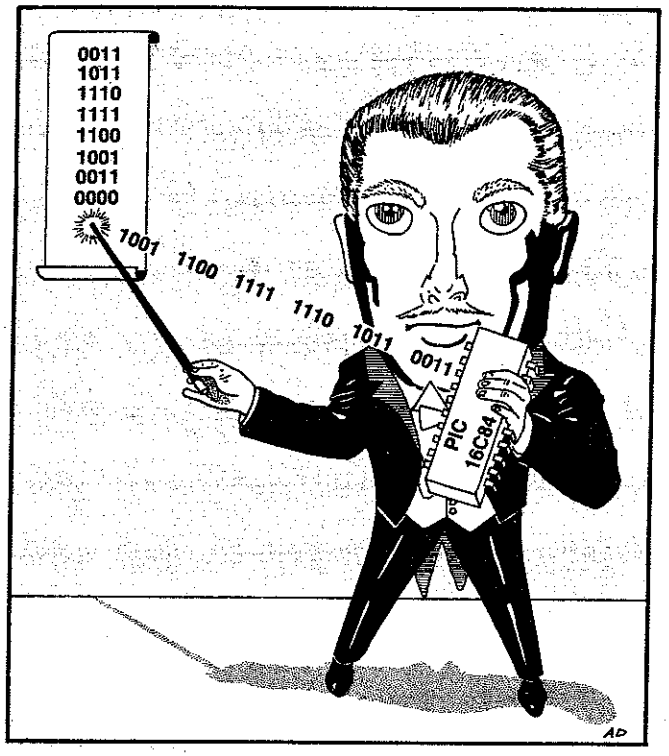
BURN PIC MICROCONTROLLERS WITH A "NO PARTS" PIC PROGRAMMER



Now you can "magically" create your own custom PICs with this simple programmer.

MICHAEL A. COVINGTON

When creating and designing a new electronics project, the current trend is to try to keep the descriptive terms "low-cost," "full-featured," and "ultra-compact" in the top slots in a list of features. That used to be hard to do. Fortunately, over the last several years, designing a project with those attributes has become much easier thanks to the development of programmable devices such as the PIC microcontroller family from Microchip Technologies. Looking over the last year's worth of construction articles that have been published in **Electronics Now** shows just how commonplace those microcontrollers have become.

Unfortunately, some sort of programming device is needed to "burn" the software into those programmable chips. The cost of such a device could easily wipe out any savings from using a microcontroller in a project—until now.

Obviously, the PIC programmer presented here needs *some* parts in order to build it. The nicest feature of the unit is that it needs no specialized parts, and it's a very simple circuit. The unit can be built in about an hour with parts from either your junk box or the nearest RadioShack store. Connect it to the printer port of any handy PC (the port need not be bi-directional), run the free software, and you can get started programming PIC16F84, 16F83, and 16C84 microcontrollers immediately. With an adjustable power supply, you can even do production-grade work.

What's a PIC? A PIC, like most microcontrollers, is a tiny computer with CPU, ROM, RAM, and I/O circuits all on one chip. Microcontrollers are the fastest-growing segment of the electronics industry; the average home now contains about 100 of them in everything from microwave ovens to wristwatches.

Many microcontroller applications don't involve computing, at least not as you'd normally think of it. A microcontroller is best thought of as an IC that you can customize by writing a program in assembly language. You then download the program into the ROM area of the microcontroller, and the microcontroller becomes a custom IC. Sometimes the program is designed to be little more than a logic gate or an oscillator, but what's important is that it does exactly what you tell it to do.

Among the low-end microcontrollers, the PIC family from Microchip, Inc. is especially popular because of their simplicity and low cost. Additionally, much of the software used to create the program code needed for programming PICs is available free for downloading from Microchip's Web site (www.microchip.com).

Of the various types of PICs that are available, the 16F83, 16F84, and 16C84 are probably the easiest to work with. Those particular models are the ones that our programmer supports. They cost less than $6 each, and their ROM is electrically erasable so that you don't need an ultraviolet light to erase and reprogram the devices.

The 16F84 is the most popular of the three—it has 68 bytes of RAM and 1024 words of program memory. The program memory is a "flash"

PIC will reboot whenever power is applied, but it won't automatically reboot several times a second. That feature may be used if there is the possibility of the PIC program "locking up" or for a special timing arrangement in a control application.

It is important to use the *__config* instruction in any programs used with the "No Parts" PIC Programmer. The assembler program will not be doing the actual programming—it will only be creating a file with the numbers that will be transferred to the PIC chip as a second step.

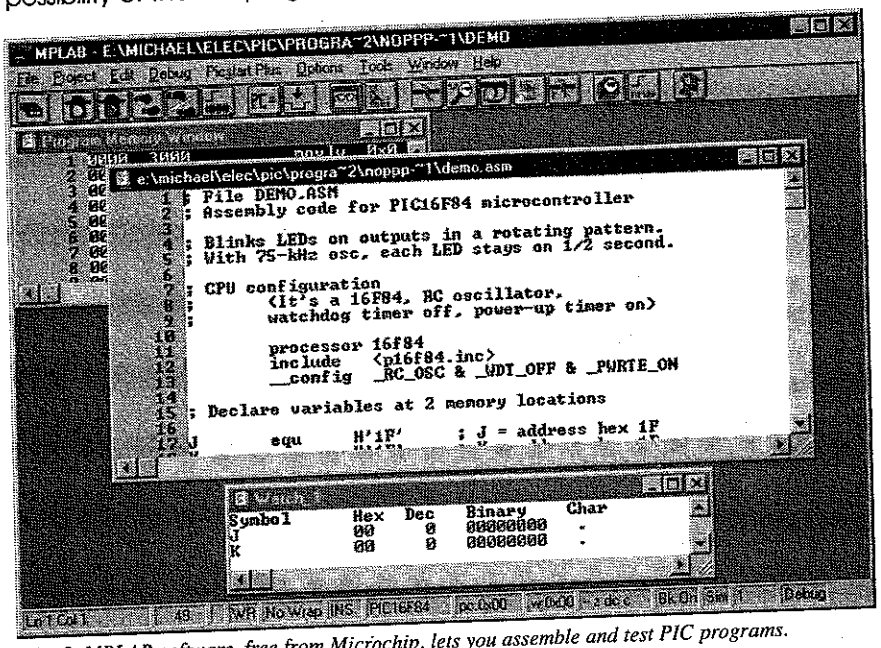


Fig. 5. MPLAB software, free from Microchip, lets you assemble and test PIC programs.

The two *equ* instructions reserve memory space in the PIC's RAM for two variables, which we'll be calling "J" and "K." It is just like declaring variables in BASIC, only we need to say which physical RAM locations will be used. In this case, those locations are (in hexadecimal numbering) 1E and 1F. Those locations will be used to store counters to keep track of how many times a loop has been repeated.

The *org* instruction tells the assembler that the program starts at location 0 in program memory and that the actual program is next.

The first real PIC instruction is a *movlw* instruction that clears the working register (called *W*). That number is then copied into the TRIS control register for port B, setting pins 6–13 to output pins instead of input pins. Next, the program puts binary 00000001 into the W register and copies it to Port B. That lights the LED connected to pin 6. But before you have time to actually see the LED come on, the program executes an *rlf* instruction that rotates the contents of Port B to the left, changing the data to 00000010. That will light the second LED attached to pin 7 instead. Repeating that instruction will give 00000100, then 00001000, and so on, making the LEDs flash in a marching pattern.

In between rotations, the program needs to wait about ½ second so that the action isn't too fast to see. That's why we have a delay loop in the program. It stores the decimal number 50 in memory locations "J" and "K," using the *decfsz* instruction to count down from 50 to 0.

Conditional instructions on the PIC are somewhat unusual, and decfsz is no exception. It stands for "Decrement and skip next instruction if zero." Translating the program into English, the instructions

```
kloop:   decfsz   K,f
         goto     kloop
```

mean "Subtract 1 from variable K,

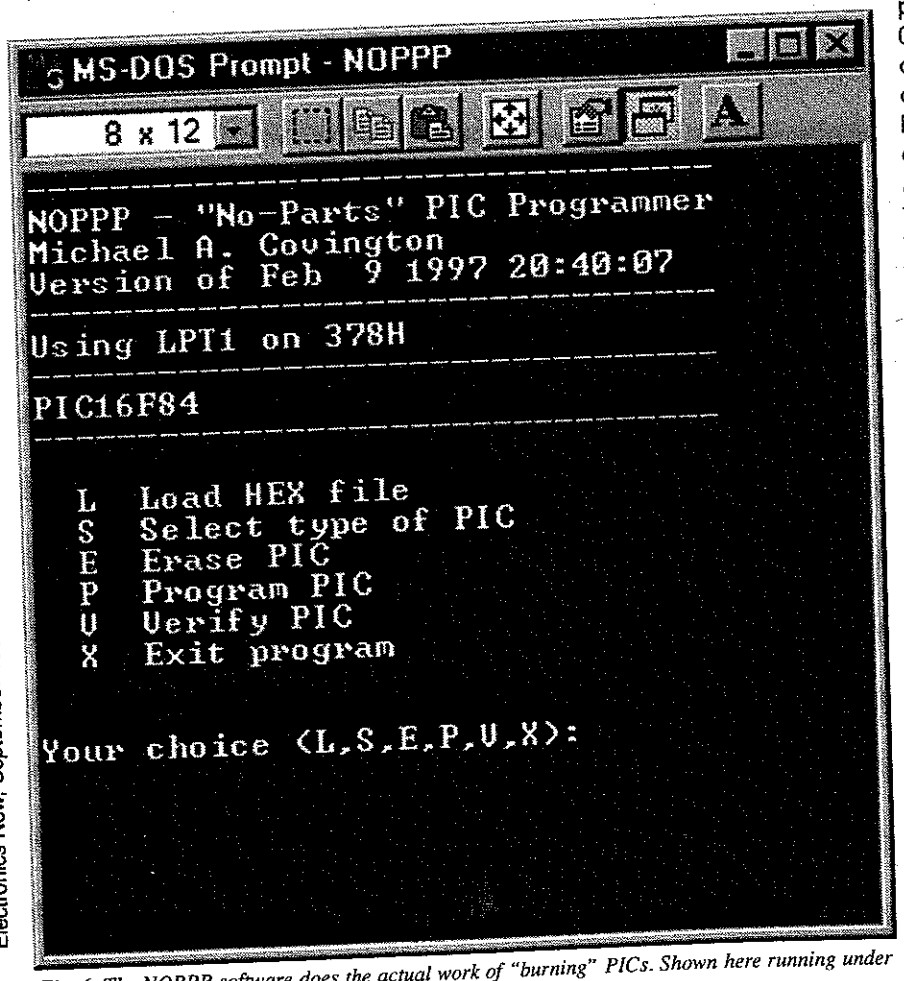


Fig. 6. The NOPPP software does the actual work of "burning" PICs. Shown here running under Windows 95, it also runs under DOS, Windows 3.1, and OS/2.

LISTING 1

```
; File DEMO.ASM
; Assembly code for PIC16F84 microcontroller

; Blinks LEDs on outputs in a rotating pattern.
; With 75-kHz osc, each LED stays on 1/2 second.

; CPU configuration
;       (It's a 16F84, RC oscillator,
;       watchdog timer off, power-up timer on)

        processor 16f84
        include   <p16f84.inc>
        __config  _RC_OSC & _WDT_OFF & _PWRTE_ON

; Declare variables at 2 memory locations

J       equ     H'1F'       ; J = address hex 1F
K       equ     H'1E'       ; K = address hex 1E

; Program

        org     0           ; start at address 0

        ; Set port B as output and initialize it

        movlw   B'00000000'     ; w := 00000000 binary
        tris    PORTB           ; port B ctrl register := w
        movlw   B'00000001'     ; w := 00000001 binary
        movwf   PORTB           ; port B itself := w

        ; Rotate the bits of port B leftward

mloop:  rlf     PORTB,f

        ; Waste some time by executing nested loops

        movlw   D'50'       ; w := 50 decimal
        movwf   J           ; J := w
jloop:  movwf   K           ; K := w
kloop:  decfsz  K,f         ; K = K-1, skip next if zero
        goto    kloop
        decfsz  J,f         ; J = J-1, skip next if zero
        goto    jloop

        ; Do it all again

        goto    mloop

        end
```

and if the result is zero, skip the goto instruction." Normally, the result is not zero and the goto part of the instruction is not skipped. Instead, the loop executes repeatedly until K reaches zero. As you can see, if you like double negatives—or rather, *don't not* like double negatives, you'll love programming PICs. The actual program uses two loops, one nested within the other.

Finally, *goto mloop* sends execution back to the beginning of the program. The *end* instruction is not a CPU instruction; instead, it tells the assembler that the program is over.

The 16F84 has 35 different CPU instructions. As you can see from the simple program we just created, you don't have to master all of them in order to write useful programs.

Compiling Programs. At the Microchip Web site, you can get data sheets, application notes, and best of all, MPLAB, which is a full-featured development program for compiling and testing PIC programs. It is designed to run under Microsoft Windows. A sample screen shot seen in Fig. 5 shows our demo program being edited. The MPLAB software lets you edit assembly-language programs (also called *source code*), assemble them into object code, and then step through the resulting binary code to see what it will actually make the microcontroller do. That way, you can spot any logical errors in your programming before you actually commit any code to hardware.

The Microchip software is well documented on its use. We are going to be using MPLAB to create an object-code file from the source-code text we typed in from Listing 1. Be ready for an error message when compiling the program. MPASM will complain mildly that you're not supposed to use the *tris* instruction. Microchip has dropped support for that instruction and some future PIC processors might not support it. A nice feature of the PIC from a software point of view is what is called *source-code compatibility*. If your design outgrows the resources of the chip that you started with, you can use another chip with more resources without having to rewrite the program from scratch. For our purposes, using the *tris* instruction on the 16C84, 16F84, and 16F83 works fine. Besides, the alternative way of setting up Port B for output is much more complicated.

"Burning" a PIC. A second piece of software is needed for actually using the "No Parts" PIC Programmer. That software is available at the Gernsback Web site (ftp://ftp.gernsback.com/pub/EN/noppp.zip). That MS-DOS program runs well under Windows 3.x or Windows 95. If, however, you are running it under Windows 3.1, it will work best if you run the application "full screen" instead of in a window. Timing is critical for the programming pulses, and full-screen DOS applications get full control of the computer. If for some reason you have difficulty running the NOPPP program under Windows 3.1, try exiting to a DOS prompt and run it from there. You can even run the program under OS/2; if you do, be sure to set the HW_TIMER to "on" in the DOS settings for the program.

The first step is to connect the "No Parts" PIC Programmer to the PC's

printer port and start the NOPPP program without any power connected to the programmer. If the 5-volt line is grounded, the software will not be able to detect D1, and will assume that the programmer is not attached to the printer port.

If all is well, you should see a screen similar to the one shown in Fig. 6. The menu of choices is self-explanatory. In general, you would want to load an object-code file (with a .HEX extension in the filename) into memory, select the type of PIC that you will be programming, program the part, and then verify that the code was programmed into the chip correctly. You can also erase a PIC that has already been programmed for re-use or updating of the programming. One note of caution: you should never insert or remove a PIC from the programmer while the power to the programmer is on. When programming a PIC, the software will tell you what to do and when to do it.

Since the programmer software requires some tricky timing, it was written to run as a DOS program. Recall that the clock pulses for programming the PIC have to last at least 0.1 microseconds. In practice, they are somewhat longer in order to overcome any signal "bounce" in the cables. However, they shouldn't be too long or the programming process will go too slowly. It is also important that the pulse timing not depend on the speed of the computer's CPU. Because the software was written with that in mind, it will run on any IBM-compatible from a 4.77-MHz XT up to the latest Pentiums.

To achieve that, the programmer software uses one of the timers built into the PC motherboard. One of the PC's built-in timers produces an interrupt 18.2 times per second (65,536 times in a 24-hour day). That timer is used to update DOS's time-of-day clock, and some software uses it to manage any concurrent processes. However, 18.2 times per second is far too slow to be useful for PIC programming. The software instead uses the other timer that is normally used to control the tones in the internal speaker. The time delay available from that timer can be set to 25 microseconds, so that even on the fastest Pentiums, the programming pulses are not too short. There will be some unpredictable software overhead, so the pulses will come out a bit too long on the slowest PCs, but not long enough to do any harm.

Production-Grade Programming. As cheap and as simple as it is, the "No Parts" PIC Programmer can qualify as a production-grade programmer for confirming the reliability of your programmed PICs. How? By varying the 5-volt supply over the entire specified range while verifying the PIC. To do that, you'll of course need an adjustable 5-volt supply. First program and verify the PIC with the 5-volt supply set to 5.0 volts. Next, set the 5-volt supply to 6.0 volts and verify again. Finally, verify the PIC a third time with the 5-volt supply set to 4.0 volts.

Why does that guarantee reliability? Because the main source of unreliability in EPROMs of any type is that some of the cells might not be completely programmed or completely erased. If a particular location is only "half on" or "half-off", it might read correctly for a while but then shift to the wrong value with age or changes in the supply voltage. The result is that the ROM contents change unpredictably and the microcontroller fails during use. By shifting the 5-volt supply voltage up and down, you change the threshold voltages that define 0 and 1 so that you can detect marginally programmed bits. The cheapest commercial programmers don't have that feature at all. Higher-quality units do it partly in that they raise the supply voltage but not lower it. The "No Parts" PIC Programmer gives you full control over the supply voltage, making it easy to test any programmed part over the chip's full voltage range.

The author would like to acknowledge David Tait in England for his work on TOPIC, a PIC programming package of which the "No Parts" PIC Programmer is a direct descendant. Mr. Tait's work has been distributed on the Internet, and he has given his permission for this adaptation. The TOPIC software works with "No Parts" PIC Programmer hardware and vice versa. Because of that compatibility, the TOPIC package has been included with the NOPPP software on the Gernsback Web site (www.gernsback.com). Ω

SOURCES OF PARTS AND INFORMATION

Digi-Key Corporation
701 Brooks Avenue, South
Thief River Falls, MN 56701
1-800-DIGIKEY
http://www.digikey.com
Sells PIC microcontrollers and development systems.

Dontronics
P.O. Box 595
Tullamarine 3043
Australia
http://www.dontronics.com
Circuit boards, software tools, kits, and other supplies for PIC programming, many of them available nowhere else. Quick shipment all over the world.

Jameco Electronic Components
1355 Shoreway Road
Belmont, CA 94002
1-415-592-8097
http://www.jameco.com
PIC microcontrollers, other digital ICs, parts, and kits for the experimenter.

Microchip, Inc.
2355 W. Chandler Boulevard
Chandler, AZ 85224
http://www.microchip.com
Maker of PIC microcontrollers. Data books and free software distributed from web site and on CD-ROM.

Square 1 Electronics
P.O. Box 501
Kelseyville, CA 95451
squareone@zapcom.net
Publishes and sells two books, Easy PIC'n and PIC'n Up the Pace, which are excellent guides to PIC programming for the beginner.