PAM/8: A New Approach to Front Panel Design

Gordon Letwin
Heath Company
Benton Harbor MI 49022

Since the first personal computers appeared about three years ago, the field has been growing and advancing at an ever increasing rate. The variety and complexity of products increases even while the cost decreases. Indeed, the field has evolved so rapidly that it has gone through two generations (using the term somewhat loosely) in those three years. The first generation machines were typified by the first 8800 system sold by MITS, a bare bones machine festooned with switches and lights. It took a fair amount of technical know how to build one of these to get it operational. Before long, however, a new generation of machines was available. These, such as the SwTPC 6800, were usually cheaper and simpler to build, using fewer but more powerful integrated circuits.

And in July 1977, the Heath Company announced its two versions of the home computer idea, the H8 and H11 systems. I write as one of the persons who took part in the design of the H8's front panel firmware, an 8080 program called "PAM/8" which shows how software and hardware are often intimately related.

Microprocessor Front Panels

The ideal front panel for a microcomputer should allow its user total control and access to the processor's workings. A good panel system should allow an instantaneous display of the processor's states, register contents, memory contents, and other operating flags. An operator should be able to force a new state, register value, or memory value upon the processor with ease at any time without otherwise interfering with the executing program. In other words, it should be possible to examine any memory location or any register at any time without disturbing the program.

Ten years ago the implementation of such a front panel was obvious. The processor was built up from components such as integrated circuits, and the flags and registers were directly available on the circuit cards. In the remainder of this article, I will refer to this type of machine as a *discrete processor*, although it may be built out of high level integrated circuits. To build a suitable front panel for such a discrete processor, it is merely necessary to run a wire to a front panel indicator. Likewise, special logic can be built to allow flags and registers to be set from the front panel switches, usually when the machine is in a halted condition. Readers may have had experience with some of these minicomputer systems, such as the CDC 1700 or the IBM 1130 and 1620. This design works reasonably well, but the binary format is inconvenient and the cost of the front panel hardware and logic can be prohibitive for use in a personal computing system.

The situation was considerably changed with the advent of microprocessors. Now, for the first time, a full-fledged computer is within the financial reach of the general public. Unfortunately, the very development which brought this exciting possibility also brought problems. With a 1 integrated circuit microprocessor, the processor flags and register contents were no longer available for a front panel system, being buried out of reach of any possible hardware hookups. A typical microprocessor integrated circuit only has 40 connection pins (or pinouts). These are partly taken up with power supply and clocking signals, as well as the data and address buses. The remaining pins are allocated to receiving and providing signals to interface the processor to the rest of the computer system. As a result, there is no direct way to determine the contents of the processor's registers.

Previous Front Panel Systems

Attempts to solve this fundamental problem of control over the microprocessor have been responsible for the major differences between competing machines. The first widely available machine, the MITS 8800, used a direct approach to front panel design: it simply had LED readouts for each pinout on the microprocessor chip and a bank of switches hooked across the data and address busses. Some additional logic was incorporated to control the running state of the microprocessor and to allow memory locations to be read to and written from via the front panel. This scheme is a straightforward adaptation of traditional panel design; unfortunately, there wasn't a great deal of correspondence between the useful items a programmer might want and the data available on the processor pinouts.

The difficulties of using such a panel system are by now nearly legendary: it is very awkward and time consuming to get information in and out of the processor. For example, to simply determine the contents of a register, it is necessary to stop the processor, write a small program to store the register in a memory location, key it in to some unused portion of memory, run it, read the stored value from memory, and then restore control to the interrupted program. Needless to say, this is a tedious process with many opportunities for error.

The problems with this approach no doubt influenced the designers of the second generation machines. They used a different approach wherein a console terminal was used in conjunction with a monitor program (usually in read only memory) to provide the equivalent of front panel service. With such a system, a programmer could display desired information such as memory or register contents directly in octal or hexadecimal. This represented a great step forward: entry speed was increased, and the clerical task of encoding and decoding binary values was eliminated. Another great benefit of this system was that most of the monitors incorporated a bootstrap loader so that the loader did not have to be keyed in each time.

This technique has been rapidly gaining popularity at the expense of the lights and switches system, for obvious reasons. Several companies are offering such monitors encoded in read only memory boards to allow users to convert their old systems. However, this new technique still has a few disadvantages: it requires a console terminal, which adds considerably to the system cost, and once a user program has started execution the services of the monitor system are no longer available.

PAM/8 Design Goals

It was mentioned above that the front panel system is the area in which many of the differences between computer systems are found; this holds true for the Heath H8 system as well. The H8 employs a new concept in microprocessor front panels: it uses a unique combination of software and hardware to allow the emulation of a complete real time front panel system which I believe to be superior in performance to even the discrete minicomputer panel systems.

When the H8 project began, Heath engineers studied the requirements for a good front panel system closely and drew up a list of the major features to be satisfied. There were nine major requirements of a good front panel:

- The front panel system must present and accept data in a convenient octal format. Encoding and decoding binary is a job more suited to a computer than a human being.
- The front panel system must incorporate facilities to load and dump memory to and from an external device such as a cassette interface. A nearly foolproof error detection scheme must be used so that mysterious errors will not be introduced by bad loads.
- The front panel system must allow memory and register contents to be conveniently displayed and changed. In addition, data display has to be in real time. That is, if the front panel is displaying the contents of a register and the running program changes those contents, the change should be immediately visible on the panel.
- The front panel system must be capable of execution control. That is, the programmer should be able to step through a program one instruction at a time, and be able to set breakpoints within his code.
- The front panel system must provide facilities for inputting and outputting to IO ports.
- The front panel system must be easy to use, and (as much as possible) should reduce the opportunity for operator error. Whenever a front panel operation is performed, the programmer must be informed of the operation's success or failure.

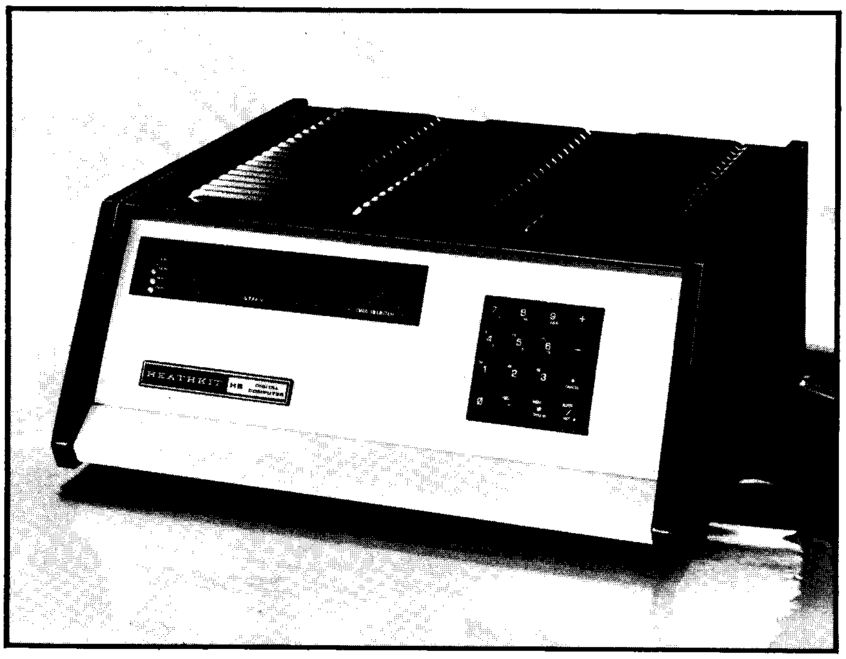

Photo 1: *Front panel of the Heath H8 computer. At left are nine 7 segment LED displays and four single LED lamps; at right is the 16 key keypad. The front panel is controlled by a novel firmware panel monitor (PAM/8) made up of both hardware and software elements.*

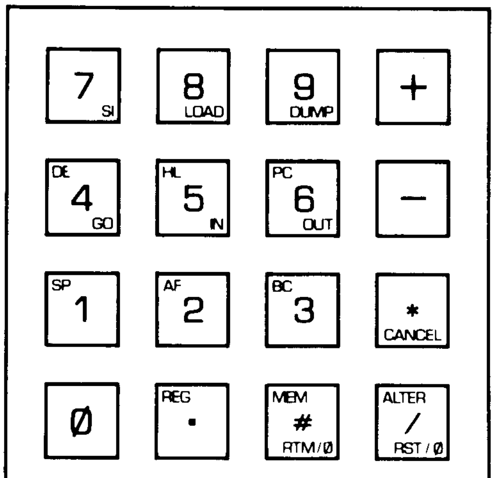

Photo 2: *H8 16 key keypad, the sole input source for the panel monitor (PAM/8). The keypad is used to enter commands and data. Some keys have more than one function, but the monitor provides an indication of which meaning will be taken for these keys.*

- The front panel system must be transparent. In other words, it must emulate a hardware panel system so that no changes are necessary to any program to allow it to be run under the PAM/8 (PAnel Monitor) system. Likewise, the front panel firmware must present a light processor load to the system so that program execution proceeds at a normal pace.
- The front panel system must be versatile. No system can be all things to all people. Some sophisticated users may have special requirements; the system must be designed to allow the sophisticated user to circumvent part or all of the system.
- The front panel system must be inexpensive. Advanced design techniques must be used to keep the cost of the panel system at or below the cost of current front panel systems.

Undoubtedly, this was a formidable list. Happily, though, Heath was able to report success with the creation of the PAM/8, the panel monitor for the H8 computer.

PAM/8 Description

The front panel of the H8 computer is shown in photo 1. Three features are immediately obvious: a 16 key keypad, nine 7 segment LED displays, and four single LED lamps.

The 16 key keypad (see photo 2) is the sole input device to the PAM/8 system. It is used for commands for PAM/8, to enter data into memory and registers, and as a bank of sense switches. Some keys have more than one function; however, no confusion results because PAM/8 provides a clear indication at all times of which meaning will be taken for such keys.

The second visible feature is the group of nine 7 segment LED displays. These are used to display addresses, data, and register names. 16 bit values are displayed in "split octal" notation. Each byte is represented as three octal digits; therefore, a 16 bit value is simply presented as two such byte groups together. Thus, in split octal notation, 377 + 001 = 001 000.

The third visible feature consists of four LED lamps (see photo 3). Three of these lamps display true hardware conditions: power on (PWR), processor running (RUN), and interrupts enabled (IE). In fact, these are the only hardware indicators in the PAM/8 system. All other displays, indicators, and keypads are under firmware control. The remaining LED, MTR, is lit when the computer is in monitor mode. Monitor mode means that the user program is not running, and the keypad is available for PAM/8 commands. When the user program is running, PAM/8 ignores most keypad commands so that the user program can use it as an input (sense switch) device.

The best way to describe the operation of the PAM/8 monitor is to go through the list of design goals again, describing how it fulfills each objective. In the process, I will touch upon some other pieces of PAM/8 hardware not visible on the front panel.

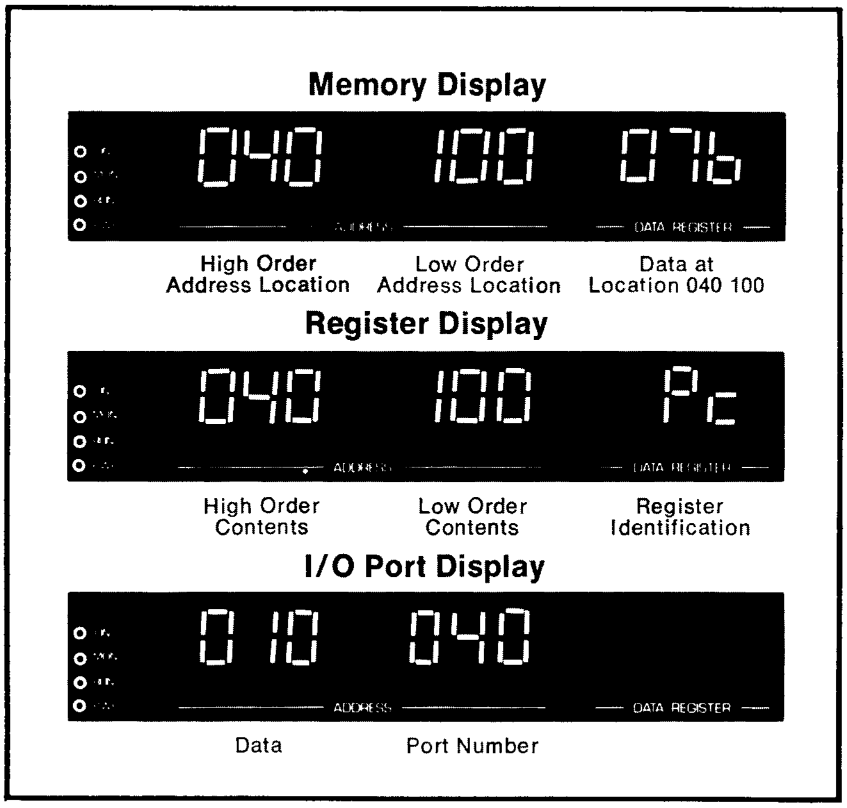

Photo 3: Three examples of the H8 LED readout format for memory display, register display and IO port display.

"The front panel system must present and accept data in a convenient octal format." This has already been discussed: PAM/8 displays and accepts octal values. 16 bit values are represented in a convenient byte octal notation.

"The front panel must incorporate facilities to both load and dump memory." The 8 and 9 keys are used for loading and dumping memory. In order to dump a block of memory to an output device (usually magnetic or paper tape), one must supply PAM/8 with the starting dump address, the ending dump address, and the entry point address. When the DUMP key is struck, PAM/8 writes a formatted record containing the memory contents. The dump record produced contains the starting address, the entry point address, and the memory data. The record is followed with a 16 bit cyclic redundancy check (CRC-16).

To reload a memory dump tape, place the tape in the transport (cassette or paper tape) and strike the LOAD (8) key. PAM-8 will read the tape and discard any information until the *beginning of record* sequence is found. The load operation then begins. When the load is complete, the computed CRC-16 is compared to the one on the tape. If the load is correct, PAM/8 gives a single beep. Since the program counter (PC) register was loaded with the entry point address, striking the GO key will begin execution. If the load is incorrect, PAM/8 displays the error code 001 (CRC error) and repeatedly beeps the horn. Pressing CANCEL (*) silences the horn.

During the load and dump operations, the six leftmost LEDs display the address being loaded or dumped, while the three remaining LEDs display the data value going into or out of that location. This allows the operator to see if the load is progressing, and gives an idea of how much is left. The H8 cassette system runs at 1200 bps, allowing the loading of 8 K BASIC in about 60 seconds.

The CRC-16 check value used in PAM/8 is nearly foolproof: single bit errors, double bit errors, and error bursts of less than 16 bits are always detected. The chance of a larger error escaping undetected is less than 0.0002%.

"The front panel system must be capable of displaying and altering both memory and registers conveniently." To display the contents of a memory location or register, strike the MEM (#) or REG (.) key followed by a 6 digit address (for MEM) or a 1 key register select (for REG). In the case of memory display, the address will appear in the left six digits, the value in the right three. In the case of a register, the value of the register (if 16 bits) or the register pair (for 8 bit registers) is displayed in the left six digits, and the register name(s) is displayed in the right two digits. See photo 3 for examples.

To change the contents of a register or memory location, first display the old contents as described above. Next strike the ALTER (/) key. You can then alter the register or location by entering six (or three) octal digits. As each 3 digit group is entered, the PAM/8 monitor provides a beep in acknowledgement. In the case of memory alteration, the memory address is automatically incremented by one. This allows you to enter a series of memory locations by entering a steady stream of values.

When the altering is complete, restriking the ALTER key clears the alter mode and restores the 0 through 7 keys to their usual function.

It is important to note that the register and memory displays are real time: if the contents of that register or location change, the display will immediately show the new value. Thus, to watch the PC register in a running program, merely select it for display and type GO. Should you now decide to watch the contents of a memory byte, press RTM/0 (# and 0 simultaneously) to halt the program, select the memory location, then press GO to resume execution from where it halted.

"The front panel system must be capable of execution control." PAM/8 provides five types of execution control:

Run
Halt
Jump
Breakpoints
Single Instruction

Pressing the GO key starts a program running at the current value of the PC register. The desired start address can be entered into the program counter beforehand. To stop a running program, press RTM (return to monitor, keys 0 and # simultaneously). Execution of the program will immediately halt, and the MTR light will come on. The operator can now examine registers and memory locations and may alter them if desired. Pressing GO causes execution to resume where it left off. To jump the processor to a section of code, press RTM, alter the PC register and press GO.

When a HLT instruction is encountered by a user program, the PAM/8 gives a single alarm beep and execution of the user program is halted. The PC register points to the byte following the HLT operation. Pressing GO causes execution to resume following the HLT opcode. The user can make use of breakpoints to debug programs by assembling or patching in HLT operations.

PAM/8 also includes a single instruction facility. Each time the SI key is struck, the instruction pointed to by the program counter is executed and the user program is immediately halted. This works for all 8080A instructions except DI (disable interrupts) including jumps, subroutine calls, and other control-transfer instructions. Holding down the SI key causes the execution of an instruction every 400 ms. It is especially instructive to display a register and use the SI key to execute instructions one by one while watching the effect these instructions have on the registers being displayed.

"The front panel system must provide facilities for communicating with IO ports." To communicate with an IO port, use the MEM key to enter the 3 digit data value and the 3 digit port address as a 6 digit memory address. Striking the OUT key causes the data value to be output to the port. Striking the IN key causes the value read from the port to be displayed in the leftmost three digits.

"The front panel system must be easy to use and should reduce the possibility for error." In order to increase convenience and minimize operator errors, PAM/8 is designed to maximize the bandwidth of the operator-machine communication channel. Thus, PAM/8 communicates in three different ways: by the digit displays, by the alarm horn, and by the display decimal points. The use of the digit displays in communication is obvious. Many panel operations, such as entering addresses and values, cause the display to change. For instance, when altering memory, the value of each key struck is shown in the displays. The front panel horn actually serves three purposes:

- Verify keystrokes.
- Provide information (such as the beep when entering byte values).
- Provide alarm indications (such as a CRC error when loading).

The PAM/8 uses the digit decimal points independently of the values on the digits themselves. As can be seen from photo 1, some keys have multiple uses. PAM/8 uses the decimal points to indicate which use of the key is currently *active*. When the REG or the MEM key is struck, PAM/8 expects an address (or register number). The decimal points are lit continuously, indicating that the address must be entered and that the keys 0 through 7 will be taken as address values. When the ALTER key is struck, PAM/8 displays a rotating pattern on the decimal points, indicating that a value must be entered, and the keys 0 through 7 will be taken as data values.

"The front panel system must be transparent." In operation, PAM/8 is totally transparent to a task program; ie: the program need not take any notice of the presence of the PAM/8 system; any existing 8080A program can run on the H8 without change (assuming it is ORGed correctly, and the IO is compatible). Since PAM/8 is implemented partially in system software, it does require processor service for operation. Normally, PAM/8 uses about 15 percent of the processor's capacity, leaving 85 percent for task programs. Most programs are compute bound for very short periods of time, and this presents no difficulties. Programs which must run at full speed can set a flag bit in the PAM/8 programmable memory area to turn off the front panel, which then gives the task program 100 percent of the processor's capacity. The task program can then reenable PAM/8 when it desires.

"The front panel system must be versatile." Although a user program need not communicate directly with PAM/8, such communication is possible. In general, there is a set of special control bytes in the PAM/8's programmable memory area which can be used to control system operation. For example, a user program can cause PAM/8 to display any arbitrary segment pattern on the LED displays. Likewise, the user program can cause PAM/8 to stop refreshing the displays so that the program can refresh them itself. In general, it is possible to totally *close down* PAM/8 operations and to have a user program take them over, thus totally replacing the PAM/8 monitor with a homebrew system. Of course, user programs can make use of the PAM/8 utility subroutines to communicate with the tape system, read the keypad (with audio feedback and auto repeat), sound the horn, and so forth.

"The front panel system must be inexpensive." PAM/8 provides powerful features at a low cost due to its firmware design. The read only memory software handles display decoding and refreshing, keypad debouncing, and all high level functions. The necessary hardware consists of the keys, the LED displays, and a few SSI and MSI logic gates. In general, the PAM/8 design costs less than a good toggle switch and lamp panel.

How It Works

As mentioned above, PAM/8 is a firmware system, meaning that its functions are implemented by a closely integrated combination of hardware and software. The hardware resides on the front panel circuit board itself, and the software resides in a 1 K read only memory on the processor board. This read only memory contains a program which does most of the work for the PAM/8 system. Actual hardware was used only when the function could not be implemented by the program.

The central concept in the PAM/8 system is its built-in clock interrupt. When the system is powered on (or master cleared) PAM/8 sends a command to the panel control port requesting an interrupt every 2 ms. This interrupt interval is derived from the system's crystal clock and is therefore called the clock interrupt. The presence of this interrupt allows PAM/8 to perform two processes, or tasks, *simultaneously*. Of course, they are not actually performed simultaneously, since the computer has only one processor, but to a being as slow as a human the operations appear simultaneous.

This division of the work load between two independent tasks, the task time and the interrupt time processes gives PAM/8 its power. For the sake of clarity, the functions of these two tasks will be discussed separately and it will be assumed that they are truly simultaneous.

Interrupt Time

The interrupt time task is always running (unless shut off by the user program) and has three main jobs:

- Process display refreshing and updating.
- Maintain system clock.
- Allow user program clock servicing.

The most important job of the interrupt time process is to refresh the front panel displays. The displays are not latched and decoded; the display hardware consists of a 4 bit digit select field and an 8 bit pattern select field. Every interrupt cycle (2 ms), a segment pattern and digit number are output by the code. The digits are refreshed round robin so that each digit is lit every 18 ms (nine digits at 2 ms each). This gives an overall refresh rate of 55 times a second, which is sufficient to eliminate flicker. The segment patterns being refreshed are obtained from a 9 byte programmable memory area. Each 8 bit byte contains the pattern for a digit (seven segments, one decimal point). Every 32 clock interrupts, or about 16 times a second, the 9 byte pattern being displayed is updated. The PAM/8 monitor examines flag locations to determine which memory location or register is being displayed and decodes its value into nine bytes of display bar code. If a register is being displayed, the program finds its value on the stack where it was pushed when the clock interrupt occurred. It should be noted that both of these processes, refreshing and updating, may be controlled by a user program. There is a bit for each function allocated in a PAM/8 control byte; setting the bit causes the function to be discontinued. Most programs which make use of this feature turn off display updating, but they leave display refreshing turned on. Then the program can display any arbitrary pattern by simply placing segment bar patterns into the 9 byte area in memory.

The second main job performed by the interrupt time task is the maintenance of the system clock. The PAM/8 monitor maintains a 16 bit count of the clock interrupts received. Since this count is updated during the clock interrupts, it appears to task time programs that the location "magically" increments itself. Many programs, including the task time portion of PAM/8, make use of this counter.

The third major job of the interrupt time task is the handling of user clock processing. Normally, PAM/8 returns directly from the clock interrupt so that the operation will be transparent to the user program. However, a user program can set a bit in a PAM/8 control byte requesting that a user subroutine be called during every clock interrupt. This allows the user to also write task time and interrupt time systems, as well as giving multitasking capability.

Task Time Task

While all this clock interrupt processing is taking place, the H8 is also running a *task time* program. Task time refers to the "problem solving" program which runs when interrupts are not being serviced. Under the PAM/8 system, the task time program may be the user program itself, or it may be the PAM/8 command processing program.

When the system is initialized after power up, the task program is the PAM/8 command processor, which continually reads the keypad for operator commands. Keypad debouncing, key strike verification (beeps) and auto repeat on the keypad are all time dependent functions; PAM/8 makes use of the system clock to implement them. When a command is recognized, it is executed immediately. Having the interrupt time task running simultaneously with the command loop greatly simplifies command processing. For example, pressing the + key (when displaying memory) is supposed to cause the next location to be displayed. All the command processor needs to do is to increment the "address being displayed" word in memory. Sometime during the next 32 clock interrupts the interrupt task will decode this new address and its contents, causing the new address and value to be "magically" displayed (after a maximum wait of 1/6 of a second). In a similar manner, the routines to handle the LOAD and DUMP functions merely update the address being displayed word after every byte is loaded or dumped; the interrupt time task sees to it that the address being loaded is continuously displayed on the panel LEDs.

After reading this discussion, you can probably guess how the GO command is implemented: the PAM/8 monitor merely restores the user registers from the stack. The PC register is restored last, which causes execution to begin at the specified location. The interrupt time task proceeds as before, decoding and displaying the selected memory or register contents. Should the location or register be altered by the running program, the front panel will very quickly (typically in 32 ms) show the change.

HLT and Return to Monitor

So far, we've seen that the interrupt time and task time processes don't intermingle; each keeps to its own. The processing of the HLT instruction and the RTM (return to monitor) command are exceptions to this principle. When a HLT instruction is encountered the processor waits with the program counter pointing to the next byte. When the next clock interrupt comes along, the interrupt processing code takes a look at the preceding instruction; if it is a HLT, the code passes control directly to the PAM/8 task time command loop, never

P.E.T.™ PRODUCTS

SOFTWARE/ACCESSORIES/HARDWARE

Memory Expansion!!-31,743 Bytes Free!-***NEECO*** now has internal memory Expansion Boards Available for your PET! 16K, 24K and 32K Memory Configurations. Call or write ***NEECO*** and ask for our ****Free**** Software and Hardware Directory. Power up to 32K Bytes! Call ***NEECO*** for more info.

Software-NEECO has too many programs to list them all here! Call or write and ask for our ****Free**** Directory! **Software Authors!-***NEECO*** offers 25% Royalties on Pet programs with nationwide distribution!--Call ***NEECO*** for additional information on our 25% Royalty Program.

PET & Peripherals-NEECO offers fast (off the shelf?) delivery schedules for the ***Pet Computer*** and ***Peripheral 2020 Printer. NEECO*** also offers excellent personal & Warranty service!!! Interested in a ***Pet***? Call and request our ***P.E.T.*** Info Pak. Feel free to call and ask questions.

The Music Box- Music Composer and sound effects generator allows you to compose and hear music on your ***Pet***! -Program & Hardware allows you to display notes, hear the notes, save pages of music on tape for later playback or modification! The ***Music Box*** actually displays the notes as a song or tune is played!-Allows you to add sound effects to your own ***Pet*** Programs.-Endless Possibilities!-All Cassette Software, plug-in Hardware, and ***Music Box*** instructions for only $49.95! ***Music Box*** Fits right inside your ***Pet*** -No assembly required! ***Music Box*** even plays random tunes!

NEECO Dust Cover- Protect your ***Pet***! Cover your ***Pet's*** delicate circuitry and keyboard from dust that can, over time, cause intermittent chip failures! Heavy, clear Plastic ***Dust Cover*** shows off your ***Pet*** while protecting it from dust, spills, and those inevitable 'Unwanted Sticky fingers'! Manufactured to last as long as your ***Pet!***--only $17.95

VISA OR MC Phone orders Accepted--(4% Surcharge on Hardware)

NEW ENGLAND ELECTRONICS CO. "Authorized PET Sales & Service"

248 Bridge Street
Springfield, Mass.

Area Code (413)
739-9626

"Guaranteed Delivery" Schedules for all of our PET Customers. Call for our PET Package.

Circle 281 on inquiry card.

returning to the user program. Naturally, a little bit of cleaning up is performed to smooth over the abrupt transition from interrupt time to task time. This feature allows the use of the HLT instruction as a breakpoint and also provides transparent support of the HLT operation. When a program halts, the front panel comes alive, and user program execution stops. Striking the GO key causes execution to resume following the HLT instruction.

The RTM command is a key command executed by pressing the 0 and # keys simultaneously. This command serves the purpose of the RUN/HALT switch on hardware front panels: striking RTM causes execution of the user program to cease, and it causes the front panel to become active. The RTM command is implemented by a joint hardware and software effort: on a hardware level, the pressing of the two keys causes a clock interrupt to be requested immediately, without waiting through the 2 ms interval. On the software level, the clock interrupt code in PAM/8 checks the keypad for the special RTM key combination. If it is present, the same process that was used for the HLT operation is used: control passes directly to the PAM/8 task time command loop, not back to the interrupted user program.

Using the PAM/8

The design of recent microcomputer systems has shown a trend away from front panel designs toward the "no front panel" monitor. This is being done for a very good reason: a terminal monitor based on programmable memory or read only memory is much easier to use and is more powerful than hardware front panels. This fact also applies to the PAM/8 system: a good console oriented monitor and debugger, such as Heath's HBUG, is much more convenient for debugging programs. This is not to say that PAM/8 does not perform an indispensable task, as I will try to show in the following real life examples.

A typical experience in the life of a computer experimenter is the debugging of some peripheral interface. I've spent many a long hour slaving over a processor, trying to make some new device or interface talk to my computer. A favorite technique I use for this is to enter a 2 statement program into memory:

```
L1      IN      <port number>
        JMP     L1
```

This program simply inputs from the port assigned to the recalcitrant device into the accumulator, then loops back to do it again and again. Then all I do is set the PC register to the L1 address, punch up the accumulator for display, and press GO. The value read from the port will be continuously displayed in the A register, even while I adjust the hardware. By watching the panel displays, I can instantly see any results of my labors, such as, "if I ground this line, will that bit come on?"

Another important use for PAM/8 is as an aid to debugging software. Often I find myself debugging a complex piece of software that maintains various state flags in memory. For example, a command completion subroutine, which examines characters as they are entered for valid syntax, is a state dependent program. As each character is entered, the program sets flag bits indicating various things such as "two alphabetic characters entered," or "have just seen a blank," etc. When debugging this code, I simply display the address (or register) containing the state flags on the front panel. Then, as I strike test keys one by one, I can immediately judge the program's reaction by examining the state flags. This technique can be used to monitor working programs as well. For example, I have a loader program which I use to download programs from other computers. It keeps the address currently being loaded in the HL register pair. By simply displaying this register pair, I can watch the load progress (or fail!).

A real time front panel can be used for more than just debugging. The presence of the displays and keypad provides another channel of communication with the processor, independent of the console terminal. The displays can be used to indicate any desired status, and the keypad can be used as a bank of "sense switches," even while the console is being used by the program. For example, the BASIC interpreter supports commands to control the displays and read the keypad.

Conclusions

The PAM/8 front panel system provides an inexpensive and effective "firmware front panel" which emulates a complete hardware front panel. Its design combines the capabilities of a true hardware panel with the flexibility of firmware and ultimately provides the user with a greater communications bandwidth to a personal computer.■