Floating Point Numbers (IEEE Standard 754)

Why? We need to represent real numbers!

Single precision FP (32 bit):

FP value = $(-1)^S$ x (1 + F) x $2^{(E - bias)}$

Sign	Exponent (E)	Fraction (F) / Mantissa
31	23	0

For single precision FP, *S* = 1 bit, *E* = 8 bits, *F* = 23 bits, *bias* = 127.

For double precision FP, *S* = 1 bit, *E* = 11 bits, *F* = 52 bits, *bias* = 1023.

Question: Why do we use a bias?

We use bias because we wish to represent both tiny and large real numbers. Subtracting bias from our exponent value will share the range we can represent to both tiny (negative E) and large (positive E) real numbers.

"Special" single precision FP values:

±Zero: E = 0, M = 0	NaN: E = 255, M ≠ 0
±Infinity: E=255, M = 0	Denormalized: E = 0, M ≠ 0

(More on denormal numbers: http://en.wikipedia.org/wiki/Denormal_number)

Question: Convert the single precision FP representation, 0xC0B40000, to decimal.

0xC0B40000 = 0b1 | 10000001 | 01101000000000000000000

0b10000001 = 0d129 ---> E = 129 – 127 = 2

-1.01101×2^2

$= -(2^0 + 2^{-2} + 2^{-3} + 2^{-5}) \times 2^2$

$= -(2^2 + 2^0 + 2^{-1} + 2^{-3})$

$= -(4 + 1 + 0.5 + 0.125)$

$= -5.625$

Now we know how to convert from FP representations to decimals, how about the other way around? Google is always your best friend. For example, try this website:

http://www.cs.cornell.edu/~tomf/notes/cps104/floating.html#dec2hex

MIPS Revisited

Since your project 2 is all about MIPS (and so is project 4, the MIPS datapath), we decide to give you a quick taste of how to decode MIPS instructions. Remember, each instruction in MIPS is a number!

Question: Convert "addi $t1, $t0, 5" to its HEX representation.

Format: addi $rt, $rs, imm ---> opcode(addi) = 001000, $t0 = 01000, $t1 = 01001

0x21090005 or 0b001000 | 01000 | 01001 | 0000000000000101

Question: Decode the following program and describe its function.

Memory Address	Instruction
0x00	0x0085402A (0b00000000100001010100000000101010)
0x04	0x11000002 (0b00010001000000000000000000000010)
0x08	0x00A01020 (0b00000000101000000001000000100000)
0x0c	0x03E00008 (0b00000011111000000000000000001000)
0x10	0x00801020 (0b00000000100000000001000000100000)
0x14	0x03E00008 (0b00000011111000000000000000001000)

Memory Address	Instruction
0x00	slt $t0, $a0, $a1
0x04	beq $t0, $0, 2
0x08	add $v0, $a1, $0
0x0c	jr $ra
0x10	add $v0, $a0, $0
0x14	jr $ra

This function returns the larger number of $a0 and $a1.