15-213 Recitation: Data Lab

Jack Biggs

26 Jan 2015

Agenda

- Introduction
- Brief course basics
- Data Lab!
  - Getting Started
  - Bits, Bytes, Ints
  - Floating Point
  - Running your code

Introduction

- Welcome to 15-213!
- Recitations are for **15-213** students *only*
  - Place to review previous lectures
  - Discuss homework-related problems
  - General problem solving
  - Preview material in future lectures

- Ask any questions you may have, we will get back to you if we can’t answer immediately.

Course Basics

- Getting help
  - Staff email list: 15-213-staff@cs.cmu.edu
  - Office hours: **5-9PM** from Sun-Thu in Wean 5207
  - Course website: http://cs.cmu.edu/~213
  - Course textbook is **extremely** helpful!
  - Linux Workshop, 2 Feb: Location & Time **TBA**
- All homework submissions will be done through Autolab.
- All homework should be done on the **shark clusters**.
  - From Linux: `ssh andrewid@shark.ics.cs.cmu.edu`

Data Lab: Getting Started

- Download handout, transfer it to your AFS directory.
  - *From shark:* `cd <folder>`, then `tar xpvf <tar-filename>`
  - If you get `Permission denied`, try `chmod +x <filename>`
- Test your code with `btest`, `bddcheck`, `driver.pl`, and `dlc`
  - For more information, *read the writeup.*
  - The *writeup* is on the same page in Autolab as the handout.
  - Click on *view writeup* to *view the writeup*. It's really that simple.
  - No, really, read the entire writeup. Always. Please. For our sake.
- `driver.pl` will tell you your score.
- To submit, upload your `bits.c` file to Autolab.

Data Lab: Bits, Bytes, and Ints

- Computers represent all of their data in 0s and 1s, known as “bits.” 8 of these binary digits are a “byte.”
- Architects perform arithmetic on human-readable numbers using operations on binary numbers.
- The goal of this lab is to get you more comfortable with bit and byte-level representations of data.

Size of data types on different systems

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
long long	8	8	8
float	4	4	4
double	8	8	8
long double	8	10/12	10/16
pointer	4	4	8

Endianness (Byte Order)

- Little-Endian stores lower bytes of a number first.

e.g., `0xdeadbeef` stored at address `0xaaaa`:

```
0xaaaa: 0xef be ad de
```

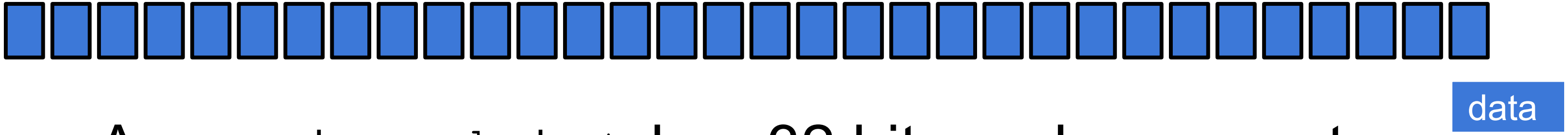
- Big-Endian stores higher bytes of a number first.

e.g., `0xdeadbeef` stored at address `0xaaaa`:

```
0xaaaa: 0xde ad be ef
```

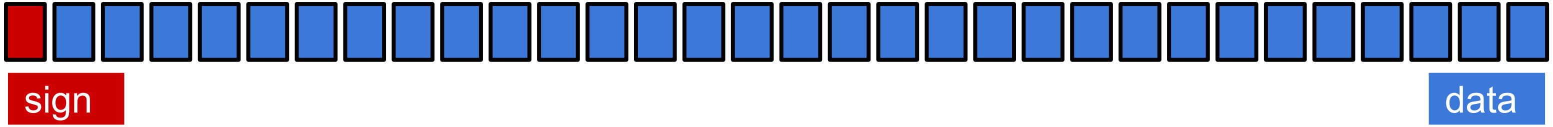
- This concept is less important in this lab, but will become more relevant in bomb and buffer lab.
- The Shark machines are *Little-Endian.*

Unsigned Numbers



- An `unsigned int` has 32 bits and represents positive numbers from 0 to 2^{32}-1.
- If we add 1 to 2^{32}-1, we *overflow* back to 0.
- General formula: With *k* bits, we can represent 2^k distinct numbers.
- Highest unsigned int value known as U_{max}.

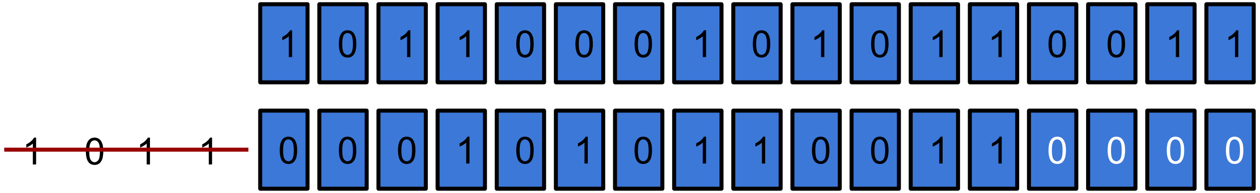
Signed Numbers



- An `int` has 32 bits: 31 bits for data, and 1 bit for sign
  - Represents $[-2^{31}, 2^{31}-1]$
- *Overflow* or *Underflow* in signed arithmetic produces **undefined behavior!**
- General formula: With *k* bits, we can represent numbers from $[-2^{k-1}, 2^{k-1}-1]$
- Lowest signed int value known as T_{min}, highest signed int value known as T_{max}.

Operators: Shifting

Shifting modifies the positions of bits in a number:



= x

= x << 4

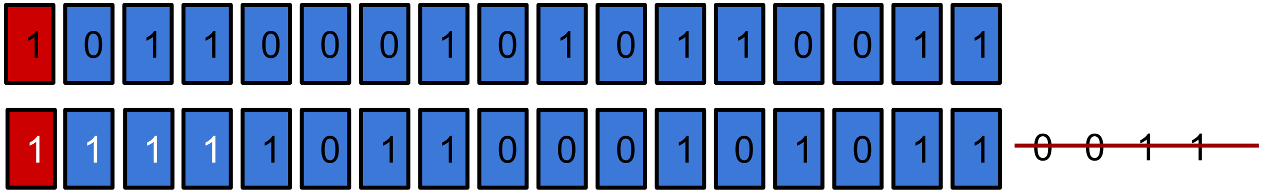
= x * 2^4

Shifting right on a signed number will *extend the sign:*

x =

x >> 4 =

x / 2^4 =



(If the sign bit is zero, it will fill in with zeroes instead.)

Operators: Bitwise

- Bitwise operators use bit-representations of numbers.

1	0	0	1	0	0	0	1	= x
0	0	1	0	0	1	0	1	= y
1	0	1	1	0	1	0	1	= x	y (**or**)

1	0	0	1	0	0	0	1	= x
0	0	1	0	0	1	0	1	= y
0	0	0	0	0	0	0	1	= x & y (**and**)

1	0	0	1	0	0	0	1	= x
0	0	1	0	0	1	0	1	= y
1	0	1	1	0	1	0	0	= x ^ y (**xor**)

1	0	0	1	0	0	0	1	= x
0	1	1	0	1	1	1	0	= ~x (**logical negation**)

Operators: Logical

- In C, the truth value of an int is **false** if 0, else **true**.
  - x && y: “x **and** y”. Ex: 7 && 3 = **true** = 1.
  - x || y: “x **or** y”. Ex: 0 || 3 = **true** = 1.
  - !x: “**not** x”. Ex: !484 = **false** = 0.
- *Ensure you are not mixing bitwise and logical operators in your code!!!*

Operators: Arithmetic

- Basic arithmetic also works in C.
- *Beware of overflow!*
- x + y, x - y: addition / subtraction.
- x * y, x / y: multiplication / division.
- x % y: *modulo*. The *remainder* after *integer* division.

- Negating a two's complement number: ~x + 1

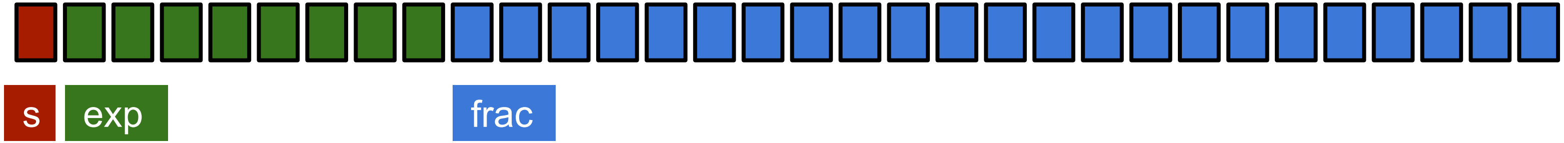
Floating Point

- In the IEEE Floating Point specification, we represent our decimal numbers in binary scientific notation:

$$x = (-1)^s M 2^E$$

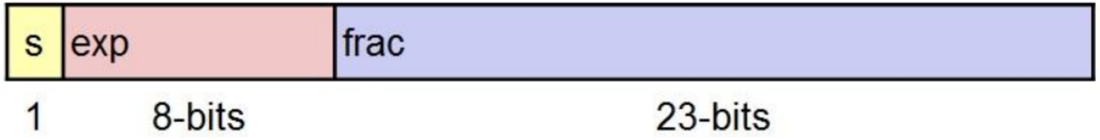
- s - the *sign* of the number
- M - the *mantissa*, a fraction in range [1.0, 2.0)
- E - the *exponent*, weighting the value by a power of two

- `s` is sign bit s, `exp` is binary representation of E, and `frac` is binary representation of M:



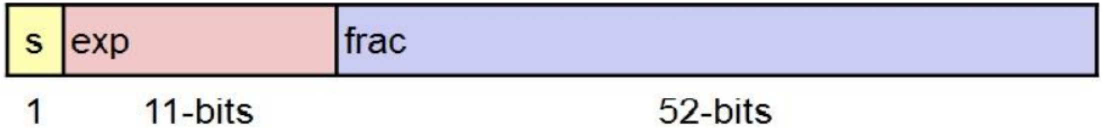
Floating Point: Different levels of precision

- Single Precision: 32 bits



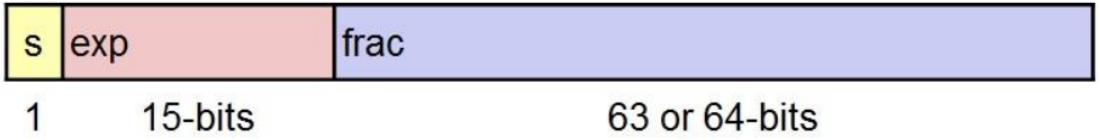
`float`

- Double Precision: 64 bits



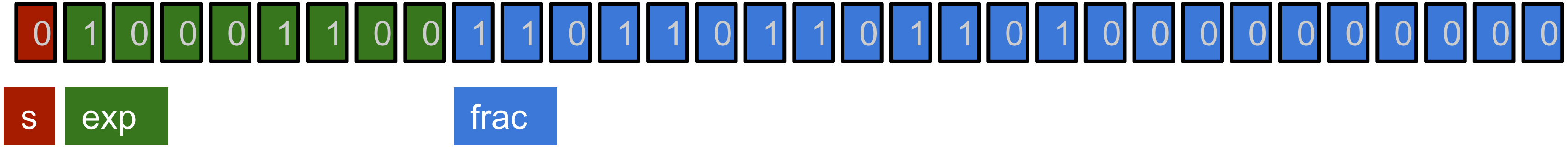
`double`

- Extended Precision: 80 bits (Intel only)



`long double`

Floating Point: Normalized Values



- Case: exp != 0, exp != 111…11
- E = exp - bias
- Bias = 2^{k-1}-1, where k = number of exponent bits
- Significand (mantissa) encoded with implied leading 1

Example: In the above diagram, exp = 10001100 = 140.

exp - bias = 140 - 127 = 13, so our multiplying factor is 2^{13}.

frac has implied leading 1, so M = 1.1101101101101_2

$(-1)^0 * 1.1101101101101_2 * 2^{13} = 11101101101101_2 = \mathbf{15213.0_{10}}$.

Floating Point: Denormalized Values

1	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0

s | exp | frac

- Case: exp = 0
- E = -bias + 1
- Bias = 2^{k-1}-1, where k = number of exponent bits
- Significand (mantissa) encoded with implied leading 0

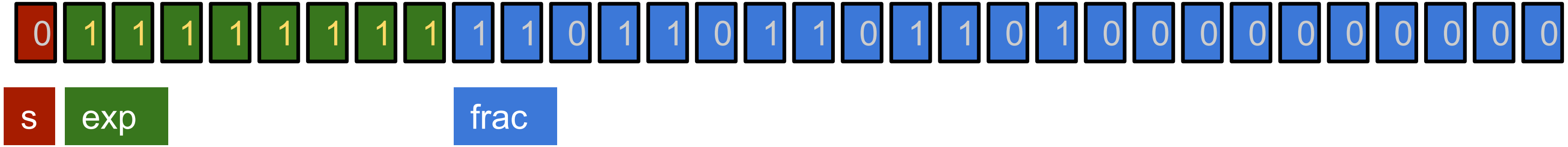
Example: Since we have an implied leading 0, M = 0.1101101101101.

Our exponent, E = -bias + 1 = -126

Our sign bit is 1

Our number: $(-1)^1 * 0.1101101101101 * 2^{-126} = 1.00746409144571... \times 10^{-38}$

Floating Point: Special Values



- Case: exp = 111…11
  - Frac = 000….000: Represents +/- infinity
    - For overflow of numbers, divergence, etc.
    - Sign **not** ignored!
  - Frac != 000….000: Represents NaN
    - “Not a Number”
    - For number division by zero, square root of -1, and other non-representable numbers
    - Sign ignored

Floating Point: Rounding

- IEEE uses the **round-to-even** rounding scheme.
  - Remove bias from long, repeated computations
- Examples
  - 10.1011: More than ½, round up: 10.11
  - 10.1010: Equal to ½, round down *to even*: 10.10
  - 10.0101: Less than ½, round down: 10.01
  - 10.0110: Equal to ½, round up *to even:* 10.10
  - All other cases involve rounding up or down

Floating Point: Practice

- Consider a 5-bit floating point representation using k=3 exponent bits, n=2 fraction bits, and no sign bit.
  - What is the bias?
  - What is the largest possible normalized number?
  - Smallest normalized number?
  - Largest **de**normalized number?
  - Smallest **de**normalized number?

Floating Point: Practice

- Consider a 5-bit floating point representation using k=3 exponent bits, n=2 fraction bits, and no sign bit.
  - What is the bias? **3**
  - Largest normalized value? 110 11 = 1110.0 = 14
  - Smallest normalized val? 001 00 = 0.0100 = ¼
  - Largest **de**normalized val? 000 11 = 0.0011 = 3/16
  - Smallest **de**normalized val? 000 01 = 0.0001 = 1/16
- These sorts of questions *will* show up on your midterm, by the way!

Floating Point: Practice

- Consider a 5-bit floating point representation using k=3 exponent bits, n=2 fraction bits, and no sign bit. Can you fill out the chart below?

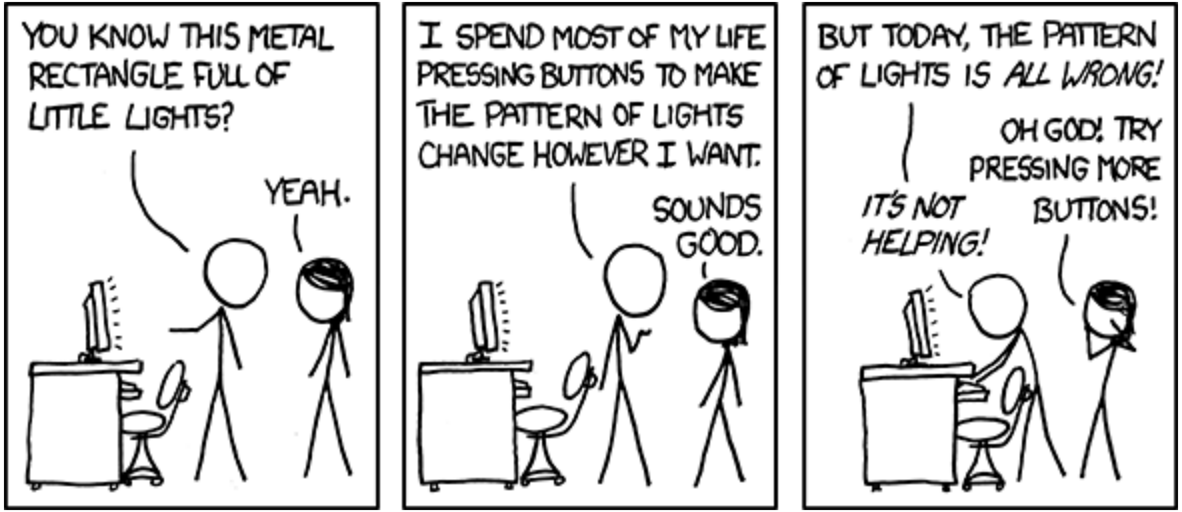
Value in Decimal	IEEE floating point representation	Rounded Value
9/32		
3		
9		
3/16		
15/2		

Floating Point: Practice

- Consider a 5-bit floating point representation using k=3 exponent bits, n=2 fraction bits, and no sign bit. Answers:

Value in Decimal	IEEE Floating point representation	Rounded Value
9/32	001 00	1/4
3	100 10	3
9	110 00	8
3/16	000 11	3/16
15/2	110 00	8

Dazed? Lost? Confused? Angry?



Read the textbook, email the staff list, go to office hours, and, *for the love of God*, ***read the writeup!!!!***

Sources

- Textbook
- Course website: http://cs.cmu.edu/~213
- Previous recitation slides
- Lecture slides