Popular Electronics®

WORLD'S LARGEST-SELLING ELECTRONICS MAGAZINE

JUNE 1978/$1

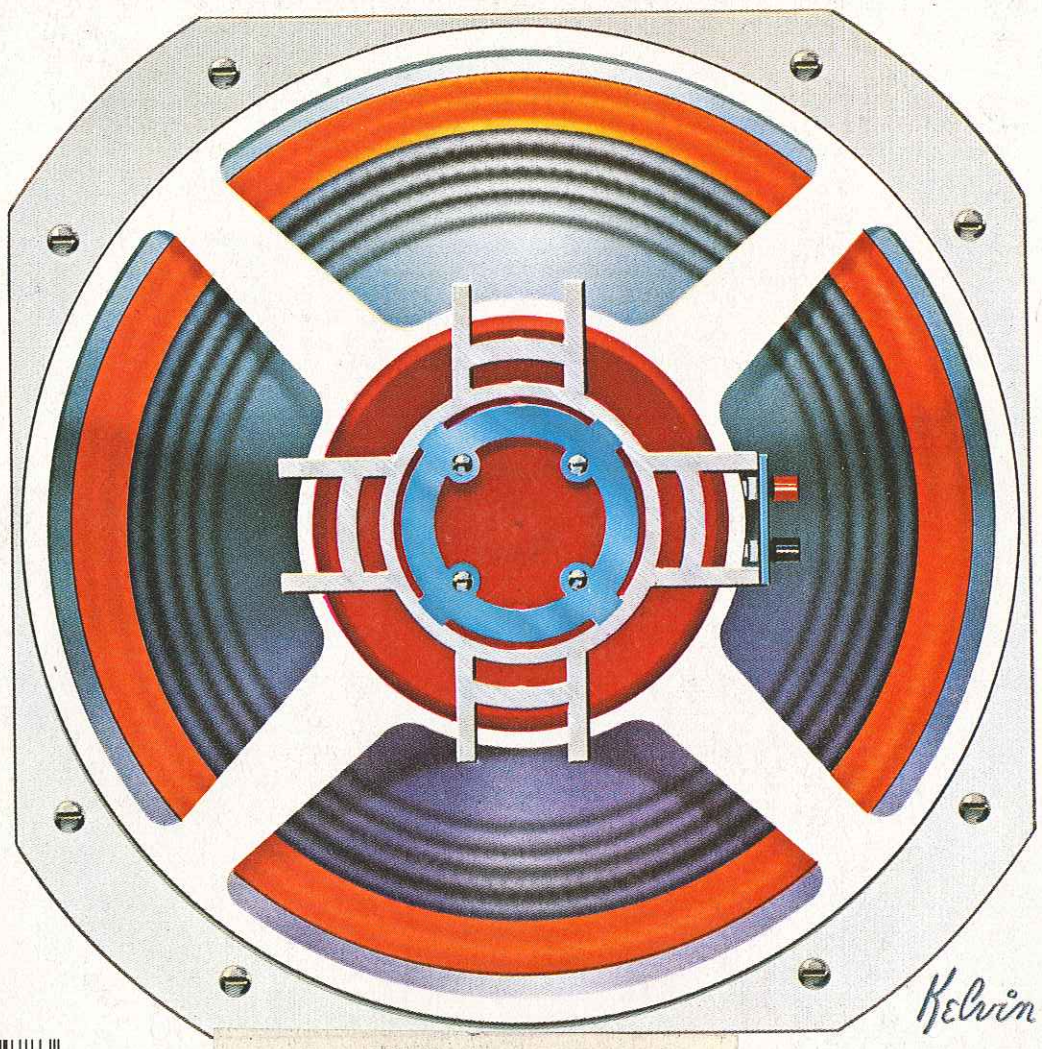
Compressor Guard Protects Air Conditioners

A "Fuzz Box" For Electric Guitar

Experiments With Programmable Logic Arrays

Special Focus On Hi-Fi Speaker Systems

MODEL-BY-MODEL COMPARISONS & HOW-TO-AUDITION GUIDE



Stereo Phono Cartridge

tereo FM Tuner

ereo Cassette Deck

0 14024 14278 06

Popular Electronics

Micro-PROCESSOR MICROCOURSE

BY FORREST M. MIMS

PART 4. PIP-2 AN ULTRA-SIMPLE EDUCATIONAL MICROPROCESSOR.

IN Part 3 of this series (May, 1978), we learned about semiconductor memories and how three-state logic allows data transfer over a bidirectional data bus. We also looked at the basic organization of a microprocessor.

This month we're going to meet PIP-2, a very simple, 4-bit educational microprocessor. Though PIP-2 is not as powerful as the 8080, Z80, 6502 and other real-world microprocessors, it illustrates some of the more important operating features of microprocessors.

Introducing PIP-2. *PIP* is an acronym for Programmable Instruction Processor. PIP-2 is a simplified successor to PIP-1, an educational computer described in detail in *Understanding Digital Computers*, a new book published by Radio Shack.

While PIP-2 is simple, it has many of the elements of a sophisticated microprocessor. For example, PIP-2 contains a built-in program memory—so it really qualifies as a *microcomputer*. Since it also contains a microprogrammable control ROM, this means that its instruction set can be revised, or replaced, by entirely new instructions, as we will see in Part 5 of this series.

PIP-2's Organization. A block diagram of the major components of PIP-2 is shown in Fig. I. As you can see, PIP-2 is a bus-organized microprocessor. All of its sections are connected to a 4-bit bidirectional bus which permits data *and* memory addresses to be transferred from one section to one or more other sections connected to this bus.

Remember from Part 3 that only *one* section can read data onto a bidirectional bus at any one time. PIP-2 meets this operating restriction by employing three-state outputs on all sections designed to read data onto the bus. This isolates the output of those sections from the bus until they are activated (one at a time) by an appropriate enable signal from PIP-2's control section.

Let's now take a look at each of the sections in PIP-2.

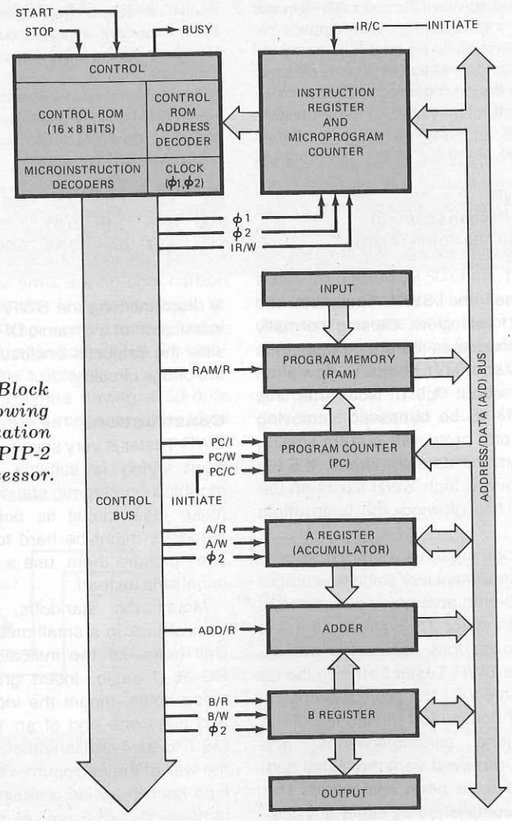


Fig. 1. Block diagram showing organization of the PIP-2 microprocessor.

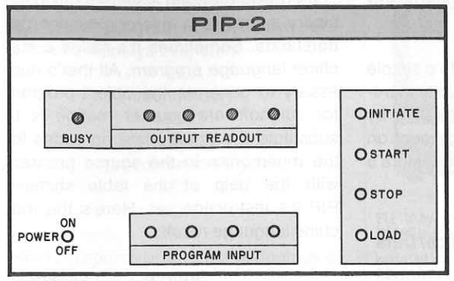


Fig. 2. Front-panel arrangement to facilitate operation of PIP-2.

Input. A row of four toggle switches, a LOAD switch and an INITIATE switch comprise PIP-2's INPUT. All these switches are shown in Fig. 2, a front-panel arrangement that allows PIP-2 to be used like a microcomputer.

Applying power to PIP-2 automatically clears the A and B registers, the program counter and the program memory to all 0's. This permits a program to be loaded into the program memory by simply switching in a binary instruction or data word and pressing the LOAD switch.

Up to sixteen 4-bit instructions and data words can be loaded into PIP-2's program memory. After the program is loaded, the program counter is cleared to 0000 by pressing the INITIATE switch. This returns the program counter to the first memory address in the program memory in preparation for running the program.

Program Memory. This is a 64-bit read/write memory (RAM) organized as sixteen 4-bit words or "nibbles." The RAM has a three-state output to keep its instructions and data isolated from the address data bus until they're needed.

The program memory has a single control input, RAM/R (R = read). When RAM/R is low, the three-state output is enabled, and the RAM reads the word addressed by the program counter onto the address data bus. When RAM/R is high, instructions and data can be loaded into the RAM.

Program Counter. This is a 4-bit binary counter. PIP-1 and many real microprocessors have a special memory address register that saves the contents of the program counter until it's time to advance to the next memory address. In PIP-2, the program counter doubles as a memory address register.

The program counter has three control inputs. A "low" that's supplied to PC/C by pressing the INITIATE switch clears the counter to 0000. The rising edge of a pulse applied to PC/I increments the program counter to the next higher count. A low at PC/W (W = write) writes any data on the address data bus into the program counter. This is a valuable feature since it means the program counter can branch to any address in the program memory.

A and B Registers. These are standard 4-bit data registers with three-state outputs. Each has two control inputs and a clock input ($\phi 2$).

When A/R or B/R is low, data is read from the selected register onto the address/data bus. When A/W or B/W is low, any data on the address/data bus is written into the selected register when the next clock pulse ($\phi 2$) arrives.

Adder. This is a 4-bit combinational logic circuit that continually sums the contents of the A and B registers. The sum is isolated from the address/data bus by a three-state buffer. When ADD/R is low, the buffer is enabled and the sum is placed on the bus.

Output. PIP-2's output consists of four light emitting diodes (LED's) that continually show the contents of the B register. It's possible, of course, to connect external devices in place of the LED's. A 4-line to 16-line decoder, for example, would permit PIP-2 to control any one of up to sixteen external devices.

Control. This is the electronic nerve center of PIP-2. Control fetches instructions from the program memory and executes them one by one under the perfectly synchronized control of timing signals ($\phi 1$ and $\phi 2$) produced by the clock.

Control consists of a 128-bit ROM organized as sixteen 8-bit bytes, an address decoder, several microinstruction decoders and a two-phase clock. PIP-2's instruction register doubles as a microprogram counter and is so closely associated with control that it can be considered part of it.

In the next installment, we'll look at a block diagram of control and study its operation in detail. For now, suffice it to say that control's ROM contains a sequence of from one to five microinstructions for each of the various microroutines necessary to execute PIP-2's six instructions. As you'll recall from Part 3, individual microinstructions implement simple operations such as data transfers from one register to another, etc.

PIP-2's Instruction Set. PIP-2 can process six separate instructions. Each instruction is identified for humans by a type of shorthand called a *mnemonic* (memory aid) and for PIP-2 by a 4-bit nibble called an operation code or in simple terms an *op-code*.

Some of the instructions require only one program memory address, while others are followed by a data word. These latter instructions require two program memory addresses and are called memory reference instructions. For example,

0001 (LDA)
1111 (data)

is the format for a memory reference instruction that loads the A register (LDA) with the data word 1111.

Shown in the box below is a table that summarizes PIP-2's instructions set. These instructions are so simple that they really need no further explanation.

PIP-2's INSTRUCTION SET

Mnemonic	Op-Code	Nibbles	Operation
NOP	1111	1	no operation.
LDA (nibble)	0001 (xxxx)	2	load A with next nibble.
ADD	0101	1	add A+B; store sum in A.
JMP (address)	1000 (xxxx)	2	jump to address in next nibble.
MOV	1011	1	move A into B; save A.
HLT	1110	1	halt the microprocessor.

It will be easier to apply them in actual programs, however, if we know something about how and why they're used. Therefore let's discuss the instructions one by one.

NOP. Pronounced "no-op," this is a do-nothing instruction with several valuable applications. You can use a NOP or two to reserve space in a program for an instruction or two you might want to add later. And you can use NOP's to replace instructions you remove from a program without rewriting the program. Finally, you can use NOP's to add a predictable time delay to a program. This is handy for calibrating a program that loops through a cycle of instructions again and again to act like a timer.

LDA. This memory reference instruction (load A) loads the A register with the data nibble in the next program memory address. It is used to temporarily store a nibble for addition or later transfer to the output or program counter.

ADD. This single-step instruction initiates a string of five microinstructions that *adds* the contents of the A and B registers and place the sum in the A register. It is used for ordinary addition, and to increment the nibble in the A register by some specified number (often 1). Incidentally, ADD uses the A register like the accumulator register found in real microprocessors.

JMP. This (jump) is a very powerful instruction that orders the program counter to branch (or jump) to the address in the program memory specified in the following nibble. JMP is used to set up a *loop*, a program or section of a program that continues to execute again and again until PIP-2 is halted by pressing its STOP button.

MOV. This register-transfer instruction has several applications. As an output instruction, it allows PIP-2's operator to see the contents of the A register on the LED readout (output). It also allows you to accomplish the equivalent of a LDB (load B) instruction by preceeding it with LDA (load A). And, it lets you double a number by following it with an ADD.

HLT. This instruction (halt) is placed at the end of all PIP-2 programs. It disables the clock in the control section, thus preventing PIP-2 from executing any additional instructions.

In the next part of the course, we'll examine the microroutines for each of these instructions in detail. We'll also learn how to add new instructions by changing the microinstructions in control's ROM. Meanwhile, let's learn how to program PIP-2.

How to Program. Let's write a simple program for PIP-2 that continually increments the number in the A register by one and displays the updated count on the LED readout of the output. Here's the program:

Program Memory Address	Mnemonics/Data
0000	LDA
0001	0001
0010	ADD
0011	MOV
0100	JMP
0101	0000
0110	HLT

It's easy to see how this program works. When PIP-2 is started, both the A and B registers are cleared to 0000. This means that the first three instructions load 0001 into A, add A to B and store the sum (0001) in both A and B. JMP loops the program back to line 0000 for another cycle. LDA replaces the contents of A with 0001 first. Register B also contains 0001 so ADD gives 0010. The sum, 0010, is moved into B and displayed on the readout.

Again, JMP loops the program back to line 0000 and the process continues. The result is that the readout flashes a binary count of 0000 to 1111 and continues repeatedly until PIP-2 is halted.

As you can see, this program is nothing more than a software version of an ordinary 4-bit counter. That alone is not very impressive since PIP-2 already contains *two* such counters in its hardware, the program counter and instruction register.

What's significant is that this simple program can be easily modified to implement *any* count increment from 0000 to 1111 by simply changing the data nibble following LDA! While this can be accomplished with some relatively simple hardware, PIP-2 performs the task after only a few seconds of software modification. This nicely illustrates the amazing versatility of using a microprocessor to simulate many different hardware functions with the help of software.

Running the Program. The simple counter program we've been discussing is called a *source program* since it's written using the mnemonics of the various instructions. Before it can be loaded into PIP-2's program memory, it must be converted to an *object program*.

An object program is written using the binary numbers a microprocessor understands. Sometimes it's called a *machine language program*. All that's necessary to generate the object program for our software counter routine is to substitute the appropriate op-codes for the mnemonics in the source program with the help of the table showing PIP-2's instruction set. Here's the machine language result:

Address	Source Program	Object Program
0000	LDA	0001
0001	0001	0001
0010	ADD	0101
0011	MOV	1011
0100	JMP	1000
0101	0000	0000
0110	HLT	1110

After the object program is compiled, it's a simple matter to load it into PIP-2's program memory. First, the power switch is turned on. This automatically clears all of the program memory, registers and counters to all 0's. Then the first object code nibble in the program (0001) is switched in via the front panel switches (a switch is 0 in the down position and 1 in the up position) and the LOAD switch is pressed. This action loads the nibble 0001 into the 0000 address of the program memory and automatically advances the program counter to the next address.

The remaining nibbles are loaded one by one until they are all stored sequentially in the program memory. Then the INITIATE switch is pressed to return the program counter to the 0000 address of the program memory.

Now all that remains is to press START. This causes control to fetch the first instruction from the program memory, load it into the instruction register, decode it and execute it. The program is processed like this a step at a time as the output displays the updated contents of the B register.

Incidentally, if the clock speed is more than about a hundred Hz, the count displayed on the readout will blur into a continuous 1111. Since the clock of most real microprocessors runs at a MHz or more, time delay loops must be added to their programs intended to display data to be viewed by an operator.

Other PIP-2 Programs. Though PIP-2's instruction set is very primitive, it's possible to write a number of different programs with it. Here, for example, is a source program that adds two numbers and displays their sum:

LDA
(first number)
MOV
LDA
(second number)
ADD
MOV
HLT

Here's a source program that doubles a number:

LDA
(number)
MOV
ADD
HLT

And here's a program that counts by two's:

LDA
0002
ADD
MOV
JMP
0000
HLT

Programming Real Microprocessors. Real microprocessors have dozens of instructions in their instruction sets. A typical microprocessor such as the 6800 or 8080 has instructions that can accomplish any of these tasks:

- Move data and addresses between registers.
- Shift and rotate the bits in a data word.
- Perform various arithmetic and logical operations.
- Branch conditionally or unconditionally to any part of a program or to a subroutine.
- Make various logical comparisons.
- Increment or decrement the contents of a register or memory address.

Real microprocessors also have special instructions that may be unique to a particular family of microprocessors. For example, some microprocessors have various instructions for accepting data from outside circuits. Others have built-in decimal arithmetic capability.

Programming real microprocessors can be both tedious and time consuming, but most people can learn to write simple programs with a little practice and some hands-on experience with a microprocessor using a keyboard (best) or toggle switch (OK) input. Of course, many microprocessor programs have been published in books and articles; and as time goes by, the number of available programs will multiply. ◇