THE COMPUTER JOURNAL®
For Those Who Interface, Build, and Apply Micros

Issue Number 3 Vol. 1, No. 3 $2.50

Editor's Page

We've Changed Our Name

The bad press for the word Hacker which we mentioned in October has been even worse this month. We origiṇally felt that the word Hacker, as it was used until the early part of 1983, best described what we wanted to do; but the new usage definitely is not what this magazine is all about.

We received negative feedback about our name from the very beginning, but we resisted change because we couldn't think of a more suitable title. Because of the continued bad press we decided not to fight the name any longer, and have changed to *The Computer Journal* effective with the November issue. Those of you who have been with us from the beginning have real collectors items in the September and October issues, because there won't be any more issues with that name.

We have not changed our goal of providing hands-on construction articles, tutorials, and information for those interested in advancing their use of the microcomputer. We have just changed the name in order to avoid rejection by those who use the currently popular definition of Hacker.

We appreciate your support, and encourage you to send your ideas, suggestions, criticism, and articles. We need your feedback on what information you would like to see published.

What Is The Next Microcomputer Frontier?

Many of the current microcomputer developments were initially developed by entrepreneurs working in their garage or basement, but the business climate has changed drastically since then. Big business has discovered microcomputers, and it is swamping the field with many copy-cat products, mass marketing sales techniques, and Madison Avenue hype advertising. They don't have innovative products, but they do have the megabucks to overwhelm the small business entrepreneur.

What are tomarrow's new markets? Forecasting the future is always difficult, especially when you are talking about the opportunities for an individual or small business with limited capital. **You can't beat big business at their own game!** The trend is from eight bit to sixteen bit machines, with work being done on thirty-two bit machines. Unfortunately, entering the market with wider bus machines takes a lot of money and a long lead time before the cash starts rolling in. When you start short on cash, you have to be innovative and break new ground.

I feel that the next NEW growth area for micros will be in applying them to control applications. We have micros in ovens, cars, telephones and,cameras but we have only scratched the surface. There are thousands of uses waiting to be developed by an entrepreneur who can see these applications.

The days of starting a computer company by assembling micros, (as we know them today,) at night in the garage are gone. So are the days of starting a software company using baggies. Now you need fancy (and expensive) four color packaging and full page ads to even try to break into these fields.

The only way that someone with limited capital can break into the field today is to be first with an innovative idea, and establish their market before others can react with a similar product. And react they will, because success is always copied! ■

Editor/Publisher	*Art Carlson*
Art Director	*Joan Thompson*
Technical Editor	*Lance Rose*
Technical Editor	*Phil Wells*
Production Assistant	*Judie Overbeek*

The Computer Journal® is published 12 times a year. Annual subscription is $24 in the U.S., $30 in Canada, and $39 in other countries.

Entire contents copyright © 1983 by The Computer Journal.

Postmaster: Send address changes to: The Computer Journal, P.O. Box 1697, Kalispell, MT 59903-1697.

Address all editorial, advertising and subscription inquires to: The Computer Journal, P.O. Box 1697, Kalispell, MT 59903-1697.

ADD AN 8087 TO YOUR DUAL PROCESSOR BOARD

by Lance Rose, Technical Editor

When Godbout Electronics (now Compupro) first brought out the 8085/8088 dual processor board several years ago, it was advertised as being "a bridge between two worlds." I liked the concept of the dual processor and was one of the early purchasers and users of it. (Remember when you could still get "unkits"?) The idea must have been a good one since several other manufacturers are now offering dual processor boards, some of which are 8085/8088 like the Godbout, and others which are Z80/8088.

I've been very pleased with the flexibility and performance of the board and have been successful at increasing the clock speed to 6 MHz for the 8085 and 8 MHz for the 8088, even though the board was only specified to run at 5 MHz. Apparently the Intel processors used are rated very conservatively.

One thing I have missed on the board is the option to add Intel's 8087 numeric data processor, a coprocessor which, when used with the 8086 or 8088, performs floating point computations in hardware instead of software, and which can increase program execution speeds by 2 or 3 orders of magnitude if the programs are heavily computation-oriented.

Since I deal with just that sort of program in my work, last year I began looking for a way to add this powerful coprocessor to my board. It's no criticism of Godbout that there isn't an empty socket on the board for the 8087. If you look, the dual processor board is packed with chips and of necessity has to be in order to give you two processors, their support circuitry, and circuitry to make the switchover between the two. In addition to the circuitry, there are the power-on jump, extended addressing, and the circuitry to merge DMA bus requests with those of both processors when the switchover occurs. There just isn't enough space left for another 40-pin socket on the board.

The answer to this is to add a small "piggyback" circuit board, an idea that is used quite often in electronics to add features that aren't present in the original product. This board simply plugs into the main circuit board in place of the existing 8088 processor and contains sockets for the 8088, the 8087 numeric processor, the 8288 bus controller (more about this later), and a few additional support chips. While it's possible to get all the power for the piggyback board from the main board, an additional 5-volt regulator is included to ease the burden on the main board regulators. The 8087 draws 600 ma from the power supply and when you add the 8288 and other support chips, a separate regulator is a good idea. The 8088 continues to draw its power from the main board.

Theory of Operation

A schematic for the circuit of the add-on board is shown in Figure 1. An associated parts list is shown in Figure 2. Before explaining the circuit in detail we need to have an understanding of the two different modes the 8088 can run in.

When Intel designed the 8088, they saw (with some foresight, I think) that this microprocessor could be used in a variety of applications from simple to complex. They therefore assigned one pin on the chip to be tied either high or low depending on whether the system was a simple one containing a single processor, or a complicated one with multiple processors, bus arbitration, and so forth. This pin is labeled MN/MX* (where an asterisk denotes an active low signal). In the minimum mode the 8088 takes the responsibility for generating the necessary bus signals to select memory or I/O, read and write, and a HOLD/HLDA type of DMA protocol. This is the mode used in the Godbout board and in all likelihood in other dual processor boards. It has the advantage of not requiring any additional bus controllers, and the HOLD/HLDA protocol lends itself to easy interfacing with the S-100 bus. The disadvantage is that there is no way of adding the 8087 numeric processor or 8089 I/O processor to the circuit. Both of them need additional coordination so they can share the bus with the main processor.

In maximum mode, a number of the pins on the 8088 take on a different use. Instead of generating bus signals directly, an 8288 bus controller is added to do this. This frees some of the pins on the CPU to be used to coordinate tasks with coprocessors, here the 8087. There are lines (QS0, QS1) to allow the 8087 to track the status of the instruction queue on the 8088 as well as monitor what is going on on the bus. In this way when an instruction comes along that is meant for the 8087 and not the 8088 (an ESC instruction), the 8087 will respond to it. There is a line to tell the 8088 when the 8087 is busy (TEST*) so that the CPU will not attempt to use results of a floating point operation before it has been deposited in memory by the 8087.

A different DMA protocol is used in maximum mode where a 3-pulse sequence on a bidirectional line represents (1) the initial request for a grant of the bus, (2) the granting of the bus by the CPU, and (3) the release of the bus by the requesting device. This is explained in detail in the "IAPX 86,88 User's Manual." This and the application note "Getting Started with the Numeric Data Processor" (AP-113), are available from Intel Corporation. Another good book on the subject is "The 8086 Book" by Russell Rector

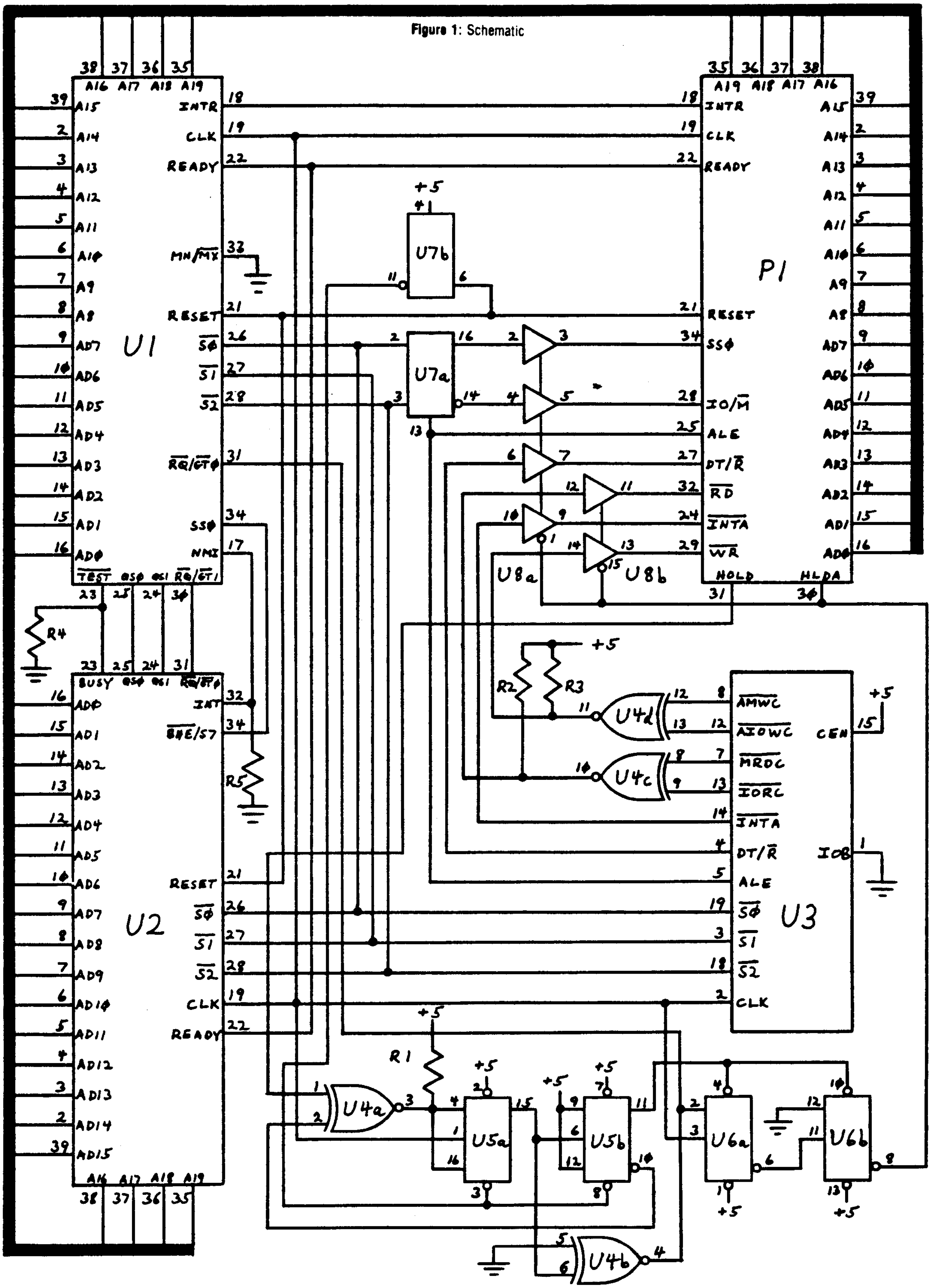


Figure 1: Schematic

and George Alexy. All of these publications contain a wealth of material on the 8086 family of microprocessors.

The purpose of the additional logic on the piggyback board is to make an 8088 running in maximum mode on the small board appear to the main board as if it were running in minimum mode. Luckily all the address and data lines are the same in both modes as well as some of the control lines so we can route these directly from the DIP header to both 40-pin sockets with no change. The most difficult part of the matching operation involves the conversion of the HOLD/HLDA sequence for minimum mode to the RQ*/GT* pulse sequence. There is a circuit in one of the books mentioned above that performs this function, and I originally used it. However the processor swap wasn't 100% successful with it, and once in a while the machine would die trying to change over from the 8085 to the 8088 or back. After designing the present circuit, I decided that the reason it hadn't worked completely was that the HLDA was being dropped too quickly after HOLD was removed. Whether this is peculiar to the conversion circuit or to the Godbout board I can't say.

U1	8088central processor
U2	8087 numeric processor
U3	8288 bus controller
U4	74LS266 quad exculsive-NOR, open collector
U5	74LS76 dual JK flip-flop
U6	74LS74 dual D flip-flop
U7	74LS75 4-bit D latch
U8	74LS367 hex tri-state buffer
R1-R3	1K ¼ watt
R4,R5	10K ¼ watt
P1	40-pin DIP header

Figure 2: Parts List for 8088/8087 Piggyback Board

In any case, the circuit here accomplishes the conversion from HOLD/HLDA to RQ*/GT* with the flip-flops U5 and U6 along with exlusive-OR gate U4a. When the board is reset, the 8088 is not on hold. Since the 8085 is supposed to get control on a power-on or reset, the HOLD line to the 8088 immediately goes high. On the falling edge of each of the next two 8088 clock pulses, U5a toggles since its inputs are high. This generates a RQ* pulse to the 8088. The 8088 RQ*/GT* line is driven by open collector gate U4b. Since the request/grant line is bidirectional, it cannot be driven by a totem-pole output and requires an open collector or tri-state driver. At the end of this pulse, U5b, which is clocked by the output of U5a, toggles its state causing the inputs of U5a to change from high to low since HOLD is still high and the second input to U4a has changed from high to low. This puts flip-flop U5a in the hold mode. At the same time this occurs, the preset inputs on U6, which kept HLDA low, are released allowing U6a to respond to a GT* level from the 8088. This level is sampled on the rising edge of the clock and may occur on the next rising edge following the falling edge which terminated the RQ* pulse.

When GT* is detected, U6a clocks the low GT* into its output. The complement output in turn clocks U6b. This causes HLDA to go high and it remains high even if U6a is clocked back low. The 8088 is now held and the 8085 has control of the dual processor board bus (and of course the S-100 bus). This state of affairs remains until the processor swap occurs. At this time, HOLD goes back to low. When this happens, the inputs to U5a again become high and the next two clocks cause toggling to occur, sending the release pulse to the 8088. This lets the 8088 know that it can begin operating. Since the output of U5a toggles U5b back to low, U6 is asynchronously preset and HLDA drops to low. This signals the processor swap circuitry on the main board that the 8088 is now in control of the bus.

Other than converting between DMA conventions, the rest of the modifications are fairly minor. The RD* signal is generated by combining the memory and I/O read lines from the 8288. Similarly, the WR* signal is derived from AMWC* and AIOWC* of the bus controller. It turns out that the "advance" outputs for these signals more closely approximate the timing of WR* in the minimum mode than the normal outputs.

The ALE and DT/R* signals are now generated by the 8288 instead of the 8088 but are otherwise unchanged. The S0* and S2* signals emulate SS0* and IO/M* except that their duration is shorter and S2* needs to be inverted. U7a latches these two signals when ALE is active and holds them, approximating the minimum mode timing. All the control signals except ALE are now routed through tri-state U8 to disconnect them from the system bus when the 8085 is in control. This is necessary since some of these which were formerly tri-stated by the 8088 in minimum mode are now generated by the 8288 and are no longer tri-stated. One part of U7b is used as an inverter with its enable input tied high. This changes the positive-going reset signal available at the DIP header into the necessary negative-going reset for the flip-flops U5 and U6.

Resistors R1, R2 and R3 are used as pull-ups for the open-collector exclusive NOR gate U4. R4 and R5 serve as pull-down resistors for two inputs of the 8088 which are normally driven by the 8087. This allows the piggyback board to function without the 8087 plugged in, for checkout or if you want to remove the numeric processor for some reason.

If you need the NMI line on the 8088 for your system's operation, you can omit the connection between that line and the 8087. The 8087 can operate without interrupts if desired. On a power-up or reset, the 8087 interrupt request line is masked so the 8088 will not necessarily be interrupted, even though the 8087 is tied to the NMI line. You have to enable interrupts in the 8087 to cause an interrupt to occur. Alternatively, a few additional gates can OR together the system bus INTR line with that from the 8087, leaving the NMI for system use. A lot depends here on how much you need the NMI for other activities.

Hardware Construction

The piggyback board is built around a small piece of perfboard attached to a 40-pin DIP header plug with a pair of standoffs (see Figure 3). The standoffs are necessarily long to keep the wire-wrap pins of the sockets from contacting the parts on the main PC board. Of course this causes a larger space requirement for the board in the card cage and you need to have additional bus slots available for the increased size. A printed circuit board would take up less space and if anyone out there is versed in PC layout and wants to go ahead and do some artwork, I would encourage it. The screw heads that attach the DIP header to the standoffs will prevent the header from being fully inserted into the socket on the main board due to their thickness unless some surgery is undertaken. I found that filing or grinding the heads down to the point where the slot just disappears (these are 6-32 binding head screws) allows the header to plug in to the socket all the way. If you don't do this the support for the piggyback board, which is pretty heavy, might be a little marginal. A lot depends on the type of socket on the main board. Some types grip the pins of the chip or header very tightly and partial insertion is adequate. With others you would probably want to get the header all the way in. You'll have to check your particular board to see.

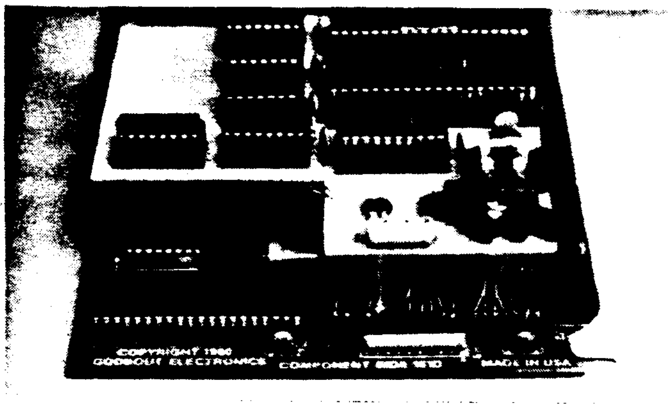


Figure 3

Note in Figure 4 that the additional regulator and its heat sink are mounted under the screw head that attaches the piggyback board to one of the spacers. Though probably unnecessary, there is a small (10 uf) electrolytic across the output of the voltage regulator. In addition there is a single .1 uf ceramic bypass capacitor for each IC package on the board. Fewer would probably do but since some of these chips have pretty healthy power requirements, it won't hurt to play it safe. There aren't that many caps involved.

To power the added 5-volt regulator, I found it easiest to add a single wire-wrap pin at the lower left-hand corner of the main board. There is a large plated area carrying 8 volts and several holes in it which are large enough to take a pin. Using the wire-wrap pin allows easy removal and re-installation of the piggyback board if necessary (see Figure 5). Regardless of whether you use a separate regulator or not, be sure to tie all grounds together and connect them to the ground pin on the DIP header.

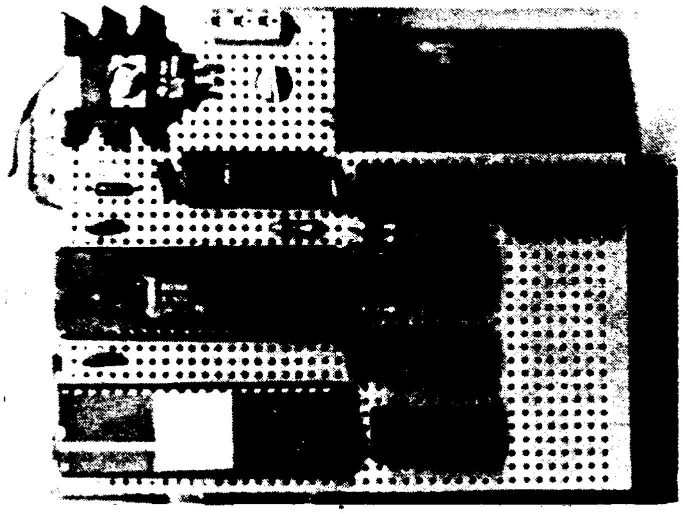


Figure 4

As far as the sockets go, board layout is pretty much up to the individual builder. A lot depends on where the piggyback board has to plug in on the main board. I built mine to go into the Godbout dual processor board, but this type of scheme should work equally well on almost any dual processor board that uses the 8088 in minimum mode. The design presented here is actually the second circuit I used (see above). The original one had several more IC packages in it than the current one, hence the perfboard shown here is larger than it actually has to be.

After you choose a suitable layout, go ahead and wire up all the sockets along with the bypass caps and resistors. I would suggest that you leave a little breathing room around the two large chips since they both generate a fair amount of heat.

When you get all the assembly done, plug the piggyback board into the main board and make sure you have enough clearance for all components with no potential short circuits. I found from my own installation that about two additional bus slots are needed for a wire-wrap version. Many of you, like myself, will have one actual bus slot for every other position in the motherboard, so will really only lose one position

Something else that might be helpful is also visible in

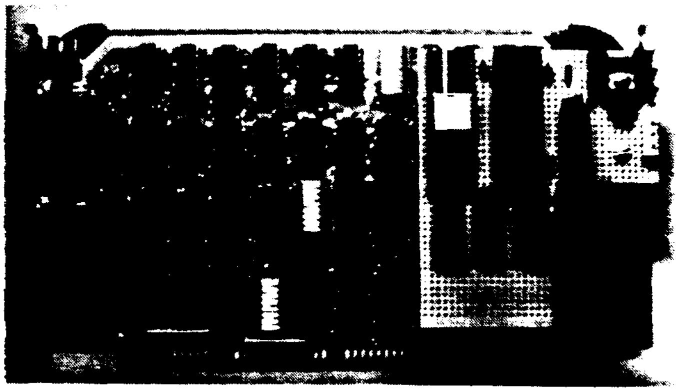


Figure 5

Figure 6. For trying out different crystals, I have replaced the soldered-in crystals with small individual Molex-type pins. This allows easy crystal changing when trying to see how fast you can get your board to operate.

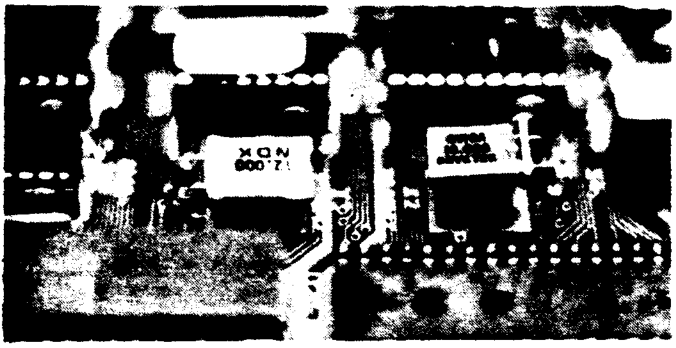


Figure 6

Checkout

Once you have everything wired together, you're ready to check out the circuit. Start by just testing the regulator to make sure its output is close to 5 volts. I don't think I have to say that when you're dealing with a $200chip (the 8087), you can't be too careful. If the regulator is OK, turn off the power and insert all the TTL chips along with the 8088 and 8288. Leave the 8087 out for now. Plug in the board and power up. Quickly check the regulator for 5 volts under load. If it's off spec, turn off the power and find out why. Check for shorts or a bad regulator. Once you get the power supply cleared up, go ahead and try switching over to the 8088 processor. If all goes well, it should run just as before except that it will not respond to NMI (non-maskable interrupt). This shouldn't be a problem as very few of us normally use that interrupt for anything. It's designed for power fail service and things like that. If your 8088 performs as it should, then (after drawing a deep breath) insert the 8087 into its socket on the board.

As an aside here, I should tell you that the 8087s sold up to the present time have been rated at only 5 MHz so you will have to keep the clock rate down. If you're adventurous, after you get the thing running OK at 5 MHz, you can experiment with faster crystals. Unofficial (stress that) word from Intel is that you won't damage the chip by trying this. I myself have tried clock speeds up to 7.4 MHz with no damage. The fastest I can get the 5 MHz version to operate is 6.67 MHz and I have settled on 6 MHz. At this speed it runs fine. If you're already running at 8 MHz and don't want to slow down, Intel has in the works an 8 MHz version of the 8087 that should be available sometime this winter. Price hasn't been announced yet but will obviously be more than the 5 MHz version. Incidentally, lest the price of even the latter put you off, you have to realize that the power contained in one of these little packages is more than that of most minicomputers and even some mainframes when it comes to floating point computations.

To get back to the checkout, power up again and check the 5 volts one more time. If everything is up to snuff, you're ready to try using the numeric processor.

Software

As is usual in this game, the software lags the hardware. Although high-level language compilers that take advantage of the 8087 have been available from Intel for some time, these are expensive packages designed to run only on Intel's hardware. For the S-100 people, the only readily-available way to take advantage of this chip has been to program in assembly language. Digital Research includes a macro library with its CP/M-86 operating system that has the extended instruction set of the 8087 in it. I have used this so far in my own work and found it to be adequate but no more than adequate. Writing scientific or engineering applications software in assembly language is not really the easiest solution. Programs tend to be long, unclear as to function, and hard to debug. They do run quite fast, however, so at least there are some rewards. Also, the library provided by DR does not totally conform to the descriptions provided by Intel in its publications, and some editing is required if you want the floating point mnemonics to function as the Intel book says they should.

Luckily, there is some evidence that a number of compilers should be on the market soon that will use the power of the 8087. I have already seen advertisements for C, PASCAL and FORTRAN compilers that incorporate the 8087. Some other companies are selling runtime libraries to retrofit current compilers with this capability. To date I have no first-hand experience with any of these so I would advise buying carefully.

I think this situation will improve as more IBM PCs are sold (which already have a socket for the 8087 to be plugged into). Unfortunately for non-IBMers, most of the software will be written for the MS-DOS (PC-DOS) operating system. Digital research, however, has recently announced an MS-DOS emulator which they say will allow programs written for MS-DOS to run under CP/M-86 with only a slight speed penalty. DR is also reputed to be working on a FORTRAN compiler of their own, which I assume would be available for CP/M-86 as well as MS-DOS. I'm all in favor of assembly language for systems programming, writing BIOSs and the like, but for number-crunching applications I'll take FORTRAN. (Yes, Virginia, there actually is somebody who likes it!)

Summary

For hard core number crunchers, this chip is a whiz. The 5 MHz version is readily available now, with faster versions on the way. Other companies have announced math processor chips but have not yet managed to get them into production, although samples of some are becoming available. While I haven't gone into the individual operations of the 8087 in detail, suffice it to say that besides performing the four basic math operations on both fixed and floating-point data types, it also has built-in hardware for calculating log, exponential and trig and inverse trig functions. There are also a number of built-in floating- point constants such as 0, 1 pi and ln(10). When this hardware finally meets some high-level software that can take advantage of it, we should see microcomputers finally come into their own in science and engineering. ■

BUILD AN A/D CONVERTER FOR THE APPLE][

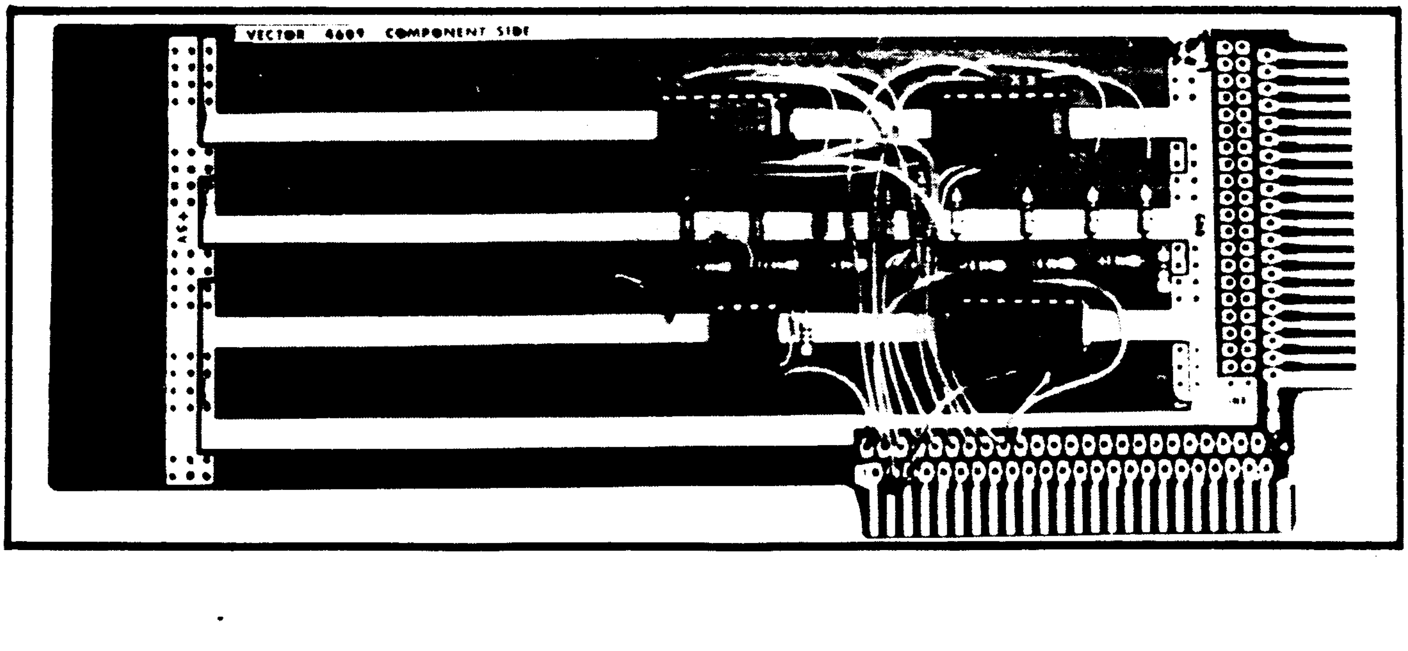
by Tom Lawson

Whether a computer's input comes from a keyboard, a disk drive or another computer, the information is received in the form of ones and zeros. The binary system is ideally suited for electronic arithmetic and logical calculations because each digit can have only two states; one or zero. An electronic circuit can distinguish a one from a zero by answering one question, "Is the voltage above the threshold?": if so, the logic level is one; if not, the logic level is zero. In the "real" world choices are not often so clearly defined. There is an infinity of gradations between black and white or hot and cold. Translating such continuously variable quantities into the binary language of computers involves making a series of comparisons. Devices for this purpose are called analog-to-digital converters.

The more comparisons that are made, the more precisely the analog voltage can be known. One approach is to use a series of electronic comparators each with a higher threshold than the last. One comparator can detect two states, two can detect three, etc. Bar graph drivers, commonly used in audio equipment for signal level indicators, are of this type. These A/D converters are only practical if fewer than ten or twenty gradations are to be resolved. The circuit complexity and power consumption would get out of hand if hundreds of levels were to be distinguished.

A second method uses the unknown input voltage to charge a capacitor. A single comparator then indicates when the capacitor has charged to a particular voltage. The time required to charge the capacitor is inversely proportional to the unknown input voltage; in other words, the greater the voltage, the faster the capacitor will charge. A digital counter keeps track of the charging time to complete the conversion. This is called single slope A/D conversion. Dual slope and triple slope converters are refinements of this basic approach. These types of A/Ds have excellent high frequency noise rejection and since computers generate lots of high frequency interference, they are a good choice when the most accuracy per dollar is the prime consideration. These converters are intrinsically slow. A converter capable of resolving one part in 128 generally cannot run faster than 1000 samples per second. Greater precision requires longer conversion times. A/D converters of this type will be discussed in more detail in these pages in the future.

The third approach involves first completing a digital-to-analog conversion and then comparing the unknown voltage to the digitally synthesized known one. The advantages of this scheme are simplicity and speed. Single-chip A/D converters are available that contain a DAC (digital-to-analog converter) and a comparator. Some also offer microcomputer buss compatibility. I have elected not to use any of those parts for this project because they obscure the principles involved. A completely satisfactory A/D converter able to resolve one part in 256 can be made from four inexpensive integrated circuits (ICs) plus a few resistors and capacitors. The converter could be operated through a serial or a parallel port, but for simplicity I built this one to sit directly on the buss of an Apple II® computer. The circuit can be modified to sit on any microprocessor's buss with, at worst, the addition of some address decoding and/or a 6.5 to 15 volt power supply. The microprocessor needs to be able to read a single-bit input at one address and to write an 8-bit word to another. Everything in the circuit is available from Jameco Electronics for under $5. You will also need a prototyping board that will plug into an expansion slot in



your computer. I used a Vector part #4609 for $22.95 from Jameco that will accomodate several small projects.

The DAC must somehow weight the binary bits to construct a voltage. Resistors having values of R, 2R, 4R, 8R, 16R, 32R, 64R and 128R could be used for the purpose, but trying to achieve that sequence of values with the necessary precision out of available parts is messy. Instead, we will use something called a R/2R network. Figure 1 shows a 2-bit R/2R network.

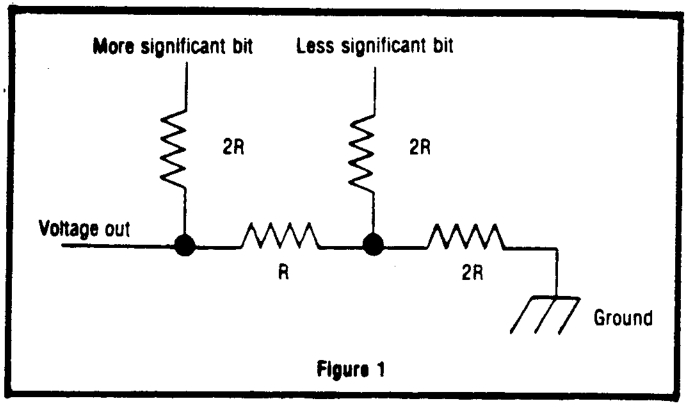


Figure 1

Assume the two binary bits are at either 5 volts or 0 volts. If the digital word is 10, then the rightmost two resistors both connect to 0 volts. That is equivalent to one resistor to ground of value R (for detailed information on this, refer to Thevenin equivalent in an engineering text). Figure 2 shows the network redrawn in simplified form.

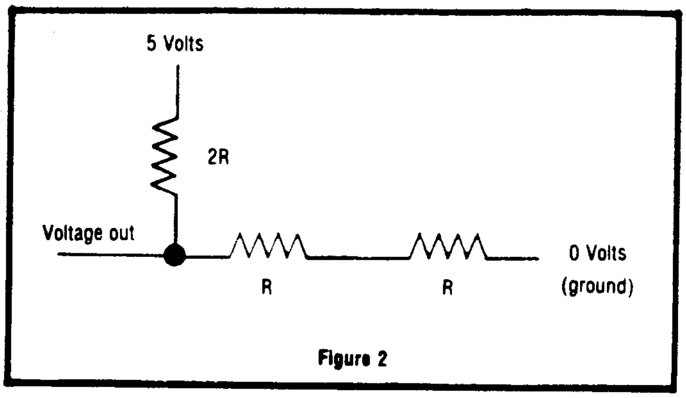


Figure 2

Because R + R = 2R the voltage output will be ½ V or 2.5 volts. Figure 3 shows the digital code 01.

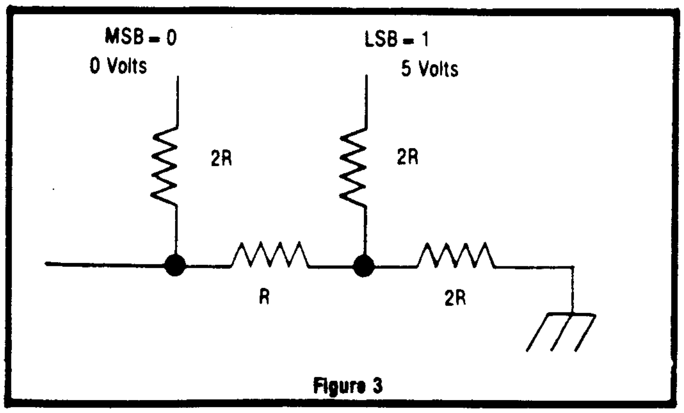


Figure 3

The right two resistors are equivalent to one resistor of value R connected to 2.5 volts. The simplified equivalent is shown in Figure 4.

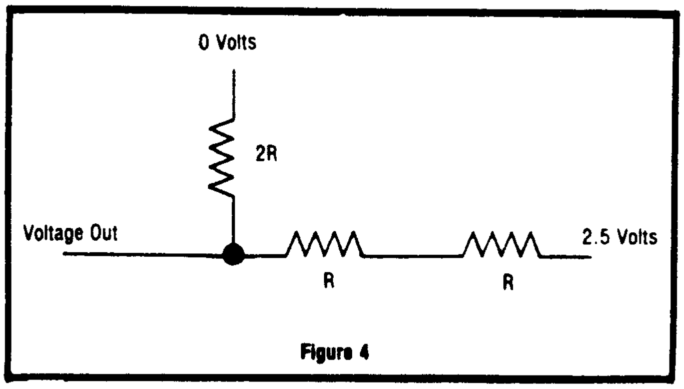


Figure 4

The voltage out will now be 2.5 / 2 volts or ¼ V. To show that the bits are additive consider the digital code 11. Figure 5 illustrates 11 in simplified form.

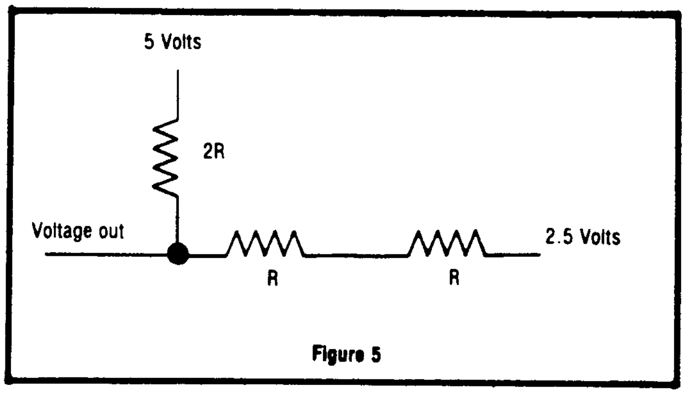


Figure 5

The voltage out will be halfway between 5 volts and 2.5 volts or ¾ V. As the digital code in binary counts from 0 to 3, the voltage out steps through the sequence 0, 1.25, 2.5, 3.75 volts. Mathematically, the voltage output equals the digital code divided by the number of codes, times the reference voltage. Finer resolution is obtained by adding more bits. Each time another bit is added on, it has the weight of half the preceding bit. In this way binary words of any length can be weighted using only two values of resistors. The more significant bit resistors must, however, be held to close tolerances in order for all the increments to be in proportion. In an 8-bit DAC the most significant bit resistors should be accurate to ½%, the next bit's resistors should be good to 1%, etc. Unless you have access to close tolerance resistors, the more significant resistors should be hand-selected. More precise values can almost always be selected from a population. Parallel and/or series combinations and trimpots are OK if necessary, but they were what the R/2R network was supposed to eliminate in the first place. If any resistor is too far from its nominal value, the DAC's output voltage will decrease at some point when the digital word is increased. This is called non-monotonicity.

CMOS outputs, unlike TTL, swing all the way from one supply rail to the other. Using a CMOS latch to store the digital word allows the latch outputs to connect directly to

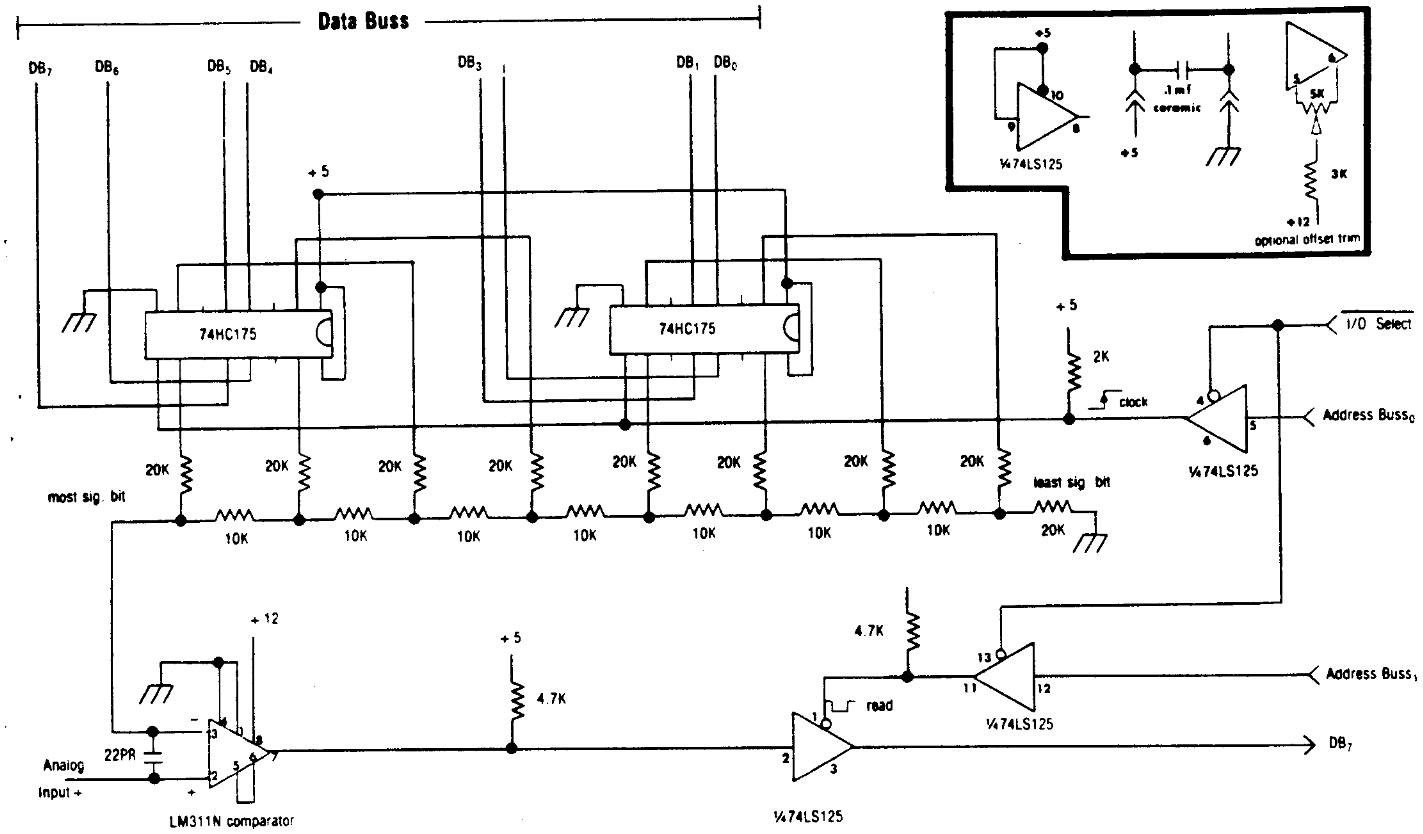


Figure 6: Schematic

the R/2R network. A 74HC series part was chosen for speed. We used two 74HC175 quadruple latches but the 74HC273 octal latch should be available soon and can be substituted.

A LM311 comparator has the DAC output connected to its inverting input and the unknown input voltage on its non-inverting input. If the unknown voltage is larger than the DAC output voltage then the comparator's output will be 5 volts; otherwise the output will be 0. The 22pf capacitor is for calming the comparator's tendency to oscillate near the crossing point. The comparator output is read by the computer through one quarter of a 74LS125 quad tri-state buffer. Two extra sections of the 74LS125 have been used for address decoding. Using tri-state with pull-up or pull-down resistors for gating produces less than ideal waveforms but they should be adequate for the purpose unless your clock is faster than one megahertz. If you want, use a 74LS138 for address decoding. The 74LS138 could also help if you want to use the rest of the card space for an unrelated project.

As shown, for the Apple, if the lower two bits of the address in the peripheral slot's I/O address space are 10 then the comparator's output will appear on data buss 7. A POKE or store to the card with the lower bits 01 will write the data into the latches. For Apple expansion slot #7 the addresses are 49394 ($C0F2) for the latch and 49393 ($C0F1) for the comparator. Subtract 16 from these addresses for each slot you move to the left. The examples that follow all assume the A/D is in Apple slot #7.

a The positive supply for the LM311 is shown as 12 volts. Actually anything from 5 to 36 volts will do. If 5 volts is used, the analog input range will be restricted to 0-3.8 volts instead of 0 to 5 volts.

The five volt supply is used as the voltage reference in the circuit. That is the limiting factor for accuracy. If you want to get maximum stability, you may want to run the 74HC175s from a more closely regulated 5 volt source such as a 5.1 volt zener diode. The 74HC175s don't use much current, so the reference wouldn't need to be fancy.

Because CMOS may self-destruct if the voltage at any of it's pins exceeds the supply rails, it is good practice to double-check the supply connections with an ohmmeter before turning on the power to your circuit. To test the circuit attach a voltmeter to the DAC output. POKING 0 into the latch should produce a voltage near zero. POKE 255 into the latch and the voltage out should be equal to the actual 5 volt supply voltage times 255/256. Check your DAC for monotonicity with a program like this.

```
10 FOR N = 0 to 255: POKE 49394,N
20 FOR P = 0 TO 200: NEXT P
30 NEXT N
```

Line 20 is there to give you a little time to see what's happening. If the voltage proceeds smoothly from 0 to 5 volts you have achieved monotonicity. If the voltage ever reverses direction either you've crossed two wires or the resistor ratios aren't accurate enough. Once the DAC is working connect a 1.5 volt battery to the A/D's input and change line 20 to read as follows.

```
20 IF PEEK(49393) < 128 THEN PRINT N: END
```

PEEK(49393) will be less than 128 only if the DAC output exceeds the analog input. As soon as the DAC voltage is greater than the input voltage the comparator output goes to 0 and the program prints the answer N and stops. The number N/256 is the fraction of the 5 volt supply that is equivalent to the analog input. To translate N into a voltage, multiply it times 5/256. You can calibrate your converter by substituting the actual voltage for the "5" in the above equation.

You may have observed that the above program is a little on the slow side. It also runs much faster at lower voltages. We can speed it up and equalize conversion time by using a technique called successive approximation. If we first set the DAC at 2.5 volts and then check the comparator, half the possible values have been eliminated with one comparison. Next we set the DAC to the center of the remaining range of possible values, either 1.25 or 3.75, and check the comparator again. By continuing in this fashion, only eight comparisons are needed to complete the conversion. You may have seen this technique used in sorting routines where it is called a Binary Search.

```
10 N = 128: A = 64
20 POKE 49394,N
30 IF PEEK (49393) < 127 THEN N = N + A
40 IF PEEK (49393) > 128 THEN N = N - A
50 A = A / 2
60 IF A > .5 THEN GOTO 20
70 PRINT N
```

If you have an oscilloscope you will want to spend some time looking at the DAC waveform generated by a successive approximation program. Synchronize the sweep with the start of the conversion. Then vary the input voltage and watch how the DAC closes in on the answer.

Any number of enhancements can be added to the hardware. An input amplifier can be used for low level signals or a non-inverting voltage follower can be added to provide a buffered DAC output. LF351 amplifiers are a good choice if you have dual supplies available. Use sections of an LM324 quad amplifier for single supply operation. Input multiplexing will allow multi-channel operation. The zero point may be off by as much as 3 counts because of the input offset voltage of the comparator. That offset may be trimmed out by adding a 5K ohm potentiometer between pins 5 and 6 with the wiper connected through a 3K ohm resistor to the plus supply. The circuit as drawn should have scatter of less than ½ count. I do not, however, recommend adding extra bits without at least an improved 5 volt reference and probably power supply decoupling as well.

A substantial speed improvement can be attained by using machine language driving routines to run the A/D converter. Here is a simple program written in 6502 machine code that works like the first BASIC listing.

```
300    A2 00       LDX    #00
302    8E F2 C0    STX    $C0F2
305    2C F1 C0    BIT    $C0F1
308    10 03       BPL    $030D
30A    E8          INX
30B    D0 F5       BNE    $0302
30D    86 08       STA    $08
30F    60          RTS
```

The X register is zeroed and the DAC ($C0F2) is set to zero. Then the comparator is checked using the BIT instruction. In line 308, if the comparator has switched, the branch is taken and the result is stored in zero page location 8. Then the routine ends. If the comparator hasn't switched yet, then the X register is incremented and the larger number is stored in the DAC. If the input voltage is overrange, then eventually the INX instruction will cause the X register to wrap around to zero again. Then 0 is stored in location 8 just as if that were the correct answer and the routine ends; otherwise, the program would get stuck in the loop whenever the input voltage was overrange.

The code is shown in the usually safe $300 space (for Apple II) but it can be placed anywhere in memory. The BASIC line

```
CALL 768: PRINT PEEK(8)
```

will perform a conversion and print the result. The above routine is short and runs reasonably fast but has the same disadvantages as the equivalent BASIC routine. A successive approximation machine language program called FAST SAX is listed in Figure 7. Most machine language programs are a compromise between speed and compactness. This is a fast program without loops. We'll look at a looped version shortly, but consider the iterative program first because the logic is easier to follow.

The exclusive or (EOR) function was used to perform the addition and subtraction instead of the more explicit functions. This was done because the EOR operation can serve both to add and subtract; also, EOR is not affected by the carry bit so time is saved by not having to set or clear the carry before each addition or subtraction. The answer is developed in the accumulator. The first trial answer, 128, is stored in the DAC. Line 305 checks the comparator. If the trial answer was too small the comparator will be up and the program will branch to 30E. There, after a NOP (no operation) to equalize the timing of the two pathes, 64 is added to the trial answer. If the comparator was down back at line 305, the trial answer was too large and therefore the most significant bit should be dropped and the next less significant bit should be raised. EORing with 192 in line 30A accomplishes both. The branch in line 30C is always taken and the accumulator will hold 64, the next trial value. In line 311 the pathes converge. The next trial answer, either 192 or 64, is stored in the DAC. The comparator is checked again in line 314 and the process repeats itself testing the next-less-significant bit. After all 8 bits have been tested, the answer is stored in location 8 and the routine ends.

The elapsed time for one conversion will be 130 clock cycles. If your system runs faster, the A/D hardware should be able to go at least four times that fast. In an Apple II 256 data points can be gathered at the rate of 7,400 samples/second. In order to run the converter at full speed, the data must be loaded into consecutive locations in the memory. This can be done by adding a few lines at the end of the routine to store the answers using indirect indexed addressing. If you want to preserve the constant sample time be sure to keep the branch sequence just before the RTS intact. If coaxing the last ounce of speed out of the A/D converter isn't important to you, a separate routine can be used to fetch the answer from location 8 and store it somewhere else in the memory.

```
Ø300-    A9 8Ø          LDA   #$8Ø
Ø302-    8D F2 CØ       STA   $CØF2
Ø305-    2C F1 CØ       BIT   $CØF1
Ø308-    3Ø Ø4          BMI   $Ø3ØE
Ø30A-    49 CØ          EOR   #$CØ
Ø3ØC-    DØ Ø3          BNE   $Ø311
Ø3ØE-    EA             NOP
Ø3ØF-    49 4Ø          EOR   #$4Ø
Ø311-    8D F2 CØ       STA   $CØF2
Ø314-    2C F1 CØ       BIT   $CØF1
Ø317-    3Ø Ø4          BMI   $Ø31D
Ø319-    49 6Ø          EOR   #$6Ø
Ø31B-    DØ Ø3          BNE   $Ø32Ø
Ø31D-    EA             NOP
Ø31E-    49 2Ø          EOR   #$2Ø
Ø32Ø-    8D F2 CØ       STA   $CØF2
Ø323-    2C F1 CØ       BIT   $CØF1
Ø326-    3Ø Ø4          BMI   $Ø32C
Ø328-    49 3Ø          EOR   #$3Ø
Ø32A-    DØ Ø3          BNE   $Ø32F
Ø32C-    EA             NOP
Ø32D-    49 1Ø          EOR   #$1Ø
Ø32F-    8D F2 CØ       STA   $CØF2
Ø332-    2C F1 CØ       BIT   $CØF1
Ø335-    3Ø Ø4          BMI   $Ø33B
Ø337-    49 18          EOR   #$18
Ø339-    DØ Ø3          BNE   $Ø33E
Ø33B-    EA             NOP
Ø33C-    49 Ø8          EOR   #$Ø8
Ø33E-    8D F2 CØ       STA   $CØF2
Ø341-    2C F1 CØ       BIT   $CØF1
Ø344-    3Ø Ø4          BMI   $Ø34A
Ø346-    49 ØC          EOR   #$ØC
Ø348-    DØ Ø3          BNE   $Ø34D
Ø34A-    EA             NOP
Ø34B-    49 Ø4          EOR   #$Ø4
Ø34D-    8D F2 CØ       STA   $CØF2
Ø35Ø-    2C F1 CØ       BIT   $CØF1
Ø353-    3Ø Ø4          BMI   $Ø359
Ø355-    49 Ø6          EOR   #$Ø6
Ø357-    DØ Ø3          BNE   $Ø35C
Ø359-    EA             NOP
Ø35A-    49 Ø2          EOR   #$Ø2
Ø35C-    8D F2 CØ       STA   $CØF2
Ø35F-    2C F1 CØ       BIT   $CØF1
Ø362-    3Ø Ø4          BMI   $Ø368
Ø364-    49 Ø3          EOR   #$Ø3
Ø366-    DØ Ø3          BNE   $Ø36B
Ø368-    EA             NOP
Ø369-    49 Ø1          EOR   #$Ø1
Ø36B-    8D F2 CØ       STA   $CØF2
Ø36E-    2C F1 CØ       BIT   $CØF1
Ø371-    3Ø Ø6          BMI   $Ø379
Ø373-    49 Ø1          EOR   #$Ø1
Ø375-    EA             NOP
Ø376-    85 Ø8          STA   $Ø8
Ø378-    6Ø             RTS
Ø379-    3Ø FB          BMI   $Ø376
Ø37B-    FF             ???
Ø37C-    FF             ???
```

Figure 7: FAST SAX

Figure 8 is a listing of a shorter but slower successive approximation routine. It requires 301 clock cycles per conversion but occupies less than ⅓ the memory space.

```
Ø3ØØ-    A2 8Ø          LDX   #$8Ø
Ø3Ø2-    AØ CØ          LDY   #$CØ
Ø3Ø4-    8A             TXA
Ø3Ø5-    8D F2 CØ       STA   $CØF2
Ø3Ø8-    85 Ø8          STA   $Ø8
Ø3ØA-    8A             TXA
Ø3ØB-    4A             LSR
Ø3ØC-    AA             TAX
Ø3ØD-    2C F1 CØ       BIT   $CØF1
Ø31Ø-    3Ø ØØ          BMI   $Ø312
Ø312-    3Ø Ø1          BMI   $Ø315
Ø314-    98             TYA
Ø315-    45 Ø8          EOR   $Ø8
Ø317-    8D F2 CØ       STA   $CØF2
Ø31A-    85 Ø8          STA   $Ø8
Ø31C-    98             TYA
Ø31D-    4A             LSR
Ø31E-    A8             TAY
Ø31F-    DØ E9          BNE   $Ø3ØA
Ø321-    6Ø             RTS
```

Figure 8: Successive Approximation

The answer is constructed in location 8, the X register keeps track of the bit that is being tested. The contents of the Y register are the same as the X register with the next-less-significant bit also raised. EORing the trial answer with the X register is equivalent to an addition and EORing with the Y register simulates a subtraction. In line 305 the DAC is loaded with 128. Then the X register is shifted one place to the right. This has the effect of dividing by two. The comparator is checked in line 30D and if the result is minus (comparator output of 1), the developing answer is EORed with the X register. If the comparator was zero then the answer is EORed with the Y register. The new trial answer is placed in the DAC in line 317 and in location 8 in line 31A. The contents of the Y register are then shifted right and the program branches back to test the next bit. When all the bits have been checked the the routine ends with the answer stored in location 8.

Notice the repeated "branch if minus" instructions in lines 310 and 312. They illustrate a technique for equalizing the timing of two branches of code. When the elapsed time difference between branches is an even number of clock cycles, then the times can be matched by adding NOPs to the shorter path. When the time difference is one clock cycle then ½ a NOP is needed but is not available. The first BMI instruction has a displacement of zero. It behaves like a three-cycle NOP if the branch is taken and like a two-cycle NOP if it is not. One of the paths consists of two branches taken, for an elapsed time of six cycles. The other path is two branches not taken and a TYA which also adds up to six cycles.

Whichever machine language or BASIC driving routine you settle upon, the A/D converter is ready to be put to work measuring voltages. Almost all sensors for quantifying temperat ure, pressure, light, velocity, position, etc. produce voltage outputs. If you work out an application that you find particularly appealing, perhaps other *Computer Journal* readers would like to hear about it. ■

ASCII REFERENCE CHART

Dec	Hex	ASCII	Remarks
000	000	NUL	Nul,tape feed. Control shift P.
001	001	SOH	Start of heading[SOM,start of message]. Control A.
002	002	STX	Start of text[EOA,end of address]. Control B.
003	003	ETX	End of text [EOM,end of message]. Control C.
004	004	EOT	End of transmission, shuts off TWX machines and disconnects some data sets. Control D.
005	005	ENQ	Enquiry [WRU,''Who are you?'']. Triggers identification (''Here is...'') at remote station if so equipped. Control E.
006	006	ACK	Acknowledge [RU,''Are you...?'']. Control F.
007	007	BEL	Rings the bell. Control G.
008	008	BS	Backspace. Control H.
009	009	HT	Horizontal tab. Control I.
010	00A	LF	Line feed. Control J.
011	00B	VT	Vertical tab. Control K.
012	00C	FF	Form feed to top of next page. Control L.
013	00D	CR	Carriage return to beginning of line. Control M.
014	00E	SO	Shift out; change character set or change ribbon color to red. Control N.
015	00F	SI	Shift in; return to standard character set or color. Control O.
016	010	DLE	Data link escape [DCO]. Control P.
017	011	DC1	Device control 1, turns transmitter (reader) on. Control Q (X ON).
018	012	DC2	Device control 2, turns punch or auxillary on. Control R (TAPE, AUX ON).
019	013	DC3	Device control 3, turns transmitter (reader) off. Control S (X OFF).
020	014	DC4	Device control 4 (stop), turns punch or auxillary off. Control T (TAPE, AUX OFF).
021	015	NAK	Negative acknowledge [ERR,error] . Control U.
022	016	SYN	Synchronous idle [SYNC]. Control V.
023	017	ETB	End of transmission block [LEM, logical end of medium]. Control W.
024	018	CAN	Cancel [S_0]. Control X.
025	019	EM	End of medium [S_1]. Control Y.
026	01A	SUB	Substitute [S_2]. Control Z.
027	01B	ESC	Escape prefix [S_3]. Control shift K.
028	01C	FS	File separator [S_4]. Control shift L.
029	01D	GS	Group separator [S_5]. Control shift M.
030	01E	RS	Record Separator [S_6]. Control Shift N.
031	01F	US	Unit Separator [S_7]. Control Shift O.

Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII
032	020	space	064	040	@	096	060	‘
033	021	!	065	041	A	097	061	a
034	022	”	066	042	B	098	062	b
035	023	#	067	043	C	099	063	c
036	024	$	068	044	D	100	064	d
037	025	%	069	045	E	101	065	e
038	026	&	070	046	F	102	066	f
039	027	’	071	047	G	103	067	g
040	028	(	072	048	H	104	068	h
041	029	)	073	049	I	105	069	i
042	02A	*	074	04A	J	106	06A	j
043	02B	+	075	04B	K	107	06B	k
044	02C	,	076	04C	L	108	06C	l
045	02D	-	077	04D	M	109	06D	m
046	02E	.	078	04E	N	110	06E	n
047	02F	/	079	04F	O	111	06F	o
048	030	0	080	050	P	112	070	p
049	031	1	081	051	Q	113	071	q
050	032	2	082	052	R	114	072	r
051	033	3	083	053	S	115	073	s
052	034	4	084	054	T	116	074	t
053	035	5	085	055	U	117	075	u
054	036	6	086	056	V	118	076	v
055	037	7	087	057	W	119	077	w
056	038	8	088	058	X	120	078	x
057	039	9	089	059	Y	121	079	y
058	03A	:	090	05A	Z	122	07A	z
059	03B	;	091	05B	[	123	07B	{
060	03C	<	092	05C	\	124	07C	|
061	03D	=	093	05D	]	125	07D	}
062	03E	>	094	05E	^	126	07E	~
063	03F	?	095	05F	_	127	07F	Delete

MODEMS FOR MICROS

by E.G. Brooner

Introduction

Before you say "Why should I care about modems?" let's remember that at one time we didn't have personal computers, and a little bit later we wondered why anyone would ever need a disk drive, or 64K of memory. Some of us have recently felt the same way about modems, but if you can't stop progress you have to go along with it. Believe me, modems are here to stay. It is no longer a question of whether or not to have one, or even whether or not you can afford it, but rather, where to find one that will work. The options are becoming so numerous that choosing a modem can be downright confusing.

You've probably been told a number of times that "MODEM" is just a contraction of "modulator-demodulator." If your background is not in communications, you might have inquired around and found out that, where computers are concerned, that just means that the digital bits can't travel over phone lines and have to be converted to audio tones to make the trip, then converted back to "bits" again at the other end. So much for a definition of a modem. It lets computers (and other digital devices) communicate over long distances.

You already know that there's a limit to how far apart you can put two devices that are connected by a 40-connector ribbon cable; even RS-232 (more about that later) is only intended for use over a distance of 50 feet or so. So if any two devices are going to be connected at any distance at all, something has to be the go-between. That's where modems get into the act.

Here's the big picture: your terminal or computer wants to talk to another one across town, or even, perhaps, across country or state. Your equipment has to do so over the telephone line, so you connect a magic box (the modem) between your equipment and the phone line. The other party does the same. The modems are the "go-betweens" that translate bits into audible beeps, and vice versa. Ergo, the computers think they are connected directly to one another (see Figure 1).

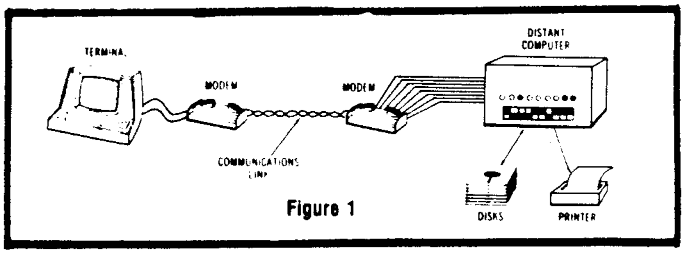


Figure 1

A Review of Features

The most difficult thing about adding a modem to your system is deciding which one to buy, and why. A quick review of the features and what they mean might help.

•**Baud rate, or data speed:** this usually expresses the approximate number of bits per second. Standard rates are established and the ones hackers most often see are 300 baud and 1200 baud. There are others in commercial use. You may run your peripherals at faster or slower rates, or even at these same standard modem rates, but the speed enters into the design of the modem itself. Most of those currently available to us, and of any use to us, run at one of these fixed rates.

We refer a lot to the 300 and 1200 baud data speed as being the practical alternatives. I have talked to commercial users who have made 2400 baud work over a limited distance, such as within a city, and have heard of 4800 being used. But generally speaking, rates over 1200 baud (and sometimes even 1200 baud) require you to contract with the phone company for some kind of special line. It may be a DC path, or a frequency compensated circuit, or some such; be assured that if you have such a need and are willing to pay for it the phone company can probably accomodate you. But, we are talking about hacking, not banking or commodity trading...

Faster is not always better; for many reasons, usually poor phone service, 1200 baud is pushing things a bit. Then, too, a lot of the services you might want to use (more about that later, too) were established years ago when 300 baud was the only practical way to do it. And last but certainly not least, 1200 baud modems are still, and will probably remain, more expensive than the little 300 baud units most hobbyists have.

For reasons best left to engineers, the type of modulation used in 300 and 1200 baud modems differs greatly, one from the other; when you see a modem advertised as handling both speeds, as many of the better ones do, you can be sure that there are really two modems in the same box!

In general 300 baud modems make use of two pairs of frequencies, and the modulation is accomplished by Frequency Shift Keying, a form of FM (frequency modulation) in which one frequency represents a mark and the other a space. There is one pair of such frequencies for transmission in one direction, and the other set is for data travelling in the other direction.

Modems operating at 1200 baud and higher use fewer tones (frequencies) and modulate them in other ways - such as by shifting the phase of a single tone one way for a mark and the other way for a space.

•**Bell-compatible:** Most (but not all) of the data communication standards were developed by Bell Labs and are known as "Bell standards;" i.e: the common 300 baud standard is described as "Bell 103" (see Figure 2). Any Bell

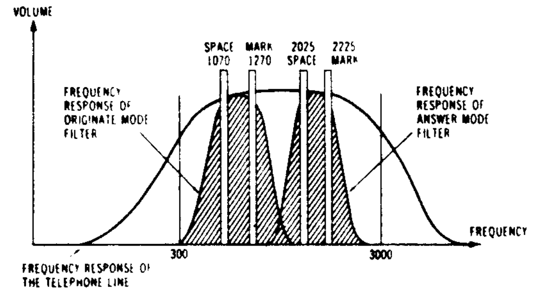


Figure 2: Bell 103 frequency bandpass

103 modem should work with any other regardless of who manufactured it.

When we get to the higher speeds we find that there are some 1200 baud standards, for example, that differ from one another and are not necessarily compatible with each other. At least one manufacturer makes 1200 baud modems that can only be used with others of the exact same kind because of the frequencies that are used. Even the more "standard" 1200 baud standards are not all alike; one might be half duplex and another full duplex. Some caution is advised when geting into this class of equipment.

•**Acoustic and Direct-connect**: In the "old days" the phone company wouldn't let us connect anything directly to their lines without paying dearly to lease their interface devices. Now, you can connect any "FCC approved" device yourself. When we couldn't tamper with the lines the only answer was "acoustic coupling." This meant physically sticking the telephone receiver into a pair of muffs built into the modem, where communication took place via the earpiece and mike (see Figure 3). This is now considered old-fashioned but it still has advantages. It's quick and easy, and it doesn't change your phone installation in any way. It works great if (a) you have a good quality handset, and (b) you aren't in a really noisy place, like the engine room in a boiler plant. If you plan on using a portable computer an acoustic modem is almost a necessity (such as, for use in a public phone booth.) On the other hand many of the new handsets are of a shape or size that do not permit them to fit into an acoustic modem. The "Star Trek" communicator type is one such example.

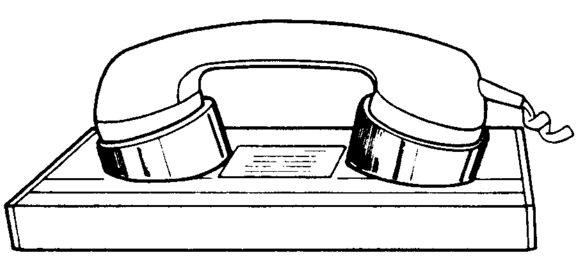


Figure 3: Acoustic Modem

Direct connect modems avoid the above mentioned hassle, but have a few of their own. First of all, most of them plug into a modular jack - which you probably are using but may not be, especially if you bootlegged the extension in your computer room. Your telephone set, then, usually plugs into the modem, where there is another modular jack for the purpose. There may or may not be some way to rapidly switch from "talk" to "data" use.

•**Originate-Answer**: just to prevent mass confusion one end of any circuit has to "originate" and the other "answer." This is an aspect of the modem design itself. For example, in a 300 baud modem there are 4 audio tones present. One pair handles transmission in one direction, the other pair handles it in the other direction. The "answer" end can send data and the "originate" end can receive; it's just an arbitrary agreement about who will use which pair of frequencies. The importance is this: most commercial services are automatically set up to be the answer device, which means that if you dial in to one of them, you have to be the originator.

Some of the older modems were "originate only" devices. Beware. Better ones have a switch letting you be either "end" of the circuit. If you want to swap files with someone across town, then, you decide in advance which one of you will be which end. (Unless of course, one of you is "originate only" in which case the other had better have the ability to swing either way.)

•**Full or Half-duplex**: some confusion is possible here. As applied to modems used with computers, though, the meaning refers to whether or not your transmissions are "echoed back" from the other end. There's more to it than that, but be aware that some devices you might want to connect to want it one way and some the other. Your modem will no doubt have a switch that lets you go either way. As we mentioned earlier some of the 1200 baud modems are half-duplex only, but you probably won't have one of those.

•**Physical configuration**: Do you want your modem to be part of your computer? For Apples and S-100 machines, some manufacturers can supply a plug-in board. For most everyone else (and sometimes just for convenience) most modems are separate units that sit on the table or shelf. In at least one case, the modem is built into the telephone instrument itself, and some of the newer ones are so small that they can actually be mounted somewhere inside your computer.

Most all modems require you to have an RS-232 I/O connection on your machine which you aren't using for something else, such as your terminal. The plug-in board kind avoid this complication. There was (and may still be) one that plugged into the TRS-80 model I, using the expansion slot rather than a port. But, in most cases, there will be a DB25 connector on the modem and it will expect to see something similar on your equipment. As is always true in the real world only part of the RS-232 standard will be fully used in this application, but keep in mind that a lot of the normally "un-used" RS-232 lines were developed for telephone use and your modem may use them for some reason, even if only to light a few LEDs. Earlier issues of *The Computer Hacker* have gone into some detail on the RS-232 standard.

You will probably only have to worry about signals (pins)

1,2 3 and 7 for the simpler modem connections. Pins 2 and 3 carry the transmitted and received data; depending on how your port is wired these may or may not have to be reversed. You can figure this out by referring to your own documentation and the aforementioned RS-232 articles.

So far we haven't mentioned power, because most modems have an AC power supply of their own, or in the case of those that plug into your bus, draw power from the computer. There is a variation described as LP, or line-powered. These modems, when you can find one, are smaller, lighter and cheaper than the more common kind. They draw power from the telephone line itself. They work if your phone line can provide at least 16 milliamperes of current on a reliable basis. Worth trying if you have a choice.

Another possible variation is the so-called LD (line-driver) or "short haul" modem. These are not really modems, but merely amplifiers that transmit the digital pulses directly without any kind of modulation. They require a direct line between the devices; in general they are cheap, work at relatively high speed, and are of little practical use to hackers. They are widely used, however, for industrial purposes. One use you might find for a pair of these is for extending your RS-232 line several thousand feet, rather than staying within the normal 50 foot limitation.

•**Bells and whistles:** auto-answer and auto-dial do just what the terms imply. This is sometimes a modem feature, but it also may depend on the "communication software" you are using. The auto-dial feature on a brand X modem might require that you also use the software brand X has written to go with it. Some kind of monitor feature is a nice touch, too, such as a little loudspeaker that lets you hear the phone line so that you know how things are progressing. As modems proliferate you shouldn't be surprised at any new feature that comes along. Even the simplest and cheapest ones can put you in the communication business—there are both VW Beetles and Caddies in every field of endeavor.

Additional Considerations

The point of all of the preceding is that not all modems are alike. Now, what is all of this going to cost? You can find modems advertised for as low as $50—usually reconditioned, or close-outs of old models. These are usually 300 baud, acoustic coupled, sometimes originate-only. New 1200 baud or 300/1200 combinations have only recently dropped below $1000. The "average" hacker's modem, if there is such a thing, will be either an acoustic or DC type (the price difference is small) and run at 300 baud only. It should at least be a combination originate/answer type, capable of either half or full duplex operation, and will probably cost between $150 and $300.

What do you do with it? I use mine, primarily, for swapping software with people with whom, for reasons like system incompatibility or distance, I cannot directly swap diskettes. I have used it in the past to communicate with information services and bulletin boards. Unfortunately, if you don't have any of these located nearby the long distance phone charges can be shocking. This alone would justify going to 1200 baud, provided that the services to be used can manage 1200 baud, and that the phone lines were always that reliable.

There's another good use for modems, too. This article is being prepared on a S-100 microcomputer. However, *The Computer Journal* works from an Apple][, which is then interfaced to a typesetter. Using a pair of modems and the telephone, I can transfer a text file containing this article to the *Journal's* microcomputer. They can then use the same or even a different word processor to edit the text and embed commands for typesetting. Without modems, I would have to print the article out and get the hard copy to the *Journal*, where someone would then have to re-enter all the text on a different computer, or directly into the typesetter. This would be more time-consuming, harder work, and would increase the chance for errors.

What kind of modem should you buy? If you have a mass-produced machine, the kind you buy in computer stores, you'd be wise to pick something made to go with it. For example, Novation makes a line of CAT modems that anyone can use; a couple of their models have been picked up and re-labelled by the mass merchandisers. If you have a "matched pair" of anything, you can be sure it will work in some fairly predictable manner.

If you use the mongrel assortment of components most hackers use you have more choice and, also, more "ifs" to consider.

Software

You need software, too. Again, the mass-producers can (usually) provide something that matches their components. If what you have is not in that class you might have to do some software-shopping or generate your own. Software is really the whole key to computer communication. The modem is just one more piece of hardware that has to be fitted in somewhere. Buying, borrowing, or building a modem is only the beginning, but it's one you have to make before you can connect your equipment to the rest of the world.

Communication software is a subject unto itself. Past and future issues will, hopefully, clarify this very intriguing subject for interested readers. Software of this kind comes in many flavors; you might wish, for example, to make your micro emulate a terminal so as to operate someone else's computer. These emulation programs are used to communicate with remote information services or databases. Or, you might want to transfer files back and forth. Such transfers require slightly different programs. October's *Computer Hacker* described a set of "file transfer" programs that have proved to be very successful—I use them myself. Some fancy software packages permit you to perform both of these functions as well as others you might find desirable.

All of these communication software packages are in addition to what usually comes with a computer. Some of them are separate programs which have to be loaded and run, and some can be patched into your "normal" operating

Continued on page 18.

THE CP/M OPERATING SYSTEM

by M. Mosher

For those of us not yet blessed (or cursed) with owning one or more floppy disk drives, the choice of an operating system has not really been much of a problem. With a cassette system, there are few enough different things you can do with it to make it practical for each manufacturer to choose his own standards as to how to format the data on the tape, what size blocks to use, what type of encoding, and so on. The cassette interface is usually very hardware-dependent and this routine is as much a necessity as it is a choice. Moving up to a floppy complicates the situation considerably, and that is where the operating system comes in.

For openers we might ask "What is an operating system anyway, and why do we need one?" To understand the answer to this question we need to explore a little bit into how data is stored on a floppy diskette. Contrary to a tape cassette where data is stored sequentially, and to find a particular program you need to read through (or fast forward through) all the preceding programs on the tape, a diskette is a "random access" device, i.e. any part of the diskette can be read from or written to about as fast as any other part. An analog to this would be a phonograph record where you could set the tonearm down and play any cut on the side about as quickly as any other. An audio cassette is just the opposite; you have to skip over a lot of songs if you want to play one in the middle of the tape.

To take full advantage of this random access capability, each diskette has on it a directory. This is simply a reserved area on the diskette where the information is stored which tells where each file is located on the diskette. (A file can contain a program in any one of several languages, or data or text—the actual content doesn't really matter.) When we want access to a particular file, we simply give the name of the file to the operating system and it then searches the diskette directory to see if the file is there, and if so, allows us access to it.

This is where the problems begin. Since there is a need to set aside a certain part of the diskette for the directory, the question arises of which part to reserve for this. Many 8-inch disk systems use part of track #2 for the directory. Radio Shack, on its TRS-80 Model I, uses part of track #17. Some other minifloppy systems use track #3. An associated problem is where on the diskette to store the operating system itself, since it must be loaded into memory each time the power is turned on. Once again, different manufacturers have choses different places on the diskette to store this program. Yet a third problem is exactly what information should be placed in the disk directory. The most obvious would be the exact loaction of the file on the diskette and the file size. However some systems also store the date of last access and other such information. Clearly a diskette that has been recorded under one operating system could not, in general, be read by another system even if we could get around a multitude of other problems such as different size disks, different types of diskette sectoring, and so on.

CP/M was designed as an answer to most of these problems. With the CP/M operating system, versions of which are available for almost any microcomputer sold today, programs written on one computer can be run on another totally different machine. Most programs written in BASIC, FORTRAN, assembly language and others have a need to get at certain files on the disk. If each machine has a diskette structure that is different, the disk access part of every program must be modified each time it is to be run on another computer.

The idea behind CP/M is that it isolates the user from the details of the diskette structure and allows him to use a standard series of operating system "calls". These calls are actually subroutines that input or output characters to or from the system console or printer, or to or from the disk system. The calls are the same in every CP/M system no matter what machine it is running on. This lets a program written on, say, an S-100 microcomputer be transferred to a TRS-80 that has CP/M and run immediately with no modification (at least in theory). The same program could later be transferred to an Apple that had CP/M and also run with no changes. Those of you that have seen some programs you would like to have but that were written for the TRS-80 while you have an Apple should realize the advantage of CP/M.

CP/M accomplishes this neat trick by dividing the operating system up into three pieces, each of which is a part of the total operating system. The first is called the Console Command Processor (CCP). This program interacts with the system console, issues prompts, inputs commands for execution and so on. It is identical for all versions of CP/M since it uses the same prompts, etc. for all systems. (I should mention here that CP/M is limited to 8080/Z80 systems.) The second part of CP/M is called the Basic Disk Operating System (BDOS). This is the part of CP/M to which "calls" are made to open files, close files, input or output data, and so forth. It is also the same for all CP/M systems because the calls are always the same and because the same information is stored in the disk directory in all versions of CP/M. The third and last part of the system is called the Basic I/O System (BIOS). This is the part of CP/M that actually interfaces the software to the hardware, namely the disk controller and drive. This part of the system is different for each machine and disk controller. It must be supplied by the manufacturer of the hardware of someone else who understands the hardware well enough to write this portion of CP/M for the chosen computer. The BIOS isolates the user (and the other ⅔ of CP/M) from the actual hardware.

By now you ought to be able to see some of the advantages of CP/M. For example, a program such as the one for file transfer found in the October issue of *The Computer Hacker* could be transferred to another computer using modems. The same program could then be run on another computer using CP/M. The program was written on an S-100 microcomputer with a fixed hard disk, but can be used on a TRS-80, North Star Horizon, Apple II, Osborne, Kaypro, Morrow, or any other computer which has been set up to use CP/M. With CP/M all computers become equal to a large dgree and the number of programs available to users of all these machines increases, simply because if someone writes a program is usable on any other micro that has CP/M availale for it, which includes almost every other micro made today.

For those of you interested in 16-bit computing as I am, there is even a CP/M-86 available for the 8086/8088 16-bit microprocessors. This CP/M-86 retains the exact same file structure as the CP/M-80 (8-bit CP/M). This allows access to the same data files on the same diskettes from either 8-bit or 16-bit software. Once again, the advantages of this are pretty obvious. CP/M-86's chief competitor, Microsoft's MS-DOS (or PC-DOS if used on the IBM PC) can't make this claim. This is largely why I have adopted CP/M-86 as my 16-bit operating system. Chances are when the time comes we may see a CP/M-32 appear.

Well, by now you should be convinced of the merits of some kind of a standard operating system. If you don't already have CP/M for your computer, you should run right out and get it. Then we could all use the same programs and life would be much easier. The only snag, if you want to call it that, is the price. Most manufacturers sell CP/M for around $150 but you can sometimes do better from discount mail-order houses. This may seem like a lot, considering the operating system you have came with the disk system at "no extra cost". But you have to realize that part of what you paid for the disk hardware went to pay off the guy who wrote the operating system software as well.

The microcomputer industry right now is characterized by dozens of different hardware systems and few standards. CP/M comes as close to a software standard as you can get at the present time. Shouldn't we all be using it? ■

MODEMS, continued from page 16.

system. Many of them require some modification to fit your particular use, an action which may involve a little bit of machine language programming or alternatively, switching some jumpers on your I/O port.

A final suggestion: find someone who presently has a modem and is using it. Try his (or her) system, paying for the phone calls if you have to. In other words, see it work and then decide whether or not a modem of some kind is a worthwhile addition to your system. ■

FURTHER READING

For more information on using your computer to transfer data, see **Computer Communication Techniques** *by E.G. Brooner and Phil Wells. The book is published by Howard W. Sams & Co., and is available from:*

Group Technology, Ltd.
P.O. Box 87
Check, VA 24072

The cost is $15.95 plus $1.00 for shipping and handling.

Help

This space is for you. We encourage you to communicate with other readers through this column by asking for their help with your problems, and by writing in with your solutions to questions like "Where can I...?" or "How can I...?" As a forum for sharing hands-on experience, this section can be an important resource for you. We will try to keep the lead time short for a rapid exchange of information. Let us hear from you!

BUILD A HARDWARE PRINT SPOOLER

Part Two: Construction

by Lance Rose, Technical Editor

Last issue, after an overview of the spooling process, I set out the rationale behind the hardware print spooler design. This month I'll go through the actual circuit, explaining theory of operation, and also describe the construction process for those of you who want to add this piece of hardware to your system.

The electrical schematic for the spooler is shown in Figure 1. An associated parts list is also included in Figure 2. The parts required are for the most part easily available even for those of you who, like me, live away from the metropolitan centers where electronic components are in plentiful supply. Everything here can be ordered from one of the reputable mail-order supply houses such as Jameco or purchased at your local Radio Shack store.

Let's look at the circuit broken down into logical blocks.

Power Supply

The power supply for the spooler is pretty standard except for the transformer configuration. T1, which must supply the total current for all three voltages is a 12.6 volt center-tapped unit with a 1-amp rating. The ends of its windings directly feed the rectifier that produces approximately 8 volts unregulated which is then regulated down to 5 volts by VR1. T2 and T3, when combined with T1, effectively emulate a single transformer with multiple secondary taps. Since T2 and T3 need only supply a small amount of current for the ± 12 volt supplies which power the RS232 line driver, they can be small. The regulators for these two supplies are simply Zener diodes which carry a no-load current of approximately 50 milliamps each. LED D4 provides a pilot light to show when the unit is turned on.

A word here about Radio Shack transformers. While often they are the only easily-available choice, I wouldn't recommend them from past experience. Too often the windings are loose and buzz annoyingly and the current ratings are very optimistic. If you have a source for Stancor or other name-brand transformers, by all means use them.

I haven't shown all the bypass capacitors specifically. You should scatter them around the circuit depending on your past experience(?) for deciding how many. Very conservative designers use a .1 uf ceramic disk per chip but I think that is a bit of overkill. I generally use one bypass cap for every 2 ICs but you can stretch this a bit further in the RAM chip area here since, being CMOS, they have a very low power drain and don't create heavy switching transients the way TTL does. See Figure 3 for a photo showing the board layout and location of bypass caps.

Let me add a short comment here about wiring the AC plug and line fuse. Since the switch is in the hot side of the line only, be sure to use a 3-prong plug here and ground the chassis to the power line ground for safety. Also be sure the switch goes in the proper leg of the line, namely the one with the narrower blade or, if they are the same for a 3-prong plug, the one on the right as you look at the wall ready to plug the thing in. Don't omit the fuse. A lot of manufacturers of low-current devices seem to think that a small transformer can serve as a fuse and will open up in case of a short but I don't subscribe to that school of thought. Be safe and use a fuse!

Clock Circuit and Dividers

The clock is built out of two gates of a hex inverter with a crystal and the necessary resistors and capacitor. At this frequency (4 MHz) a 74LS04 will be fine. For much higher frequency crystal oscillators a 7404 works better. The third gate buffers the clock output to prevent loading from killing the oscillation. Part of flip-flop U10 is used as a divide-by-2 counter to provide the Z80 with a 2-Mhz clock signal. This allows the use of the most inexpensive version of the Z80 and is still plenty fast enough to handle the spooling process.

The different baud rates are generated by a pair of 74LS193 counters, U13 and U14. U13 takes as its input the 4 Mhz clock signal and, using a divide-by-13 process outputs a 308 kHz signal. Since the UART must have a clock that is 16x the baud rate, this allows for use at 19,200 baud as the maximum communications rate. U14 further divides the signal to obtain successively lower frequencies for baud rates down to 1200. Since most printers can accept data at 1200 baud with some form of handshaking, it shouldn't be necessary to go down any lower. If for some reason your printer can't handle 1200 baud or higher, you can add one additional 74LS193 wired with the same configuration as U14 with its input from pin 7 of U14. This would allow baud rates as low as 75 baud. The version of the spooler shown in the schematic has its input wired to 19,200 baud and its output to 1200 baud. If you anticipate changing baud rates frequently you might want to wire in a DIP switch instead of wrapping directly to the pins.

Reset Circiuts

During power-on, a reset pulse of approximately 100 msec is generated by R2 and C1. Two inverter gates provide the proper polarity of reset to the UART (positive-going) and the Z80 and flip-flop (negative going). In the case of a buffer overflow, the processor will be in a halt state and must be reset to accept further input. You can do this via S2. When halted for an overflow, part of U7 along with red

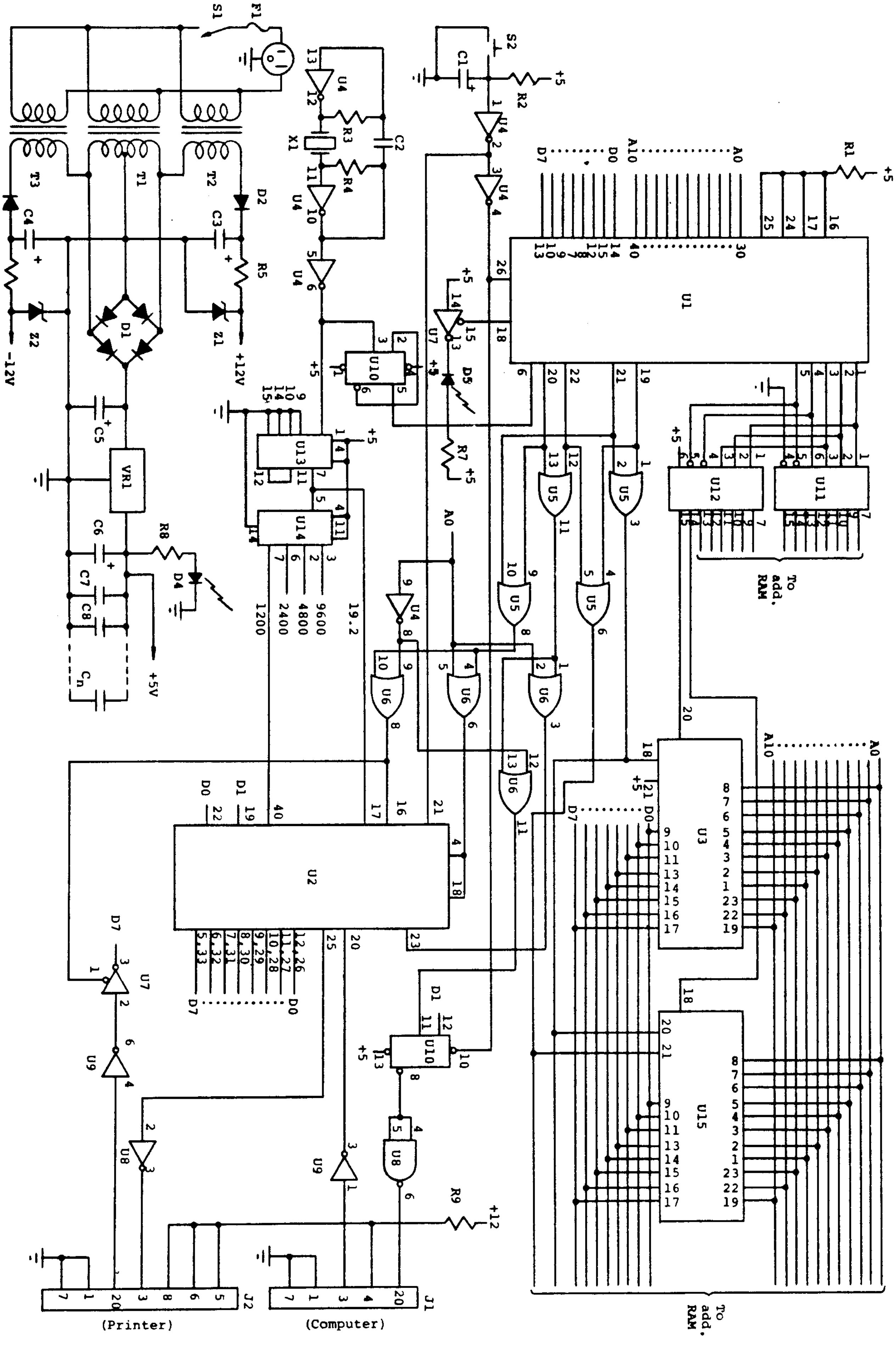

S1
F1
T1
T2
T3
D1
D2
D3
C3
C4
C5
C6
C7
C8
Cn
R5
R6
R8
Z1
Z2
VR1
D4
+12V
-12V
+5V
U4
X1
R3
R4
C2
S2
C1
R2
U7
D5
R7
U10
U13
U14
9600
4800
2400
1200
19.2
U1
R1
A10
A0
D7
D0
U5
U6
U11
U12
To add. RAM
U2
U3
U15
U8
U9
R9
+12
+5
J1
J2
(Printer)
(Computer)

Figure 1

LED D5 provide a visual overflow indicator.

Memory

Address decoders U11 and U12 provide the chip select signals to the individual memory chips. For more than 32K RAM, more 74LS138 decoders can be added. If you do this be sure that you select the proper inputs of the decoder for A14 and A15. The 74LS138 provides two negative and one positive enable inputs. Also note that the chip select is on a different pin for the 2716 than it is for all the 6116 RAM chips. The data lines and low address lines are wired in parallel for all memory ICs.

Control Circuitry

The individual signals for read and write and for memory select versus I/O select are all derived from U5 and U6. The configuration chosen may seem illogical as to choice of bits for flag signals but I set the circuit up to emulate the 8251 as far as bits used for data ready, output ready, DTR and so forth. It doesn't require any additional hardware and helps me remember what's what since I have 8251s in my main system.

U1	Z80 Microprocessor
U2	AY-5-1013 UART
U3	2716 EPROM
U4	74LS04 Hex Inverter
U5,U6	74LS32 Quad OR gate
U7	74LS368 Hex buffer
U8	1488 EIA Line Driver
U9	1489 EIA Line Receiver
U10	74LS74 Dual D flip-flop
U11,U12	74LS138 3-to-8 Decoder
U13,U14	74LS193 Binary counter
U15-Un	6116 2K × 8 RAM
R1	1K ¼ watt
R2	10K ¼ watt
R3,R4	470 ¼ watt
R5,R6	1.5K ¼ watt
R7,R8	220 ¼ watt
R9	1K ½ watt
C1	10 uf 35V electrolytic
C2	.01 uf ceramic disk
C3,C4	1000 uf 35V electrolytic
C5	4700 uf 35V electrolytic
C6	3.3 uf 25V Tantalum electrolytic
C7-Cn	.1 uf ceramic disk
D1	1A 100 PIV bridge
D2,D3	1A epoxy rectifier
D4	LED-green
D5	LED-red
Z1,Z2	12V 1W Zener
VR1	7805 voltage regulator
X1	4 MHz crystal
S1	SPST 120V toggle switch
S^2	SPST momentary close
T1	12.6VCT 1A transformer
T2,T3	6.3V 300 ma transformer
F1	¼ amp fast blow fuse
J1,J2	DB25S connector

Figure 2: Parts List

The UART outputs can be tri-stated so the status flags are connected across the data bus and turned on by the proper control signals. Part of U7 provides a DTR signal for the computer to test and is turned on at the same time as the UART status flags. The other half of U10 is used as a 1-bit register to latch a DTR signal to the computer to tell it to stop sending when the buffer is full. Although there are a number of handshaking conventions for this purpose, the DTR method seems to be the most widely used at the present time. (For a good discussion of serial interface conventions see "The RS-232-C Serial Interface" by Phil Wells in Issue #1 of *The Computer Hacker*.) Don't forget to wire up both the inputs and outputs of the UART to the data bus since they are internally separate.

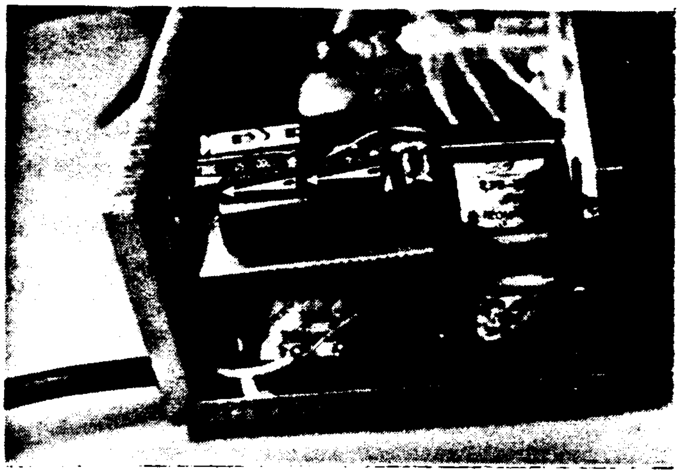
Serial I/O and Interface

The serial input and output to and from the UART are passed through the usual 1488 and 1489 EIA bus driver and receiver ICs U8 and U9. The DTR signal also uses another section of the same chips. Pin 4 (RTS) is held high to the computer and pins 5,6 and 8 to the printer likewise. Pin 20 is the only line on which handshaking occurs.

Construction

The basis of any construction project like this, unless you build it on an open board, is a good choice of chassis box. There is a universal law that states that whatever size box you choose it will be just a little bit too small to cram everything into that you want. This law perpetuates itself because most people don't want a huge box sitting around with just a few parts in it and would rather do a little shoehorning to avoid wasting space. The problem is exacerbated by the poor choice of chassis sizes available to us out in the wide open spaces. The spooler I built is based on a Radio Shack box #270-229. As you can see from some of the figures, a 32K version doesn't leave an extra cubic millimeter at the end. I think I would recommend using a somewhat larger box if you can get it.

Begin construction by drilling all the necessary holes for mounting the power switch, fuse, transformers, line cord, DB25S connectors and indicator LEDs. A nibbling tool comes in handy here but if you don't have one some careful drilling will do the job followed by some file work. Although not visible in the figures, the 7805 regulator is mounted on the rear panel behind the transformers and uses the chassis for a heat sink. See Figure 4 for a layout of the power supply section. In my version, I put all the filter caps, Zeners and rectifiers on a small piece of perfboard mounted on top of the main transformer. This is a tight clearance with the top of the case but since there are vent holes in the top, heat isn't a problem. You can play around a bit here, especially if you have different transformers. You might be able to mount things a bit differently. Try to keep all the leads carrying 120v line current away from all the low voltage leads. The wire insulation is supposed to prevent



the two from having any intercourse but I like an air gap for additional safety.

Once the power supply is assembled and mounted, turn it on and check out the voltages for proper wiring. If all is well here you can continue.

All the ICs are mounted on a single piece of perfboard. As you can see from Figure 3, it's a very tight fit with the board mounted horizontally. I considered splitting the components and mounting two smaller boards vertically but decided against it because of the need for a lot of interconnecting wires. If you are building a version smaller than 32K, things will be much easier since the RAM chips take up most of the space on the board. A version larger than 32K will require a larger box.

To begin with, start by wiring up the clock oscillator circuit. When this is done, you can check the output waveform with a scope (or at least with a logic probe or frequency counter if you don't have a scope). Wire the 74LS74 divider and 74LS193 baud rate generating counters next and check out the outputs in a similar fashion. Next wire up the Z80 socket. (I did say to use sockets, didn't I?) Include the reset circuitry and pullup resistor R1. You don't need to do the address decoders or control decoders yet. When you plug in the Z80 and apply power you should be able to see some repetitive waveforms at the RD, WR aad MEMRQ pins as well as the M1 pin, showing the processor is operating in some kind of a loop. If nothing is hooked to the data lines yet, the loop will probably be from the processor executing a continuous series of RST 7 instructions (FF hex) and you can look to see if the waveform period matches this.

The next part to tackle is the 74LS138 and 74LS32 decoders. Once again, you can apply power to look at the outputs and see if they are what you would expect. If you don't know what to expect, then don't worry at this point unless you see smoke. Just proceed on.

Continue by adding the UART and interface ICs U8 and U9. U10 should already be wired in. At this time you can wire the 2716 EPROM socket and at least one RAM socket. If you are building a version larger than 4K you might as well finish all the wire wrapping now before proceeding to final checkout.

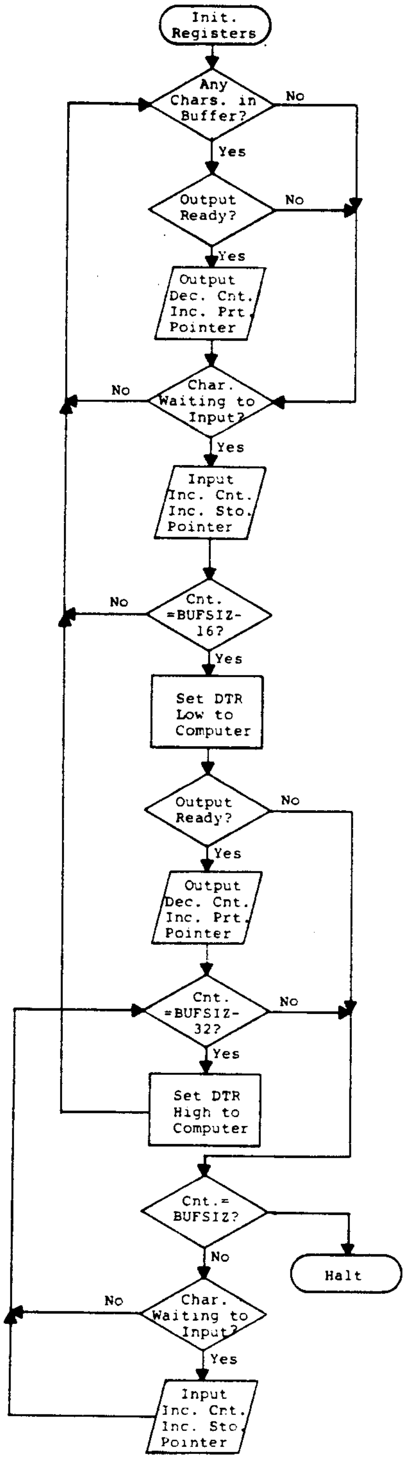
Once all the connections are made, apply power before inserting all the Ram chips. In case there is some problem you can cut your losses this way. The operating program to be burned into the 2716 is given at the end of this article, and a flow chart showing the program logic is given in figure 5. The program isn't long, so keying it into your system shouldn't be a problem. If you don't have facilities for burning EPROMS and don't know anyone who does, you can get a preprogrammed 2716 with the operating program in it from *The Computer Journal* for $15.

Final Checkout

Once all the components are installed in their sockets and a 2716 programmed, you are ready for the final checkout. The best way to do this is to print a file that you know will exceed the spooler's RAM capacity. If the handshaking protocol is working correctly, the spooler will fill up and then tell the computer to stop sending. There should be no overflow if this works correctly. Of course during this time the printer should be receiving characters and printing. If the printer buffer itself overflows, check the handshaking between the spooler and the printer. The printer must be configured for DTR handshaking on pin 20. If the spooler overflows, check the handshaking to the computer. It too must be DTR handshaking on pin 20. If there is no output to the printer check the cable wiring to both the computer and the printer. The spooler is wired to look like a printer to the computer and to look like a modem to the printer. Some printers and computers may not agree with this and you may have to do some reversing of pins to get the signal going. A logic probe and some signal tracing should clear things up. If the spooler is still inoperative, you will probably have to get hold of an oscilloscope to troubleshoot further. It is impossible to give specific troubleshooting instructions to cover a wide variety of possible faults. A good starting point is to reset the spooler and then look at the waveform on the M1 line from the Z80. It should have a repeating pattern every 45 clock cycles or 23 microseconds for a 2 MHz clock. This indicates that the CPU is reading instructions from the ROM and polling the I/O in a loop. If things are OK here, it is pretty certain that the problem is in the serial I/O section. If not OK, check the address decoders and ROM to be sure it is being selected for reading instructions.

Once you have the spooler working, it should give you a long period of trouble-free service (once the initial burn-in period is over, of course). It will let you have the use of your computer during periods when it would ordinarily be tied up servicing the printer, one slow character at a time. If you have elected to build a small memory version of the spooler, you can always expand it later simply by adding additional RAM chips and possibly another 74LS138 decoder. Just plan ahead and use a large enough chassis box.

In the third and final part of this series, I plan to discuss some alternative handshaking methods. I'll also show a modified circuit for those of you who want to use the spooler with a printer that has a Centronics interface.



```
;       Operating program for print spooler
;       Version of 5/18/83


;
;
RAMSIZ  EQU     30*1024         ;RAM storage capacity
BEGRAM  EQU     2048            ;Beginning of storage RAM
ENDRAM  EQU     BEGRAM+RAMSIZ   ;End of storage RAM
;
STAT    EQU     01H             ;Serial status port
DATA    EQU     00H             ;Serial data port
IFLAG   EQU     02H             ;Serial input flag
OFLAG   EQU     01H             ;Serial output flag
READY   EQU     80H             ;Serial DTR flag
;
;
        ORG     0000H
;
        LXI     B,0000H         ;Initialize character count
        LXI     D,BEGRAM        ;Initialize printing pointer
        LXI     H,BEGRAM        ;Initialize storage pointer
CHKOT1: MOV     A,B
        ORA     C
        JZ      CHKIN1          ;Nothing in buffer to print
        IN      STAT
        ANI     OFLAG
        JZ      CHKIN1          ;UART not ready for character
        IN      STAT
        ANI     READY
        JZ      CHKIN1          ;DTR low, printer not ready
        LDAX    D
        OUT     DATA            ;Send character to printer
        DCX     B
        INX     D
        MOV     A,D
        CPI     ENDRAM SHR 8
        JNZ     CHKIN1          ;No overflow of RAM
        MVI     D,BEGRAM SHR 8  ;Go back to beginning
CHKIN1: IN      STAT
        ANI     IFLAG
        JZ      CHKOT1          ;Nothing to input
        IN      DATA
        MOV     M,A
        INX     B
        INX     H
        MOV     A,H
        CPI     ENDRAM SHR 8
        JNZ     CHKBF1          ;No wrap
        MVI     H,BEGRAM SHR 8
CHKBF1: MOV     A,B
        CPI     (RAMSIZ SHR 8)-1
        JNZ     CHKOT1          ;>256 locations remaining
        MOV     A,C
        CPI     0F0H
        JNZ     CHKOT1          ;>16 locations remaining
        MVI     A,00H
        OUT     STAT            ;Set DTR low to computer
CHKOT2: IN      STAT
        ANI     OFLAG
        JZ      CHKBF3          ;Output not ready
        IN      STAT
        ANI     READY
        JZ      CHKBF3          ;DTR still low
        LDAX    D
        OUT     DATA
        DCX     B
        INX     D
        MOV     A,D
        CPI     ENDRAM SHR 8
        JNZ     CHKBF2          ;No wrap
        MVI     D,BEGRAM SHR 8
CHKBF2: MOV     A,C
        CPI     0E0H
        JNZ     CHKBF3          ;<32 remaining
        MVI     A,02H
        OUT     STAT            ;Set DTR back to high
        JMP     CHKOT1
CHKBF3: IN      STAT
        ANI     IFLAG
        JZ      CHKOT2          ;Nothing waiting for input
        MOV     A,C
        CPI     0FFH
        JZ      STOP            ;Spooler RAM overflow
        IN      DATA
        MOV     M,A
        INX     B
        INX     H
        MOV     A,H
        CPI     ENDRAM SHR 8
        JNZ     CHKOT2          ;No wrap
        MVI     H,BEGRAM SHR 8
        JMP     CHKOT2
STOP:   HLT                     ;Halt if error detected
;
        END
```

Books of Interest

Electronic Prototype Construction
by Stephan D. Kasten
Published by Howard W. Sams & Co.Inc.
4300 West 62nd St
Indianapolis, IN 46268
398 Pages, 5 3/8 x 8 1/2, softbound $17.95

This book, which is part of the BLACKSBURG Continuing Education Series, covers the mechanical aspects of building prototypes. The book includes information on wire-wrappings and system packaging, but the main focus is on printed circuit board construction. In fact approximately 75% of the text is on printed circuit board construction.
The contents are as follows:

- •Chapter 1-WIRE-WRAPPING. Includes pencil wiring,solder pad wiring,reverse wrap pins, boards for wire wraps, methods for mounting boards, power supplies, and design and construction.
- •Chapter 2-PRINTED CIRCUIT TECHNOLOGY. Includes general industrial process, prototype PCB considerations, and prototype process outline.
- •Chapter 3-PCB DESIGN, LAYOUT, AND ARTWORK. Includes design and artwork preparation.
- •Chapter 4-PHOTOGRAPHY. Includes professional graphic arts photography, 35 mm photographic reduction process, 35 mm photography, preparation of transparencies, and special artwork considerations.
- •Chapter 5-PHOTO RESIST. Includes photo resist processes, general properties of KPR 3 photo resist, laminate preparation, flow coating, image transfer, preparation for etching, and alternative photo resist processes.
- •Chapter 6-ETCHING. Includes Source of materials, ferric chloride etchant, cupric chloride etchant, and etching problems.
- •Chapter 7-ELECTROPLATING AND COATING. Includes general principles of electroplating, electroplating parameters, electroplating equipment, baths for tab plating, tab plating procedure, and PCB protective coatings.
- •Chapter 8-SCREEN PRINTING. Includes materials source, electronic applications of screen printing, general principles, elements of screen printing, manual printing techniques, and screen printing procedure.
- •Chapter 9-PCB MACHINING, SOLDERING, AND ASSEMBLY. Includes shaping, drilling, soldering, desoldering, cleaning PCBs, and PCB modifications and repairs.
- •Chapter 10-HIGH DENSITY PCBs. Includes the two layer PCB.
- •Chapter 11-ELECTRONIC SYSTEM PACKAGING. Includes basic planning, standard enclosures, parts layout, accessibility, panels, electrical considerations in packaging, chassis interconections, openings and holes, cabinet finishing, anodized aluminum, and panel labels and decorations.

The book is well written, easy to read, and gives detailed information on prototype printed circuit board layout and construction. Kasten is a chemist with the Tennessee Eastman company and is interested in computers, so his information on circuit board construction is accurate and complete.

COMPUTER BUILDER'S SOURCE DIRECTORY

The Computer Journal is preparing a directory of items used in prototype construction. You are invited to submit your literature, news releases, evaluations, and samples for consideration.

The Computer Journal, P.O. Box 1697, Kalispell, MT 59903-1697