Popular Electronics®

WORLD'S LARGEST-SELLING ELECTRONICS MAGAZINE

JULY 1978/$1

Build an IC Voltage Regulator for Your Car
Tune In Africa for Exciting English Broadcasts
Ultrasonic Detector Reveals Sounds of Insects
How to Get Hi-Fi Sound in Any Auto!
NEW SPEAKERS, TAPE DECKS, FM RADIOS, POWER BOOSTERS, EQUALIZERS + INSTALLATION TIPS

rntable — New Quartz-Lock System

aker — New Induction Tweeter

Issue Hafler Preamplifier — New Equalization Circuit

0 14024 14278

POPULAR ELECTRONICS

JULY 1978

er. Then rub the palms of your hands in front of *TR1*. The receiver will detect the ultrasonic energy from the rubbing.

You will notice that *TR1* has a very directional response. This is due to the fact that ultrasonics have very short wavelengths (compared to those at audio frequencies) and are thus subject to less diffraction at the edges of large objects. Also, ultrasonic waves behave like light waves in that they tend to travel in straight lines.

It's interesting to note that if coupling capacitor *C10* in the receiver is disconnected from the diode mixer, the receiver will still detect ultrasonic signals if more than one frequency is present. The frequencies present at the input will beat against each other to produce an audible output. This can be verified by repeating the palm-rubbing experiment described earlier after the coupling capacitor has been disconnected. The speaker will still generate an audio output even though no local oscillator signal is being injected into the diode mixer.

If an ultrasonic wave generated by transmitter transducer *TR2* now impinges upon *TR1*, the random noise reproduced by the speaker will drop to a low level. No tone will be heard because only one frequency is applied to the mixer. Stray coupling that allows a portion of the local oscillator output to reach the mixer will create an audible beat.

When the receiver and transmitter are operating in the same room, a signal will be heard as *R12* tunes the receiver across its range. The two transducers do not *have* to be directly facing each other if enough hard surfaces in the room reflect the ultrasonic waves, and the room is not so large that it introduces excessive signal attenuation.

The circuits presented have been successfully used with ultrasonic transducers from many different sources, including those used in television receiver remote control accessories. Of course, if you want to tune in several ultrasonic "bands," you can use a multiple-pole rotary switch to select the appropriate transducer and its corresponding oscillator capacitance. Experimentation indicates that the receiver can "hear" the transmitter at distances up to 125 feet if the transducers are aimed at each other. The use of a suitable parabolic reflector in tandem with *TR1* and/or multiple driven transmitter transducers should result in even greater useful range.

Other Suggestions. We have already mentioned the possibility of using these circuits for signalling purposes. Many other practical applications exist. For example, leaks in the rubber sealing of car doors and windows or in the sealing of a freezer door. The transmitter is placed in the car or freezer and fills the interior with ultrasonic waves. The walls of the interior reflect the waves to create a wide dispersion of ultrasonic energy. If the receiver's transducer is moved over the exterior, a tone will be heard whenever it passes any leaks. ◇

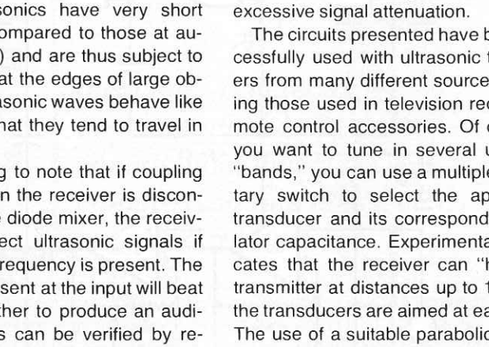


Fig. 2. This ultrasonic transmitter employs four NOR gates.

PARTS LIST FOR FIG. 2

C1—180-pF (or 330-pF) disc ceramic, polystyrene, glass or silver mica capacitor
IC1—CD4001 quad dual-input NOR gate
R1—470,000-ohm 10%, ¼-W resistor
R2—22,000-ohm 10%, ¼-W resistor
R3—20,000-ohm linear-taper potentiometer
S1—Spdt switch
TR2—Piezoelectric ultrasonic transducer
Misc.—Printed circuit or perforated board; suitable enclosure; hook-up wire; dc power source; machine hardware, etc.

Micro-PROCESSOR MICROCOURSE

BY FORREST M. MIMS

PART 5: THE CONTROL SECTION OF PIP-2.

THUS FAR in this series, we have covered the basics of number systems, digital logic and microprocessor organization. We have also introduced PIP-2, a simple 4-bit educational microprocessor, and learned how it's organized and programmed.

Now let's take a detailed look at the control section of PIP-2. We will see how instructions are fetched from the program memory, decoded and executed. We will also learn how to revise PIP-2's instruction set by modifying the microinstructions stored in control's ROM.

PIP-2's Control Section. The most important and complex section of a microprocessor is its control circuitry. This is the element that fetches instructions from the microprocessor's memory in the proper sequence, then decodes and executes the instructions.

The overall operation of the control section is a perfectly synchronized sequence of individual operations that fetch instructions, transfer data, advance counters and perform arithmetic operations.

The control section responds to a load instruction, for example, by simultaneously connecting the memory address containing the data word to be loaded (the *source*) and the input of the appropriate register (the *destination*) to the microprocessor's bidirectional bus. The control then sends a clock pulse to the register to complete the load operation and proceeds to fetch the next instruction.

While all this might seem extremely complicated to the unititiated, it's really quite simple since the program instruction is merely a binary bit pattern that can be interpreted by the control section to perform a specific task. In simplest terms, the control section is no more complicated (at least in principle) than the decoder circuit that lights up the proper segments of a seven-segment display in response to a binary-coded decimal (BCD) input nibble.

The heart of the control section of some microprocessors is a complex combinational network of gates that decodes program instructions and activates the appropriate control inputs of the various sections of the processor. More advanced microprocessors employ a special ROM that contains the sequences of microinstructions necessary to accomplish each program instruction. These so-called *microprogrammable* microprocessors are much more versatile since their instruction sets can be extensively revised by simply modifying the microinstructions stored in the ROM.

PIP-2, the educational microprocessor we've been studying, is microprogrammable and the block diagram shown in Fig. 1 illustrates the general organization of PIP-2's control section. You might want to refer back to Part 4 of this series to see how control interfaces with the remainder of PIP-2.

A detailed breakdown of PIP-2's control, including the organization of the microprogrammable ROM containing the microinstructions, the microinstruction decoders and the clock, is shown in Fig. 2. We will now discuss each part of the control section.

Clock. The clock is a relatively simple but vital part of CONTROL since it provides the synchronized train of pulses that cycle PIP-2 through a program. The clock's output is said to be *two-phase* since it supplies two streams of pulses having identical frequency but different phases from outputs $\phi 1$ and $\phi 2$. Fig-

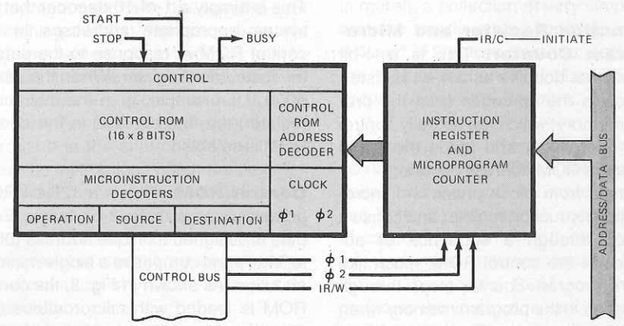


Fig. 1. Organization of the PIP-2 control section.

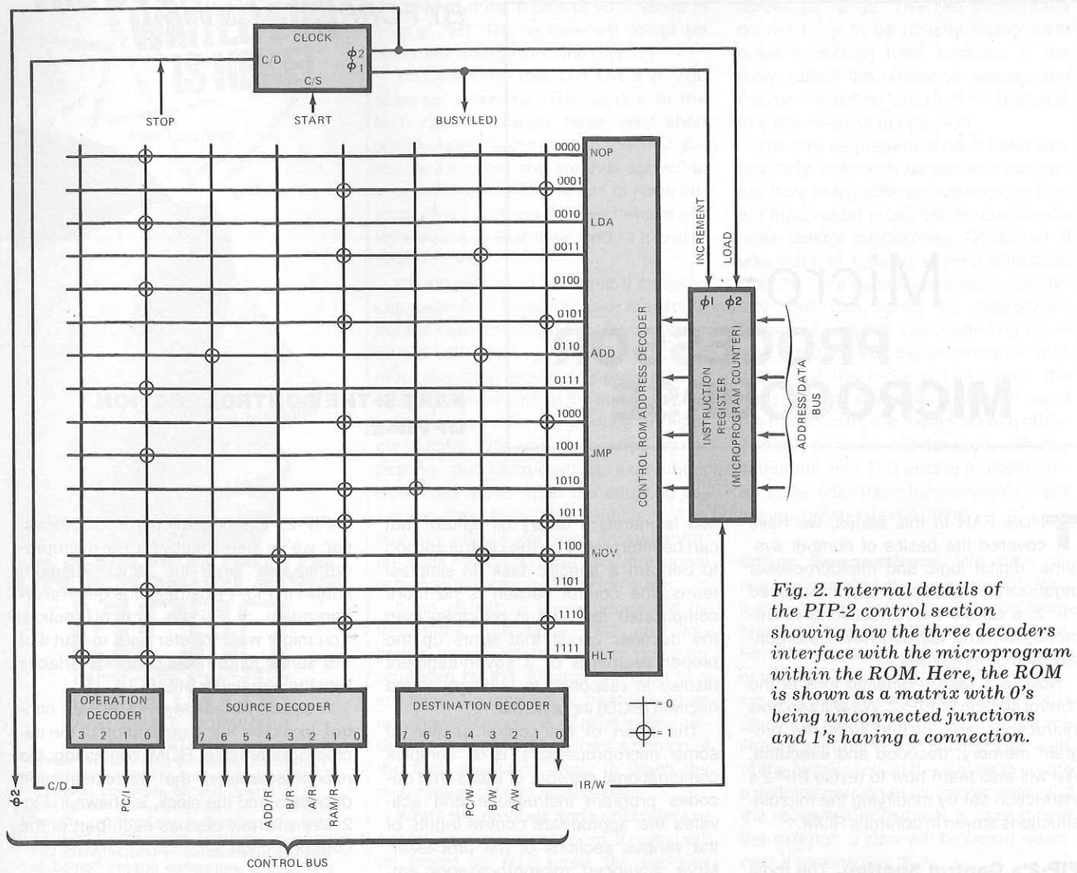


Fig. 2. Internal details of the PIP-2 control section showing how the three decoders interface with the microprogram within the ROM. Here, the ROM is shown as a matrix with 0's being unconnected junctions and 1's having a connection.

ure 3 shows the timing diagram for these two clock signals.

The clock has two control inputs. A low at C/S applied by pressing the START switch *starts* the clock. A low at C/D applied by pressing STOP, or by a signal from the microinstruction decoder (activated by a HLT instruction in the program), *disables* the clock.

Instruction Register and Microprogram Counter. This is a 4-bit counter that doubles as a 4-bit register. It receives the op-codes from the program memory, which are actually control ROM addresses, and feeds them into the control ROM address decoder.

Signals from clock phase ϕ 1 increment the instruction register and cause it to step through a sequence of addresses in the control ROM, much like PIP-2's program counter steps through addresses in the program memory when executing a program. That's why the instruction register can also be called a microprogram counter.

The instruction register has a couple of other control inputs. When IR/W is low, a ϕ 2 pulse from the clock *writes* the instruction on PIP-2's address/data bus into the instruction register. When IR/C is low, the instruction register is *cleared* to 0000.

Control ROM Address Decoder. This is simply a 1-of-16 decoder that activates appropriate addresses in the control ROM in response to the data in the instruction register. When the nibble 0000, for example, is in the instruction register, the first address in the control ROM is selected.

Control ROM. This is a 128-bit ROM organized as sixteen 8-bit bytes. Each byte is assigned a unique address (0000 to 1111) and comprises a single microinstruction. As shown in Fig. 2, the control ROM is loaded with microroutines (sequences of microinstructions) for six separate program instructions. As we'll soon see, these microroutines can be easily changed by simply reprogramming the ROM.

Microinstruction Decoders. Control has a pair of 1-of-8 decoders (Source and Destination), and a single 1-of-4 decoder (Operation). The selected output of each decoder goes *low* while the remaining outputs stay *high*.

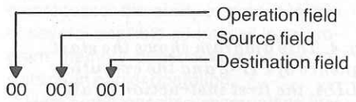
These decoders convert the microinstructions encoded in the selected ROM address into the appropriate operations necessary to execute the microinstruction. As you can see in Fig. 2, the control ROM is divided into sixteen 8-bit bytes. The first two bits of each byte are fed into the operation decoder. The next three bits go to the source decoder and the final three bits go to the destination decoder.

The outputs from the three decoders and from the clock form PIP-2's control bus. The outputs of the *source* decoder go to the read (R) control inputs of the various sections of PIP-2. The outputs of the *destination* decoder go to the write

(W) control inputs of the various sections. And the outputs of the *operation* decoder go to the special operation control inputs, clock disable (C/D) and program counter increment (PC/I).

Note that several outputs of the *source* and *destination* decoders and two outputs from the *operation* decoder are not used. This means that additional circuits (maybe a C register, perhaps an arithmetic-logic unit) can be connected to PIP-2's address data bus. These lines may also be used to control external devices. In both cases, of course, new microinstructions would have to be added to the control ROM to activate the new circuits.

Note also how the bit pattern stored in the ROM activates the decoders. Address 0001, for example, contains the microinstruction 00001001. Let's divide this byte into each of the three bit fields applied to the decoders and see what happens:



The operation field (00) does nothing since it activates the unconnected 0 output of the *operation* decoder.

The source field (001) activates the 1 output of the *source* decoder. This applies a low to RAM/R.

The destination field (001) activates the 1 output of the *destination* decoder. This applies a low to IR/W.

The result? The output of the program memory (RAM) and the input of the instruction register (IR) are simultaneously connected to the address/data bus, and the arrival of the next ϕ 2 pulse from the clock loads the instruction register with the selected instruction op-code in the program memory.

Now that we know something about each of the sections of PIP-2's control and how an individual microinstruction is executed, let's see how control fetches and executes an instruction from the program memory.

Fetching and Executing. Understanding how control fetches (retrieves) an instruction from the program memory and then executes it will take you a long way toward understanding how real microprocessors work. You might find it handy to have Part 4 of the Microcourse available since we'll be referring to PIP-2's instruction set mnemonics and op-codes.

Let's assume the first instruction in the program memory (address 0000) is LDA. This is a memory reference instruction that is followed by a 4-bit data nibble in the next program memory address. When executed, LDA will load the A register with the data nibble in program memory address 0001.

After the program containing the LDA instruction is loaded into the program memory, the INITIATE switch is pressed to return the program counter to program memory address 0000. The instruction register doubles as a microprogram counter and pressing INITIATE clears it to 0000 also.

The two microinstructions that comprise NOP occupy the first two bytes of the control ROM. When START is pressed, the first clock pulse advances the combination instruction register/microprogram counter to the second NOP microinstruction (control ROM address 0001).

What's the byte stored in this address? Figure 2 shows that this microinstruction is 00001001—which activates the RAM/R and IR/W control inputs discussed earlier. When the ϕ 2 clock pulse arrives, the instruction register

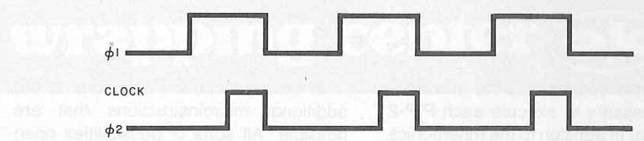


Fig. 3. Timing diagram of the PIP-2 two-phase clock.

copies the op-code of the instruction in program memory address 0000. The op-code for LDA is 0001, so in this case the instruction register doesn't change states. (What would happen if the op-code was 1011 or 0101?)

Thus far, all of control's operations have been preprogrammed and completely automatic with the specific goal of fetching the first instruction from the program memory. What happens next?

Recall that the op-code for each instruction is a binary number that is 0001 less than the starting address of the microroutine in the control ROM that executes the instruction. When the next ϕ 1 clock pulse arrives, the instruction register advances to the first microinstruction in the LDA microroutine and things start to happen. Let's follow the various steps in the execution of the LDA microroutine to see how.

The first LDA microinstruction (Fig. 2) is 01000000. Only the PC/I control input is activated; the program counter is advanced to the next address in the program memory (which contains the data nibble to be loaded into the A register). Signal ϕ 2 is a do-nothing clock pulse since there is no data located on the address/data bus.

The third ϕ 1 clock pulse advances the instruction register to the second microinstruction in the LDA microroutine (control ROM address 0011). This microinstruction (00001010) applies lows to RAM/R and A/W. When clock pulse ϕ 2 arrives, the A register copies the contents of the data nibble following the LDA op-code in the program memory.

Now that the A register has been loaded with the specified data nibble, the most important part of the LDA instruction has been accomplished. The remaining two microinstructions fetch the next step from the program memory.

The fourth ϕ 1 clock pulse advances the instruction register to LDA microinstruction 01000000. This increments the program counter to the next address in the program memory (0011). The next ϕ 2 clock pulse is another do-nothing pulse. The fifth ϕ 1 clock pulse advances the instruction register to the final LDA microinstruction, 00001001. This loads the op-code of the next instruction in the program memory into the instruction register.

All the steps necessary to execute LDA appear rather complicated at first. But if you'll browse back through the preceeding paragraphs again you'll see that LDA, like all of PIP-2's instructions, is merely a collection of very simple operations neatly strung together by the ϕ 1 and ϕ 2 pulses from the clock. Figure 4 is a sequence of diagrams that shows exactly what happens.

Summing Up Control. Now that you've seen how PIP-2 fetches, decodes and executes an instruction, you can better appreciate the sophistication of control. You can even think of control as a simple microprocessor inside PIP-2. The control ROM contains the program, the instruction register serves as the program counter and the microinstruction decoders implement the various instructions.

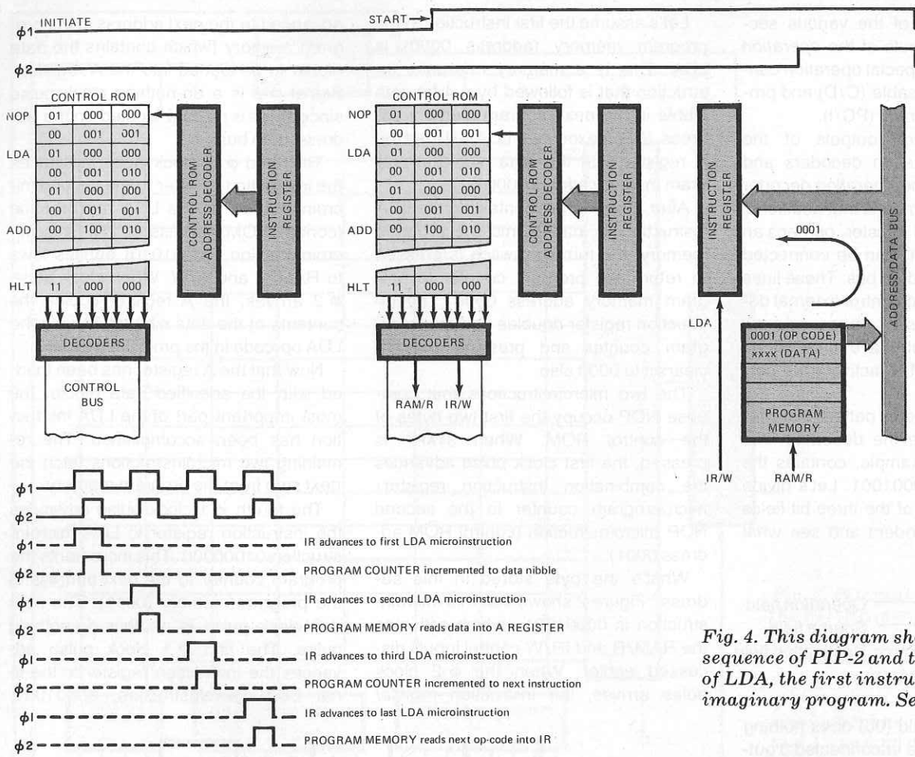


Fig. 4. This diagram shows the start sequence of PIP-2 and the execution of LDA, the first instruction in an imaginary program. See text for details.

The Table summarizes the microroutines necessary to execute each PIP-2 instruction. In addition to the mnemonics and their op-codes, the table contains the entire truth table of the control ROM. It also shows the operations that take place for each microinstruction.

Microprogramming PIP-2. Look back at the table of microroutines for a moment. Notice how often the fetch operations PC/I and RAM/R→IR/W occur? Remove these microinstructions from the table and we're left with only five additional microinstructions.

Obviously there are more possible microinstructions than just these seven. All that's necessary to arrive at a new microinstruction is to place one source and one or more destinations on the address data bus. Here are some possibilities:

A/R⟶IR/W
A/R⟶PC/W
B/R⟶IR/W
B/R⟶PC/W
B/R⟶A/W
RAM/R⟶B/W
ADD/R⟶B/W
ADD/R⟶PC/W
ADD/R⟶IR/W

Of course these are only some of the additional microinstructions that are possible. All sorts of possibilities open up if we activate *more* than one destination device. For example, RAM/R⟶A/W;B/W;PC/W.

If we assume that you have assembled a working version of PIP-2, it's quite possible the original instruction set will not fill your requirements. If that's the case, you can substitute new microinstructions to devise your own special instruction set.

Suppose you want to replace LDA with LDB (load the B register). All you have to do is find the LDA microroutine in the control ROM and reprogram the byte that loads the A register (address 0011) so that the B register is loaded instead. The original byte is 00001010. The new byte is 00001011. The remaining bytes are unchanged. The op-code for LDA becomes the op-code for LDB since we haven't changed the location of the microroutine in the control ROM.

You can use this same procedure to microprogram other new instructions into PIP-2. Just remember these points:

1. Be sure to assign the correct op-code to each new instruction. Remember, the op-code is a binary number that is 0001 *less* than the first address of the microroutine in the control ROM.

2. If necessary, be sure to include the appropriate fetch microinstructions in each new microroutine so the next instruction in the program memory will be retrieved.

3. Be sure the microinstruction at the 0001 address in the control ROM is always 00001001. This is necessary since this microinstruction plays a key role in fetching the first instruction from the program memory during PIP-2's automatic start sequence.

4. Plan ahead! Are you eliminating an existing instruction(s) you might need later? Does the control ROM have room for the new instruction(s)? Are there any possible programming shortcuts you can use to implement instructions *not* in the control ROM?

5. Document your work so you'll know what you've done.

Don't let these simple precautions stop you from having a go at microprogramming PIP-2! Some of the possibilities are very interesting.

For example, an instruction that loads the program counter with the contents of the A register permits the program to branch to an address specified by the result of an addition. This procedure is called *indirect addressing*. It gives a microprocessor the ability to branch to one of several possible addresses in its program memory depending upon the results of an earlier operation.

Here's one possible microroutine that performs the indirect addressing operation we've been discussing:

Microinstruction			Operation
OP	S	D*	
00	100	100	ADD/R→PC/W
00	001	001	RAM/R→IR/W

** OP = operation; S = source; D = destination decoders.*

PIP-2'S MICROROUTINES

Program Memory		Control ROM				Operation
		Address	Microroutine			
Mnemonic	OP-Code		OP	S	D	
NOP	1111	0000	01	000	000	PC/I
		0001	00	001	001	RAM/R→IR/W
LDA	0001	0010	01	000	000	PC/I
		0011	00	001	010	RAM/R→A/W
		0100	01	000	000	PC/I
		0101	00	001	001	RAM/R→IR/W
ADD	0101	0110	00	100	010	ADD/R→A/W
		0111	01	000	000	PC/I
		1000	00	001	001	RAM/R→IR/W
JMP	1000	1001	01	000	000	PC/I
		1010	00	001	100	RAM/R→PC/W
		1011	00	001	001	RAM/R→IR/W
MOV	1011	1100	00	010	011	A/R→B/W
		1101	01	000	000	PC/I
		1110	00	001	001	RAM/R→IR/W
HLT	1110	1111	11	000	000	C/D

Since the second microinstruction of this microroutine is the same as the second microinstruction of NOP in PIP-2's original instruction set, we can easily substitute it for NOP. We just reprogram the first address in the control ROM with 00100100 and assign NOP's op-code to the new instruction.

For convenience, it's nice to assign a mnemonic to the new instruction. Since the instruction is an indirect jump, one possibility is JMI. You might want to be more specific since other indirect jump instructions are possible. Since this is a "JUMP INDIRECT TO ADDRESS IN A," a better mnemonic might be JIA.

Now that you know how PIP-2 is microprogrammed, how about adding a new instruction or two on your own? With a little care you just might come up with an instruction set that's better.

Summing Up. If you've stayed with the course, you should have a fairly respectable knowledge of some of the fundamental basics of microprocessors. To be sure, *real* microprocessors are far more sophisticated than PIP-2. But PIP-2 has prepared you to move up to real microprocessors. ◇