Anita's Super Awesome Recitation Slides

15/18-213: Introduction to Computer Systems

Bit Logic and Floating Point, 9 September 2013

Anita Zhang, Section M

Welcome to the Fall Edition

- Data Lab due Thursday, 12 Sept 2013, 11:59 PM
  - 2 grace days per lab, 5 per semester
  - Don’t waste your late days
- Bomb Lab out Thursday, 13 Sept 2013, 12:00 AM
  - After the relevant lecture(s)
- FAQ on the main site
  - O’Hallaron put love and care into updating it
  - 10/10 must read

Additional Probing

- Quick 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

I Need Help):

- Email us: 15-213-staff@cs.cmu.edu
  - Please attach C files if you have a specific question
  - Goes to TAs and Professors
- Autolab: autolab.cs.cmu.edu
  - No Blackboard (woohoo?)
- Office Hours: Wean 5207, Sun-Thur, 5:30-8:30PM
  - The only Linux cluster in Wean
  - At least 2 (!!!) TAs at your service
    - Depending on the day there may be a ton of students in line
- Tutoring: Mudge Reading Room, Tues 8:30-11PM
  - Hosted by a smart cookie who is not a TA

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
  - Only here because new people forget

What are "ints"?

- int $\neq$ integer
- Minimum and maximum values are capped by the number of bits

Casting Magic

- What happens when casting between signed and unsigned?
  - Signed ↔ Unsigned
    - Values are “reinterpreted”
    - Bits remain the same
- Mixing signed and unsigned values
  - Values are casted to unsigned first (mostly)

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 (on these machines)
  - Other bits “fall off” (discarded)
  - Both will result in the same left shift
  - Undefined if negative shift amount (to be discussed)

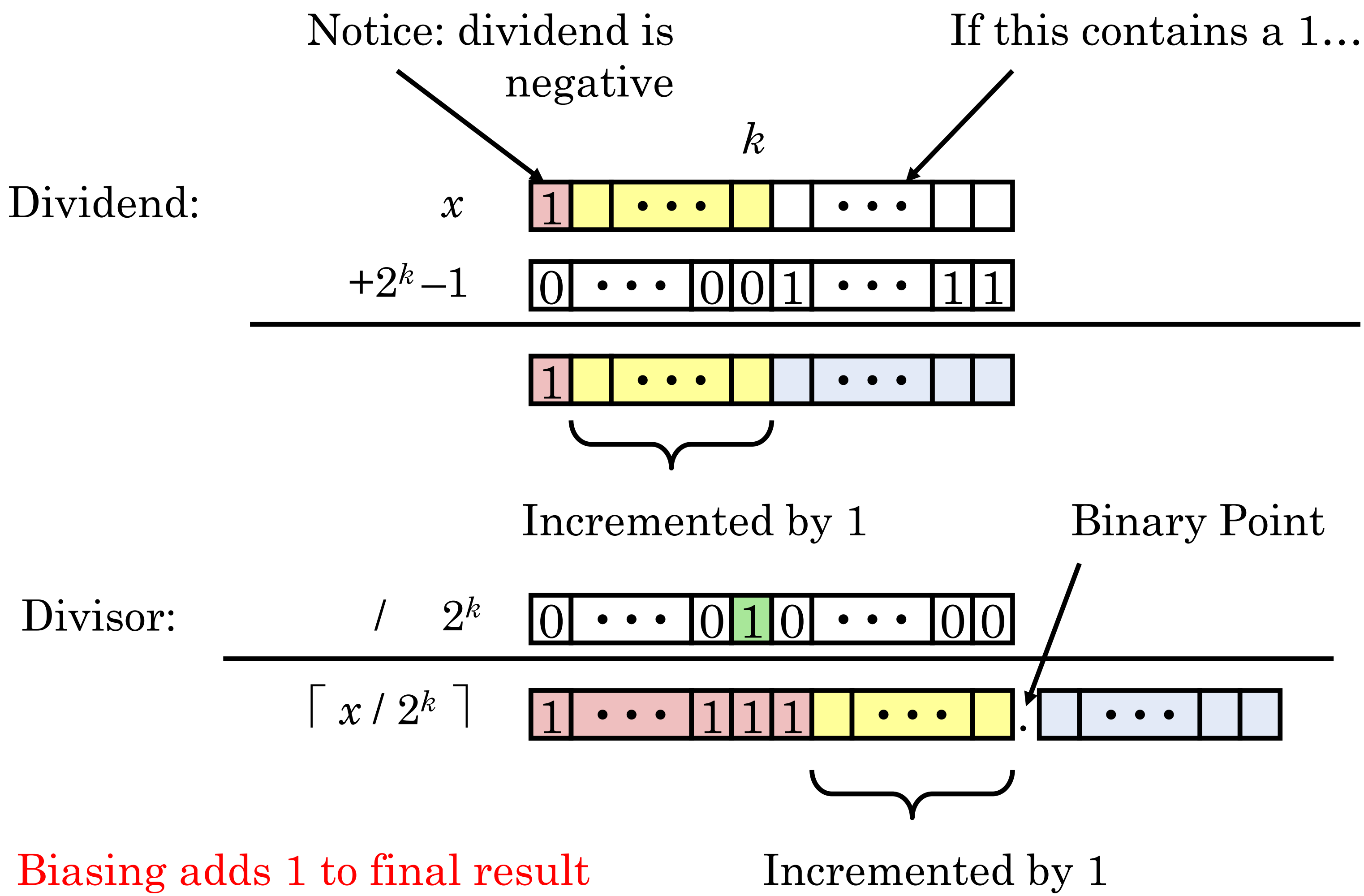
Shifting Math

- Multiplication/ division by 2^k
  - Multiply: left shift by k
  - Division: right shift by k
- Shifting rounds towards negative infinity
  - Math-performing humans round towards 0
  - How do we round negative values toward 0?

Division by Shifting (Negatives)

- Division of a negative number by 2^k
  - Needs a bias
    - Bias will push the number up so it rounds towards 0
  - 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
    - Remember, only applies to negative values of x

FOR THE VISUAL/MATH INCLINED



Biasing adds 1 to final result

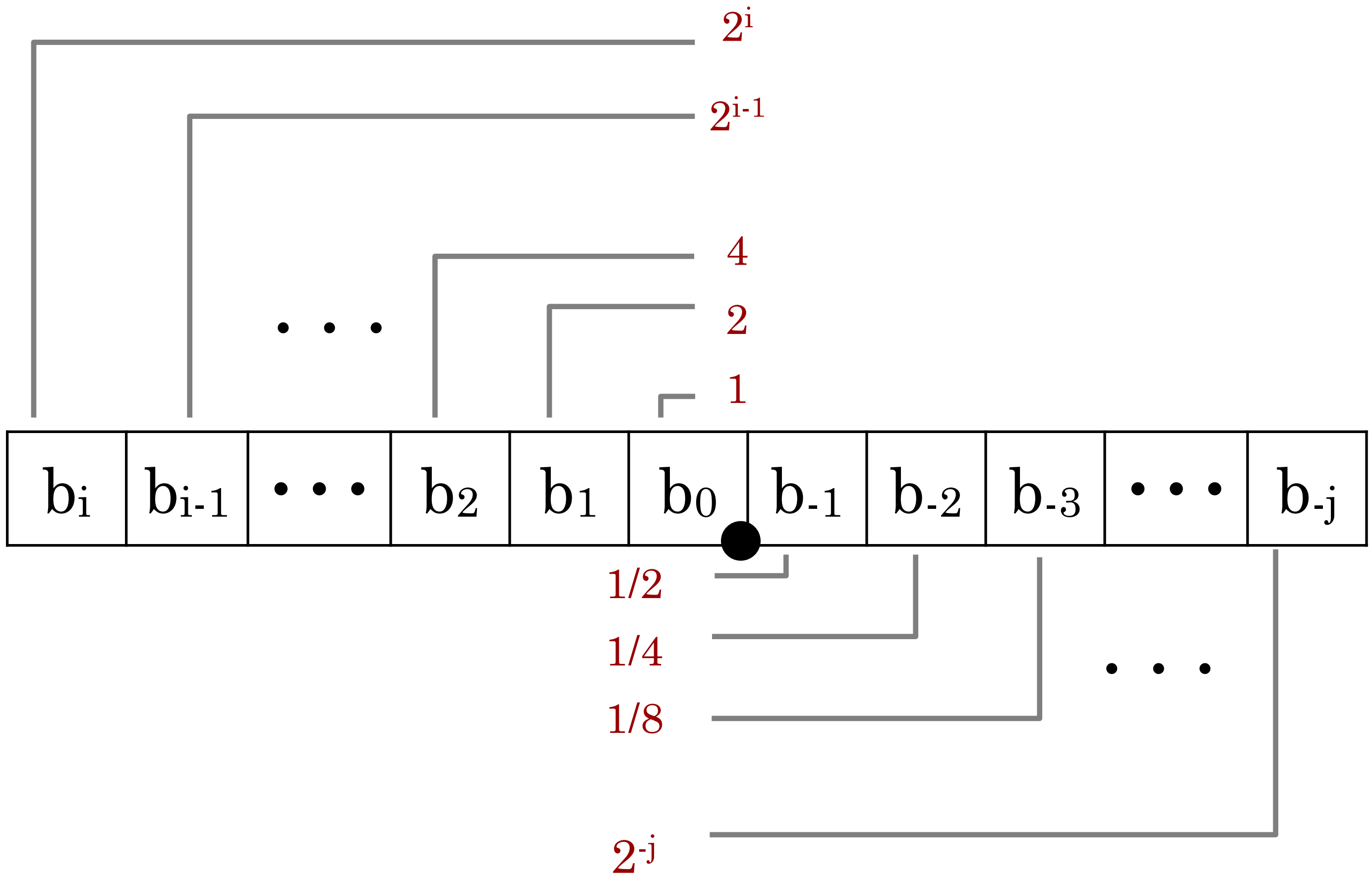
Random Number Stuff

- 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

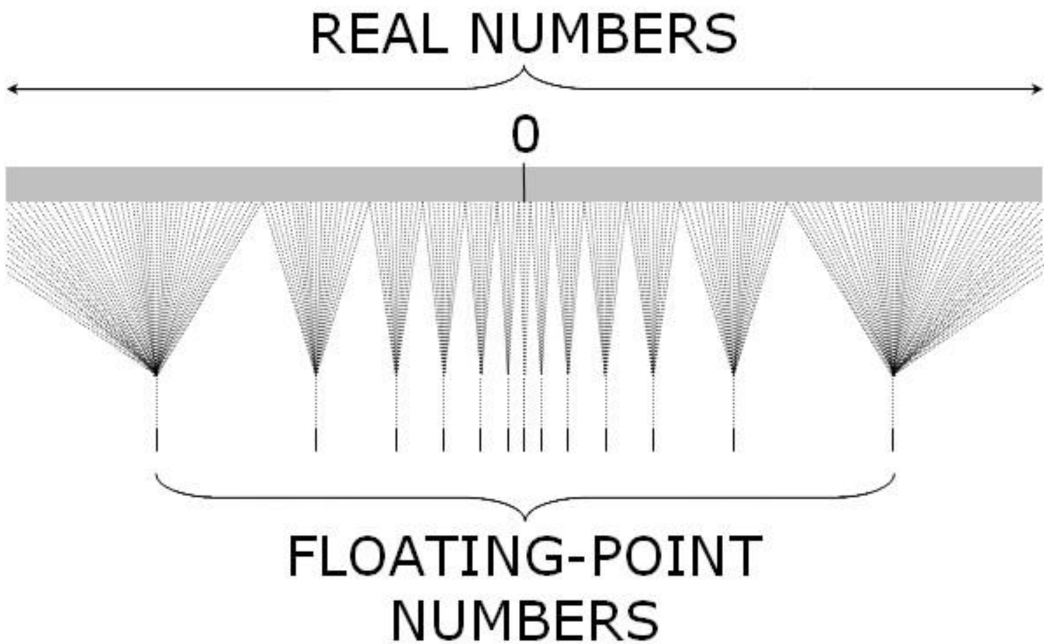
- Random example: 0x59645322
  - **Big**: (lower) 59 64 53 22 (higher)
  - **Little**: (lower) 22 53 64 59 (higher)

Fractional Binary



(Quick and Dirty) Floating Point

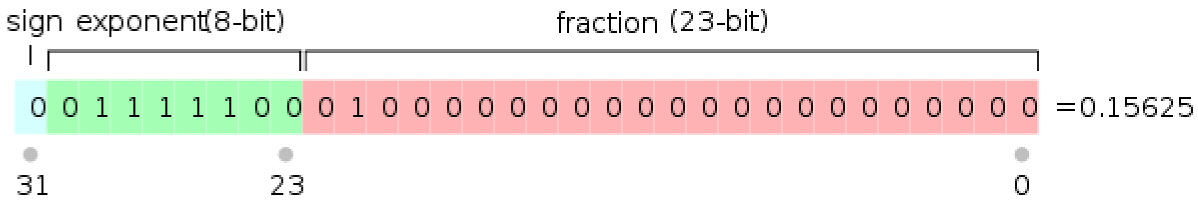
- What is this floating point stuff?
  - Another type of data representation
  - Enables support for a wide ranges of numbers
  - Symmetric on its axis (has ±0)



(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 $\neq$ 00…0
- exp $\neq$ 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	Not a Number
exp = 11....1	exp = 11....1
frac = 00...0	frac $\neq$ 00...0
Division by 0, $\pm\infty$	sqrt(-1), $\infty - \infty$, ∞ x 0

Special cases

- BTW, infinity and NaN are not the same
  - Infinity is “overflow”
  - NaN is not a number
    - “Mathematically undefined” in my book

Legit Floating Point Rules

- Rounding
  - Rounds to even
    - Used to avoid statistical bias
    - 1.1011 → 1.11 (greater then1/2, up)
    - 1.1010 → 1.10 (equal to 1/2, down)
    - 1.0101 → 1.01 (less than 1/2, down)
    - 1.0110 → 1.10 (equal to 1/2, up)
- Addition and Multiplication…
  - Are lies
    - Associativity/ distributivity may not hold
    - 3.14 + (1e20 – 1e20) vs. (3.14 + 1e20) – 1e20
  - Don't need to do this in this class

Insight into Rounding

- Round to even
  - How does it avoid statistical bias of rounding up or down on half?

1.0100_2	truncate	1.01_2
1.0101_2	below half; round down	1.01_2
1.0110_2	interesting case; round to even	1.10_2
1.0111_2	above half; round up	1.10_2
1.1000_2	truncate	1.10_2
1.1001_2	below half; round down	1.10_2
1.1010_2	Interesting case; round to even	1.10_2
1.1011_2	above half; round up	1.11_2
1.1100_2	truncate	1.11_2

Sample Floating Point on Exams

- Consider the following 5-bit floating point representation based on the IEEE floating point format. This format does not have a sign bit – it can only represent nonnegative numbers.
  - There are k=3 exponent bits.
  - There are n=2 fraction bits.

- What is the…
  - Bias?
  - Largest denormalized number?
  - Smallest normalized number?
  - Largest finite number it can represent?
  - Smallest non-zero value it can represent?

Sample Floating Point on Exams

- Consider the following 5-bit floating point representation based on the IEEE floating point format. This format does not have a sign bit – it can only represent nonnegative numbers.
  - There are k=3 exponent bits.
  - There are n=2 fraction bits.

- What is the…
  - Bias? **011_2 = 3**
  - Largest denormalized number? **000 11_2 = 0.0011_2 = 3/16**
  - Smallest normalized number? **001 00_2 = 0.0100_2 = 1/4**
  - Largest finite number? **110 11_2 = 1110.0_2 = 14**
  - Smallest non-zero value? **000 01_2 = 0.0001_2 = 1/16**

Semi-Large Floating Point Exam Tip

- When converting from `float` to `int`, assume normalized first
  - It will be normalized most of the time
    - Easier to convert too
  - If it is denormalized, you will be able to tell quickly when doing the normalized math
    - The exponent won't make sense, for example

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
    - Don't have whitespace before a closing curly brace

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

Starting Data Lab (Newb Edition)

- Untar the lab handout
  - `> tar xvf` *`labhandout.tar`*
- Solve puzzles provided in `bits.c`
  - Only file to get turned in
- Test using provided tools
  - You should not be using Autolab to check your work!
  - Everything should be tested by the time you submit
    - `driver.pl` assigns your final grade, NOT `btest`

Datalab Other Stuff

- Operator precedence
  - There are charts. Google them.
  - Alternatively use parenthesis and never worry again.
- Hint: !, 0, and T_{min} are cool and useful
- No bonus points for having smallest op count
- Other hints in no particular order or reference:
  - Divide and conquer
  - Round to even with floating points
- Undefined behavior
  - Shifting by 32 and why you get strange results
    - My small rant to follow

Rant on Undefined Behavior

"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…

Labs, Warnings

- Permission denied
  - Are you working on a Shark machine?
  - Did you untar on a Linux machine?
  - Learning basic commands can fix this
    - > chmod +x *executable*
      - Sets executable bits for executable

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