CPU Architecture & Boolean Math

ECE 376 Embedded Systems

Jake Glower - Lecture #1

Please visit Bison Academy for corresponding lecture notes, homework sets, and solutions

Background:

Microcontrollers are a type of computer which is designed for controlling devices, such as toasters, vacuum cleaners, etc. Most are built around a microcomputer with several features added:

- Memory is typically incorporated within the microcontroller, allowing a single-chip design.
- Timers are added, and
- Analog inputs / output are often added.

For example, the microcompuer and corresponding microcontroller are

- Intel: 8080 / 8051
- Motorola: 6800 / 6812

MicroChip & PIC18F4620

- Microchip: Small company which specializes in microcontrollers for start-up companies.
- PIC18F4620: Inexpensive, works, easy to use, tools are free
- Very similar to other microcontrollers. (They're all about the same)
- We have access to the registers (allows for low-level programming)

Microcontrollers

- Inexpensive: $0.45 for PIC10F200
- Versitile: Change code and you change the operation
- Overkill: Many problems can be solved using hardware, but for $0.45, why not?

Changing hardware is expensive:

- Recertify, change the asssembly line, rewrite code
- Changing software is free (sort of)

Three Levels of Programming

Low-Level (ECE 376)

- Direct access to hardware and registers
- Low level routines to read sensors, drive actuators, set timing.

Mid-Level (ECE 476, Rasberry Pi, Arduino)

- Call low-level routines
- Make a quad-copter hover, follow commands

High-Level (Computer Science)

- Call mid-level routines
- AI: get quad copters to swarm, find hottest spot in the room, etc.

CPU Architecture

All computers have five main sections:

- Program Memory
- Data Memory
- Stack Memory (to store data such as the return address when you call a subroutine)
- Registers (to store data which is being manipulated), and
- Arithmetic Logic Unit (ALU) which does the addition, subtraction, etc.

Microcontroller: All five on a single chip

- Good: No interfacing chips
- Bad: You're stuck with what you have

4-bit, 8-bit, 16-bit, 32-bit

- Number of bits for a memory read / write / add / subtract
- PIC18F4620: 8-bit (simple: you have to start somewhere)

Von-Neuman Architecture

Most computers (Motorola, Intel, etc.)

Program / Data / Stack are all the same size (8-bits)

- Allows you to allocate memory as needed
- Requires multiple memory reads for a single operation
- Data can over-write the program (bad)

Address	Allocation	Memory Type
0	Data	RAM
4,095		
4,096	Stack	RAM
16,583		
16,584	Program	RAM or FLASH EPROM
65,536		

Harvard Architecture (PIC)

Program / Stack / Data are all separate

- You can optimize the size of each

Good: Each instruction only takes one clock (one memory fetch)

Bad: Can't reallocate memory

PIC18F4620:

Program Memory 16-bits	Data Memory 8-bits	Stack Memory 15-bits
0000 start	00 start	00 start
.	.	.
.	.	.
.	.	31 end
.	3968 end	
.		
32,767 end		

ROM:

ROM is used for

- Vectors (Tell the CPU where to go on certain events, such as power on, 10ms clock tic, etc.)
- Tables (information the program will use. Feedback gains, wait times, etc. This allows you to change the program by changing data in the table.)
- Main Routine (tells the CPU what to do)
- Subroutines (small parts of the mainline routine)
- Interrupt Service Routines (programs that are called by hardware events, like a switch closing or 10ms clock tic).

All of this must fit in the 32k ROM.

It's preferred if you keep these blocks together to simplify debugging. (The address tells you what the data means. In a table, it's data. In the main routine, it's an instruction, etc.)

RAM

RAM is used for

- Sending and receiving data from the microcontroller
- Saving data from program execution.

RAM split into banks of 4096 bytes.

- Data RAM is located in the low bank at address 0x0000 to 0x0FFF (0 to 4095).
- Special functions are located 0xFF00 to 0xFFFF.

For us, this doesn't mean much: you can have a single array that's 3198 bytes of 3198 variables, each of which is one byte.

Note: This is an issue for future versions of the PIC18F4620

To access memory above 0x0FFF, you need to switch banks

Stack

Stores the return address from a subroutine call

- Each call pushes the return address onto the stack
- Each return pops the return address off of the stack

31-level stack means

- You can nest subroutines 31-levels deep
- If you go beyond that, the return address is lost (program crashes)
- Recursion is not allowed with a PIC processor

Pipeline & Program Timing

A PIC processor has a 2-level pipeline

- Level 1: The next instruction in the program (moved to Level 2)
- Level 2: The present instruction being executed.

Result

- One clock per instruction
- Two clocks if the instruction is a jump
  - Prefetch got the wrong instruction

Clock	Memory Pre-Fetch	Program Execution
1	0x800	-
2	0x801	0x800
3	0x802	0x801
4	0x803	0x802

Registers

Helps with hardware design

- Easier to add two known locations (X & Y) than two arbitrary locations

Motorola 6812:

- Six registers for you to use

PIC18F4620:

- One register for you to use (W)

Good:

- No choice.
- Everything goes through W

Bad:

- Code can get convoluted

Boolean Math

In a computer, everything is binary: data is only stored as ones and zeros. Likewise, all math is done using binary arithmetic.

Definitions:

- Bit: 1 or 0. A single flip flop or capacitor whose output is 5V (1) or 0V (0).
- Nibble: 4 bits.
- Byte: Eight bits.
- Word: More than one bit. 'Word' has no specific size in general.
- Binary: Base 2 arithmetic. 0b01010 means 'binary 01010'
- Decimal: Base 10 arithmetic Default is base 10.
- Hexadecimal: Base 16 arithmetic. 0x1234 means 'hexadecimal 1234'

Base N Numbers:

Base 10	Base 2	Base 16
1234 means	10101 means	1234 means
$1 \times 10^3 +$ $2 \times 10^2 +$ $3 \times 10^1 +$ 4×10^0	$1 \times 24 +$ $0 \times 23 +$ $1 \times 22 +$ $0 \times 21 +$ 1×20	$1 \times 16^3 +$ $2 \times 16^2 +$ $3 \times 16^1 +$ 4×16^0

Range of N bits

Base 10	Base 2	Base 16
$0 .. 10^N - 1$	$0 .. 2^N - 1$	$0 .. 16^N - 1$
3 digits	8 bits	4 nibbles
000 .. 999	000 .. 255	0000 .. 65,535

Hexadecimal

- More convenient than binary
- Easier to see bits than decimal

Program Memory 16-bits	Hexadecimal	Binary	Decimal	Hexadecimal	Binary
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

Hexadecimal Examples:

Convert the number 0x1234 to binary.

Solution: Go nibble by nibble:

Hexadecimal	1	2	3	4
Binary	0001	0010	0011	0100

Convert 0b00101010100100101010 to hexadecimal.

Solution: Separate into groups of 4 bits (nibbles)

0b 0010 1010 1001 0010 1010

0x 2 A 9 2 A

The answer is 0x2A92A.

Boolean Math

Addition: Add just like you do in base 10. Just remember to carry a 2 (base 2) or carry a 16 (hexadecimal)

Example:

Carry	(1)		(1)	
	0x4	A	2	6
+	0x9	C	8	D
	14	22 (16 + 6)	11	19 (16 + 3)
Result	0xD	6	B	3

0x4A26 + 0x9C8D = 0xD6B3

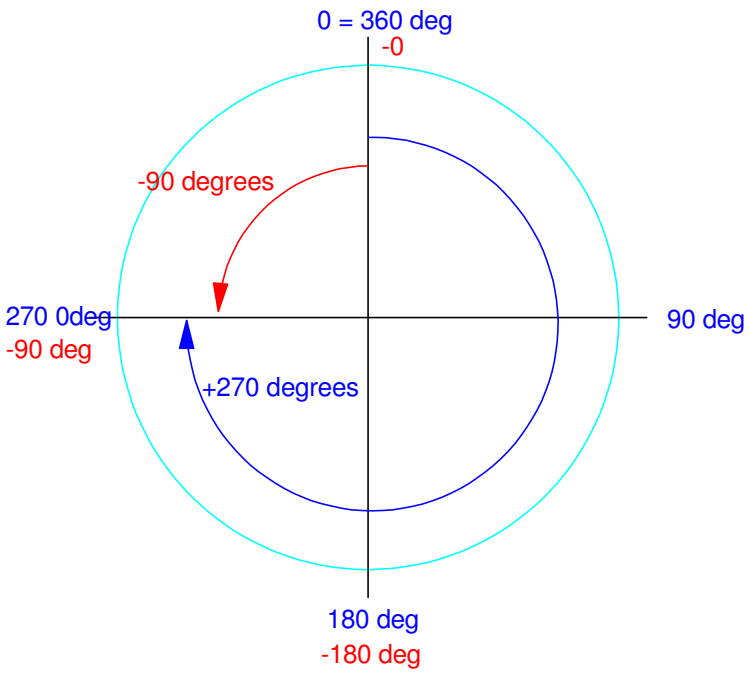
Logical Operations:

- AND:
  - 0 = bit clear
  - 1 = no change
- OR
  - 0 = no change
  - 1 = set
- XOR
  - 0 = no change
  - 1 = toggle

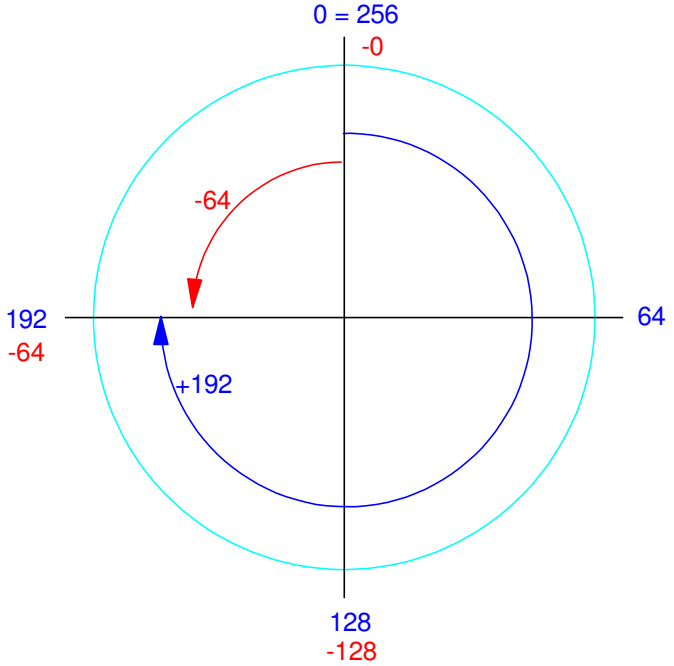
A	B	A & B (and)	Comment	A | B (or)	Comment	A ^ B (xor)	Comment
0	0	0	Bit clear	0		0	
0	1	0		1		1	
1	0	0		1	bit set	1	bit toggle
1	1	1		1		0	

2's Compliment Notation

- The way you express negative numbers
- Angles: Add or subtract 360 degrees
- 8-bit: Add or subtract 256



Base 360 (angles)



Base 256 (8-bit binary)