

## Floating Point

### Part V: IEEE Standard 754

Why? We need to represent real numbers!

#### Single precision FP (32 bit):

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

Sign (S) Bits: [31]	Exponent (E) Bits: [30, 23]	Fraction (F) (aka Mantissa) Bits: [22, 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.

1) 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](http://en.wikipedia.org/wiki/Denormal_number))

2) Convert the single precision FP representation, 0xC0B40000, to decimal.

0xC0B40000 = 0b1 | 10000001 | 011010000000000000000000

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

-1.01101 × 2<sup>2</sup>

= - (2<sup>0</sup> + 2<sup>-2</sup> + 2<sup>-3</sup> + 2<sup>-5</sup>) × 2<sup>2</sup>

= - (2<sup>2</sup> + 2<sup>0</sup> + 2<sup>-1</sup> + 2<sup>-3</sup>)

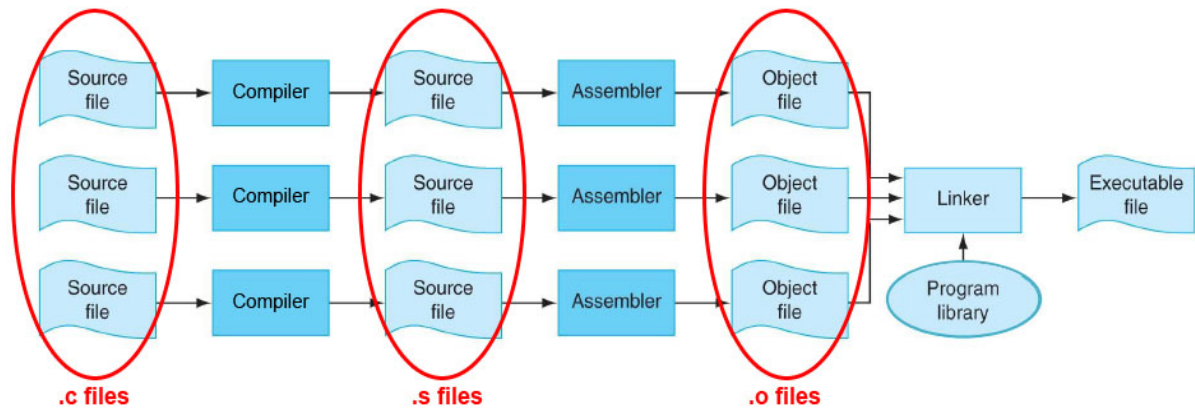
= - (4 + 1 + 0.5 + 0.125)

= -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>

## CALL (Compiling, Assembling, Linking, and Loading)

Part VI: It's Turtles all the way down! (aka, How it all fits together):



**Assembler:** Converts pseudoinstructions to MIPS instructions. Uses the \$at register. Converts assembly language into an object file containing:

- 1) Header – size and position of other parts
- 2) Text segment – machine language code
- 3) Static data segment
- 4) Relocation information – instructions and data words using absolute addressing
- 5) Symbol table – matches symbols/labels with addresses
- 6) Debugging information

**Linker:** Combines independently assembled machine language programs and resolves all remaining undefined labels from the relocation information and symbol table. Result is the executable file.

1) How many passes over assembly code does an assembler have to make and why?

**Two. First to find all label addresses, second for instruction conversion.**

2) Does the linker resolve issues in relative or absolute addressing?

**Absolute addressing.**

3) What does RISC stand for? How is this related to pseudoinstructions?

**Reduced Instruction Set Computing. Minimal set of instructions leads to many lines of code. Pseudoinstructions are more complex instructions intended to make assembly programming easier for the coder. These are converted to TAL by the assembler.**