An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

Getting Familiar with PICs and PicBasic

by Chuck Hellebuyck

A Brief History of the Microchip PIC

Although you don't need to know anything about Microchip to use their parts, it does help to have some history (even if you just wonder where the name "PIC" came from).

Microchip Technology incorporated was started by a group of venture capitalists who saw potential in the semiconductor division of General Instruments. General Instruments, which produced various electronic components, many years ago developed a series of programmable, high output current, input/output controllers. They called them **P**eripheral **I**nterface **C**ontrollers, or PICs. They were actually simple microcontrollers built around a RISC (reduced instruction set code) architecture. They ran efficiently at 1 instruction per clock cycle at a high oscillator frequency of 20 MHz. This made PICs relatively fast for an 8-bit micro, but their main feature was 20 ma of source and sink current capability on each I/O pin. Typical micros of the time were advertising high I/O currents of 1 ma source and 1.6 ma sink.

For some business reason, General Instruments decided that the semiconductor division was not worth keeping and sold the operations (along with the factory in Chandler, Arizona) to the venture capitalists. This group formed what is now known as Microchip Technology. Those PICs became the main components offered by Microchip.

Initially the selection of parts was small, and none of the parts had some common micro features like interrupts. They also used a somewhat unusual banking arrangement for memory that still exists today in many of Microchip's parts. Despite these limitations, the parts sold well and allowed Microchip to develop new components with new features including interrupts, on-board A/D, on-board comparators, and more. Microchip's lineup soon included flash memory parts as well as low cost OTP (one time programmable) parts. This low cost OTP feature set Microchip apart from their competitors. Other 8-bit micro companies offered OTP's but they usually came at a high price premium to the masked ROM (read only memory) version.

Masked ROM micros are built by placing layers of semiconductor atop each other to form the transistors and other components that make up a micro. The proper arrangement makes the micro operate according to the software. After a masked ROM is created, it cannot be changed. One

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

single software command change requires a new masked ROM. Microchip found a way to produce OTPs at only a small cost premium over their masked ROM parts. This allowed designers to use OTPs in final designs so small changes could be made without stopping production or spending more money for a new masked ROM. Microchip also made the parts serially in-circuit programmable. This allowed a manufacturer to build up electronic modules with an unprogrammed PIC on board and then program it right on the factory floor. That flexibility, along with the other features, made Microchip popular with professionals and experimenters.

Microchip has since grown to become the second largest producer of 8-bit microcontrollers. Microchip also expanded to become a leader is low cost, long life EEPROM memory and other niche markets. Microchip continues to develop new micros parts at a rapid pace with all the micros falling into three main categories; 12-bit core, 14-bit core and 16-bit core program memory. All the parts have a 8-bit wide data bus, which classifies them as 8-bit microcontrollers. No matter what your application, Microchip probably has a part that will work well with your design concept.

PIC Overview

The PIC family is divided into three main groups:

- 12 bit instruction core (16C5X, 12C5XX, 12CE5XX)
- 14 bit instruction core (16C55X,16C62X, 16C6X, 16C7X, 16C71X, 16C8X, 16F8X, 16F87X, 12C6XX, 16C9XX, 14C000)
- 16 bit instruction core (17C4X, 17C7XX)

All three groups share the same core set of RISC instructions with additional instructions available on the 14- and 16-bit cores. This means that assembly code written for the 12-bit family can be easily upgraded to work on a 14- or 16-bit core part. This is one of the great advantages to the PIC. All instructions (except branch and goto instructions) execute within one clock cycle (crystal frequency / 4), which makes it easy to check the execution timing. To fully understand all the features of the PIC family would require you to read the Microchip Databook which is about 2” thick and all in small print. Instead, I briefly summarize the parts below in Table 1 and descriptively as follows:

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

Device	ROM Words	EEPROM Bytes	RAM Bytes	# I/O	A/D	Timers	Misc.
12-bit Core							
12C5XX	0.5K to 1K		25 to 41	6	none	1+ WDT	8-pin package
12CE5XX	0.5K to 1K	16	25 to 41	6	none	1+ WDT	8-pin package
16C5X	0.5K to 2K		25 to 73	12 to 20	none	1+ WDT	18-pin, 28-pin package
14-bit Core							
12C67X	1K to 2K		128	6	4	1+ WDT	8-pin package
12CE67X	1K to 2K	16	128	6	4	1+ WDT	8-pin package
16C55X	.5K to 2K		80 to 128	13		1+ WDT	18-pin package
16C6X	1K to 8K		36 to 368	13 to 33		3+ WDT	18-pin, 28-pin, 40-pin package
16C62X	.5K to 2K		80 to 128	13		1+ WDT	18-pin package
16C7X, 71X	.5K to 8K		36 to 368	13 to 33	4 to 8	3+ WDT	18-pin, 28-pin, 40-pin package
16F87X, 8X	.5K to 8K (FLASH)	64 to 256	36 to 368	13 to 33	0 to 8	3 + WDT	18-pin, 28-pin, 40- or 44-pin package
16F9XX	4K		176	52	0 to 5	3 + WDT	64- or 68-pin package, built in LCD driver
14000	4K		192	20		1+ WDT	28-pin package
16-bit Core							
17C74X	4K to 16K		232 to 454	33		4+ WDT	40- or 44-pin package
17C7XX	8k to 16K		678 to 902	50		4+ WDT	64- or 68-pin package

TABLE 1

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

12-bit instruction core

This is the original core produced and is found in the most cost effective parts available from Microchip. They use only 33 assembly language instructions. These parts will not work with the PicBasic compiler because they only have a 2 byte stack for storing return addresses during jumps and goto's. The PicBasic compiler requires the 8-bit stack that the 14-bit core parts have. If you write in assembly though, this family of parts is cheap and quite powerful.

16C5X

This group contains some of the original PIC parts from Microchip. They are very useful parts and offer several variations. They come in three different packages: 18-pin, 28-pin (DIP or SOIC), or 20-pin package (SSOP) . They offer two different amounts of I/O to meet your design requirements, 12 I/O or 20 I/O. Code space is available in three sizes: 0.5K, 1K, or 2K for your program (instructions are 12 bits wide). This can be deceiving though because the mnemonic and operand are included in the same word. This means a program will probably use less code space than another micro such as a Motorola 8-bit micro. Available RAM is 25 or 73 bytes (the 73 bytes are available in a banking arrangement only which is somewhat annoying). One 8-bit timer is included with no interrupt on timer overflow. They all have one internal watch dog timer that runs on its own RC circuit so it works during low power sleep mode.

All parts also share high I/O current capability of 20 ma source / 25 ma sink, which is incredibly useful for driving LED's full bright or driving power transistors directly. They work over a large voltage range of 2.5 to 6.25 volts (it may be less if you choose the high frequency crystal option). They will run at crystal speeds of 20 MHz, which is an instruction time of 200 nanoseconds. Since most instructions execute within one clock cycle, very fast routines can be accomplished. These parts are limited to a two level stack. This makes nested subroutines difficult but not impossible because the program counter on all PICs can be written to.

The biggest difference between the 5X and the other PICs is the lack of interrupts. If you need an external interrupt, then you have to move up to the 14-bit core parts. Finally, these parts in sleep mode can draw as little as 1.0 µA, which is very useful for battery applications.

12C5XX, 12CE5XX

These parts also use the 12-bit core and have many of the same 5X features with one major difference—they come in an 8-pin package. These parts are very unique and are the smallest micro available today. Because of the 8-pin package constraint, creative solutions were required to allow for enough I/O to be useful. They have 1 input-only pin and five I/O, Vdd (B+), and Vss (gnd), for a total of eight pins.

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

To achieve this, Microchip developed an on-board oscillator. It's not as accurate as a crystal (or even resonator) but it works great if timing is not critical (worst case accuracy is about +- 7.5 % over -40 to +85 C). This eliminates the need for any external oscillator parts, although it can be used with a crystal or resonator at the cost of two I/O. These parts also share the MCLR reset pin with the I/O, so if you need that function you lose another I/O. Microchip did add a form of interrupt. It's a wake-up on state change interrupt. If the micro is in sleep mode, a change of state from low to high or high to low on any of 4 I/O will wake up the processor. This is not much different than a external reset on the MCLR pin, but it does allow simple switch hook-up to the micro. The 16CE5XX includes 16 bytes of EEPROM data space.

14 bit instruction core

The 14-bit core parts are the second generation parts microchip developed, with interrupts and other features. The clever thing Microchip did, however, is keep the footprint or pin-out the same between these parts and the 12-bit parts. They also kept most of the 12-bit core assembly code instructions. This was clever because it offered a direct upgrade from the 12-bit core parts to the 14-bit core parts without changing the circuit board or having to make a major software “tear up.”

Because of the added features, the number of assembly instructions increases by two for a total of 35. Microchip actually added four instructions and replaced two 12-bit core assembly commands with special function registers. The two instructions replaced by a special function register are the TRIS (port direction) and OPTION (special function). The four added instructions include two math function commands and two return commands. The two return commands include one return command for the interrupts and one for subroutine returns which can be nested deeper on the 14-bit core because the stack increases to eight levels. This increased stack size is necessary to use the PicBasic compiler. Table 1 lists the feature summaries for these parts. They also offer most of, if not all, the features any electronics hobbyist needs to develop micro based products.

16C55X

The 16C55X are pin-for-pin compatible with their 5X 12-bit core cousins with a major addition, namely interrupts. They also add 1 more I/O pin by sharing the TOCKI external clock pin (used for incrementing the 8-bit timer from an external source). The interrupts include the 12CXXX wake-up on state change interrupt along with a real interrupt pin for capturing an event. Also included is a timer overflow interrupt for the 8-bit timer. All the interrupts jump to a single redirection register so your main interrupt routine will have to bit test the interrupt flags within the INTCON register. Your program can mask any and all interrupts through the INTCON register also. A final difference is the I/O characteristics increase to 25 ma sink and source.

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

16C62X

These parts are similar to the 55X group but add two on-board comparators to the package. The 62X have 13 I/O and 0.5K, 1K, or 2K of 14-bit wide code space. They share all the features of the 14-bit core group including the interrupts. If you need comparators in your design then these could reduce your overall parts count.

16C6X

These parts were of the original 14-bit core group. They consist of several parts with unique features. They start out with the 16C61, which isn't much different than the 16C556 part, but the rest of the 16C6X group is very different. They add the following features to those parts mentioned above: 2K, 4K, or 8K of code space for your program, 22 or 33 I/O, synchronous serial port (shared with I/O), one or two Capture/Compare/ PWM pins (shared with I/O), and three timers (two 8-bit, one 16-bit). The 16-bit timer is great for accurate timing requirements. It can run from its own crystal separate from the main clock source. It will even run during sleep mode, allowing time to increment while very little current is being consumed by the PIC. It has an overflow interrupt so you can wake up from sleep process the timer information and then sleep some more. The synchronous serial port can be used to communicate with serial devices. It operates in two modes: 1) Serial Peripheral Interface (SPI), or 2) Inter-Integrated Circuit (I2C). These are very powerful parts.

16C7X, 16C71X

These parts are identical to their 6X cousin with the addition of four, five, or eight channels of 8-bit on-board analog to digital converter (A/D). For example, if your design uses a 16C62 and you need to add A/D, then drop a 16C72 in its place. They are pin-for-pin compatible with each other. The A/D channels are shared with some of the port A and Port E I/O pins, so its best to save these when doing a non-A/D design that may later need A/D. The 16C71X are upgraded versions of some 16C7X parts that add more RAM space.

16C67X

These parts are the 8-pin package versions of the 14-bit core group. They share the I/O the same way the 12CXXX 8-pin parts do, to maintain one input only and five I/O. The amazing thing is

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

that they also have four channels of A/D that operate the same as the 16C7X parts (shared with the I/O). Code that was written to work with the 16C7X A/D will work on the 16C67X parts. They also have all the 14-bit core interrupts. It also has one 8-bit timer with timer overflow interrupt and built-in oscillator option. They offer 0.5K and 1K of code space; this is a lot of micro in a small package.

16C8X,16F8X

If you're looking for a flash or EEPROM version of the PIC, this is the group. Originally Microchip only offered EEPROM versions (16C8X), but have now released them in flash (16F8X). They have all the features of the base 14-bit core group, including interrupts, 13 I/O, one 8-bit timer, 0.5K or 1K of code space as EEPROM or flash, and 36 or 68 bytes of RAM. Unique to these parts is the 64 bytes of EEPROM data memory. This data will stay even when power is removed, so its great for storing calibration or variable data to be used when the program starts again. They are very handy for development because they can be programmed over and over again without ever leaving the circuit.

16F87X

This is one of the newest groups of parts from Microchip. They have FLASH program memory so they can be reprogrammed over and over again. They are built to be identical to the 7X family of parts with some data memory and program memory updates. They offer 13 to 33 I/O, three timers, and up to 8K of program memory. They have all the special functions of the 6X and 7X parts as mentioned earlier. These are new as I write this, but will be very popular with hobbyists and professionals for project development and maybe even production.

16C9XX

This part shares many of the 16C63 and 16C73 features (three timers, interrupts, etc.) but adds another feature, on-board Liquid Crystal Display (LCD) drive circuitry. It can drive up to 122 segments using four commons. The 16C924 also has five channels of A/D on-board, making this a great part for measuring analog signals and then displaying the results on an LCD. With the 16-bit timer it could display time for possible datalog applications. With the synchronous serial port, any kind of external data storage or PC interface is possible. These parts seem to have it all except on board EEPROM for non volatile memory storage.

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

14C000

This is a different numbering scheme and offers a different approach. It's a mixed signal processor. It has a slope type A/D, rather than the sample and hold, and also has D/A capability. It shares the higher end 14-bit core characteristics including the three timers and such. These are unique parts when compared to the rest of the PICs but share the same code.

16 bit instruction core

This is the high end group from Microchip. They offer up to 33 MHz clock speed for a 121 nanosecond instruction time. They have the same 35 instructions as the 14-bit core plus 23 more instructions. The stack increases to 16 levels. 33 I/O is standard with two open drain high voltage (12 V) and high current (60 ma) pins. They add another 16-bit timer for four total timers. These parts can also operate as a microprocessor rather than a microcontroller by accessing the program to be executed from external memory. These are not the parts to start experimenting with until you've mastered the 12- or 14-bit core parts. If you are rea;;y experienced with other micros then you may be able to use them right away, but I recommend using C language or assembly language. The PicBasic compiler is not really designed to be used with these parts.

Software for PICs

A micro is nothing without software and to program PICs requires a binary file of coded ones and zeros. Microchip offers an assembly language for PICs and a free assembler to get you going. Assembly language can be tough for a beginner, though. What is easier for a beginner or hobbyist is a higher level language and a compiler to convert that higher level language into an assembly language program. PicBasic is a higher level language that is easy for beginners, hobbyists, and even professionals to use for simple code development and rapid " prove out" of a concept. I recommend it and use PicBasic often. I also write in assembly and recommend everyone learn it at some point, but PicBasic is a great way to start. Since this article is intended for hobbyists, I'll just touch on assembly below and then dive into the guts of PicBasic.

Assembly Language

All micros run on simple binary codes. These codes are various arrangements of ones and zeros. Assembly language is a higher level language to this binary code. Microchip PICs have their own set of assembly commands. These commands when combined as a program are assembled by a software program called an assembler. The assembler outputs a file in the binary command form the micro uses. Microchip offers a free assembler for software writers to assemble their programs. The file produced by the assembler for PICs uses the Merged Intel Hex format, or INHX8M, and

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

is given the .hex file suffix. This .hex file is what the PIC programmer tool uses to burn the program into the PICs program memory. Assembly commands, although easier to understand than binary code, can be difficult to understand and can take a beginner months of practice to get a program to work. That's why higher level languages such as PicBasic become popular. At some point though, you'll need to do something with the PIC that PicBasic or any higher level language won't do. That's when you may want to use assembly language. Sometimes a single assembly language command can solve the problem. PicBasic fortunately has the capability to mix assembly code within the PicBasic program. I've written hundreds of programs in PicBasic and never had to use assembly language but it helps to know it's there when you really need it.

PicBasic Compiler

In 1995, a company named Parallax incorporated developed a small computer module based on the PIC that could be programmed in a modified version of the BASIC software language. Parallax Inc. had been producing programmers and emulators for the Microchip PICs but saw a potential to make PIC based design easier for everyone. They knew that assembly language programming was difficult for the beginner and hobbyist so they decided to develop a form of BASIC language called PBASIC. They developed the computer module around a PIC 16C56 part and called it the BASIC Stamp. The module used external EEPROM memory to store the programs and the PIC retrieved commands from that memory one at a time and executed them. This is known as interpreted execution which the BASIC language is famous for.

Although this isn't the fastest way to run a program, it became popular with many experimenters, electronic hobbyists, and even professional technical people. It offered a totally new approach to programming PICs that was simple, and quick. It wasn't long before some users were asking if working programs could be compiled into assembly language so a PIC could be programmed instead of the somewhat expensive computer modules. Micro Engineering Labs answered the call. They developed a PicBasic compiler or PBC that would take a working PBASIC programs and covert it into the Merged Intel Hex format required to program a PIC. They added more commands as well to increase the capabilities of PicBasic. It really made PIC based development easy. The compiler works with all the 14-bit core parts mentioned above and when compiled a program will run around 15 times faster than the same program running on the Parallax module.

Because it's compiled code rather than directly written in assembly, it isn't as efficient as an assembly language program but it can be close. The true advantage is reduced software development time. Programs that may take weeks or months to write in assembly can now be written in days or weeks in PicBasic. For the professional this offered quick concept “prove out” or even rapid production. For the hobbyist or experimenter, it offered quick project development and a shorter software learning curve.

PBC allows access to any register within the PIC including all the I/O ports. Commands for I2C communication are included, and assembly language inserts are possible. Because you can choose

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

various PICs, more variables are available in PICs with more RAM. The PBC instruction set is shown in List 1. Some of these commands will get used in every program you write while others will only get used in specific applications. The list may seem extensive, but in time you'll find they are easy to remember and easy to understand.

ASM..ENDASM - Insert assembly language code section.
BRANCH - Computed GOTO (equivalent to ON..GOTO).
BUTTON - Debounce and auto-repeat input on specified pin.
CALL - Call assembly language subroutine.
EEPROM - Define initial contents of on-chip EEPROM.
END - Stop execution and enter low power mode.
FOR..NEXT - Repeatedly execute statement(s).
GOSUB - Call BASIC subroutine at specified label.
GOTO - Continue execution at specified label.
HIGH - Make pin output high.
I2CIN - Read bytes from I2C device.
I2COUT - Send bytes to I2C device.
IF..THEN - GOTO if specified condition is true.
INPUT - Make pin an input.
LET - Assign result of an expression to a variable.
LOOKDOWN - Search table for value.
LOOKUP - Fetch value from table.
LOW - Make pin output low.
NAP - Power down processor for short period of time.
OUTPUT - Make pin an output.
PAUSE - Delay (1mSec resolution).
PEEK - Read byte from register.
POKE - Write byte to register.
POT - Read potentiometer on specified pin.
PULSIN - Measure pulse width (10 µs resolution).
PULSOUT - Generate pulse (10 µs resolution).
PWM - Output pulse width modulated pulse train to pin.
RANDOM - Generate pseudo-random number.
READ - Read byte from on-chip EEPROM.
RETURN - Continue execution at statement following last executed GOSUB.
REVERSE - Make output pin an input or an input pin an output.
SERIN - Asynchronous serial input (8N1).
SEROUT - Asynchronous serial output (8N1).
SLEEP - Power down processor for a periond of time (1 Sec resolution).
SOUND - Generate tone or white-noise on specified pin.
TOGGLE - Make pin output and toggle state.
WRITE - Write byte to on-chip EEPROM.

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

List 1
PBC Instruction Set

How It Works

Here's how PBC works. You write your program in PicBasic using any text editor you like. When your done save the file and then issue the command:

```
C:\>PBC filename
```

The PicBasic compiler then converts the file into a source file. In it, each PicBasic command is broken down into an assembly language subroutine and grouped together to form that source file. The source file is then assembled by PBC into the binary file format as filename.hex. Most PIC programmers use that format officially known as Merged Intel Hex format (INHX8M). From here you need a PIC programmer. I talked about programmer options in the December 1998 *Nuts & Volts*, and I've even added one to my web site at www.elproducts.com. Use that HEX file generated by PBC with a PIC programmer and in a few seconds your program is inside the PIC ready to be inserted into a circuit.

PBC requires the 8-level stack used in the 14-bit core PICs to store redirect addresses and gosub return locations. Because of that requirement PBC cannot be used with the 16C5X and 12C5XX PICs. That may seem like a limitation but look again. Microchip offers 14-bit versions of the 5X named 16C55X. The 12C5XX can be replaced with the 12C67X. The PIC options are huge since the 14-bit core family is the biggest family of parts in the overall PIC offering. . The list of PICs supported by PBC currently include 16C554, 556, 558, 61, 62, 62A, 620, 621, 622, 63, 64, 64A, 65, 71, 710, 711, 715, 72, 73, 73A, 74, 74A, 84, 923, 924,l and the 16F83, 84 along with the 14000. Obviously you have a lot of choices. The command format is:

```
PBC  options  filename
```

PBC defaults to the 16F84 Flash ROM PIC that is probably the most popular PIC with hobbyists. It can be reprogrammed over and over again without erasing it first. To use a PIC other than the 16F84, you have to tell the compiler which PIC profile to use. You do that with the -P option.

```
PBC -P622  filename
```

The command above will compile the filename.bas into a HEX file for the 16C622 PIC

PBC also remembered its roots. Add the -C option and the compiler only allows the original Parallax commands and variable names to be recognized. PBC also offers the -D option that creates a symbol table file, a source listing file and a map file for anyone needing more detail. The -OB option generates a binary file rather than the Merged Intel HEX format if you need that. The

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

-S option stops the assembler from invoking thus leaving the just the source file. You probably won't use most of these options, but I mention them so you understand that PBC is more than just a simple PicBasic to hex file converter.

Using PBC

Now that we've covered the fundamentals lets put PBC into action. Lets say you want to turn on an LED that's connected to pin RB0. A brief assembly language example to set bit 0 of port B to a high state looks like this:

```
bsf   STATUS,RP0             ; Move to register bank 1
movlw 0FF                    ; First make all pins of PORT B
movwf TRISB                  ; high impedance inputs
bcf   STATUS,RP0             ; Move to register bank 0
movlw  01                    ; Set bit 0 of PORT B
movwf PORTB                  ; to high.
bsf   STATUS,RP0             ; Move to register bank 1
movlw 0FE                    ; Set PORT B pin 0 to output
movwf  PORTB                 ; and the rest of the pins to inputs
bcf   STATUS,RP0             ; Move back to bank 0
```

Although this probably isn't the most efficient way to do this in assembly language, it does show the several main steps required. The same function in PicBasic looks like this:

```
high  0                             ; Set PORTB pin 0 to high
```

What a time saver and its much easier to read and understand two months from now! When the commands get more involved, such as serial communication, the assembly code file gets bigger but the PicBasic command takes just one line. This explains why PicBasic is more efficient for you personally, but this is also where the inefficiency of higher level languages appear when you have limited code space in your micro. Some assembly language commands within PBC command subroutines could be shared but aren’t because of the structure. The author of any compiler program tries to keep those inefficiencies to a minimum, but it's almost impossible to get rid of them all. That's the price we pay for quick, easy to follow program development.

I've found the PicBasic compiler to be quite efficient. I do a lot of development with the 16F84 flash PIC which has only 1K of ROM space. When I've run out of space, simple modifications to my PicBasic program allowed some complex routines to fit. What really helps development time is the vast array of commands PicBasic offers. Serial RS232 type communication, lookup tables, math functions are just some of the complex features PicBasic has reduced down to a single command.

Flash A LED

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

This is always a popular first project. We'll use the "high" command mentioned above along with a few others from Table 1. The software is shown as Program 1. This program is written to work with the schematic in Figure 1. It will work on any PIC but I chose to use the default 16F84.

Program 1

```
'Flash an LED
'Flash the LED on RB1 on and off every second.

symbol        LED =  1        ' Port LED is connected to.

Loop:
      high  LED         ' LED is on
      pause 1000        ' Wait 1 second
      low   LED         ' LED is off
      pause 1000        ' Wait another second
      goto  loop        ' Let's do it again making the LED flash
```

Although its simple, many beginners and even experienced hobbyists will get some satisfaction the first time you compile/assemble this program, program it into a PIC, and then watch it work when the PIC is powered up. Notice how the label "Loop:" starts with a capital and in "goto loop" the L is lower case. This works because PBC is case insensitive. "LED" and "led" look the same to PBC.

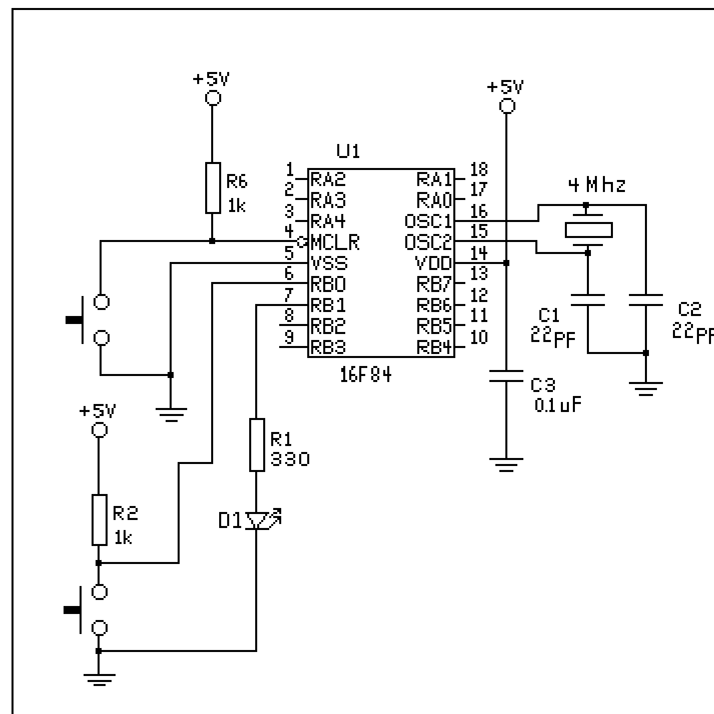

Figure 1
Flash LED Schematic

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

Analog To Digital Converter

This time lets use a PIC 16C711 with on board A/D converter. For this we have to access special registers of the 711. That's just what the PEEK and POKE commands were designed for. Program 2 shows the software. This is actually a sample program that comes with PBC. It reads the voltage (0 to 5 volts) as an 8-bit value of 0 - 255 decimal on A/D port 0 which is PIC pin 17 (RA0/AN0). That value is stored in variable B0. The content of B0 is then sent out in RS232 8N1 inverted format at 2400 baud that can be received by almost any PC. Make sure you use the -P711 option when you invoke PBC. The schematic for this is shown in Figure 2.

Program 2

```
' PEEK and POKE Commands
'
' Access PIC16C71 A/D using Peek and Poke

Symbol  ADCON0 = 8                    ' A/D Configuration Register 0
Symbol  ADRES = 9                     ' A/D Result
Symbol  ADCON1 = $88                  ' A/D Configuration Register 1
Symbol  SO = 0                        ' Serial Output

      poke ADCON1, 0                ' Set PortA 0-3 to analog inputs
      poke ADCON0, $41              ' Set A/D to Fosc/8, Channel 0, On

Loop: poke ADCON0, $45              ' Start Conversion
      pause 1                       ' Wait 1ms for conversion
      peek ADRES, B0                ' Get Result to variable B0

      serout SO,N2400,(#B0,10)      ' Send variable to serial out

      goto Loop
```

Note how B0 was never defined as a symbol. PBC uses the same base variable names as the original Parallax module. The Parallax module has 13 byte (B0-B13) or 6 word (W0-W6) variables. The 16C711 has 68 bytes of RAM so it allows 51 byte (B0-B51) or 25 word (W0-W25) predefined variables. This program also accesses the PORT A I/O of the PIC. This leaves the full PORT B pins for other functions such as the serial communication port. PORT A plus PORT B results in 13 total I/O for any of the 18-pin PICs. To really understand this program you have to understand the Microchip data sheets for the 16C711 A/D module. It can be confusing at first, but after a while you'll find this stuff is easy.

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

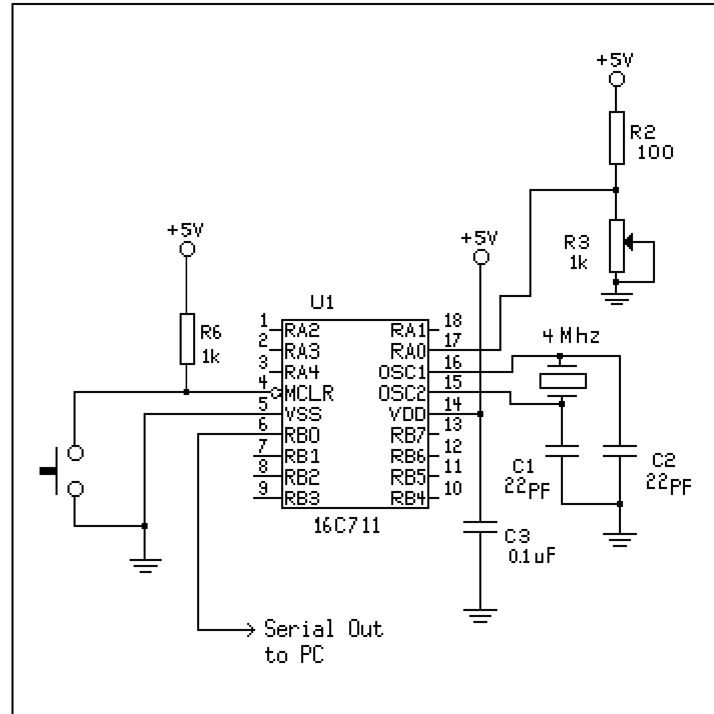

Figure 2
A/D Schematic

If you have questions there is a PicBasic listserver. To add your email address to the list send a message to: majordomo@qunos.net. In the message body enter: subscribe picbasic-l
It will then reply with a email message to verify your email address and ask you to reply.
Once this is done you can send messages to and receive messages from picbasic-l@qunos.net.
There are many people including myself ready to help you out.

PicBasic PRO

Now if your really serious about PICs but want to stay with PicBasic, microEngineering Labs has introduced a professional version of the PicBasic compiler called PicBasic Pro (PBP). microEngineering Labs had so many things they wanted to add to PBC that it became a compiler in a league by itself. PBP added many more features to PBC. The main benefits of PBP over the PBC are:

- Interrupts in BASIC.
- Programs longer than 2K.
- Arrays.
- Direct access to 16 I/O without using PEEK and POKE.
- Direct access to special function registers without using PEEK and POKE.

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

- There is also a way to tell Pro which clock oscillator you want to operate at instead of the 4 MHz PBC expects.
- Pro can also be used with Microchips assembler MPASM for better ICE and simulator compatibility.
- Any variable designated as bit, byte, or word.

It is really a full featured compiler. By the time you read this, microEngineering Labs should have announced the release of additional support for the 17CXXX high performance 16-bit core PICs. List 2 has the current PBP commands.

@ - Insert one line of assembly language code.
ASM..ENDASM - Insert assembly language code section.
BRANCH - Computed GOTO (equiv. to ON..GOTO).
BRANCHL - BRANCH out of page (long BRANCH).
BUTTON - Debounce and auto-repeat input on specified pin.
CALL - Call assembly language subroutine.
CLEAR - Zero all variables.
COUNT - Count number of pulses on a pin.
DATA - Define initial contents of on-chip EEPROM.
DEBUG - Asynchronous serial output to fixed pin and baud.
DISABLE - Disable ON INTERRUPT processing.
DTMFOUT - Produce touch-tones on a pin.
EEPROM - Define initial contents of on-chip EEPROM.
ENABLE - Enable ON INTERRUPT processing.
END - Stop execution and enter low power mode.
FOR..NEXT - Repeatedly execute statements.
FREQOUT - Produce up to 2 frequencies on a pin.
GOSUB - Call BASIC subroutine at specified label.
GOTO - Continue execution at specified label.
HIGH - Make pin output high.
HSERIN - Hardware asynchronous serial input.
HSEROUT - Hardware asynchronous serial output.
I2CREAD - Read bytes from I2C device.
I2CWRITE - Write bytes to I2C device.
IF..THEN..ELSE..ENDIF - Conditionally execute statements.
INPUT - Make pin an input.
{LET} - Assign result of an expression to a variable.
LCDOUT - Display characters on LCD.
LOOKDOWN - Search constant table for value.
LOOKDOWN2 - Search constant / variable table for value.
LOOKUP - Fetch constant value from table.
LOOKUP2 - Fetch constant / variable value from table.
LOW - Make pin output low.
NAP - Power down processor for short period of time.

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

ON INTERRUPT - Execute BASIC subroutine on an interrupt.
OUTPUT - Make pin an output.
PAUSE - Delay (1 mSec resolution).
PAUSEUS - Delay (1 µSec resolution).
PEEK - Read byte from register.
POKE - Write byte to register.
POT - Read potentiometer on specified pin.
PULSIN - Measure pulse width on a pin.
PULSOUT - Generate pulse to a pin.
PWM - Output pulse width modulated pulse train to pin.
RANDOM - Generate pseudo-random number.
RCTIME - Measure pulse width on a pin.
READ - Read byte from on-chip EEPROM.
RESUME - Continue execution after interrupt handling.
RETURN - Continue execution at statement following last executed GOSUB.
REVERSE - Make output pin an input or an input pin an output.
SERIN - Asynchronous serial input (8N1) (BS1 style with timeout.)
SERIN2 - Asynchronous serial input (BS2 style.)
SEROUT - Asynchronous serial output (8N1) (BS1 style.)
SEROUT2 - Asynchronous serial output (BS2 style.)
SHIFTIN - Synchronous serial input.
SHIFTOUT - Synchronous serial output.
SLEEP - Power down processor for a period of time.
SOUND - Generate tone or white-noise on specified pin.
STOP - Stop program execution.
SWAP - Exchange the values of two variables.
TOGGLE - Make pin output and toggle state.
WHILE..WEND - Execute code while condition is true.
WRITE - Write byte to on-chip EEPROM.
XIN - X-10 input.
XOUT - X-10 output.

List 2
PBP Instruction Set

Interrupts In Basic

Program 3 is an example of using "Interrupts in Basic" feature available in PBP. I've read many emails from Parallax module users wishing they had access to interrupts because it adds a whole background process to any program. This is a very useful option. In this example notice the difference in variable declaration and the lack of declarations for special function registers like the OPTION register. These are some of the features PBP added to make writing the program easier. This program will run on the circuit in Figure 1.

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

Program 3

```
' On Interrupt - Interrupts in BASIC
'  Turn LED on. When switch is closed, interrupt main program and
'  turn LED off. Program waits .5 seconds and turns LED back on.

led     var     PORTB.1         ' LED on PortB bit 0


        OPTION_REG = $7f        ' Enable PORTB pullups and
                                ' interrupt on RB0 falling edge

        On Interrupt Goto myint ' Define interrupt handler
        INTCON = $90            ' Enable INTE interrupt

loop:   High led                ' Turn LED on
        Goto loop               ' Do it forever


' Interrupt handler
        Disable                 ' No interrupts past this point
myint:  Low led                 ' If we get here, turn LED off
        Pause   500             ' Wait .5 seconds
        INTCON.1 = 0            ' Clear interrupt flag
        Resume                  ' Return to main program
        Enable
```

Summary

PBP is an article in itself but I couldn't talk about PicBasic without including PBP. PBP is much more expensive than PBC. PBC costs $99 and PBP costs $249, but I've yet to hear anybody complain about the price after they've used it. Some people swear by PBP but I still use both. Many times I just use PBC with a 16F84. It's quick, easy, and still very powerful. I realized that it was the best way to get started with PicBasic and PICs so I negotiated with microEngineering Labs and they kindly agreed to produce a Special Edition of PBC dedicated to the 16F84. I grouped that with a PIC programmer and a 16F84 PIC so anyone can get started for around $100. Check it out at www.elproducts.com. I also have been contracted by LLH Publishing to write a book about PICs and the PicBasic compiler to pass on more of my experience for anyone interested in using PICs and PicBasic. Look for an announcement of its publication here at www.Hobby-Electronics.com or www.LLH-Publishing.com.

PICs and PicBasic has brought back the fun electronics was when I was just a kid. When I was young, I learned a lot from building the projects featured in electronic hobbyists magazines. I hope to publish more articles about some of the projects that clutter my bench. Between future articles, the book, and pages on my web site dedicated to PBC and PBP, I hope I can help you have as much fun with PICs and PicBasic as I do.

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com

An article downloaded from **www.hobby-electronics.com**
Copyright © 1999. All rights reserved.

PIC® and PICmicro® are a registered trademark of Microchip Technology.

PicBasic® is a registered trademark of microEngineering Labs.

About the author:

Chuck Hellebuyck has worked as an engineer by profession for over 15 years in the automotive industry and spent many years as a youngster learning electronics from books and magazine articles. He also operates a internet-based business at **www.elproducts.com** where he designs various products and markets them. He also offers help to hobbyists that want to build electronic projects of their own with information on his web site and articles he publishes in various electronics magazines.

Let him know what you thought about this article via email at **chuck@elproducts.com.**

Hobby-Electronics.com is a publication of LLH Technology Publications
www.LLH-Publishing.com