How to Choose a MicroController ?

Step 1: What IS a "Microcontroller" ?

If you ever took a very introductory computer course, you probably learned about the major components of ANY computer:

A **Central Processing Unit** or CPU. The part that actually performs logic and math
Memory. Where the computer stores data and instructions
Input and Output or I/O. How the computer moves data between its other components and the real world.

A **microprocessor** uses microelectronic fabrication techniques to shrink the CPU to a very small size; usually a single "chip."

A **microcontroller** uses the same techniques to shrink the entire computer to a single chip (or very small module.) CPU, Memory, and I/O all in a little package as small as a grain of rice. Just connect up power and it starts doing its thing; computing and talking to the world. Usually the I/O on a microcontroller is aimed at "low level" hardware like talking to individual switches and LEDs instead of keyboards, internets, and displays (like your desktop computer.) A microcontroller is just the thing you want, if you want to talk to individual switches and LEDs...

Step 2: Show Stoppers

There are a number design considerations that might immediately reduce your number of choices a great deal.

Programability and Reprogramability:

At this point in time, I would say that a hobbyist should only consider microcontrollers that have internal flash or eeprom program memory and can be erased and reprogrammed a substantial number of times. There are also micros that can be used with external memory (adds complexity and expense), UV erasable micros (usually quite expensive due to the special packaging), one-time programmable chips (potentially usable after you have a working design, but losing their price advantage anyway), and mask-programmed chips (essentially useless.)

Peripherals:

If you want your microcontroller to have built in Ethernet, CAN, USB, or even multiple serial ports, many common choices are going to be eliminated. Some peripherals can be handy to have: UARTs, SPI or I2C controllers, PWM controllers, and EEPROM data memory are good examples, even though similar functionality can frequently be implemnented in software or external parts.

It's convenient if output pins can supply reasonable amounts of current for driving LEDs or transistors directly; some chips have 5mA or less drive capability.

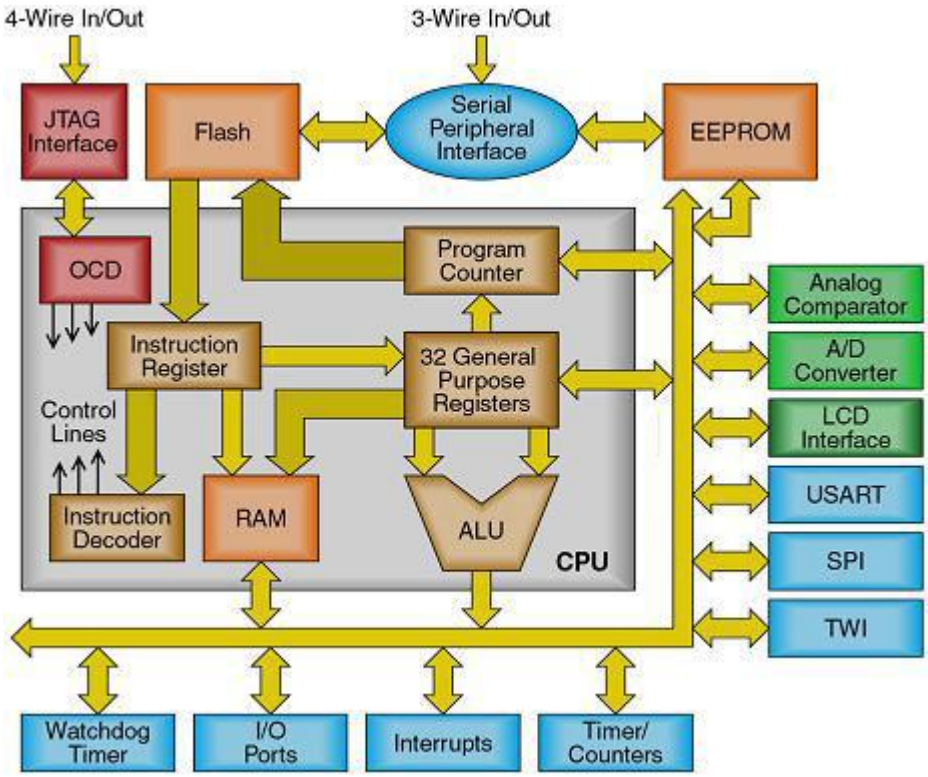
Physical packaging:

Hobbyists are somewhat limited is the packages they are able to deal with, from a prototyping, PCB fabrication, and/or soldering. That 256 ball BGA coldfire CPU may be sweet, but it's effectively unusable by mere mortals. Microcontrollers available in 0.1in DIP packages are much easier to deal with. Probably anything with a pin spacing smaller than 0.05in should be viewed with some skepticism.

Memory:

The size of memory may be an important consideration. Some micros have as few as 256 innstructions and 16 bytes of RAM. You can do a lot in that sort of space (believe it or not), but not everything. Moreover, while you can frequently upgrade to a "bigger chip", some chip families have relatively small memory limits imposed by their architecture, so even the biggest chip you can get will have only 8k instructions (for example.) RAM (data memory) is usually scarce in microcontrollers; some algorithms require substantial RAM to be implemented in a straightforward manner, and it may be worthwhile looking for a micro with a lot of RAM (or external RAM expansion capabilities) if that's the sort of thing you had in mind. (For instance, implementing TCP/IP networking protocols in a micro whose total RAM space is less than used by a typical ethernet packet is likely to be ... interesting.)

Step 3: Thoughts About Architectures



The "architecture" of a microcontroller refers to the philosophy of the internal implementation, sort of. It includes details like how many "registers" there are, and how "general purpose" those registers are, whether code can execute out of data memory, whether the peripherals are treated like memory, registers, or yet something else, whether there is a stack and how it works, and so on.

You will hear people talk about how some architectures are **better** than others. I suppose this is true. But I'm going to claim that it is largely irrelevant for the typical hobbyist. If you wish to avoid the architectural quirks of a particular family of microcontrollers, you can use a high level language. If you're willing to program in assembler, you're at a level of interest where learning and overcoming the quirks is party of the stuff you should be learning. Besides, we're not talking here about evaluating some new architecture that some company is proposing. All of the chips I mention in this instructable have been around long enough that they've **proven** that their architectures are good enough for MANY real applications.

Here are some architectural "features" you may read about and some explanation of what they mean.

CISC/RISC. Complex Instruction Set Computer. Reduced Instruction Set Computer.

In the old days, CPU designers were getting clever and wanted their CPUs to support high-level language features in hardware, leading to cobol-like string manipulation instructions that accepted arguments in blocks of 8 registers. IIRC, it was Berkeley and IBM who noticed that compiler writers didn't really know how to USE such complicated instructions from a compiler, the amount of silicon real estate used by these instructions was getting large, and in fact the hardware implementation sometimes wasn't as fast as doing the same thing in software anyway. So they said "I bet we can make the CPU go a lot faster if we leave out these complex instructions and dedicate the silicon to more registers or cache memory or something", and thereby invented the RISC CPU. Nowadays "RISC" is widely used by marketing departments to mean "we don't have very many instructions", even if the rest of the architecture isn't very much like the original RISC researchers had envisioned.

Harvard Architecture.

In a Harvard architecture, the instruction memory and the data memory are separate, controlled by different buses, and sometimes have different sizes. For microcontrollers, the instructions are usually stored in "read only" memory, and data is in RAM or registers. An example is the
PIC microcontroller, where instructions are in 12, 14, or 16 bit wide flash, and data is in 8bit
wide RAM.

Von Neuman Architecture.

In a von Neuman Architecture, data and instructions share memory space, so you could do things like dynamic compilation to generate instructions in RAM and then execute them. The TI MSP430 is an example of a von Neuman architecture.

Accumulator based

In an "accumulator based" architecture, there is usually one "special" register where most of the actual computation (math, logic, etc) occurs. Some effort has to be spent to get operands into the accumulator and results back out to where you need them. The opposite is a processor with "general purpose" registers, where any of several registers can be used for math/etc.

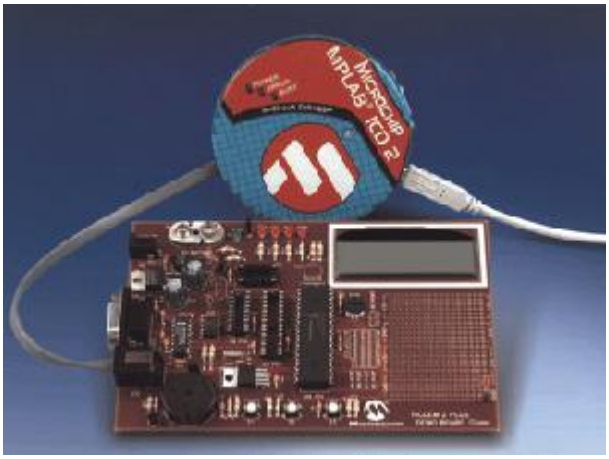
Load/Store

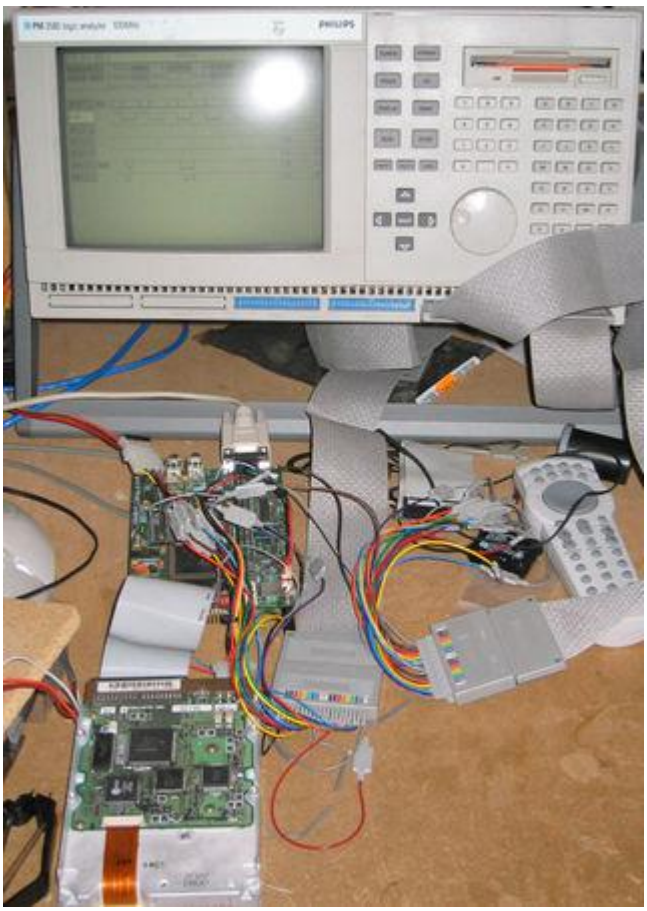
A load/store architecture typically means that operands to ALU operations have to be in registers, so that you have to load them from memory beforehand and store them back when operations are complete. The opposite typically allows operations between a register or
accumulator and a memory location directly. RISC architectures tend to be load/store, since a lot or the CISCness of CISC processors shows up in implementing complex "addressing modes" for accessing the memory operands. In a RISC architecture, all that complexity only has to be implemented in the load and store instructions (if at all.)

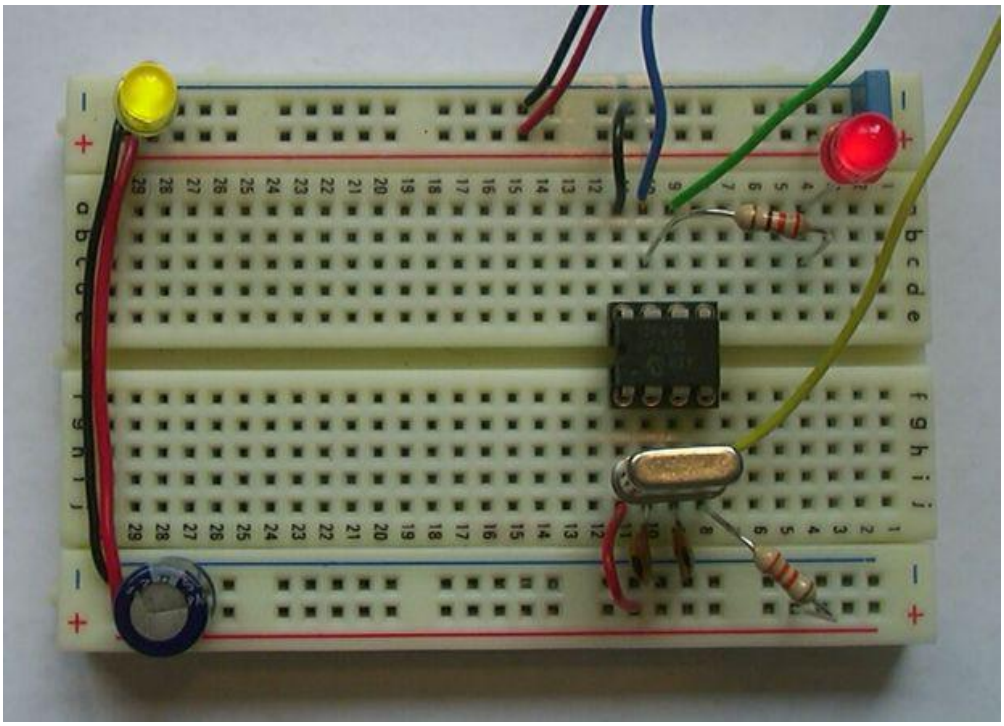
Registers

Memory Mapped I/O

Step 4: Thoughts About Hardware Tools


Microchip
MPLAB ICD 2


PHILIPS

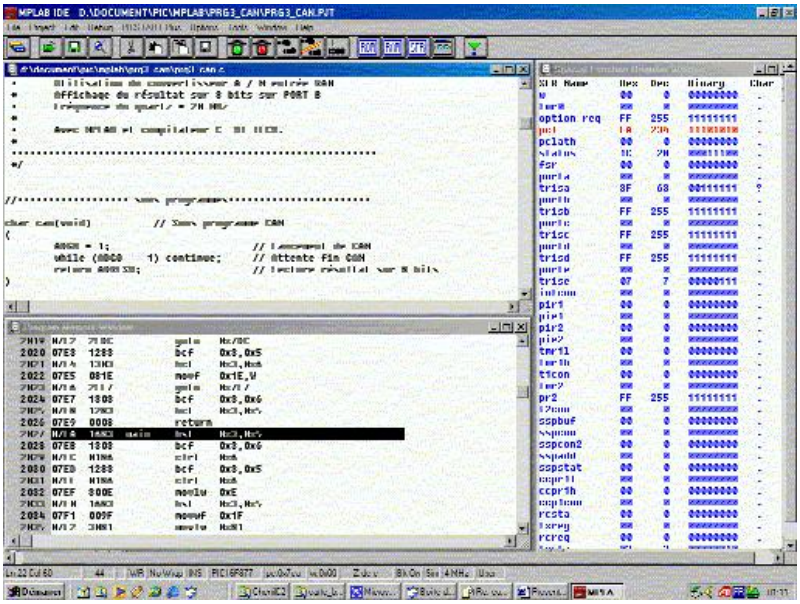


Minimally, you'll need some sort of programmer to load the program into the microcontroller.
These vary wildly in cost. It's pretty common for manufacturers to offer some low-cost programmer so that people can "evaluate" their microcontrollers without risking too much cash (or requiring director level approval :-) "low cost" used to mean "less than $500" (which happens to be a common "expensable" limit), but the number has been going down. I think you can find manufacturer supplied development tools for most of the microcontrollers mentioned here for
$50 or less.

Cheaper tools can be bought from third parties, or built yourself. If you don't mind the possibility that bugs in your tools might cause more problems than bugs in your code.

Maximally, production gang programmers, In circuit emulators, logic analyzers, oscilloscopes, lab-bench power supplies... You can spend as much money as you've got, probably.

Step 5: Thoughts About Software Tools



All of the microcontrollers mentioned here have some level of standard tools (at least an assembler) provided by the manufacturer. Most have "Integrated Development Environments" (IDE) that allow integrated use of an editor (that you won't like) with the assembler, some compilers, and a simulator (of perhaps limited value.) Some have significant additional support from the open source movement.

Tools from manufacturers aren't always great, but you appreciate them more after you've tried to use a microcontroller that doesn't have any such tools available...

Step 6:

Step 7: Microchip PIC Microcontrollers

The Microchip PIC microcontrollers were perhaps the first that were marketed to the hobbyist and student community, one of the first microcontrollers to be offered in a relatively small package (18 pin DIP) and one of the first to implement flash or eeprom program memory (in the PIC16C84 in 1993)

Architecture: harvard, accumulator based (mostly)
Package sizes: 6, 8, 14, 18, 20, 28, 40, ... 100
Program memory size: 256 words (12bit words = one instruction) to 256k bytes (2 bytes=1 instruction)
Data RAM: 16 to ~3900 bytes (4096 byte address space, shared with peripheral registers.)
Special features: EEPROM, 20mA output drive, several "sub-architectures"

Flash memory based PIC microcontrollers curently range from vanishingly small 6-pin chips in SOT23 packages to 100pin TQFPs. The basic architecture has been extended to chips with 16bit ALUs and integral DSP functionality.

Microchip has a liberal sample policy and chips are widely available from many vendors.

Step 8: Resources for Microchip PIC

Microchip the Manufacturer

PICList mailing list repository of knowlege

[http://techtrain.microchip.com/masters2004/(kgmnvafutocq2355egt11231)/downloads/classlist.htm Microchip Masters Conference 2004 Downloads] Tutorials and presentations

[http://techtrain.microchip.com/masters2005/(kgmnvafutocq2355egt11231)/downloads/classlist.htm Microchip Masters Conference 2005 Downloads] Tutorials and presentations

Starting with PICmicro Controllers Tutorial

PIC Elmer 160 Another Tutorial

WinPicProg Tutorial Tutorial w hardware designs.

PIC vs. AVR smackdown A comparison between PIC and AVR

There are a bunch of existing Instructables that use use Microchip PICs:

Intro to PIC Microcontrollers

PIC micro Timer Code

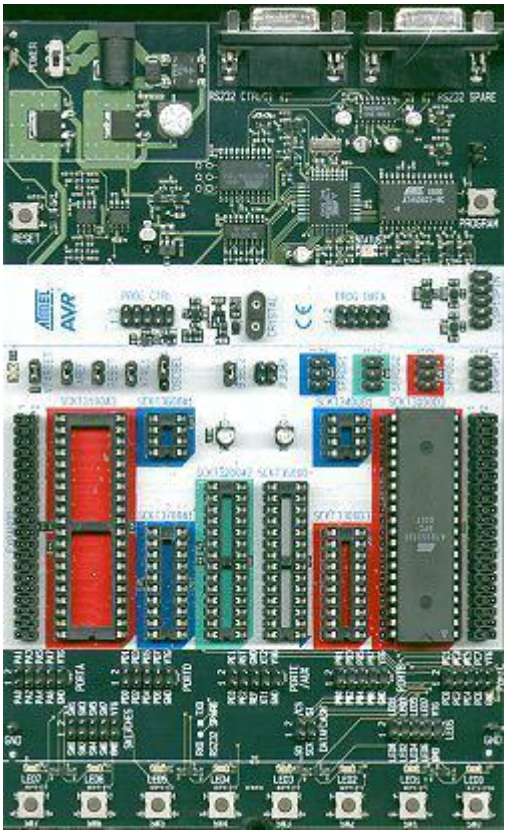
"TouchTimer" EggLight

Business Card PIC Programmer

Buildng the Inchworm ICD2 PIC Programmer / Debugger

Other PIC Instructables

Step 9: Atmel AVR



As near as I can tell, Atmel came along and decided to steal some of Microchip's business by offering "similar but better" chips. Some things they did right, some things they didn't do so well. But the Atmel AVR chips have also gained a lot of popularity among

hobbyists, and we get to cash in on the sometimes-war between Atmel and Microchip. My subjective observation is that microchip seems to do low-end chips a bit better, and Atmel does high-end chips a bit better.

Architecture: harvard, GP register based (mostly)
Package sizes: 8, 14, 20, 28, 40, ... 100
Program memory size: 1k byte to 256k bytes (2 bytes=1 instruction)
Data RAM: 32 to 8k+32 (32 bytes of gp registers, 0 to 8k of RAM)
Special features: EEPROM, 20mA output drive

Step 10: Resources for Atmel AVR

Atmel the Manufacturer

AVR Freaks Despite the amateurish name, this is a REALLY good site.

PIC vs. AVR smackdown A comparison between PIC and AVR

AVR Butterfly The AVR "Butterfly" evaluation board is a phenomenal value at the current price of $20.

Another Butterfly vendor

GCC supports all but the smallest AVRs; giving you a C compiler of substantial quality for free.

GCC and WinAVR

Arduino is a sort of combination of some standardized hardware based on the ATmega8 or ATmega168 and a simplified development environment that hides some of the complexities of using gcc for AVRs. See also under "modules."

Product guides/comparison charts:

Official Atmel Product Guide

Device list at AVR Freaks

Atmel AVR MCU Comparison Table Non-official table contributed by skylen

Instructables using Atmel AVRs:
Getting started with AVR microprocessors on the cheap
Programmable LED
Synchronizing Fireflies
Other AVR Instructables

...Collaborators feel free to add additional information here...

Step 11: Intel 8051 and Variants

Intel invented the 8051 architecture a long time ago, and garnered some hobbyist interest with the 8052BASIC chip, which contained a basic interpretter in masked ROM and allowed one to build a very small BASIC based computer. Since then the architecture has been licensed and/or stolen by MANY vendors, and good things have happened to it. It's been shrunk, expanded, speeded up, made lower power, had peripherals added, increased memory, flash and eeprom (and even ferromagnetic ram) added. Most variants are somewhat harder to find than PIC or AVR chips.

Step 12: Resources for 8051

8052.com site Massive amounts of info.

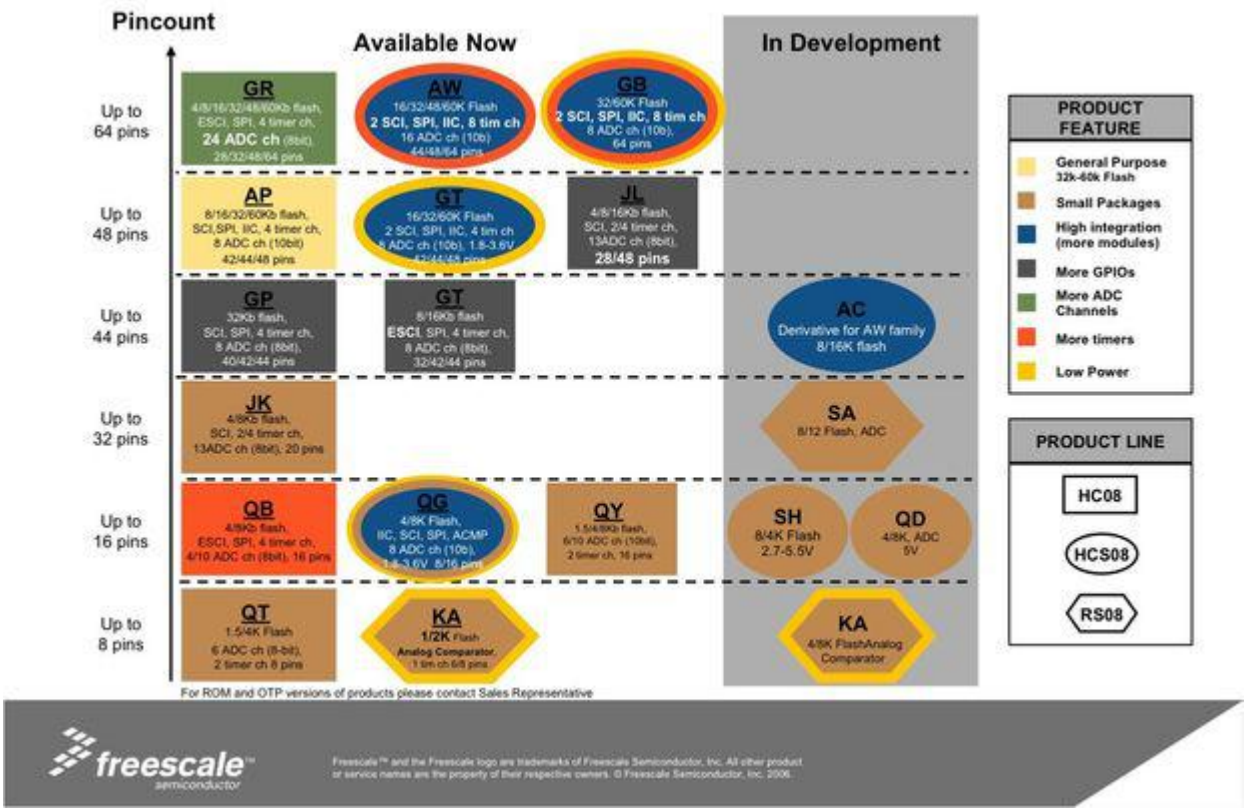
Atmel 8051s Atmel does 8051s as well as AVR and ARM

NXP (Philips) 8051 89LPC controllers.

RAMTron micros 8051s with FRAM non-volatile memory

Silicon Labs Neat 8051s including high performance ADCs. Also the smallest 8051; 11pins in a 3x3mm QFN. Cheap USB "toolstick" eval platforms.

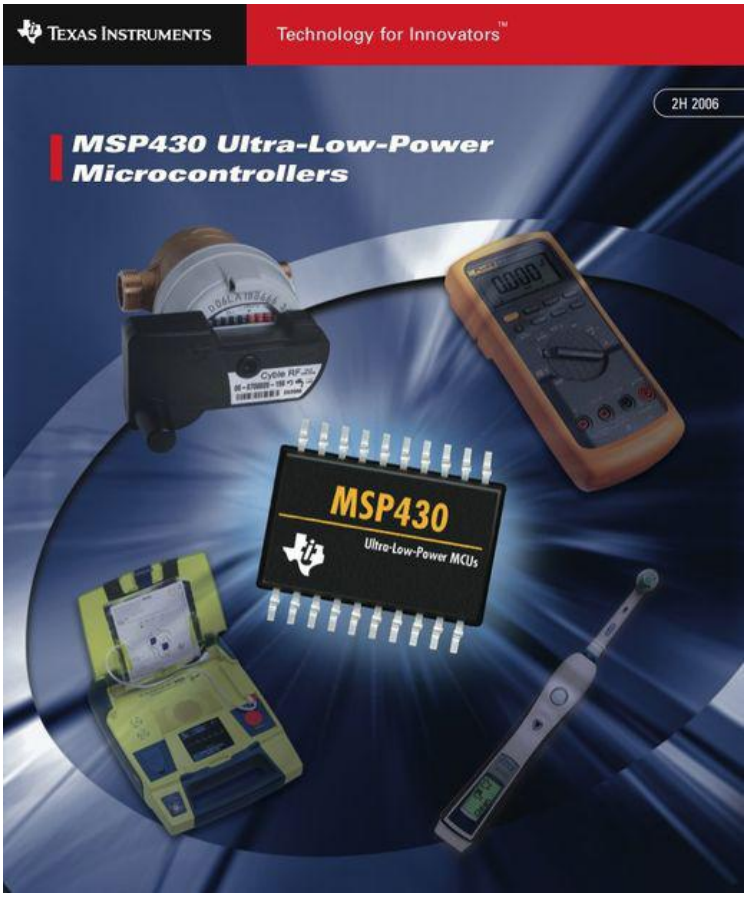
Step 13: Freescale (Motorola) 68HC908, HCS08



Motorola (well, now Freescale) has several lines of popular microcontrollers, the most accessible of which seems to be the flash-based 68HC908 and/or HCS08 or RS08 (all the same or very similar architectures, with some renaming and assorted minor differences) series. Traditionally, Motorola chips have been very difficult to obtain through low-volume channels (ie to the hobbyist) and Mot hasn't been very good about distinguishing chips you can actually buy vs chips that are so targetted to a particular industry segment that they're essentially unobtainium. But they seem to be catching on. Many can be bought at Digikey, and low-cost development systems are pretty common.

The Freescale xx08 microcontrollers are one of the few microcontrollers with a genuine von Neuman architecture. Peripherals, Flash, and RAM all share the same 64k address space.

Step 14: Texas Instrument MSP430 Micropower Microcontrollers



Texas Instruments garnered some interest when they introduced (bought?) their MSP430 series of extemely low-power microcontrollers. Until recently, most of the MSP430s were only available in assorted hobbyist-unfriendly SMT packages, but a couple of recent chips have been introduced in DIP packages. And there's a low cost USB-development dongle that offers better than the usual functionality for such things, so maybe things are looking up.
The MSP430 is one of the few natively 16-bit architetcures in the microcontroller world.

> There's currently a rather cool $20 USB stick development system offered: eZ430-F2013
>
> TI's MSP430 Overview page They have free online training, too. Every year about the beginning of may (4/30, get it?) they have a free seminar given at various locations around the country. Usually this includes a giveaway (last year it was one of the eZ430 things.) Worthwhile.
>
> Trip report to an Early TI seminar Describes 430 basics (by *me*)
>
> MSP430 Application Reports Nice document by Lutz Bierl
>
> MSP430 Yahoo Group pretty good forum.

Step 15: ARM Microcontrollers

ARM is a company that designs microprocessor architetcures, and licenses them to manufacturers who build actual chips. The ARM is a 32bit true RISC architecture, and scales upwards to CPUs with floating point hardware and clocks speeds of several hundred MHz. If you have a palmtop, it probably contains an ARM-based chip. Your cellphone probably has an ARM based chip.

Recently, some of the manufacturers of ARM architecture chips have started offering combinations of on-chip memory and peripherals, and price, that put them into the same marketplace with 8 and 16bit microcontrollers. If you're likely to need lots of memory and performance, it may be worth looking at ARM chips. Or maybe even if you're NOT. As a professional, the possibility of having a single architecture that spans from 28pin microcontrollers to 400MHz router CPUs is attractive in many ways.

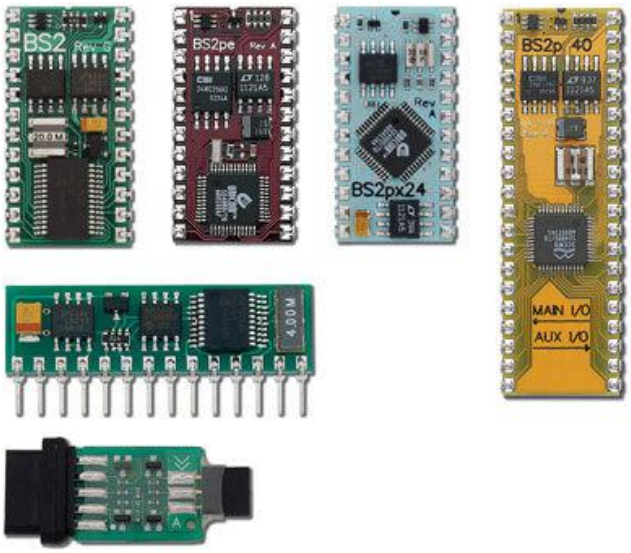
Currently, I'm finding that the breadth of the ARM space seems to generate some confusion. Putting together a tool set and development environment for a particular ARM chip can be challenging.

[Http://www.arm.com ARM the company]
Luminary Stellaris $1 ARMs with as few as 28 pins
NXP (philips) LPC ARM controllers down to 48 pins and $3.
Atmel ARM chips Atmel does ARMs as well as AVRs and 8051s
ST M32 Cortex M3 chips Cheap chips with good peripherals, and Cute "primer" development system
Olimex ARM development boards for Atmel, NXP, and other small ARMs
GNUARM Homepage for the GNU ARM toolchain
Instuctable: Pixecuter - Run Software on Matell JuiceBox

Step 16: Other Interesting Microcontrollers

- Cypress PSOC
- Renesas (Hitachi) H8, M6

Step 17: Modules, Bootloaders, and "hidden" Microcontrollers



A number of companies have made a business of selling "modules" , usually incorporating some sort of microcontroller and some of its support components with a high-level-lanaguge development environment, some sort of chip-programming capability, and communications. This gets rid of the need for any special hardware tools (usually just a cable to your PC), provides the HLL, and in general lets you get started much faster than the "pure" microcontroller route. The main disadvantage would be price ($30 for a basic stamp vs $5 for a chip with similar capability) and (sometimes) performance (the basic stamp uses interpreted basic, which is

very slow compared to native code.)

Parallax: home of the Basic Stamp
Basic Micro "ATOM"
BASIC Tiger
SunSpot Java thing
Modules/IDEs with ethernet/etc
Arduino: open source hardware and IDE for Atmel AVR

Another disadvantage is WRT the education aspect. You won't learn much about "microcontrollers" by programming a module in a HLL that hides ALL the microcontroller details.

Step 18: Zilog Z8 and Z80 Chips

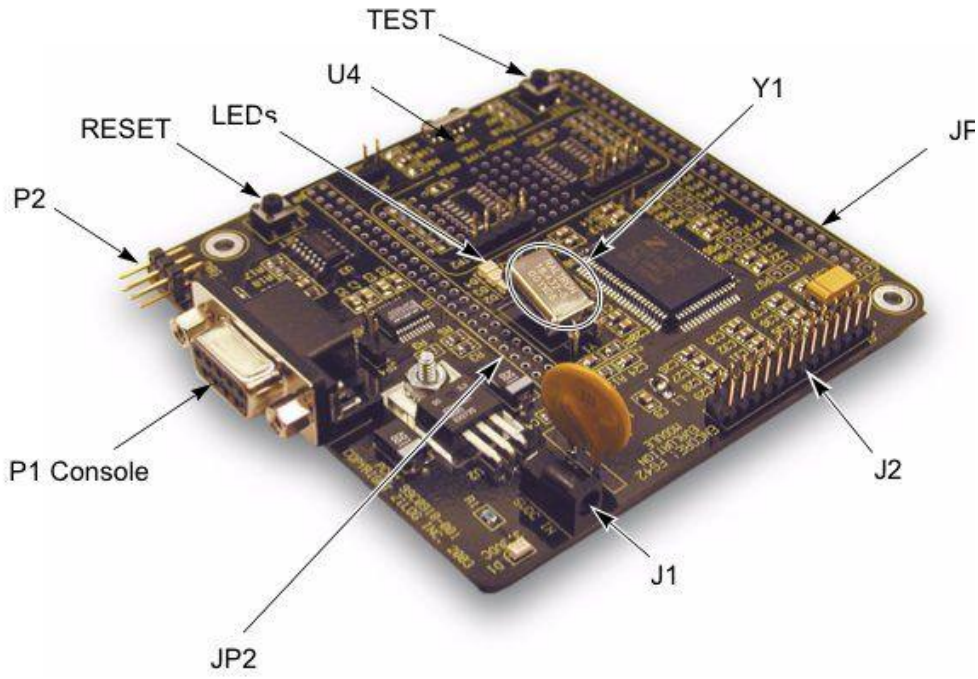


Figure 2. Z8 Encore! 64K Series Development Board

Zilog (inventor of the famous Z80 microprocessor chip) has updated versions of the Z80 in microcontroller form, and also updated versions of the even older Z8 architecture. Both have flash memory and some interesting peripherals , and inexpensive "evaluation boards" that include a C compiler. At one time, the zilog evaluation boards were quite "full" compared to many low cost evaluation boards, but the current versions seem less so. (My EZ8kit has 4 5x7 matrix displays, an IRDA transceiver, two rs232 async ports, an rs485 port, and assorted buttons and lights. IIRC, it was $50 (special deal at a trade show.))

Most of the people who have commented list the PIC, PICAxe, ARM, 8052, or Basic Stamps. These are all popular chips, but if you are just getting started with microcontrollers, you might want to start with something fairly cheap. How does $40 sound?

That's what the Zilog Z8 Encore! 8-bit microcontroller development kit costs. The Z8 Encore! microcontrollers are available with 1K to 64K of flash RAM. For speed, they operate at 5- to 20-MHz, and depending on the chip, they are available in various packages from 8 to 80 pins. The $39.95 development kit contains a development board with input/output pins, a wall wart power supply, computer cable (USB or serial to connect your computer to the development board), and a free C compiler and assembler that runs under Windows. You don't need to buy or build a programmer, since you can program the chip right in the circuit you've built -- without having to remove it and reprogram it in a programmer.

All the Encore! chips can communicate serially through a UART (some have 2 UARTs). Some have analog-to-digital converters built-in, and some have an infra-red (IRDA) communications port. Oh, and there are up to 4 built-in timers on the chip, depending on the chip version. The price of an Encore! microcontroller is pretty reasonable, given all the capabilities (anywhere from $3 to $13 each from Digi-Key).

The C compiler is excellent. The Zilog C compiler has most of the capabilities of "big boy" compilers, like a floating point library, trig functions, etc. It produces better assembly code than I can, so I use it almost exclusively.

Remember that almost any non-trivial code you write will have bugs. The debugger supplied with the development kit permits you to step through your code and look at the results of variables stored in the various registers to help you locate your mistakes.

Digi-Key stocks two different Encore! development kits (either one costs $39.95); one for 8K/4K chips, and the other for 16K through 64K chips. The Digi-Key part number for the 8K/4K development kit is 269-3183-ND, and the part number for the 64K development kit is 2693249-ND. http://www.digikey.com

Zilog's website has a large number of sample applications, full documentation of their microcontrollers, and free updates to the C compiler supplied with the development kit.

I'd say that the only downside is that the Z8 Encore! isn't as popular as the PIC as a hobbyist chip, so there haven't been any books or many of projects published for it except for some articles in Circuit Cellar magazine. Oh, and if you are interested in microcontrollers, Circuit Cellar and Nuts & Volts magazines are two that cover the subject. Both are available at Barnes & Noble bookstores -- maybe Borders too. Or, just go directly to their websites:

Circuit Cellar Magazine
Nuts & Volts Magazine

Step 19: Win Valuable Prizes

Periodically, many of the manufacturers of microcontrollers will sponsor "Design Contests" where engineers all over will be challenged to come up with a particularly clever design using a particular microcontroller. The idea is to entice engineers into looking at THEIR chips even if they're already using some other microcontroller. At any given time of year, there's likely to be at least one contest "in progress." There are a number of good things about these contests:

PRIZES. Some of the contests have significant cash prizes. $5-10k, which is not to be sneezed at.
Winning even a minor prize (or being published) will look good on your resume.
Frequently the contests are accompanied by "special offers" in the form of low cost development tools or free samples.
Motivation!
A contest may generate at least temporary interest and discussion on some processor of interest.

- The end of the contest usually includes publishing some or all of the entered designs, serving as useful examples for everyone else.

I've talked to reps at trade shows; the final number of entrants to these contests tends to be pretty small, so as contests go your chances of winning (if you complete an entry) are unusually high.
Usually there is nothing in the rules that prevents the entry from also being (say) your college senior design project. Alas, many contests are restricted to people over 18, and completing an entry in the timeframe usually allowed is not so easy as it sounds before you've tried it.

Currently running:

Freescale Low-end x08 "Black Widow" Contest half-price development kit on completion of 2 parts of the 4-part contest. $10k+ prize.

PICList Free PCB Contest Free olimex-fabbed PCB for your design. Small prize, small contest; runs every month! Must be PIC or SX based.

General reference for a lot of contests (includes past winners, etc.):
Circuit Cellar Magazine Contest Page

I entered Freescale's recent "Black Widow" contest for designs based on their very small 8-bit microcontrollers, which had a top prize of $10000 cash plus a trip to their design conference (and $1000 for each of 10 finalists to aid in completing a prototype.) I didn't make the finals :-(
However, I netted a T-shirt, a coupon good for $15 off a (low cost) development system, and oneof some ROOMBA robot vacuum cleaners raffled off to "early entrants." And I learned some stuff, too. That's not a bad haul. The contest apparently had 775 participants at the final phase...

Step 20: Try It Online: Virtual Labs

Tech Online is a pretty good website for technical News in general, and they've apparently implemented something they call "Virtual Labs" that will allow you to try out a vendor's development system over the internet with nothing but a browser on your end. I used this for the first time as part of the Freescale "Black Widow Design Contest", since one of the required steps was to get the virtual development board to behave in a certain way. It was pretty neat. But it wasn't entirely bug-free, so be a bit careful.

They have several Freescale, Luminary, Renesas, and TI development systems available.

Step 21: Free Stuff!

In the old days, companies would mail datasheets and databooks to just about anyone who asked. The web has done away with the need for most of that, and many vendors seem to have taken the money they saved on postage and used it to make their sample program more accessible.

The way "free samples" work is that the manufacturer will happily send you several samples of a part, in the hopes that you'll try them out and be so impressed that you'll used them in a project that nets them sales of millions of units. Or (if you're a student) that they'll win some loyalty and "mindshare" when you get out into the real world and be more likely to use their microcontrollers than someone else's. Or that you'll write some magazine or web article that will call attention to their products. It's a marketing expense, and it's *probably* cheaper than many marketing expenses...

Microchip, Freescale, and TI all have very liberal sample policies. Look up a part on their website, and it's likely to have a "sample" button somewhere on the page; send them some info and in a week or so you're likely to have chips in hand. Other vendors are harder; you may have to talk to a representative. Vendors have a weak spot (as they should) for students; you might think that a sample request from a "my-u.edu" email address carries less weight than one from "nonexistantcompany.com", but that's not necessarilly true.

Ethics of Free Sampling :no resell