A Complete Home Computing Package

The computer hobbyist has several options when he wants a computer. He can go out and buy a complete computer system ready to be plugged in and operate. This is becoming a real option now that some companies are supplying the devices packaged. The hobbyist may also purchase a kit and build his computer. This is probably the way most of the hobbyists have obtained their systems. Actually, most of the 'kits' are little more than a bag of parts and a printed circuit board, with a diagram showing where the parts are placed. Many hobbyists derive some satisfaction from building these kits, and to some degree it is also a method in which to learn a bit about the hardware as well as construct it. The other possibility is for the hobbyist to completely design and build a computer system from scratch. There is certainly a lot more work involved in this approach, but it has several distinct advantages. The cost is generally lower than any other method to obtain a working computer. This is highly dependent on the resouces of the builder, and for some it may not be possible to throw a computer together for much less money than a kit would cost. Naturally the size of the junk box directly affects the cost. The builder/designer must have a complete understanding of his box if it is ever to become operational. For some, this may be a curse rather than a blessing, but most of us can do something if we are forced to, and if it comes down to understanding the circuitry or not using the computer, the monetary outlay for the parts can act as a great incentive to learn about the inner workings. If you build a computer then you are more likely to add modifications than if it were a kit. There is another advantage to designing and building your own computer, however - it is the unique way in which to have a piece of equipment which does exactly what <u>you</u> want it to do, and nothing else. There is no other way to have a design that does precisely what you wish. Everybody has their own ideas about what an implementation should include and what it should not. Perhaps many of the considerations are not all that important, but nevertheless it is nice to own a device which is nothing more and nothing less than what you had imagined a perfect design to be, whether that is indeed the best or not. There are many choices to

be made in a microprocessor based computer system, and surely everybody cannot have the same preferences. If we all go the kit building route, then our systems are all mostly identical, and the personal touch is lost. What I am saying is if you are considering purchasing a computer, think a little about it and see if perhaps you can build one yourself without too much trouble. Clearly everyone is not prepared to do such, but more people could do it if they just tried.

There are two aspects of the home computer - for that matter, for any computer. Obviously these are hardware and software. These can be treated independently for the most part, and if the hobbyist decides to buy a preassembled computer or a kit design, he may still be creative and write some of his own software. The same considerations go for software as for hardware. Personalized programs are a natural for the computer hobbyist. In the business community, this is almost directly opposed to what is done, and what should be done. For the individual computer operator, putting together your own software can be a rewarding experience, as much if not more so than with hardware. Most of us are not in a position to write the big programs like BASICs, text editors, and assemblers, though these are not as impossible as some of us believe. It is important to have these large programs, but it is equally important to be able to use them efficiently together. The force which binds the separate programs together is what gives a system character. Too many hobbyists neglect the operating system as unnecessary and spend their time playing Star Trek in BASIC. Writing your own operating system can be an interesting project, and offers a constructive way to learn about programming by doing. Some thought is in order before endeavoring on the task, as it is highly desirable to be able to expand your system continually without the necessity of tearing what you already have apart in order to implement your new ideas. Decide what you could eventually wish in your ultimate operating system, and construct a program which while not being the complete implementation of the ultimate system, can be upward expanded to your ultimate goal. Thinking small at this point can be disastrous, as the thought of completely discarding a current system in order to

expand it and start anew is enough to unnerve anyone.

I have designed and built an 8080 based computer. Naturally, I have included what I believe to be the necessary features, while leaving out the others. I have also written a memory based operating system to manage the programs and it includes many utility programs as well. What will be done in future articles is a complete description will be given of the hardware and software. This can serve two purposes: for those who are independent and desire to do the same thing themselves, my overall design may be of benefit to such readers. The operating system could well serve as a basis for someone who does not like mine, or who wants minor changes made. However, both the hardware and the software may be copied directly by those who simply wish to get a system together and running. For those who already have a computer, but could use my operating system, it should be no problem to interface with most any 8080 computer. The details will be forthcoming in other months, but for now I will describe operationally my implementations.

Hardware

The processor itself is an Intel 8080. Enough has been written about the 8080 to leave it at that. The machine has a Read Only Front Panel (for those who like acronyms, a ROFP). That means that you can see what is going on, but can do little about it. There are sixteen LEDs to read the address bus, eight LEDs for the data bus, and seven LEDs for monitoring control signals. In deciding upon a front panel, I see three possibilities: no lights (depend exclusively on an ASCII terminal), LEDs to indicate the signals, or seven segment displays for displaying the hexadecimal descriptions directly. There is much to say for a readoutless computer, since once a processor is running it is rare to require readouts on the computer itself. On the other side, it is difficult if not impossible to determine what is going on in a processor which is just built when it has no high level programs running, perhaps no external interfaces, and maybe without read/write memory when first powered up. If for no other reason, I included a front panel to be able to debug it when first constructed. A front panel is also an excellent manner to

explain computer operation to a newcomer. Looking at abstractions such as hexadecimal characters on a teletype seems one step removed from the computer itself, and one gains a handle on the operations of the processor much better by playing with a front panel. Therefore, I decided that I would have a front panel on my computer. When comparing hexadecimal readouts against LED readouts, the hexadecimal readouts have a definite advantage for checking programs out, as the processor instructions are given in hex anyhow, and the hex display saves you from having to learn the binary equivalents of the hex characters. The LEDs can do anything that the hex displays can, but there is one thing that the LEDs can do that the hex readouts cannot do: monitor the system while it is running. When the processor is executing a program, certainly the actual addresses, data, and control signals cannot be read, but the relative brightness of each of the individual lights can easily be seen, and usually you can see approximately where in memory the processor is by looking at the address lights. With hex displays, all that can be seen are 88s and 8888s. Anyone familiar with conventional computing systems knows that the experienced operator can tell if the system is operating properly by a glance at the address lights of the processor. Some of the newer computers have done away with the light shows, but they do serve a definite purpose by displaying varying patterns for different programs which could not be distinguished with hex readouts.

The front panel has six control switches. Main power (anyone should be able to figure out what that does), processor power, reset, interrupt, run/wait, and single step. The processor power turns off the power to everything but the volatile RAM. In normal operation, I leave the main power on continuously, and turn the processor power on when I wish to use the system. The reset switch clearly resets the 8080, and is only needed when the system crashes (crashing is a graphic term for describing what a system does when it ceases to respond in the normal manner to operator commands). The interrupt switch parallels the one interrupt input which I have included in the processor, and is useful for testing out programs which are to make use of the interrupt capabilities of the processor, and also for getting

back to a given point in a program without resetting everything. The run/wait switch is quite conventional. In normal operation the switch is in the run mode. To halt the processor the switch is placed in the wait mode. This activates the last switch, the single step. When in wait mode, the single step permits any program to be stepped through for observation on the front panel.

Note that there are no address or data switches (remember it's only a ROFP). With a complete front panel including those additional switches, it is possible to turn the power on and load bootstrap programs into memory for execution. This is an archaic method, time consuming, and generally a pain. With a low level monitor in read only memory and an ASCII terminal the front panel can go the way of the horse and buggy. Eliminating the load feature of a front panel removes a lot of circuitry for DMA or other methods used to load the memory. It also lowers the cost considerably, as toggle switches cost between a dollar and a dollar and a half each. The LEDs are very low priced.

The computer itself is built on plug in cards, as most are today. What is not conventional is the edge connector. I have used inexpensive 44 pin connectors instead of the 100 pin connectors used in the somewhat universal Altair design. The 44 pins are enough to contain all of the signals, it is much easier to build printed circuit boards with the smaller edge connections, and most of all the edge connectors themselves are much less expensive than the 100 pin connectors. I have on several occasions purchased a dozen or more for about 25 cents apiece, about twenty times less than the 100 pin connectors. The most unfortunate thing about my choice of the connector is that it makes it more difficult to utilize available boards which are compatible with the Altair bus, but this is a problem which can be circumvented by constructing a very simple board with a 44 pin edge and a 100 pin connector. It is also necessary to put a few integrated circuits on the board to make the buses compatible, but that is not difficult at all. The price savings alone for the connectors justified the bus configuration that I selected. The savings is considerable in light of the fact that the processor has 25 slots.

The power supply delivers 15 amperes of unregulated eight volts, as well as plus and minus 17 volts.

The CPU, serial interface, and parallel I/O cards are constructed on a combination of Radio Shack boards and my own patchboards. Two of the Radio Shack boards could easily be used as well. The Radio Shack cards have the 44 pin edge, and they only cost several dollars as compared to those designed for the Altair bus. These three boards are made using point to point wiring. The CPU board contains the CPU circuitry, address and data drivers, and some additional circuitry. The additional circuitry takes care of the single step, run/wait, latches an incoming interrupt until that interrupt is acknowledged, and memory wait cycles. The memory speeds may be selected independently for the ROM and the RAM by switches. The circuitry on the CPU bOard also does a ROM swap. To describe this feature, something must be said about the memory address structure. The processor has 4K of ROM, and the simplest place to put this is at address 0000. This is because on reset the 8080 goes to address 0000 to fetch instructions. In my opinion, the bottom is a terrible place to locate ROM, particularly on the 8080. Most software developed for the 8080 is assembled for the bottom, and some even makes use of the RST instructions as one byte subroutine calls. These programs would have to be reassembled to run on a system with ROM at the bottom of memory, and placing ROM at the bottom necessitates jumps from the interrupt locations. Therefore, the ROM is at the top of memory, from F000 to FFFF. Since the 8080 on reset goes to the bottom instead of F000, the circuitry on the CPU board first enables the ROM at address 0000, and later switches it to the proper place at the top of memory after the first instruction in ROM. After insuring one jump instruction in ROM, this action is completely transparent to the user.

The serial interface board contains two independent, completely programmable UARTs (universal asynchronous receiver transmitters). The input/output is voltage loop, TTL, and CMOS compatible. All normal modes of the UART (number of data bits, stop bits, and parity on/off and odd/even) are software controllable as well as the baud rate from 110 baud to 1200

(actually any eight speeds up to 9600 baud could be selected). This gives the capability of writing a program which tries different modes and speeds attempting to match the input signal. It also removes the necessity of reaching inside the computer to change speeds.

The parallel input/output board contains two eight bit general purpose input and output ports. This board also contains all of the buffers necessary to drive the front panel lights.

The rest of the boards are printed circuit cards. There is one ROM board. This holds sixteen 1702A ROMs for up to 4K of ROM. The rest of the memory is 60K of RAM. The RAM board is an 8K board which uses 64 2102 memories. There is an option on the board to allow only half of it to be filled, giving a 4K RAM board which responds correctly to addressing. The RAM boards are addressed in any of the eight possible positions in the address space by one jumper wire. There is one 4K board to mate with the 4K ROM board to completely fill the address space of the system. Yes, I did say 60K of RAM and 4K of ROM. No, I am not wealthy - only by building your own boards can such a system be built for reasonable cost. It is also necessary to scrounge parts at good prices. I was particularly lucky to purchase 512 2102s on ripped out boards. By unsoldering these chips, it was possible to afford such a large amount of memory. I daresay that my system cost far less than most systems with much less capability. The approximate cost on the computer was $500. The foil layout patterns of the RAM and ROM boards will be printed in a forthcoming installment.

That completes the hardware for the basic computer. I also have a Friden justowriter printer interfaced. The hardware for that consists of high voltage transistors and opto isolators for TTL compatibility. This interface will also be described.