SAME AS THE LAST READING OF THE KEYBOARD, THEN
                     SAME IS SET. */

  24   2     READER: PROCEDURE EXTERNAL;
  25   1     END READER;


             /*  PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 SCAN WITH SPACES (20H ASCII)
                 DURING THE SCROLL ROUTINES.*/

  26   2     BLANK: PROCEDURE EXTERNAL;
  27   1     END BLANK;


             /*  PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSER TO THE END OF
                         THE DISPLAY LINE */

  28   2     BLINE: PROCEDURE EXTERNAL;
  29   1     END BLINE;


             /*  PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS THE CURSER LINE
                     WITH SPACES*/

  30   1     FILL:
             PROCEDURE EXTERNAL;
  31   1     END FILL;
```

```
            $EJECT

            /*  PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP
                THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */

 32   1     CHECK$BAUD$RATE:
            PROCEDURE;
 33   2     SCON=70H;                    /* MODE 1
                                            ENABLE RECEPTION*/
 34   2     TMOD=TMOD OR 20H;            /* TIMER 1 AUTO RELOAD */
 35   2     TR1=1;                       /* TIMER 1 ON */
 36   2     ES=1;                        /* ENABLE SERIAL INTERRUPT*/
 37   2     ENSP=1;                      /* SERIAL INTERRUPT MASK FLAG */
 38   3     DO CASE (P1 AND 07H);
 39   3        ;                         /* 00 IS NOT ALLOWED */
 40   3        TH1=040H;                     /*  150 BAUD */
 41   3        TH1=0A0H;                     /*  300 BAUD */
 42   3        TH1=0D0H;                     /*  600 BAUD */
 43   3        TH1=0E8H;                     /* 1200 BAUD */
 44   3        TH1=0F4H;                     /* 2400 BAUD */
 45   3        TH1=0FAH;                     /* 4800 BAUD */
 46   3        TH1=0FDH;                     /* 9600 BAUD */
 47   3     END;
 48   1     END CHECK$BAUD$RATE;



            /*  PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
                LOADS IT INTO THE 8276 CURSER REGISTERS. */

 49   1     LOAD$CURSER:
            PROCEDURE;
 50   2     IF CURSER$ON=1 THEN
 51   2        CURSER$COL=CURSER$COLUMN;
 52   2     EX1=0;                           /* DISABLE BUFFER INTERRUPT */
 53   2     COMMAND$ADDRESS=80H;             /* INITIALIZE CURSER COMMAND */
 54   2     PARAMETER$ADDRESS=CURSER$COL;
 55   2     PARAMETER$ADDRESS=CURSER$ROW;
 56   2     EX1=1;                           /* ENABLE BUFFER INTERRUPT */
 57   1     END LOAD$CURSER;


            /*  PROCEDURE CARRIAGE$RETURN   */

 58   1     CARRIAGE$RETURN:
            PROCEDURE;
 59   2     DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-CURSER$COLUMN;
 60   2     CURSER$COLUMN=0;
 61   2     CURSER$ON=1;
 62   2     CALL LOAD$CURSER;
 63   1     END CARRIAGE$RETURN;
```

```
          $EJECT

          /*  PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
              BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */


 64   1   DOWN$CURSER:
          PROCEDURE;
 65   2   IF CURSER$ROW < 18H THEN
 66   3   DO;
 67   3      CURSER$ON=1;
 68   3      CURSER$ROW=CURSER$ROW + 1;
 69   3      CALL LOAD$CURSER;
 70   3      IF DISPLAY$RAM$POINTER < 780H THEN
 71   3         DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER +50H;
 72   3      ELSE
                DISPLAY$RAM$POINTER=(DISPLAY$RAM$POINTER-780H);
 73   3      L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
 74   3      IF DISPLAY$RAM(L)=0F1H THEN                        /* LOOK FOR END OF*/
 75   4      DO;                                                /*   LINE CHARACTER */
 76   4         CALL FILL;                                      /*IF TRUE FILL LINE*/
 77   4         DISPLAY$RAM(L)=20H;                             /*WITH SPACES */
 78   4      END;
 79   3   END;
 80   1   END DOWN$CURSER;



          /*  PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
          BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

 81   1   UP$CURSER:
          PROCEDURE;
 82   2   IF CURSER$ROW >0 THEN
 83   3   DO;
 84   3      CURSER$ROW=CURSER$ROW - 1;
 85   3      CURSER$ON=1;
 86   3      CALL LOAD$CURSER;
 87   3      IF DISPLAY$RAM$POINTER<50H THEN
 88   3         DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+780H;
 89   3      ELSE
                DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER - 50H;
 90   3      L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
 91   3      IF DISPLAY$RAM(L)=0F1H THEN                     /* LOOK FOR END OF LINE*/
 92   4      DO;                                             /*     CHARACTER */
 93   4         CALL FILL;                                   /* IF TRUE FILL WITH */
 94   4         DISPLAY$RAM(L)=20H;                          /* SPACES */
 95   4      END;
 96   3   END;
 97   1   END UP$CURSER;
```

PL/M-51 COMPILER   CRTCONTROLLER

```
              $EJECT

              /*  PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
                  BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

 98    1      RIGHT$CURSER:
              PROCEDURE;
 99    2      IF CURSER$COLUMN < 4FH THEN
100    3      DO;
101    3         CURSER$COLUMN=CURSER$COLUMN + 1;
102    3         CURSER$ON=1;
103    3         CALL LOAD$CURSER;
104    3         DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER +1;
105    3      END;
106    1      END RIGHT$CURSER;



              /*  PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
                  BY SUBTRACTING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

107    1      LEFT$CURSER:
              PROCEDURE;
108    2      IF CURSER$COLUMN >0 THEN
109    3      DO;
110    3         CURSER$COLUMN=CURSER$COLUMN - 1;
111    3         CURSER$ON=1;
112    3         CALL LOAD$CURSER;
113    3         DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER -1;
114    3      END;
115    1      END LEFT$CURSER;
```

PL/M-51 COMPILER   CRTCONTROLLER

```
                $EJECT

                /*  PROCEDURE MOV$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR
                    IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL
                    READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE
                    IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */


 116   1        MOV$CURSER:
                PROCEDURE;
 117   3        DO WHILE FIFO<4;        /* WAIT UNTILL THE MOV$CURSER PARAMETERS*/
 118   3        END;               /* ARE RECEIVED INTO THE FIFO */
 119   2        TEMP=CURSER$ROW;
 120   2        CURSER$ROW=SERIAL(2);
 121   2        IF CURSER$ROW>TEMP THEN
 122   3        DO;
 123   3           L=DISPLAY$RAM$POINTER+((CURSER$ROW-TEMP)*50H);
 124   3           IF L>7CFH THEN                         /* IF OUT OF RAM RANGE */
 125   3              DISPLAY$RAM$POINTER=L-7D0H;         /* RAP AROUND TO BEGINNING */
 126   3           ELSE                                   /* OF RAM */
                      DISPLAY$RAM$POINTER=L;
 127   3        END;
 128   2        ELSE
                DO;
 129   3           IF CURSER$ROW<TEMP THEN
 130   4           DO;
 131   4              L=(TEMP-CURSER$ROW)*50H;
 132   4              IF DISPLAY$RAM$POINTER<L THEN                   /* IF OUT OF RAM RANGE*/
 133   4                 DISPLAY$RAM$POINTER=(7D0H-(L-DISPLAY$RAM$POINTER));/* RAP AROUND TO END OF RAM*/
 134   4              ELSE
                         DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-L;
 135   4           END;
 136   3        END;
 137   2        TEMP=CURSER$COLUMN;
 138   2        CURSER$COLUMN=SERIAL(3);
 139   2        IF CURSER$COLUMN>TEMP THEN
 140   2           DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+(CURSER$COLUMN-TEMP);
 141   2        ELSE
                   DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-(TEMP-CURSER$COLUMN);
 142   2        CURSER$ON=1;
 143   2        CALL LOAD$CURSER;
 144   2        L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
 145   2        IF DISPLAY$RAM(L)=0F1H THEN                 /* LOOK FOR END FO LINE CHARACTER*/
 146   3        DO;
 147   3           CALL FILL;                               /* IF TRUE FILL WITH SPACES */
 148   3           DISPLAY$RAM(L)=20H;
 149   3        END;
 150   2        ES=0;
 151   3        DO I=2 TO FIFO-2;
 152   3           SERIAL(I)=SERIAL(I+2);
 153   3        END;
 154   2        FIFO=FIFO-2;
 155   2        ES=ENSP;
 156   1        END MOV$CURSER;
```

PL/M-51 COMPILER   CRTCONTROLLER

```
            $EJECT

            /*    PROCEDURE ERASE FROM CURSER TO END OF SCREEN:  */


 157   1    ERASE$FROM$CURSER$TO$END$OF$SCREEN:
            PROCEDURE;
 158   2    CALL BLINE;                                /* ERASE CURRENT LINE */
 159   2    IF CURSER$ROW < 18H THEN
 160   3    DO;
 161   3      L=DISPLAY$RAM$POINTER-CURSER$COLUMN+50H;      /* GET NEXT LINE */
 162   4      DO WHILE (L < 7D0H) AND (L <> (LINE0 AND 7FFH));
 163   4         DISPLAY$RAM(L)=0F1H;           /* ERASE UNTIL LINE0 OR */
 164   4         L=L+50H;                   /* END OF DISPLAY RAM*/
 165   4      END;
 166   3      L=0;
 167   4      DO WHILE L <>(LINE0 AND 7FFH);        /* ERASE UNTIL LINE0 */
 168   4         DISPLAY$RAM(L)=0F1H;
 169   4         L=L+50H;
 170   4      END;
 171   3    END;
 172   1    END ERASE$FROM$CURSER$TO$END$OF$SCREEN;



            /*  PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION */

 173   1    HOME:
            PROCEDURE;
 174   2    CURSER$ROW=00;
 175   2    CURSER$COLUMN=00;
 176   2    CURSER$ON=1;
 177   2    CALL LOAD$CURSER;
 178   2    DISPLAY$RAM$POINTER=(LINE0 AND 7FFH);
 179   1    END HOME;
```

```
            $EJECT

            /*  PROCEDURE CLEAR SCREEN     */

 180   1    CLEAR$SCREEN:
            PROCEDURE;
 181   2    CALL HOME;
 182   2    CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
 183   1    END CLEAR$SCREEN;



            /*      PROCEDURE SCROLL    */

 184   1    SCROLL:
            PROCEDURE;
 185   2    CALL BLANK;
 186   2    EX0=0;                 /* DISABLE VERTICAL REFRESH INTERRUPT */
 187   2    IF LINE0= 1F80H THEN
 188   2       LINE0= 1800H;
 189   2    ELSE
               LINE0= LINE0+50H;
 190   2    EX0=1;                 /* ENABLE VERTICAL REFRESH INTERRUPT */
 191   1    END SCROLL;



            /*  PROCEDURE LINE$FEED */

 192   1    LINE$FEED:
            PROCEDURE;
 193   2    IF CURSER$ROW=18H THEN
 194   2       CALL SCROLL;
 195   2    ELSE
            DO;
 196   3       CURSER$ROW= CURSER$ROW+1;
 197   3       CURSER$ON=1;
 198   3       CALL LOAD$CURSER;
 199   3    END;
 200   2    IF DISPLAY$RAM$POINTER >77FH THEN
 201   2       DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-780H;
 202   2    ELSE
               DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+50H;
 203   2    L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
 204   2    IF DISPLAY$RAM(L)=0F1H THEN               /* LOOK FOR END OF LINE CHARACTER*/
 205   3    DO;
 206   3       CALL FILL;                             /* IF TRUE FILL WITH SPACES */
 207   3       DISPLAY$RAM(L)=20H;
 208   3    END;
 209   1    END LINE$FEED;
```

```
PL/M-51 COMPILER    CRTCONTROLLER

                $EJECT

                /*  PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
                      RECEIVE AND PUT IT INTO THE DISPLAY RAM. */

 210   1        DISPLAY:
                PROCEDURE;
 211   2        DISPLAY$RAM(DISPLAY$RAM$POINTER)=RECEIVE;
 212   2        IF DISPLAY$RAM$POINTER=7CFH THEN            /* IF END OF RAM */
 213   2          DISPLAY$RAM$POINTER=000H;                 /* RAP AROUND TO BEGINNING */
 214   2        ELSE
                  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+1;
 215   2        IF CURSER$COLUMN=4FH THEN
 216   3        DO;
 217   3          CURSER$COLUMN=00H;
 218   3          L=DISPLAY$RAM$POINTER;
 219   3          IF DISPLAY$RAM(L)=0F1H THEN
 220   4          DO;
 221   4            CALL FILL;
 222   4            DISPLAY$RAM(L)=20H;
 223   4          END;
 224   3          IF CURSER$ROW=18H THEN
 225   3            CALL SCROLL;
 226   3          ELSE
                    CURSER$ROW=CURSER$ROW+1;
 227   3        END;
 228   2        ELSE
                  CURSER$COLUMN=CURSER$COLUMN+1;
 229   2        CURSER$ON=1;
 230   2        CALL LOADCURSER;
 231   1        END DISPLAY;
```

```
          $EJECT

          /*  PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES
              WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE
              THE PROCEDURE THEN ACTS ACCORDINGLY */

232   1   DECIPHER:
          PROCEDURE;
233   2   START$DECIPHER:
          VALID$RECEPTION=0;
234   2   I=0;
235   3   DO WHILE (I<FIFO) AND (SERIAL(I)>1FH) AND (SERIAL(I)<7FH);
236   3      RECEIVE=SERIAL(I);
237   3      CALL DISPLAY;
238   3      I=I+1;
239   3   END;
240   2   IF I>0 THEN
241   3   DO;
242   3      ES=0;                       /* DISABLE SERIAL INTERRUPT WHILE MOVING FIFO */
243   3      K=FIFO-I;
244   4      DO J=0 TO K;                /* MOVE FIFO */
245   4         SERIAL(J)=SERIAL(I);
246   4         I=I+1;
247   4      END;
248   3      FIFO=K;
249   3      ES=ENSP;                    /* ENABLE SERIAL INTERRUPT */
250   3      VALID$RECEPTION=1;
251   3   END;
252   2   IF FIFO=0 THEN
253   3   DO;
254   3      SERIAL$INT=0;
255   3      GOTO END$DECIPHER;
256   3   END;
257   2   IF (SERIAL(0)=1BH) THEN
258   3   DO;
259   3      IF (ESC$SEQ=1) AND (FIFO<2) THEN
260   3         GOTO END$DECIPHER;
261   3      K=(SERIAL(1) AND 5FH)-40H;
262   3      IF (K >00H) AND (K<0CH) THEN
263   4      DO;
264   5         DO CASE K;
265   5            ;
266   5            CALL UP$CURSER;                          /* ESC A */
267   5            CALL DOWN$CURSER;                        /* ESC B */
268   5            CALL RIGHT$CURSER;                       /* ESC C */
269   5            CALL LEFT$CURSER;                        /* ESC D */
270   5            CALL CLEAR$SCREEN;                       /* ESC E */
271   5            CALL MOV$CURSER;                         /* ESC F */
272   5            ;
273   5            CALL HOME;                               /* ESC H */
274   5            ;
275   5            CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN; /* ESC J */
276   5            CALL BLINE;                              /* ESC K */
277   5         END;
278   4      END;
279   3      ES=0;                       /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */
```

PL/M-51 COMPILER   CRTCONTROLLER

```
4         DO I=0 TO (FIFO-2);
4            SERIAL(I)=SERIAL(I+2);     /* MOVE FIFO */
4         END;
3         FIFO=FIFO-2;
3         ES=ENSP;                      /* ENABLE SERIAL INTERRUPTS */
3         VALID$RECEPTION=1;
3         IF FIFO=0 THEN
4         DO;
4            SERIAL$INT=0;
4            GOTO END$DECIPHER;
4         END;
3      END;
2      IF (SERIAL(0)> 07H) AND (SERIAL(0)<0EH) THEN
3      DO;
4         DO CASE (SERIAL(0) -08H);
4            CALL LEFT$CURSER;                  /* CTL H */
4            ;
4            CALL LINE$FEED;                    /* CTL J */
4            ;
4            CALL CLEAR$SCREEN;                 /* CTL L */
4            CALL CARRIAGE$RETURN;              /* CTL M */
4         END;
3         ES=0;                                 /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */
4         DO I=0 TO (FIFO-1);
4            SERIAL(I)=SERIAL(I+1);             /* MOVE FIFO */
4         END;
3         FIFO=FIFO-1;
3         ES=ENSP;                              /* ENABLE SERIAL INTERRUPTS */
3         VALID$RECEPTION=1;
3      END;
2      IF VALID$RECEPTION=0 THEN
3      DO;
3         ES=0;
4         DO I=0 TO (FIFO-1);              /* IF CHARACTER IS UNRECOGNIZED THEN */
4            SERIAL(I)=SERIAL(I+1);        /* TRASH IT */
4         END;
3         FIFO=FIFO-1;
3         ES=ENSP;
3      END;
2      IF FIFO=0 THEN
2         SERIAL$INT=0;
2      END$DECIPHER:
       END DECIPHER;
```

```
             $EJECT

             /* PROCEDURE TRANSMIT- THIS PROCEDURE LOOKS AT THE CLEAR TO SEND PIN FOR AN ACTIVE
                LOW SIGNAL. ONCE THE MAIN COMPUTER SIGNALS THE 8051 THE ASCII CHARACTER IS PUT
                INTO THE SERIAL PORT.*/


 322   1     TRANSMIT:
             PROCEDURE;
 323   2     IF LOCAL$LINE =1 THEN
 324   3     DO;
 325   4        DO WHILE (CLEAR$TO$SEND=1) OR (TRANSMIT$INT=0);
 326   4        END;
 327   3        SBUF=ASCII$KEY;
 328   3        TRANSMIT$INT=0;
 329   3     END;
 330   2     ELSE
             DO;
 331   3        SERIAL(FIFO)=ASCII$KEY;
 332   3        FIFO=FIFO+1;
 333   3        SERIAL$INT=1;
 334   3     END;
 335   1     END TRANSMIT;




             /*  PROCEDURE AUTO$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION
                 AFTER A FIXED DELAY PERIOD */


 336   1     AUTO$REPEAT:
             PROCEDURE;
 337   2     IF NEW$KEY=1 THEN
 338   3     DO;
 339   3        TRANSMIT$TOGGLE=0;
 340   3        TRANSMIT$COUNT=0D0H;
 341   3        CALL TRANSMIT;                    /* FIRST CHARACTER */
 342   3        NEW$KEY=0;
 343   3     END;
 344   2     ELSE
             DO;
 345   3        IF TRANSMIT$COUNT <> 00H THEN
 346   4        DO;
 347   4           TRANSMIT$COUNT=TRANSMIT$COUNT+1;
 348   4           IF TRANSMIT$COUNT=0FFH THEN   /*DELAY BETWEEN FIRST CHARACTER AND THE SECOND */
 349   5           DO;
 350   5              CALL TRANSMIT;             /*SECOND CHARACTER */
 351   5              TRANSMIT$COUNT=00;
 352   5           END;
 353   4        END;
 354   3        ELSE
                  DO;
 355   4           CURSER$ON=1;
 356   4           CURSER$COUNT=0;
 357   4           IF TRANSMIT$TOGGLE = 1 THEN       /* 2 VERT FRAMES BETWEEN 3RD TO NTH CHARACTER */
 358   4              CALL TRANSMIT;                 /* 3RD THROUGH NTH CHARACTER */
 359   4           TRANSMIT$TOGGLE= NOT TRANSMIT$TOGGLE;
 360   4        END;
 361   3     END;
 362   1     END AUTO$REPEAT;
```

```
            $EJECT

            /******************** START MAIN PROGRAM ***************************/
            /*  BEGIN BY PUTTING ASCII CODE FOR BLANK IN THE DISPLAY RAM;*/


 363   1    INIT:
            DO L=0 TO 7CFH;
 364   2       DISPLAY$RAM(L)=20H;
 365   2    END;

            /*  INITIALIZE POINTERS, RAM BITS, ETC.     */


 366   1    ESC$SEQ=0;
 367   1    SCAN$INT=0;
 368   1    SERIAL$INT=0;
 369   1    FIFO=0;
 370   1    CURSER$COUNT=0;
 371   1    LLC=0;
 372   1    DATA$TERMINAL$READY=1;
 373   1    TCON=05H;
 374   1    LINE0=1800H;
 375   1    RASTER=1800H;
 376   1    DISPLAY$RAM$POINTER=0000H;
 377   1    TRANSMIT$INT=1;

            $IF SW1

 378   2    DO I=0 TO 7;
 379   2       LAST$KEY(I)=00H;
 380   2    END;

 381   1    VALID$KEY=0;
 382   1    LAST$SHIFT$KEY=1;
 383   1    LAST$CONTROL$KEY=1;
 384   1    LAST$CAP$LOCK=1;
            $ENDIF

            $IF SW2
            RCVFLG=0;
            SYNC=0;
            BYFIN=0;
            KBDINT=0;
            ERROR=0;
            $ENDIF

                    /* INITIALIZE THE 8276 */

 385   1    COMMAND$ADDRESS=00H;             /* RESET THE 8276 */
 386   1    PARAMETER$ADDRESS=4FH;           /* NORMAL ROWS, 80 CHARACTER/ROW */
 387   1    PARAMETER$ADDRESS=58H;           /* 2 ROW COUNTS PER VERTICAL RETRACE
                                               25 ROWS PER FRAME */
 388   1    PARAMETER$ADDRESS=89H;           /* LINE 9 IS THE UNDERLINE POSITION
                                               10 LINES PER ROW  */
 389   1    PARAMETER$ADDRESS=0F9H;          /* OFFSET LINE COUNTER, NON-TRANSPARENT FIELD ATTRIBUTE
```

PL/M-51 COMPILER   CRTCONTROLLER

```
                                          NON-BLINKING UNDERLINE CURSER, 20 CHARACTER COUNTS PER
                                          HORIZONTAL RETRACE */
390   1      TEMP=COMMAND$ADDRESS;

391   1      CURSER$COLUMN=00H;
392   1      CURSER$ROW=00H;
393   1      CURSER$COL=00H;
394   1      CALL LOAD$CURSER;
395   1      TEMP=COMMAND$ADDRESS;

396   1      COMMAND$ADDRESS=0E0H;           /* PRESET 8276 COUNTERS */
397   1      TEMP=COMMAND$ADDRESS;

398   1      COMMAND$ADDRESS=23H;            /* START DISPLAY */
399   1      COMMAND$ADDRESS=0A0H;           /* ENABLE INTERRUPTS */
400   1      TEMP=COMMAND$ADDRESS;

                 /* SET UP INTERRUPTS AND PRIORITIES */

             $IF SW1
401   1      IP=10H;                         /* SERIAL PORT HAS HIGHEST PRIORITY */
402   1      IE=85H;                         /* ENABLE 8051 EXTERNAL INTERRUPTS */
             $ENDIF

             $IF SW2
             IP=10H;                         /* SERIAL PORT HAS HIGHEST PRIORITY */
             IE=87H;                         /* ENABLE TIMER0 INTERRUPT*/
             TMOD=05H;                       /* TIMER 0 =EVENT COUNTER */
             TL0=0FFH;
             TH0=0FFH;                       /* INITIALIZE COUNTER TO FFFFH*/
             TR0=1;
             $ENDIF

             /* PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A
                SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT
                   WILL BE TRANSMITTED TO THE HOST COMPUTER.*/

403   1      SCANNER:
             EA=1;
404   1      DATA$TERMINAL$READY=0;
405   1      IF CURSER$COUNT=1FH THEN        /* PROGRAMMABLE CURSER BLINK */
406   2      DO;
407   2         CURSER$ON=NOT CURSER$ON;
408   2         CURSER$COUNT=00;
409   2         IF CURSER$ON=0 THEN
410   2            CURSER$COL=7FH;
411   2         CALL LOAD$CURSER;
412   2      END;
413   1      IF LLC<>LOCAL$LINE THEN         /* IF LOCAL/LINE HAS CHANGED STATUS */
414   2      DO;
415   2         IF LOCAL$LINE=0 THEN
416   3         DO;
417   3            ENSP=0;
418   3            ES=0;
419   3         END;
420   2         ELSE
                  CALL CHECK$BAUD$RATE;
421   2         LLC=LOCAL$LINE;
422   2      END;
             $IF SW1
423   2      DO WHILE SCAN$INT=0;            /* WAIT UNITL VERTICAL RETRACE BEFORE */
424   2         IF SERIAL$INT=1 THEN         /* SCANNING THE KEYBOARD*/
425   2            CALL DECIPHER;
426   2      END;
```

PL/M-51 COMPILER   CRTCONTROLLER

```
                $EJECT

 427   1        CALL READER;
 428   1        IF VALID$KEY =1 AND SAME=1 AND (LAST$SHIFT$KEY=SHIFT$KEY) AND
                   (LAST$CAP$LOCK=CAP$LOCK) AND (LAST$CONTROL$KEY=CONTROL$KEY) THEN
 429   1           CALL AUTO$REPEAT;
 430   1        ELSE
                DO;
 431   2           IF KEY0=0 AND SAME=0 THEN
 432   3           DO;
 433   3              TEMP =0;
 434   3              K=0;
 435   4              DO WHILE LAST$KEY(K)=0;
 436   4                 K=K+1;
 437   4              END;
 438   3              TEMP=LAST$KEY(K);
 439   4              DO I=(K+1) TO 7;
 440   4                 TEMP=TEMP+LAST$KEY(I);
 441   4              END;
 442   3              IF TEMP=LAST$KEY(K) THEN
 443   4              DO;
 444   4                 J=0;
 445   5                 DO WHILE (TEMP AND 01H)=0;
 446   5                    TEMP=SHR (TEMP,1);
 447   5                    J=J+1;
 448   5                 END;
 449   4                 IF TEMP >1 THEN
 450   5                 DO;
 451   5                    VALID$KEY=0;
 452   5                    NEW$KEY=0;
 453   5                 END;
 454   4                 ELSE
                         DO;
 455   5                    IF CONTROL$KEY=0 THEN
 456   5                       ASCII$KEY=(LOW$SCAN(K).KEY(J)) AND 1FH;
 457   5                    ELSE
                            DO;
 458   6                       IF SHIFT$KEY=0 THEN
 459   6                          ASCII$KEY=LOW$SCAN(K+08H).KEY(J);
 460   6                       ELSE
                               DO;
 461   7                          ASCII$KEY=LOW$SCAN(K).KEY(J);
 462   7                          IF (CAP$LOCK=0) AND (ASCII$KEY>60H) AND (ASCII$KEY<7BH) THEN
 463   7                             ASCII$KEY=ASCII$KEY-20H;
 464   7                          IF LLC=0 THEN
 465   8                          DO;
 466   8                             IF ASCII$KEY=1BH THEN
 467   8                                ESC$SEQ=1;
 468   8                             ELSE
                                        ESC$SEQ=0;
 469   8                          END;
 470   7                       END;
 471   6                    END;
 472   5                    LAST$SHIFT$KEY=SHIFT$KEY;
 473   5                    LAST$CAP$LOCK=CAP$LOCK;
 474   5                    LAST$CONTROL$KEY=CONTROL$KEY;
 475   5                    VALID$KEY=1;
 476   5                    NEW$KEY=1;
 477   5                 END;
 478   4              END;
 479   3              ELSE
                      DO;
 480   4                 VALID$KEY=0;
 481   4                 NEW$KEY=0;
 482   4              END;
 483   3           END;
 484   2        END;

                $ENDIF
```

PL/M-51 COMPILER   CRTCONTROLLER

```
            $EJECT

            $IF SW2
            IF SERIAL$INT=1 THEN
               CALL DECIPHER;
            IF KBDINT =1 THEN
            DO;
               IF ERROR =0 THEN
               DO;
                  ASCII$KEY=LST$KEY(1);
                  NEW$KEY=1;
                  CALL AUTO$REPEAT;
                  KBDINT=0;
               END;
            ERROR=0;
            KBDINT=0;
            END;
            $ENDIF

 485   1    GOTO SCANNER;
 486   1    END;



MODULE INFORMATION:                  (STATIC+OVERLAYABLE)
    CODE SIZE                        = 08E6H       2278D
    CONSTANT SIZE                    = 0080H        128D
    DIRECT VARIABLE SIZE             =    2DH+00H    45D+   0D
    INDIRECT VARIABLE SIZE           =    00H+00H     0D+   0D
    BIT SIZE                         =    10H+00H    16D+   0D
    BIT-ADDRESSABLE SIZE             =    00H+00H     0D+   0D
    AUXILIARY VARIABLE SIZE          = 0000H          0D
    MAXIMUM STACK SIZE               = 000CH         12D
    REGISTER-BANK(S) USED:             0
    1056 LINES READ
    0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION
```

```
MCS-51 MACRO ASSEMBLER    CRTASM


ISIS-II MCS-51 MACRO ASSEMBLER V2.1
OBJECT MODULE PLACED IN :F1:CRTASM.OBJ
ASSEMBLER INVOKED BY:  ASM51 :F1:CRTASM.SRC


LOC  OBJ          LINE  SOURCE



                     ;
                     ;

                            PUBLIC BLANK
                            PUBLIC BLINE
                            PUBLIC FILL
                            EXTRN DATA (LINE0,RASTER,POINT,SERIAL,FIFO,CURSER,COUNT,L)
                            EXTRN BIT (SERINT,ESCSEQ,TRNINT,SCAN)


----                        CSEG AT(03H)
0003 8020                   SJMP     VERT              ;RESET RASTER TO LINE0 AND SCAN KEYBOARD

                     ;      EXTRN CODE (DETACH)
                     ;      CSEG AT(0BH)
                     ;      LJMP     DETACH            ;NEEDED IF DECODED KEYBOARD IS USED

----                        CSEG AT(013H)
0013 802A                   SJMP     BUFFER            ;FILL 8276 ROW BUFFER

----                        CSEG AT(023H)
0023 802D                   SJMP     SERBUF            ;STICK SERIAL INFORMATION INTO THE FIFO

----                        CSEG

0025 C0D0            VERT:  PUSH     PSW               ;PUSH REG USED BY PLM51
0027 C0E0                   PUSH     ACC
0029 C000                   PUSH     00H
002B 850000   F             MOV      RASTER,LINE0      ;REINITIALIZE RASTER TO LINE0
002E 850000   F             MOV      RASTER+1,LINE0+1
0031 7801                   MOV      R0,#01H           ;CLR 8276 INTERRUPT FLAG
0033 E2                     MOVX     A,@R0
0034 0500     F             INC      COUNT             ;INCR CURSER COUNT REGISTER
0036 D200     F             SETB     SCAN              ;FOR DEBOUNCE ROUTINE
0038 D000                   POP      00H               ;POP REGISTERS
003A D0E0                   POP      ACC
003C D0D0                   POP      PSW
003E 32                     RETI


003F C0D0          BUFFER:  PUSH     PSW               ;PUSH ALL REG USED BY PLM51 CODE
0041 C0E0                   PUSH     ACC
0043 C082                   PUSH     DPL
0045 C083                   PUSH     DPH
0047 11FA                   ACALL    DMA               ;FILL 8276 ROW BUFFER
0049 D083                   POP      DPH               ;POP REGISTERS
004B D082                   POP      DPL
004D D0E0                   POP      ACC
004F D0D0                   POP      PSW
0051 32                     RETI

                     +1  $EJECT
```

```
MCS-51 MACRO ASSEMBLER    CRTASM

LOC  OBJ            LINE  SOURCE

0052 309904               SERBUF: JNB    099H,OVER  ;IF TRANSMIT BIT NOT SET THEN CHECK RECEIVE
0055 C299                         CLR    099H             ;CLR TRANSMISSION INTERRUPT FLAG
0057 D200      F                  SETB   TRNINT           ;SETB TRANS INT FOR PLM51 STATUS CHECK
0059 209828               OVER:   JB     98H,GOBACK ;IF RI NOT SET GOBACK
005C C001                     PUSH       01
005E A999                         MOV    R1,SBUF          ;READ SBUF
0060 C298                         CLR    098H             ;CLEAR RI BIT
0062 C0D0                         PUSH   PSW              ;PUSH REGISTERS USED BY PLM51
0064 C0E0                         PUSH   ACC
0066 C000                         PUSH   00H
0068 C200      F                  CLR    ESCSEG           ;CLR ESC SEQUENCE FLAG
006A 7400      F                  MOV    A,#SERIAL        ;GET SERIAL FIFO RAM START LOCATION
006C 2500      F                  ADD    A,FIFC           ;AND FIND HOW FAR INTO THE FIFO WE ARE
006E F8                           MOV    R0,A             ;PUT IT INTO R0
006F E9                           MOV    A,R1
0070 C2E7                         CLR    0E7H             ;CLR BIT 7 OF ACC
0072 F6                           MOV    @R0,A            ;PUT DATA IN FIFO
0073 B41B02                       CJNE   A,#1BH,OVER1     ;IF DATA IS NOT A ESC KEY THEN GO OVER
0076 D200      F                  SETB   ESCSEG           ;SET ESC SEQUENCE FLAG
0078 0500      F          OVER1:  INC    FIFC             ;MOV FIFC TO NEXT LOCATION
007A D200      F                  SETB   SERINT           ;SET SERIAL INT BIT FOR PLM51 STATUS CHECK
007C D000                         POP    00H              ;POP REGISTERS
007E D0E0                         POP    ACC
0080 D0D0                         POP    PSW
0082 D001                         POP    01H
0084 32                   GOBACK: RETI

0085 C0D0                 BLANK:  PUSH   PSW              ;PUSH REG USED BY PLM51
0087 C0E0                         PUSH   ACC
0089 C082                         PUSH   DPL
008B C083                         PUSH   DPH
008D C000                         PUSH   00H
008F 850082 F                     MOV    DPL,LINE0+1      ;GET LINE0 INFO
0092 850083 F                     MOV    DPH,LINE0        ;AND PUT IT INTO DPTR
0095 7850                         MOV    R0,#50H          ;NUMBER OF CHARACTERS IN A LINE
0097 7420                 NOTYET: MOV    A,#20H           ;ASCII SPACE CHARACTER
0099 F0                           MOVX   @DPTR,A          ;MOV TO DISPLAY RAM
009A A3                           INC    DPTR             ;INCR TO NEXT DISPLAY RAM LOCATION
009B D8FA                         DJNZ   R0,NOTYET        ;IF ALL 50H LOCATIONS ARE NOT FILLED
                                                          ;GO DO MORE
009D D000                         POP    00H              ;POP REGISTERS
009F D083                         POP    DPH
00A1 D082                         POP    DPL
00A3 D0E0                         POP    ACC
00A5 D0D0                         POP    PSW
00A7 22                           RET
                          +1  $EJECT
```

```
MCS-51 MACRO ASSEMBLER    CRTASM

LOC  OBJ            LINE   SOURCE

                     104
00A8 C0D0            105  BLINE:   PUSH  PSW            ;PUSH REGISTERS USED BY PLM51
00AA C0E0            106           PUSH  ACC
00AC C082            107           PUSH  DPL
00AE C083            108           PUSH  DPH
00B0 C000            109           PUSH  00H
00B2 850083   F      110           MOV   DPH,PCINT      ;GET CURRENT DISPLAY RAM LOCATION
00B5 850082   F      111           MOV   DPL,PCINT+1
00B8 438310          112           ORL   DPH,#10H       ;SET BIT 15 FOR RAM ADDRESS DECODING
00BB A800     F      113           MOV   R0,CURSER      ;GET CURSER COLUMN INFO TO TELL HOW
                     114                                ;FAR INTO THE ROW YOU ARE
00BD 7420            115  CONT1:   MOV   A,#20H         ;ASCII SPACE CHARACTER
00BF F0              116           MOVX  @DPTR,A        ;MOV TO DISPLAY RAM
00C0 A3              117           INC   DPTR           ;INCR TO NEXT DISPLAY RAM LOCATION
00C1 08              118           INC   R0
00C2 B850F8          119           CJNE  R0,#50H,CONT1  ;IF NOT AT THE END OF THE LINE
                     120                                ; CONTINUE
00C5 D000            121           POP   00H            ;POP REGISTERS
00C7 D083            122           POP   DPH
00C9 D082            123           POP   DPL
00CB D0E0            124           POP   ACC
00CD D0D0            125           POP   PSW
00CF 22              126           RET
                     127
00D0 C0D0            128  FILL:    PUSH  PSW            ;PUSH REGISTERS USED BY PLM51
00D2 C0E0            129           PUSH  ACC
00D4 C082            130           PUSH  DPL
00D6 C083            131           PUSH  DPH
00D8 C000            132           PUSH  00H
00DA C3              133           CLR   C
00DB 850083   F      134           MOV   DPH,L          ;GET BEGINNING OF LINE RAM LOCATION
00DE 850082   F      135           MOV   DPL,L+1        ;CALCULATED BY PLM51
00E1 438310          136           ORL   DPH,#10H       ;SET BIT 15 FOR DISPLAY RAM ADDRESS DECODE
00E4 784F            137           MOV   R0,#4FH        ;SET UP COUNTER FOR 50H LOCATIONS
00E6 A3              138           INC   DPTR           ;GO PAST THE 0F1H
00E7 7420            139  CONT2:   MOV   A,#20H         ;ASCII SPACE CHARACTER
00E9 F0              140           MOVX  @DPTR,A        ;MOVE TO DISPLAY RAM
00EA A3              141           INC   DPTR           ;INCR TO NEXT DISPLAY RAM LOCATION
00EB D8FA            142           DJNZ  R0,CONT2       ;IF ALL 79 LOCATIONS HAVE NOT BEEN FILLED
                     143                                ;THEN CONTINUE
00ED D000            144           POP   00H            ;POP REGISTERS
00EF D083            145           POP   DPH
00F1 D082            146           POP   DPL
00F3 D0E0            147           POP   ACC
00F5 D0D0            148           POP   PSW
00F7 22              149           RET
                     150
                     151
                     152 +1  $EJECT
```

```
MCS-51 MACRO ASSEMBLER    CRTASM

LOC  OBJ         LINE  SOURCE

                       ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                       ;THIS ROUTINE MOVES DISPLAY RAM DATA TO ROW BUFFER OF 8276
                       ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

00F8 21DB              DDONE:  AJMP    DMADNE

00FA 850083   F        DMA:    MOV     DPH,MASTER        ;LOAD XFER POINTER HIGH BYTE
00FD 850082   F                MOV     DPL,MASTER+1      ;LOAD XFER POINTER LOW BYTE
0100 E0                        MOVX    A,@DPTR
0101 A3                        INC     DPTR
0102 20B3F3                    JB      0B3H,DDONE        ;IF INT1 HIGH, THEN DMA IS OVER
0105 E0                        MOVX    A,@DPTR
0106 A3                        INC     DPTR
0107 E0                        MOVX    A,@DPTR
0108 A3                        INC     DPTR
0109 E0                        MOVX    A,@DPTR
010A A3                        INC     DPTR
010B E0                        MOVX    A,@DPTR
010C A3                        INC     DPTR
010D E0                        MOVX    A,@DPTR
010E A3                        INC     DPTR
010F E0                        MOVX    A,@DPTR
0110 A3                        INC     DPTR
0111 E0                        MOVX    A,@DPTR
0112 A3                        INC     DPTR
0113 E0                        MOVX    A,@DPTR
0114 A3                        INC     DPTR
0115 E0                TEN:    MOVX    A,@DPTR
0116 A3                        INC     DPTR
0117 E0                        MOVX    A,@DPTR
0118 A3                        INC     DPTR
0119 E0                        MOVX    A,@DPTR
011A A3                        INC     DPTR
011B E0                        MOVX    A,@DPTR
011C A3                        INC     DPTR
011D E0                        MOVX    A,@DPTR
011E A3                        INC     DPTR
011F E0                        MOVX    A,@DPTR
0120 A3                        INC     DPTR
0121 E0                        MOVX    A,@DPTR
0122 A3                        INC     DPTR
0123 E0                        MOVX    A,@DPTR
0124 A3                        INC     DPTR
0125 E0                        MOVX    A,@DPTR
0126 A3                        INC     DPTR
0127 E0                        MOVX    A,@DPTR
0128 A3                        INC     DPTR
0129 E0                TWENTY: MOVX    A,@DPTR
012A A3                        INC     DPTR
012B E0                        MOVX    A,@DPTR
012C A3                        INC     DPTR
012D E0                        MOVX    A,@DPTR
012E A3                        INC     DPTR
012F E0                        MOVX    A,@DPTR
```

```
MCS-51 MACRO ASSEMBLER    CRTASM

LOC  OBJ            LINE     SOURCE

0130 A3              208              INC     DPTR
0131 E0              209              MOVX    A,@DPTR
0132 A3              210              INC     DPTR
0133 E0              211              MOVX    A,@DPTR
0134 A3              212              INC     DPTR
0135 E0              213              MOVX    A,@DPTR
0136 A3              214              INC     DPTR
0137 E0              215              MOVX    A,@DPTR
0138 A3              216              INC     DPTR
0139 E0              217              MOVX    A,@DPTR
013A A3              218              INC     DPTR
013B E0              219              MOVX    A,@DPTR
013C A3              220              INC     DPTR
013D E0              221     THIRTY:  MOVX    A,@DPTR
013E A3              222              INC     DPTR
013F E0              223              MOVX    A,@DPTR
0140 A3              224              INC     DPTR
0141 E0              225              MOVX    A,@DPTR
0142 A3              226              INC     DPTR
0143 E0              227              MOVX    A,@DPTR
0144 A3              228              INC     DPTR
0145 E0              229              MOVX    A,@DPTR
0146 A3              230              INC     DPTR
0147 E0              231              MOVX    A,@DPTR
0148 A3              232              INC     DPTR
0149 E0              233              MOVX    A,@DPTR
014A A3              234              INC     DPTR
014B E0              235              MOVX    A,@DPTR
014C A3              236              INC     DPTR
014D E0              237              MOVX    A,@DPTR
014E A3              238              INC     DPTR
014F E0              239              MOVX    A,@DPTR
0150 A3              240              INC     DPTR
0151 E0              241     FORTY:   MOVX    A,@DPTR
0152 A3              242              INC     DPTR
0153 E0              243              MOVX    A,@DPTR
0154 A3              244              INC     DPTR
0155 E0              245              MOVX    A,@DPTR
0156 A3              246              INC     DPTR
0157 E0              247              MOVX    A,@DPTR
0158 A3              248              INC     DPTR
0159 E0              249              MOVX    A,@DPTR
015A A3              250              INC     DPTR
015B E0              251              MOVX    A,@DPTR
015C A3              252              INC     DPTR
015D E0              253              MOVX    A,@DPTR
015E A3              254              INC     DPTR
015F E0              255              MOVX    A,@DPTR
0160 A3              256              INC     DPTR
0161 E0              257              MOVX    A,@DPTR
0162 A3              258              INC     DPTR
0163 E0              259              MOVX    A,@DPTR
0164 A3              260              INC     DPTR
0165 E0              261     FIFTY:   MOVX    A,@DPTR
0166 A3              262              INC     DPTR
```

```
MCS-51 MACRO ASSEMBLER    CNTASM

LOC  OBJ          LINE     SOURCE

0167 E0            263              MOVX    A,@DPTR
0168 A3            264              INC     DPTR
0169 E0            265              MOVX    A,@DPTR
016A A3            266              INC     DPTR
016B E0            267              MOVX    A,@DPTR
016C A3            268              INC     DPTR
016D E0            269              MOVX    A,@DPTR
016E A3            270              INC     DPTR
016F E0            271              MOVX    A,@DPTR
0170 A3            272              INC     DPTR
0171 E0            273              MOVX    A,@DPTR
0172 A3            274              INC     DPTR
0173 E0            275              MOVX    A,@DPTR
0174 A3            276              INC     DPTR
0175 E0            277              MOVX    A,@DPTR
0176 A3            278              INC     DPTR
0177 E0            279              MOVX    A,@DPTR
0178 A3            280              INC     DPTR
0179 E0            281     SIXTY:   MOVX    A,@DPTR
017A A3            282              INC     DPTR
017B E0            283              MOVX    A,@DPTR
017C A3            284              INC     DPTR
017D E0            285              MOVX    A,@DPTR
017E A3            286              INC     DPTR
017F E0            287              MOVX    A,@DPTR
0180 A3            288              INC     DPTR
0181 E0            289              MOVX    A,@DPTR
0182 A3            290              INC     DPTR
0183 E0            291              MOVX    A,@DPTR
0184 A3            292              INC     DPTR
0185 E0            293              MOVX    A,@DPTR
0186 A3            294              INC     DPTR
0187 E0            295              MOVX    A,@DPTR
0188 A3            296              INC     DPTR
0189 E0            297              MOVX    A,@DPTR
018A A3            298              INC     DPTR
018B E0            299              MOVX    A,@DPTR
018C A3            300              INC     DPTR
018D E0            301     SEVNTY:  MOVX    A,@DPTR
018E A3            302              INC     DPTR
018F E0            303              MOVX    A,@DPTR
0190 A3            304              INC     DPTR
0191 E0            305              MOVX    A,@DPTR
0192 A3            306              INC     DPTR
0193 E0            307              MOVX    A,@DPTR
0194 A3            308              INC     DPTR
0195 E0            309              MOVX    A,@DPTR
0196 A3            310              INC     DPTR
0197 E0            311              MOVX    A,@DPTR
0198 A3            312              INC     DPTR
0199 E0            313              MOVX    A,@DPTR
019A A3            314              INC     DPTR
019B E0            315              MOVX    A,@DPTR
019C A3            316              INC     DPTR
019D E0            317              MOVX    A,@DPTR
```

```
MCS-51 MACRO ASSEMBLER    CRTASM

LOC   OBJ               SOURCE

019E  A3                          INC     DPTR
019F  E0                          MOVX    A,@DPTR
01A0  A3                          INC     DPTR
01A1  E0                EIGHTY:   MOVX    A,@DPTR
01A2  A3                          INC     DPTR

01A3  E583              CHECK:    MOV     A,DPH
01A5  B41F0C                      CJNE    A,#1FH,DONE
01A8  E582                        MOV     A,DPL
01AA  B4D007                      CJNE    A,#0D0H,DONE
01AD  750018     F                MOV     RASTER,#18H
01B0  750000     F                MOV     RASTER+1,#00H
01B3  22                          RET

01B4  858300     F      DONE:     MOV     RASTER,DPH
01B7  858200     F                MOV     RASTER+1,DPL
01BA  22                          RET

01BB  C3                DMADONE:  CLR     C
01BC  E582                        MOV     A,DPL
01BE  244F                        ADD     A,#79D          ;ADD 79 TO BUFFER POINTER
01C0  F582                        MOV     DPL,A           ;TO GET TO NEXT DISPLAY LINE
01C2  50DF                        JNC     CHECK           ;IN THE DISPLAY MEMORY
01C4  0583                        INC     DPH
01C6  80DB                        SJMP    CHECK


                        END
```

```
MCS-51 MACRO ASSEMBLER    CRTASM

SYMBOL TABLE LISTING
------ ----- -------

N A M E      T Y P E   V A L U E       A T T R I B U T E S

ACC. . . .   D ADDR    00E0H   A
BLANK. . .   C ADDR    0085H   A PUB
BLINE. . .   C ADDR    00A8H   A PUB
BUFFER . .   C ADDR    003FH   A
CHECK. . .   C ADDR    01A3H   A
CONT1. . .   C ADDR    008DH   A
CONT2. . .   C ADDR    00E7H   A
COUNT. . .   D ADDR    ----      EXT
CURSER . .   D ADDR    ----      EXT
DDONE. . .   C ADDR    00F8H   A
DMA. . . .   C ADDR    00FAH   A
DMADNE . .   C ADDR    01BDH   A
DONE . . .   C ADDR    0184H   A
DPH. . . .   D ADDR    0083H   A
DPL. . . .   D ADDR    0082H   A
EIGHTY . .   C ADDR    01A1H   A
ESCSEQ . .   B ADDR    ----      EXT
FIFO . . .   D ADDR    ----      EXT
FIFTY. . .   C ADDR    0165H   A
FILL . . .   C ADDR    00C0H   A PUB
FORTY. . .   C ADDR    0151H   A
GOBACK . .   C ADDR    0084H   A
L. . . . .   D ADDR    ----      EXT
LINE0. . .   D ADDR    ----      EXT
NOTYET . .   C ADDR    0097H   A
OVER . . .   C ADDR    0059H   A
OVER1. . .   C ADDR    0078H   A
POINT. . .   D ADDR    ----      EXT
PSW. . . .   D ADDR    00D0H   A
RASTER . .   D ADDR    ----      EXT
SBUF . . .   D ADDR    0099H   A
SCAN . . .   B ADDR    ----      EXT
SERBUF . .   C ADDR    0052H   A
SERIAL . .   D ADDR    ----      EXT
SERINT . .   B ADDR    ----      EXT
SEVNTY . .   C ADDR    018DH   A
SIXTY. . .   C ADDR    0179H   A
TEN. . . .   C ADDR    0115H   A
THIRTY . .   C ADDR    013DH   A
TRNINT . .   B ADDR    ----      EXT
TWENTY . .   C ADDR    0129H   A
VERT . . .   C ADDR    0025H   A

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
MCS-51 MACRO ASSEMBLER    KEYBD

ISIS-II MCS-51 MACRO ASSEMBLER V2.1
OBJECT MODULE PLACED IN :F1:KEYBD.OBJ
ASSEMBLER INVOKED BY:  ASM51 :F1:KEYBD.SRC

LOC  OBJ  LINE     SOURCE

          ;****************************************************************************
          ;****************************************************************************
          ;****                                                                   ****
          ;****               SOFTWARE FOR READING AN UNDECODED                   ****
          ;****                           KEYBOARD                                ****
          ;****                                                                   ****
          ;****************************************************************************
          ;****************************************************************************
          ;
          ;
          ;
          ;
          ;       THIS CONTAINS THE SOFTWARE NEEDED TO SCAN AN UNDECODED KEYBOARD
          ;       THIS PROGRAM MUST BE LINKED TO THE MAIN PROGRAMS TO FUNCTION
          ;
          ;
          ;        MEMORY MAP FOR READING KEY BOARD (USING MOVC)
          ;
          ;       ADDRESS FOR KEY BOARD 10FFH TO 17FFH
          ;
          ;
          ;
          ;
                  PUBLIC READER
                  EXTRN DATA (LSTKEY)
                  EXTRN BIT (KEYD,SAME)
          ;
          ;***************************************************************************
          ;*                                                                         *
          ;*                        "READER"ROUTINE"                                 *
          ;*
          ;*************************************************************************
      +1  $EJECT
```

MCS-51 MACRO ASSEMBLER   KEYBO

```
LOC  OBJ            SOURCE


                    UNDECODED_KEYBOARD SEGMENT CODE
----                RSEG UNDECODED_KEYBOARD



0000 C0D0           READER: PUSH  PSW            ;PUSH REG USED BY PLM51
0002 C0E0                   PUSH  ACC
0004 C082                   PUSH  DPL
0006 C083                   PUSH  DPH
0008 C000                   PUSH  00H
000A C001                   PUSH  01H
000C C002                   PUSH  02H
000E C003                   PUSH  03H
0010 9010FF                 MOV   DPTR,#10FFH    ;INITIALIZE DPTR TO KEYBOARD
                                                 ;ADDRESS
0013 7900                   MOV   R1,#00H        ;CLR ZERO COUNTER
0015 7800      F            MOV   R0,#LSTKEY     ;GET KEYBOARD RAM POINTER
0017 7B08                   MOV   R3,#08H        ;INITIALIZE LOOP COUNTER
0019 C200      F            CLR   KEY0           ;INITIALIZE PLM51 STATUS BITS
001B D200      F            SETB  SAME
001D 8602           MORE:   MOV   02H,@R0        ;MOV LAST KEYBOARD SCAN TO 02H
001F E4                     CLR   A
0020 93                     MOVC  A,@A+DPTR      ;SCAN KEYBOARD
0021 F4                     CPL   A              ;INVERT
0022 6005                   JZ    ZERO           ;IF SCAN WAS ZERO GO INCREMENT ZERO COUNTER
0024 B50224                 CJNE  A,02H,NTSAME   ;COMPARE WITH LAST SCAN IF NOT THE SAME
                                                 ;THEN CLR SAME BIT AND WRITE NEW INFORMATION
                                                 ;TO RAM
0027 8005                   SJMP  EQUAL          ;IF EQUAL JMP OVER INCR OF ZERO COUNTER
0029 0501           ZERO:   INC   01H            ;INCR ZERO COUNTER
002B B5021D                 CJNE  A,02H,NTSAME
002E 08             EQUAL:  INC   R0             ;STEP TO NEXT SCAN RAM LOCATION
002F 0583                   INC   DPH            ;NEXT KEYBOARD ADDRESS
0031 DBEA                   DJNZ  R3,MORE        ;IF LOOP COUNTER NOT 0, SCAN AGAIN
0033 B90804                 CJNE  R1,#08H,BACK   ;CHECK TO SEE IF ALL 8 SCANS WHERE 0
0036 D200      F            SETB  KEY0           ;IF YES SET KEY0 BIT
0038 C200      F            CLR   SAME
003A D003           BACK:   POP   03H
003C D002                   POP   02H            ;POP REGISTERS
003E D001                   POP   01H
0040 D000                   POP   00H
0042 D083                   POP   DPH
0044 D082                   POP   DPL
0046 D0E0                   POP   ACC
0048 D0D0                   POP   PSW
004A 22                     RET

004B F6             NTSAME: MOV   @R0,A          ;IF SCAN WAS NOT THE SAME THEN PUT NEW
                                                 ;SCAN INFO INTO RAM
004C C200      F            CLR   SAME           ;CLR SAME BIT
004E 80DE                   SJMP  EQUAL          ;GO DO MORE


                    END
```

```
MCS-51 MACRO ASSEMBLER    KEYBD

SYMBOL TABLE LISTING
------ ----- -------

N A M E              T Y P E   V A L U E        A T T R I B U T E S

ACC. . . . . . . . . D ADDR    00E0H   A
BACK . . . . . . . . C ADDR    003AH   R        SEG=UNDECODED_KEYBOARD
DPH. . . . . . . . . D ADDR    0083H   A
DPL. . . . . . . . . D ADDR    0082H   A
EQUAL. . . . . . . . C ADDR    002EH   R        SEG=UNDECODED_KEYBOARD
KEYD . . . . . . . . B ADDR    ----      EXT
LSTKEY . . . . . . . D ADDR    ----      EXT
MORE . . . . . . . . C ADDR    001DH   R        SEG=UNDECODED_KEYBOARD
NISAME . . . . . . . C ADDR    004BH   R        SEG=UNDECODED_KEYBOARD
PSW. . . . . . . . . D ADDR    00D0H   A
READER . . . . . . . C ADDR    0000H   R PUB    SEG=UNDECODED_KEYBOARD
SAME . . . . . . . . B ADDR    ----      EXT
UNDECODED_KEYBOARD   C SEG     0050H            REL=UNIT
ZERO . . . . . . . . C ADDR    0029H   R        SEG=UNDECODED_KEYBOARD

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
MCS-51 MACRO ASSEMBLER    DECODE

ISIS-II MCS-51 MACRO ASSEMBLER V2.1
OBJECT MODULE PLACED IN :F1:DECODE.OBJ
ASSEMBLER INVOKED BY:  ASM51 :F1:DECODE.SRC

LOC  OBJ  LINE     SOURCE



          ;*****************************************************************
          ;*****************************************************************
          ;****                                                         ****
          ;****          SOFTWARE FOR DECODED KEYBOARD                  ****
          ;****                                                         ****
          ;*****************************************************************
          ;*****************************************************************
          ;
          ;
          ;
          ;
                  PUBLIC DETACH
                  EXTRN DATA (LSTKEY)
                  EXTRN BIT  (KBDINT)


          ;
          ;********************************************************************
          ;*                                                                  *
          ;*       "DECODE" INTERRUPT ROUTINE FOR DECODED KEYBOARDS           *
          ;*                                                                  *
          ;********************************************************************
       +1 $EJECT
```

```
MCS-51 MACRO ASSEMBLER    DECODE

LOC  OBJ           LINE   SOURCE

                    27
                    28    DECODED_KEYBOARD SEGMENT CODE
----                29    RSEG DECODED_KEYBOARD
                    30
0000 C0D0           31    DETACH: PUSH   PSW           ;PUSH REGISTERS
0002 C082           32            PUSH   DPL           ;USED BY PLM51
0004 C083           33            PUSH   DPH
0006 C0E0           34            PUSH   ACC
0008 9080FF         35            MOV    DPTR,#80FFH   ;ADDRESS FOR KEYBOARD
000B E4             36            CLR    A
000C 93             37            MOVC   A,@A+DPTR     ;FETCH ASCII BYTE
000D F500      F    38            MOV    LSTKEY+1,A    ;MOV TO MEMORY TO BE READ BY PLM51
000F D200      F    39            SETB   KBDINT        ;LET PLM51 KNOW THERE IS A BYTE
0011 758CFF         40            MOV    TH0,#0FFH     ;SET COUNTER TO FFFFH SO INTERRUPT
0014 758AFF         41            MOV    TL0,#0FFH     ;ON THE NEXT FALLING EDGE OF T0
0017 D0E0           42            POP    ACC
0019 D083           43            POP    DPH           ;POP REGISTERS
001B D082           44            POP    DPL
001D D0D0           45            POP    PSW
001F 32             46            RETI
                    47
                    48
                    49
                    50
                    51    END
```

```
CS-51 MACRO ASSEMBLER    DECODE

YMBOL TABLE LISTING
----- ----- -------

 A M E              T Y P E   V A L U E       A T T R I B U T E S

.CC. . . . . . . .  D ADDR    00E0H   A
ECODED_KEYBOARD     C SEG     0020H           REL=UNIT
ETACH . . . . . .   C ADDR    0000H   R PUB   SEG=DECODED_KEYBOARD
PH. . . . . . . .   D ADDR    0083H   A
PL. . . . . . . .   D ADDR    0082H   A
BDINT . . . . . .   B ADDR    ----      EXT
.STKEY . . . . . .  D ADDR    ----      EXT
SW. . . . . . . .   D ADDR    00D0H   A
H0. . . . . . . .   D ADDR    008CH   A
L0. . . . . . . .   D ADDR    008AH   A

EGISTER BANK(S) USED: 0

SSEMBLY COMPLETE, NO ERRORS FOUND
```

```
MCS-51 MACRO ASSEMBLER    DETACH


ISIS-II MCS-51 MACRO ASSEMBLER V2.1
OBJECT MODULE PLACED IN :F1:DETACH.OBJ
ASSEMBLER INVOKED BY:  ASM51 :F1:DETACH.SRC


LOC  OBJ  LINE     SOURCE



                   ;*************************************************************************
                   ;*************************************************************************
                   ;****                                                                  ****
                   ;****           SOFTWARE FOR A SERIAL OR DETACHABLE                    ****
                   ;****                          KEYBOARD                                ****
                   ;****                                                                  ****
                   ;*************************************************************************
                   ;*************************************************************************
                   ;
                   ;
                   ;
                   ;  THIS CONTAINS THE SOFTWARE NEEDED TO PERFORM A SOFTWARE SERIAL
                   ;  PORT FOR SERIAL KEYBOARDS AND DETACHABLE KEYBOARD. THIS PROGRAM MUST
                   ;  BE LINKED TO THE MAIN PROGRAMS FOR USE.
                   ;
               +1  $EJECT
```

```
MCS-51 MACRO ASSEMBLER    DETACH

LOC  OBJ        LINE     SOURCE

                         ;
                         ;

00B4                             INPUT EQU  T0


                                 PUBLIC DETACH
                                 EXTRN DATA (LSTKEY)
                                 EXTRN BIT (RCVFLG,SYNC,BYFIN)
                                 EXTRN BIT (KBDINT,ERROR)

                         ;
                         ;
                         ;

                         ;       TIMER 0 LOAD VALUES FOR DIFFERENT BAUD RATES
                         ;       USED WITH DETACHABLE KEYBOARDS
                         ;
                         ;
                         ;
                         ;   BAUD       START BIT DETECT          MESSAGE DETECT
                         ;   110           0EFA2H                    0DF45H
                         ;   150           0F400H                    0E800H
                         ;
                         ;
                         ;
                         ;
0000                            START0          EQU    000H    ;LOW BYTE FOR 150 BAUD
00F4                            START1          EQU    0F4H    ;HIGH BYTE FOR 150 BAUD
0000                            MESSAGE0        EQU    000H    ;LOW BYTE FOR 150 BAUD
00E8                            MESSAGE1        EQU    0E8H    ;HIGH BYTE FOR 150 BAUD
                     +1  $EJECT
```

```
MCS-51 MACRO ASSEMBLER    DETACH

LOC  OBJ            LINE     SOURCE

                      52
                      53     ;
                      54     ;*******************************************************************************
                      55     ;*                                                                             *
                      56     ;*              "DETACH" INTERRUPT ROUTINE FOR DETACHABLE KEYBOARDS            *
                      57     ;*                                                                             *
                      58     ;*******************************************************************************
                      59
                      60
                      61
                      62     DETACHABLE_KEYBOARD SEGMENT CODE
----                  63     RSEG DETACHABLE_KEYBOARD
                      64
0000 C0D0             65     DETACH: PUSH   PSW                 ;PUSH REGISTERS USED BY PLM51
0002 C0E0             66             PUSH   ACC
0004 200013    F      67             JB     RCVFLG,VALID        ;IF RECEIVE FLAG SET GET NEXT BIT
0007 20B44A           68             JB     INPUT,RSI           ;IF T0 IS A 1 THEN NOT A START BIT
000A D200      F      69             SETB   RCVFLG              ;IF T0 IS 0 THEN IT  A  START BIT
000C 758CF4           70             MOV    TH0,#START1         ;SET TIMER TO INTERRUPT IN THE MIDDLE OF START BIT
000F 758A00           71             MOV    TL0,#START0
0012 E589             72             MOV    A,TMOD
0014 C2E2             73             CLR    0E2H                ;SET TIMER COUNTER TO TIMER MODE
0016 F589             74             MOV    TMOD,A
0018 8028             75             SJMP   FINI                ;GO BACK TO PROGRAM
                      76     ;
001A 200010    F      77     VALID:  JB     SYNC,NXTBIT         ;CHECK IF VALID START BIT HAS BEEN SEEN
001D 20B434           78             JB     INPUT,RSI           ;IF NOT CHECK IF VALID START BIT
0020 D200      F      79             SETB   SYNC                ;IF YES SET SYNC
0022 750080    F      80             MOV    LSTKEY,#80H         ; INIT LSTKEY
0025 758CE8           81             MOV    TH0,#MESSAGE1
0028 758A00           82             MOV    TL0,#MESSAGE0       ;SET TIMER FOR 1 BIT TIME
002B 8018             83             SJMP   FINI                ;AND GO BACK TO MAIN PROGRAM
                      84     ;
002D 758CE8           85     NXTBIT: MOV    TH0,#MESSAGE1
0030 758A00           86             MOV    TL0,#MESSAGE0       ;SET TIMER FOR 1 BIT TIME
0033 200014    F      87             JB     BYFIN,STOP          ;CHECK TO SEE IF ALL 8 BITS HAVE BEEN RECEIVED
0036 E500      F      88             MOV    A,LSTKEY            ;GET WORKING REGISTER
0038 A2B4             89             MOV    C,INPUT             ;GET NEXT BIT FROM T1
003A 13               90             RRC    A
003B F500      F      91             MOV    LSTKEY,A
003D 5006             92             JNC    FINI                ;IF NO CARRY THEN NOT DONE
003F D200      F      93             SETB   BYFIN
0041 C2E7             94             CLR    0E7H                ;CLR BIT 7
0043 F500      F      95             MOV    LSTKEY+1,A          ;MOV FINAL CODE TO LSTKY+1
0045 D0E0             96     FINI:   POP    ACC
0047 D0D0             97             POP    PSW
0049 32               98             RETI
```

```
MCS-51 MACRO ASSEMBLER    DETACH

LOC  OBJ            LINE     SOURCE

                      99     ;
004A 30B405           100    STUP:   JNB     INPUT,ERR        ;IF NOT 1 THEN NOT A VALID STOP BIT
004D D200      F      101            SETB    RBUINT           ;TELL PLM A BYTE IS READY
004F 020000    F      102            JMP     RST              ;AND GO BACK TO MAIN PROGRAM
                      103    ;
0052 D200      F      104    ERR:    SETB    ERRCR
0054 C200      F      105    RST:    CLR     RCVFLG           ;CLEAR FLAGS
0056 C200      F      106            CLR     SYNC
0058 C200      F      107            CLR     BYTIN
005A E589             108            MOV     A,TMOD
005C D2E2             109            SETB    0E2H             ;SET TIMER 0 TO COUNTER MODE
005E F589             110            MOV     TMOD,A
0060 758CFF           111            MOV     TH0,#0FFH        ;SET COUNTER TO FFFFH SO INTERRUPT
0063 758AFF           112            MOV     TL0,#0FFH        ;ON NEXT FALLING EDGE OF T0
0066 800D             113            SJMP    FINI
                      114
                      115
                      116
                      117
                      118            END
```

```
MCS-51 MACRO ASSEMBLER    DETACH

SYMBOL TABLE LISTING
------ ----- -------

N A M E                 T Y P E   V A L U E       A T T R I B U T E S

ACC . . . . . . . . .   D ADDR    00E0H    A
BYFIN . . . . . . . .   B ADDR    ----       EXT
DETACH. . . . . . . .   C ADDR    0000H    R PUB   SEG=DETACHABLE_KEYBOARD
DETACHABLE_KEYBOARD     C SEG     0068H            REL=UNIT
ERR . . . . . . . . .   C ADDR    0052H    R       SEG=DETACHABLE_KEYBOARD
ERROR . . . . . . . .   B ADDR    ----       EXT
FINI. . . . . . . . .   C ADDR    0045H    R       SEG=DETACHABLE_KEYBOARD
INPUT . . . . . . . .   B ADDR    0080H.4  A
KBDINT. . . . . . . .   B ADDR    ----       EXT
LSTKEY. . . . . . . .   D ADDR    ----       EXT
MESSAGE0. . . . . . .     NUMB    0000H    A
MESSAGE1. . . . . . .     NUMB    00E8H    A
NXTBIT. . . . . . . .   C ADDR    002DH    R       SEG=DETACHABLE_KEYBOARD
PSW . . . . . . . . .   D ADDR    00D0H    A
RCVFLG. . . . . . . .   B ADDR    ----       EXT
RST . . . . . . . . .   C ADDR    0054H    R       SEG=DETACHABLE_KEYBOARD
START0. . . . . . . .     NUMB    0000H    A
START1. . . . . . . .     NUMB    00F4H    A
STOP. . . . . . . . .   C ADDR    004AH    R       SEG=DETACHABLE_KEYBOARD
SYNC. . . . . . . . .   B ADDR    ----       EXT
T0. . . . . . . . . .   B ADDR    00B0H.4  A
TH0 . . . . . . . . .   D ADDR    008CH    A
TL0 . . . . . . . . .   D ADDR    008AH    A
TMOD. . . . . . . . .   D ADDR    0089H    A
VALID . . . . . . . .   C ADDR    001AH    R       SEG=DETACHABLE_KEYBOARD

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

APPENDIX B
REFERENCES

1. John Murray and George Alexy, *CRT Terminal Design Using The Intel 8275 and 8279,* Intel Application Note AP-32, Nov., 1977.
2. John Katausky, *A Low Cost CRT Terminal Using The 8275,* Intel Application Note AP-62, Nov., 1979.