Notes for 9/2/09

Single-precision floating point (float type in C) according to IEEE754 (32 bits):

Sign	Exponent	Fraction / Significand	Total
1 bit	8 bits	23 bits	32 bits

Double-precision floating point (double type in C) according to IEEE754 (64 bits):

Sign	Exponent	Fraction / Significand	Total
1 bit	11 bits	52 bits	64 bits

This is like normalized scientific notation, e.g. (-1)^sign * fraction * 10^exponent, where the fraction is in [1, 10), and you can see that our exponent is base 10, except with IEEE754 floating point, our fraction is in [1, 2) and our base is 2. Note that the exception to this is zero, when our fraction is zero.

Loops in MIPS:

```
char A[] = "Hello World";
char *p;
for (p = A; *p != '\0'; ++p) {
    *p = *p + 3;
}

    la   $s3, A
// ----------------------
MyLoop:
    lb   $t6, 0($s3)
    beq  $t6, $zero, Exit
// ----------------------
    addi $t6, $t6, 3
    sb   $t6, 0($s3)
    addi $s3, $s3, 1
    j MyLoop
// ----------------------
Exit:
    . . .
```

Basic blocks:

A basic block is code that has only one entry point and one exit point. Branches, jumps, and system calls end basic blocks. Labels and beginning of code begin basic blocks. The basic block divisions are commented above, so note that the above code has four basic blocks. As a rule of thumb, draw a line after branches, jumps, and system calls, and draw a line before labels.

Pseudo-instructions:
Can expand into multiple MIPS instructions. For instance, load immediate:
`li $1, 0x1234ABCD => lui $1, 0x1234; ori $1, $1, 0xABCD`

Why do this in two instructions? Because we only have 32 bits to encode an instruction. After we account for the other stuff we need to encode in the instruction, we only have 16 bits left for an immediate value. So to load a 32 bit number, we need two instructions to load the high order 16 bits and low order 16 bits! Note that if our immediate value is 16 bits, then li will only expand into one instruction.

Types of MIPS instructions: R-Type, I-Type, J-Type.

R-Type: Any time you have three registers, e.g. add $t0, $1, $2. You don't have any room for an immediate value. R-Type also includes a 5-bit shift amount for the shift instructions, e.g. sll $t0, $t0, 5.

opcode	rs	rt	rd	shift amount	funct
6 bits: 000000	5 bits	5 bits	5 bits	5 bits	6 bits

Note that for r-type, the opcode is zero, and the operation is specified in funct.

I-Type: Any time you have an immediate value encoded in the instruction, e.g. addi $t0, $t1, 1234.

opcode	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits