Microprocessors in CRT terminals

by JOHN WHITING and SANDY NEWMAN

Beehive Medical Electronics
Salt Lake City, Utah

INTRODUCTION

The recent introduction of integrated circuit microprocessors has produced a new variety of CRT terminal—the firmware terminal. The firmware terminal incorporates a microprocessor to control data flow, using a control program supplied by the terminal manufacturer in Read-Only-Memory. Priced only slightly above hardwired editing terminals, it can perform far more complex functions, assisting both the operator and the computer system into which it is connected. It is priced well below the user-programmable terminal which requires magnetic storage for program and a much higher level of sophistication from both the sales force and users.

WHAT IS A CRT TERMINAL?

The CRT computer terminal is one of a variety of devices used to communicate with computers. It is distinguished by using a television screen (CRT) for presenting computer data to the human operator, rather than using an electric typewriter, loudspeaker, flashing billboard or other device. Messages appear on the screen as several lines of words, much as they would appear on a typewriter sheet of paper; a common format is 25 lines of 80 characters each, including spaces. Punctuation marks, numerals, and other symbols may appear as well as upper and usually lower case characters.

The operator inputs data through a keyboard much like a typewriter keyboard. Keyboard data may go directly to the computer, but more commonly is temporarily stored in the memory used for the display. Thus the message entered from the keyboard appears on the CRT screen just as it would if it were being typed on paper by a typewriter, but with several advantages. For one, changes can be made by simply backing up the cursor, which marks the point at which data is entered, to the point to be corrected, entering the correct data in place of the incorrect in the display memory, and then moving the cursor back to wherever data entry left off. For another, if the change requires inserting or deleting characters or words, the terminal can shift characters following the change either right or left to make additional room or close up space as necessary, even though this may involve moving entire words from one line to another. Then, since the message is stored as digital codes in the display memory, it can be transmitted to the computer on command without the need for converting printed characters with an optical reader.

The fundamental elements of a CRT terminal are shown in Figure 1. Data comes into the terminal either from the keyboard or from the serial port. The serial port sends and receives data to a computer mainframe as a serial stream of digital bits, at rates up to 1200 characters per second. (If the mainframe is located very far away, a device called a data set translates the digital data into audible tones to be sent over telephone lines to a second data set which translates the tones back into digital data.) The control logic reads/writes display memory data at the location defined by the cursor control. The cursor advances as characters are written and can also be moved about by control codes from either the keyboard or the computer. Separate character generation circuitry is used to directly access memory data and translate it into dot patterns to drive the CRT. The circuitry must rewrite the characters on the CRT 50 to 60 times a second to maintain a flicker-free display, so high-speed circuits are used for this function. Relatively slow-speed circuits can be used for the control logic which performs a far wider variety of functions, some of which may be very complex, depending on the application.

WHAT A MICROPROCESSOR CAN DO

A microprocessor with its associated program memory and interface circuits can be used very effectively in place of hardwired control logic to pass data between the keyboard, serial port, and display memory. In this position, it can intercept and interpret codes as they are received, storing characters in memory and executing control functions as necessary. It can also manipulate data in the display memory and format messages to send to the mainframe. Not only does this make feasible the execution of much more complex functions than possible with hardwired logic, it reduces the cost and time required to modify terminal functions for special applications. Since the hardware is unchanged by such program modifications, extensive retesting of the terminal design is not required, reliability and serviceability are unaffected, and all terminals of a given hardware type can be manufac-
tured on the same production line with the specialized program installed only after initial testing is completed. Thus, small quantity special orders can enjoy many of the mass production advantages of a standard product and yet have many specialized functional characteristics.

For example, a keyboard whose keys are coded to be teletypewriter compatible can be made typewriter compatible by changing a few keycap legends and translating the codes received from those keys with the microprocessor. Function keys can similarly be moved around on the keyboard by moving the corresponding addresses in the firmware table used to locate the function routine. Functional modifications are also straightforward. For example, a call to the Cursor Down subroutine can be inserted in the Carriage Return routine to make the Return key do both a Return and Line Feed, and the Skip-Cursor Right key sequence may be changed to move the cursor right by 20 positions instead of 16 by simply changing a numerical constant in the program.

The ability of the microprocessor to give specialized interpretation to both functional controls and displayable graphics originating from the keyboard also applies to codes sent from the central processor. Thus, the special codes which cause portions of the display to blink or to display black characters on a white background rather than white on black for emphasis may be transmitted over the communication lines as a two-code sequence but stored in the display memory as a single code. Code sequences may also be used for control functions and setting cursor positions, margins or tab locations, with the microprocessor translating numeric data between binary and decimal or other code formats.

In multi-drop polling networks the microprocessor can carry on interactive exchanges with the central processor. These can be very simple exchanges or quite complex, according to the requirements of the system. For example, a start-of-header code, an identifying address code, and a single status or command code may form the entire message, or the format can be expanded with preceding codes for word synchronization, multiple address and status codes, terminating codes and error check codes. The terminal might transmit only on operator command, or the central processor may be able to interrogate the terminal to find out whether the terminal is on-line or off-line, dumping data to a slave printer, actively entering information from keyboard, waiting to send data or waiting for a reply. Since the terminal program has complete control over message format, many different systems can all use the same terminal hardware.

The microprocessor can also control some of the communication signal lines directly; and when the interface is designed appropriately, it can be programmed to cooperate with other terminals in a serial string (daisy-chain) to a single data set or computer port for increased efficiency and reduced cost. As a general rule, the microprocessor should have as much control as possible over data flow and logic signals but timing controls such as Clear-to-Send delays are best handled by hardware which can be adjusted as necessary at each installation.

One of the most complex functions performed by the better hardwired terminals is the Delete Line, where a counter is employed to delete all 80 characters on a line of text; a microprocessor can use an internal register for the same purpose. Besides repetitive functions, such data-dependent functions as Delete Sentence can be performed, with the microprocessor deleting all characters between any two sentence delimiters. A variety of algorithms may be used to maintain word integrity and adjust column widths, giving special treatment to hyphens and other formatting codes. The microprocessor can search through the display memory data to locate all instances of whatever word the operator wants to find for correction or verification. In an accounting application, a column or row of numbers could be summed by even simple microprocessors, but more difficult calculations may be better handled by either the mainframe or a $20 pocket calculator. Other special functions use the microprocessor’s access to the mainframe as well as to the display memory to take advantage of a centralized data base.

Terminal functions to be performed on command from the mainframe should generally be executable in one or two milliseconds at most, corresponding to the time required to transmit data codes over fast serial communication lines. Fortunately, the data manipulations which take long times for the terminal microprocessor to execute can be performed within the mainframe before the text is transmitted.

**RAM VERSUS SHIFT REGISTER**

The role of the cursor control circuitry depends on whether random-access memory (RAM) or shift register memory is used for the display memory. To write a character into a RAM memory simply requires addressing the matrix point and writing the code directly from the data bus, a natural task for a microprocessor. But to erase
the entire memory with this direct approach requires sequentially addressing approximately 2000 memory locations and writing spaces in them, a very time consuming task for a microprocessor. Shift register memory is like a garden hose full of marbles; for each character stored at the input end of the memory, one character is pushed out at the output. When each character output is put back into the input, a memory is formed with sequential access to memory characters. To change a character, the new character must be put in the data stream in place of the former character at just the right point in the memory cycle, so the cursor circuitry must hold the new character when it is output from the data bus, and identify the right point in time to break it into the shift register data stream. But to erase the shift register memory, it is only necessary to stuff space characters into the input for one full cycle of the memory. Other tasks involving sequential access of many memory locations are insertion and deletion of characters in text, skipping over protected fields, tabulation by field, paragraph, or other text block and search for Start-of-Message characters. The abundance of such functions is a consequence of the sequential nature of the data displayed on the CRT, itself derived from the sequential nature of human thought. Thus shift registers are often used for display memories despite the awkwardness of writing single characters in them.

Both random and sequential access to the display memory is possible with RAM memory supported by specialized hardware for handling those sequential addressing functions which must be executed rapidly and with minimal microprocessor support. How much specialized hardware is required depends on what functions the terminal must perform faster than the microprocessor can do them. Such hardware can sequentially access currently available RAMs nearly as fast as popular shift registers, but adds to the cost and complexity of the terminal. Another alternative is the use of RAM memory, organized as many small record blocks linked by address characters. While this uses some memory locations for the linking and complicates display generation, it may be the most powerful and versatile scheme for an adequate microprocessor. With the low cost RAMs and better microprocessors recently made available, we expect to see changes from the traditional use of shift registers in terminals.

CHOICE OF A MICROPROCESSOR

The speed required of a microprocessor to be used in a CRT terminal depends largely upon the interface between it and the display memory, and the demands of the application. Conventional measures of processor speed, such as time to add two numbers, are not particularly meaningful here, as long as the microprocessor can identify and respond to approximately 1200 codes per second, which even the primitive 8008-1 microprocessor from Intel can do (just barely). With a shift register memory, throughput is usually limited by the time required to read characters as the cursor is moved through memory. With a RAM memory the limitation is in sequential access tasks, such as selectively erasing flagged data, unless either hardware support is given to perform all required sequential addressing tasks, or the terminal specification can be written around the microprocessor's limitations. (Since firmware CRT terminals are usually sold as improvements over hardwired, shift register memory CRT terminals, fast execution of sequential tasks is generally assumed by salesmen and customers alike.) Given adequate hardware support, almost any modern microprocessor is fast enough; without hardware support, look for microcoded instructions to repeatedly read-swap-write and index in approximately one microsecond.

CRT terminals are very price-competitive, and the cost of Read-Only-Memory for program storage invariably exceeds the cost of the microprocessor itself, with Programmable ROMs costing many times more. Since memory cost is included in sales price, the instruction set should be optimized for minimum memory utilization. The Exclusive-OR instructions of the 8008-1 are very nice for computing check characters and the even-odd parity flag is used in the most speed-critical routines. Other features which we have found useful are the logical and arithmetic operations with the following 8-bit byte, and especially the compare-immediate instruction. Since almost all routines executed in a CRT terminal application are simple and short, conditional branch and subroutine call instructions find heavy usage and relative addressing is most convenient. Subroutine nesting goes four or five levels deep at most and a LIFO stack for the program counter is very helpful.

Interrupts are not needed as long as the terminal is doing just one task at a time, as is often the case; however, in polling systems where both the operator and the central processor are sending data simultaneously, a reasonable interrupt handling capability is almost essential. This requirement also arises if the terminal is to both send and receive data simultaneously, or is to pass data from the central processor to a slave printer while continuing normal keyboard service. Thus the terminal application as well as the hardware design affect the choice of a microprocessor from among the many available.

WRITING TERMINAL CONTROL PROGRAMS

The need for specialized hardware to debug programs and program PROMs confines the writing of programs to a few specialists at the factory and sophisticated OEM customers. Consequently, machine language is used and description of the hardware-software interface is largely word-of-mouth between the hardware designers and the programmers, which makes it even more difficult for outsiders to do their own programming.

The first step in modifying a program is understanding what is required and how the terminal is to be used. Then the routines which are affected by the change must be identified which presumes knowledge of the methods and structure of the program. Register usage in particular
must be understood, since temporary data cannot be stored in PROM memory and scratchpad space is limited. Familiarity with the hardware devices to be used in execution of the function is presumed, including worst case timing for access to hardware indicators following a change of state. Following coding of the change, it must be debugged and some means of testing it provided, usually at the customer's site. Finally, the change must be documented so that it can be duplicated or modified in the future.

**DYNAMIC DEBUG**

The problem of dynamic debugging is complicated by the fact that the microprocessor has no independent status indicators other than the display of the terminal being debugged. Three solutions to this problem are: replace the microprocessor chip directly in the printed circuit board with a black box version of the microprocessor incorporating a control panel; simulate the microprocessor on a minicomputer; or write a program debug to be resident in memory alongside the terminal controlling program.

**Microprocessor control panel**

Figure 2 shows the control panel of a black box used in place of an Intel 8008 microprocessor. It plugs into the microprocessor socket through an umbilical cord and contains an 8008 chip interfaced with latches and timing circuits to permit either single stepping of instructions or normal execution to a breakpoint address. LED's indicate address and bus data as well as machine cycle type and status flags. The breakpoint address is specified by toggle switches, as is a data byte which may be substituted for bus data. This substitution capability is very useful in debugging hardware with repeated execution of an instruction, as well as in correcting instructions or modifying data while single stepping through a program. The main drawbacks of the black box approach are that substitutions must be inserted by hand each time the program encounters the bad byte, and only one breakpoint can be set at a time. The advantages are the real-time execution, straightforward single-stepping, economy, portability and universality of application in any 8008 system.

**Simulator**

Execution of a microprocessor instruction set can be easily simulated within a minicomputer by interpreting each program step and imitating its execution. However, input and output instructions which control a terminal's internal devices can best be simulated by passing these instructions to a terminal programmed to recognize the command, cause it to be executed and reply with an acknowledgment which includes data from input instructions. This combines the power of minicomputer simulation and normal control of keyboard, display, and other internal devices, except for the serial transmitter and receiver. The disadvantage of this approach is the slow simulation speed of I/O instructions, since five serial codes must be passed for every I/O instruction executed.

The console display of a simulator for a Super Bee terminal using the Intel 8008 microprocessor is represented in Figure 3. It shows a program halted at location 650 prior to executing a 117 (INPUT device 7) as indicated by...
the Data field. The condition flags set by the last arithmetic operation are all low (not zero, no sign and odd parity) except for the carry indicator, which is high. The contents of each register, A, B, C, D, E, H, and L are as shown on line 4. Four breakpoints have been set at 060, 3153, 1120 and 10005, which halt execution and cause the console display to be regenerated if they are encountered during execution. The RTN stack is a last in first out (LIPO) stack of return addresses. The contents of memory from 641 to 657 are displayed. The trace buffer records the branches of a program. Given is an example of repeated execution of a short service loop. The last jump was to location 644 (top of the stack) and originated from location 044. The program arrived at 044 by sequential execution from 031. The program jumped from location 047 to 031 and arrived at 047 by returning from 657. The top of RTN stack indicates that the next return will also be to location 047, repeating the sequence as seen below in the trace buffer.

Each of the items on the console display may be modified from the console keyboard at any point in the execution of the program. Additional keyboard functions available by single key depressions at the console are Single Step, Program Run, Examine Contents at PC, Modify Contents at PC, Read Paper Tape, Examine Trace Buffer, and Punch Paper Tape. A control “T” from the console may be used to halt program execution. The Carry switch on the minicomputer control panel causes characters from the console keyboard to be interpreted as serial inputs to the terminal as if from a mainframe and are shown in the RCVR field of Figure 3.

This system eliminates the nuisance of erasing and reprogramming PROMs for program development, since the program stored in the minicomputer memory can be easily modified from the console. Overlay techniques can be used to modify blocks of programs, which is particularly useful in changing routines and polling protocols when isolated as stand-alone items. The addition of peripheral memory allows a program to be assembled, executed, modified, debugged and punched into PROM programming format without leaving the control console. Of course, this is not a portable system and the long execution time of 1/0 instructions (about 400 times normal) restricts testing of functions whose speed and efficiency depend on multiple display memory operations. However, the slow execution enhances visualization of complex or extended display functions.

**Resident debug**

If memory space is available in the terminal, a resident debug program may be written which can either single step or run the terminal control program to breakpoints. This debug program must share the CRT display with the terminal program, either by allocating a portion of the screen to the debug program or by swapping terminal data in and out of auxiliary RAM memory to make room for the debug display. When the same RAM is used for storage of the terminal program, substitutions can be quickly and easily made and breakpoints may be inserted to return program control to the debug program; but means must then be provided to load the terminal program into the RAM from the assembly system. Register contents, memory contents, program counter, breakpoints, return stack and status flags can be displayed and modified if the necessary Push and Pop instructions are available (as on the Intel 8080, but not the 8008). Care should be taken to avoid placing breakpoints or starting execution in the middle of the multi-byte instructions. Such a debugging system has the advantages of executing a self-contained program at real-time speeds with normal access to I/O devices, albeit in a modified terminal.

**SUMMARY**

We expect to see accelerated growth in the utilization of firmware CRT terminals as users become more familiar with their advantages and capabilities. By taking advantage of terminal capabilities in systems design and assigning to the terminal those tasks which it can best perform, system throughput can be maximized, while operator-related functions can enjoy the flattery of dedicated processing.