Anita's Super Awesome Recitation Slides

15/18-213: Introduction to Computer Systems

Bit Logic and Floating Point, 28 May 2013

Anita Zhang, Section M

Welcome to the Summer Edition

- Data Lab due Mon, 3 Jun 2013, 11:59 pm EST
  - 2 grace days per lab; don't waste them
- Bomb Lab out Tues, 4 Jun 2013, 11:59 pm EST
  - After the relevant lecture(s)
- FAQ on the main site
  - To be updated…?

Additional Probing

- Questions?
- Progress?
- Autolab?
- Shark?
  - > ssh shark.ics.cs.cmu.edu

Because Everyone Needs a Guide..

- Getting Help
- Literature
- Bits and Bytes and Good Stuff
- IEEE Floating Point
- Data Lab Hints
- General Lab Information
- Question Time

Meet the TAs

Michael Hansen (mhansen1)

Anita Zhang (anitazha)

I Need Help):

- Email us: 15-213-staff@cs.cmu.edu
  - Please attach C files if you have a specific question
- IRC: irc.freenode.net, ##213
  - Anita (anitazha) lurks there daily
- Videos on Blackboard
- Everything else, Autolab: autolab.cs.cmu.edu
- Office hours: Sun-Thurs, 6pm – 9pm, Gates 5205
  - The cluster with the window
  - Both Michael and Anita will be there (mostly)
- Potential Google Hangout to come

Books I Like

- Randal E. Bryant and David R. O'Hallaron, *Computer Systems: A Programmer's Perspective, Second Edition*, Prentice Hall, 2011
- Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language, Second Edition*, Prentice Hall, 1988
- Koenig, Andrew. *C Traps and Pitfalls*. Reading, MA: Addison-Wesley, 1988
- Kernighan, Brian W., and Rob Pike. *The Practice of Programming*. Reading, MA: Addison-Wesley, 1999

Random Motivational Stuff

Representation Nutshell

- Signed
  - The most significant bit represents the sign
    - 0 for non-negative, 1 for negative
  - On x86, the 31^{st} bit (counting from 0)
  - Focus on two's complement
- Unsigned
  - Range from 0 to $2^k - 1$
    - Where k is the number of bits used to represent this value
    - Non-negative values
- Byte = 8 bits

What are "ints"?

- int ≠ integer
- Minimum and maximum values are capped by the number of bits

Casting Magic

- What happens when casting between signed and unsigned?

Casting Magic

- Signed $\leftrightarrow$ Unsigned
  - Values are “reinterpreted”
  - Bits remain the same
- Mixing signed and unsigned values
  - Values are casted to unsigned first

What is the Size of….

C Data Type	Typical 32-bit	IA32 (x86)	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 or 12	10 or 16
pointer	4	4	8

Operations

- Bitwise
  - AND → &
  - OR → |
  - NOT → ~
  - XOR → ^
- Logical
  - AND → &&
  - OR → ||
  - NOT → !
- Values
  - False → 0
  - True → nonzero

Pro-Tip

- Do not get bitwise and/or logical mixed up!!
  - If you are getting weird results, look for this error

Specific Operation Stuff

- Shifting
  - Arithmetic
    - Preserves the sign bit (sometimes sign-extended)
  - Logical
    - Fills with zeros (in our case)
  - Other bits "fall off" (discarded)
  - Both will result in the same left shift
  - Undefined if negative shift amount (to be discussed)

Shifting Math

- Multiplication/ division by 2^k
  - Multiply: left shift by k
  - Division: right shift by k

Specific Operation Stuff

- …I lied. Kind of.

Specific Operation Stuff

- Division of a negative number by 2^k
  - Needs a "bias"
  - Division looks like this: (x + (1 << k) - 1) >> k
    - x is the value we are dividing
    - (1 << k) – 1 is the value we are adding to bias

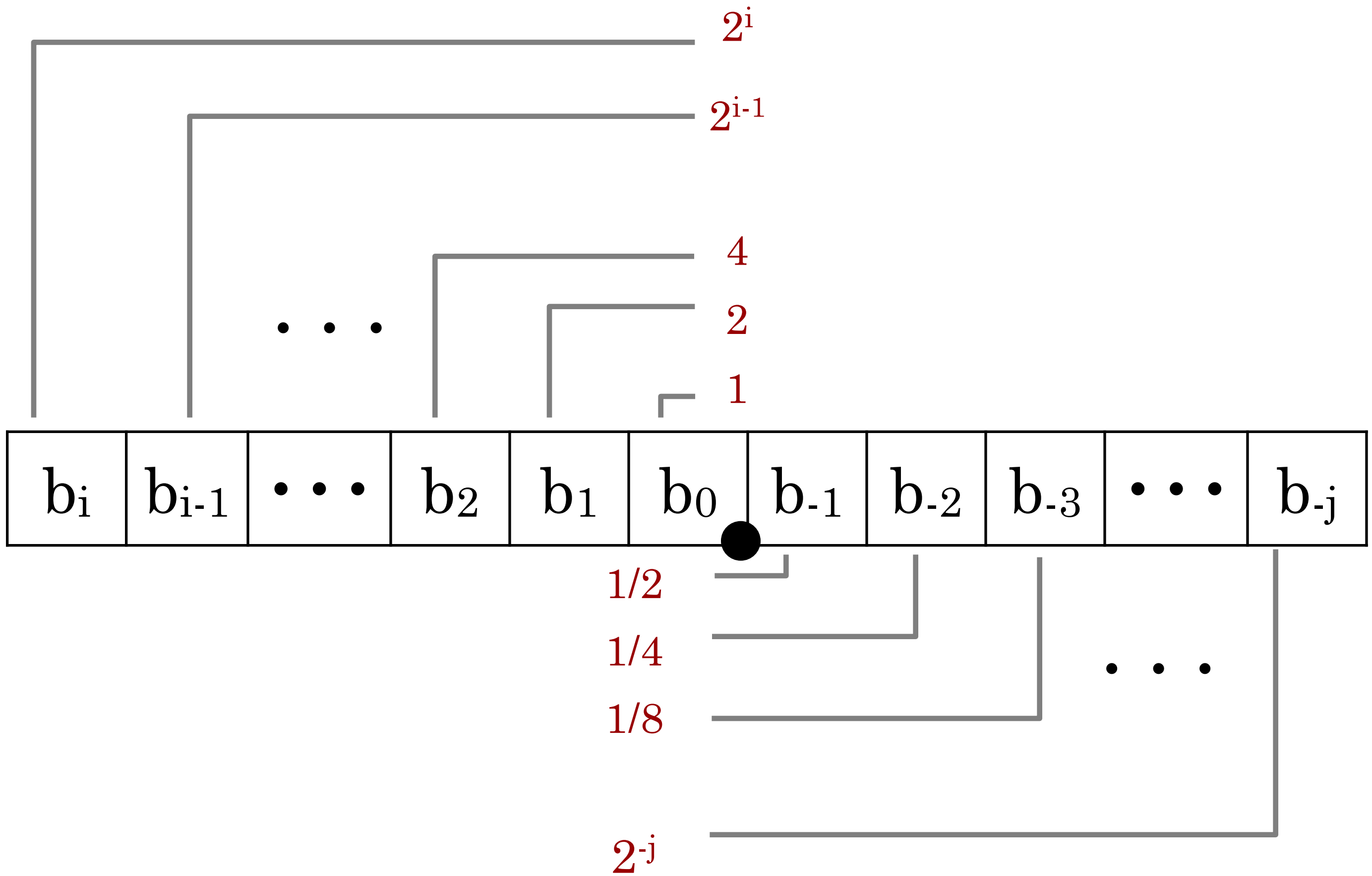
Random Number Stuff

- Endianness is real
  - How bytes are ordered
  - Representation in memory
- You'll see it in Bomb Lab (next week)

Endian	First byte (lowest address)	Middle bytes	Last byte (highest address)
big	*Most* significant	**...**	*Least* significant
little	*Least* significant	**...**	*Most* significant

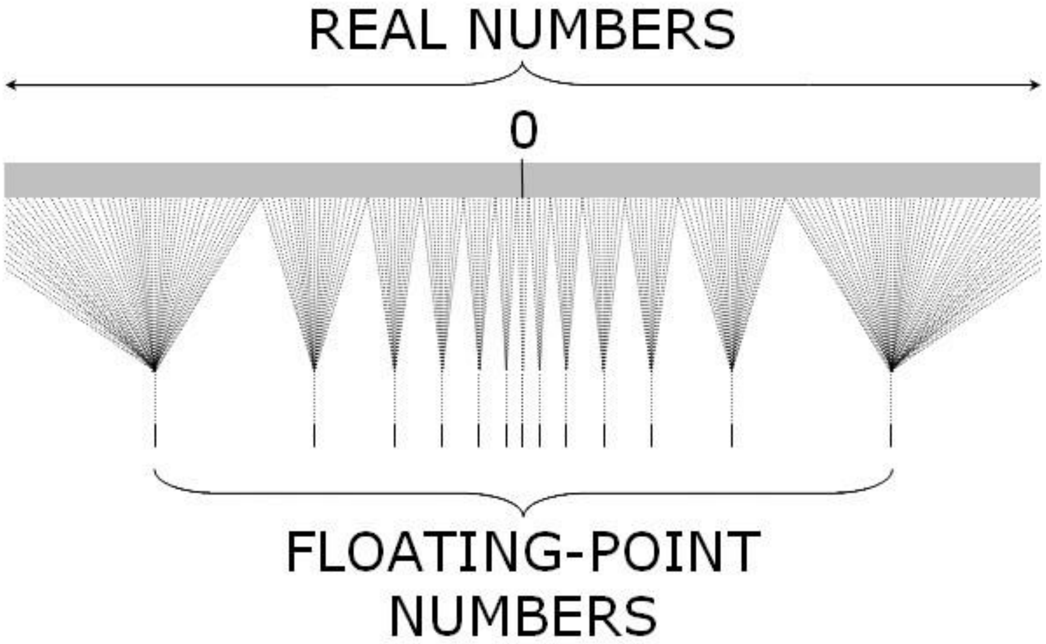
- Random example: 0x59645322
  - **Big**: (lower) 59 64 53 22 (higher)
  - **Little**: (lower) 22 53 64 59 (higher)

Fractional Binary



(Quick and Dirty) Floating Point

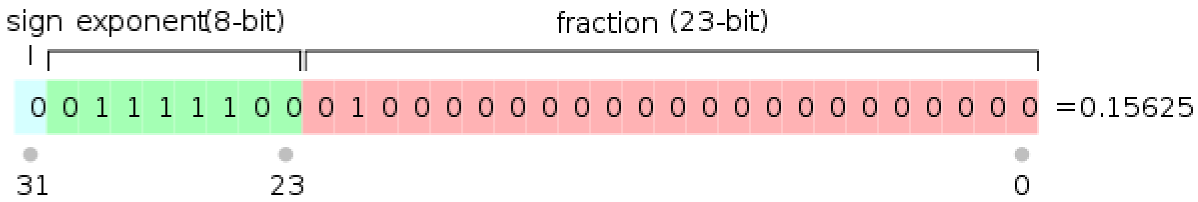
- What is this floating point stuff?
  - Another type of data representation
  - Enables support for a wide ranges of numbers
  - Symmetric on its axis (has ±0)



(Quick and Dirty) Floating Point

- Consists of 3 parts
  - Sign bit
  - Exponent bits
  - Fraction bits (the "mantissa")
- Getting the floating point
  - Value → $(-1)^s \times M \times 2^E$
    - S → sign
    - M → mantissa
    - E → shift amount (exponent bits uses 'e' or 'exp')
  - Bias → $2^{k-1} - 1$
    - Used in the math to convert between actual values and floating point values

(Quick and Dirty) Floating Point



- For single precision (32 bit) floating point:
  - Fraction (frac): 23 bits
  - Exponent (exp): 8 bits
  - Sign (s): 1 bit
  - Bias = 127

(Quick and Dirty) Floating Point

Normalized

- exp ≠ 00…0
- exp ≠ 11…1
- E = exp – bias
- M = 1.xxxxxx
  - xxxxxx is the frac
  - Implied leading 1

Denormalized

- exp = 00..0
- E = 1 – bias
- M = 0.xxxxxx
  - xxxxxx is the frac
  - Leading 0
  - frac = 0 means ±0

Special cases

Infinity

- exp = 11….1
- frac = 00…0
  - Division by 0, $\pm \infty$

Not a Number

- exp = 11….1
- frac $\neq$ 00…0
  - sqrt(-1), $\infty - \infty$, $\infty \times 0$

Special cases

- BTW, infinity and NaN are not the same
  - Infinity is “overflow”
  - NaN is not a number
    - “Mathematically undefined” in my book
  - Be aware of this for float_abs()

Legit Floating Point Rules

- Rounding
  - Rounds to even
    - Used to avoid statistical bias
    - 1.1011 → 1.11 (>1/2, up)
    - 1.1010 → 1.10 (1/2, down)
    - 1.0101 → 1.01 (>1/2, down)
    - 1.0110 → 1.10 (1/2, up)

- Addition and Multiplication…
  - Are lies
    - Associativity/ distributivity may not hold
    - 3.14 + (1e20 – 1e20) vs. (3.14 + 1e20) – 1e20

Floating Point on Exams

- Let's pretend we have a 5-bit floating point representation with no sign bit… (sadness)
  - k = 3 exponent bits (bias = 3)
  - n = 2 fraction bits

Value	Floating Point Bits	(Rounded) Value
9/32	001 00	1/4
3		
9		
3/16		
15/2		

Floating Point on Exams

- Let's pretend we have a 5-bit floating point representation with no sign bit… (sadness)
  - k = 3 exponent bits (bias = 3)
  - n = 2 fraction bits

Value	Floating Point Bits	(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

Floating Point on Exams

- Consider two 7 bit floating point representations based on the IEEE format. Neither has a sign bit.
- Format A
  - k = 3 exponent bits (bias = 3)
  - n = 4 fraction bits
- Format B
  - k = 4 exponent bits (bias = 7)
  - n = 3 fraction bits

Format A	Format B
011 0000	0111 000
101 1110	
010 1001	
110 1111	
000 0001	

Floating Point on Exams

- Consider two 7 bit floating point representations based on the IEEE format. Neither has a sign bit.
- Format A
  - k = 3 exponent bits (bias = 3)
  - n = 4 fraction bits
- Format B
  - k = 4 exponent bits (bias = 7)
  - n = 3 fraction bits

Format A	Format B
011 0000	0111 000
101 1110	1001 111
010 1001	0110 100
110 1111	1011 000
000 0001	0001 000

Data Lab Other Stuff

- Use the tools
  - ./driver.pl
    - Exhaustive autograder (uses provided tools)
  - ./bddcheck/check.pl
    - Exhaustive
  - ./btest
    - Not exhaustive
  - ./dlc
    - This one will hate you if you're not writing C like it's 1989
    - Declare all your variables at the beginning of the function

Data Lab Tools

- Extra tools
  - ./fshow *value*
    - Where value is a hex or decimal number for a floating point
    - Shows the hex for value and breaks it down into the floating point parts (sign, exponent, fraction)
    - Single precision floating point
  - ./ishow *value*
    - Where value is a hex or decimal number
    - Outputs value in hex, signed, and unsigned
    - 32-bits

Datalab Other Stuff

- Operator precedence
  - There are charts. Google them.
- bitCount
  - Divide and conquer
- isPower2
  - Actually do and write down operations on paper
- float_i2f
  - You will need to round
- Undefined behavior
  - Shifting by 32
    - And why you get strange results

Undefined Behavior (Adv. Topic)

"These instructions shift the bits in the first operand (destination operand) to the left or right by the number of bits specified in the second operand (count operand). Bits shifted beyond the destination operand boundary are first shifted into the CF flag, then discarded. At the end of the shift operation, the CF flag contains the last bit shifted out of the destination operand.

The destination operand can be a register or a memory location. The count operand can be an immediate value or register CL. The count is masked to five bits, which limits the count range to 0 to 31. A special opcode encoding is provided for a count of 1."

Labs, in General

- Aim to do all your work on our Shark machines
  - Obtain a terminal/ SSH client of sorts
  - Use the following command
    - ssh *andrewID@shark*.ics.cs.cmu.edu
      - andrewID is your Andrew ID
      - shark can be replaced with a specific shark hostname
        - If left as shark, you will be assigned a random shark
    - tar xvf *labhandout*.tar
      - Untarring on the Unix machines may prevent headaches
      - Work out of your private directory
  - Use a text editor straight from the Shark machine
    - Vim, emacs, gedit, nano, pico…

Questions & Credits Page

- http://www.superiorsilkscreen.com
- http://www.wikipedia.org/
- http://www.cs.cmu.edu/~213/
- http://jasss.soc.surrey.ac.uk/9/4/4/fig1.jpg
- Intel x86 Instruction Set Reference