18-600 Recitation #2
Data Lab Overview

September 5th, 2017

Announcements

- Schedule for Office Hours is still being refined. You should receive an update soon.
- Classes for recitations are now fixed and updated on the course website.
- Lecture 3 will be taught from PGH by Prof. Kesden.

Contents

- How do I do data lab?
- Integers
  - Representation
  - Biasing
  - Endianness
- Floating Point
  - Binary
  - IEEE standard
  - Denormalization
  - Rounding
- Examples

Motivation!


There are only 10 kinds of people.
Those who understand binary
and those who don't.

How do I do data lab?

- Step 1: Download lab files
  - All files must be downloaded from autolab.
  - Read write-up before starting lab (Click “View writeup” link).

How do I do data lab?

- Step 2: Work on the shark machines
  - Copy the lab onto the shark machines
    - `scp ~/Downloads/datalab-handout.tar <andrewid>@shark.ics.cs.cmu.edu:~/private/18600/` or your favorite file transfer protocol/client.
  - Untar the handout.
    - `tar xvf datalab-handout.tar`
  - If you get a permission denied error, try `chmod +x <filename>`

How do I do data lab?

- Step 3: Work on the lab
  - bits.c is the file you're looking for.
  - Mention your name and *andrewid* at the top of the file.
  - Implement all functions in the file.
  - Use "`make clean && make`" to build the file.

How do I do data lab?

- Step 4: Testing
  - 3 ways to test your solution -
    - btest
    - dlc
    - BDD checker
  - Other useful tools
    - ishow
    - fshow
  - driver.pl runs the same tests as autolab.

How do I do data lab?

- Step 5: Submission
  - Transfer and upload “bits.c” to autolab.
  - Submit via web form.
  - Unlimited submissions, but do not use autolab in place of driver.pl.

How do I do data lab?

- Tips
  - Write C like it's 1989
    - Declare variables at top of function.
    - Make sure closing brace "}" is in 1st column.
    - We won't use the dlc compiler for future labs.
  - Be careful of operator precedence
    - Do you know what order a^2*b<<--c&d+4 would evaluate in?
    - Use parentheses. They're free, they're unlimited!
  - Take advantages of special operators and values like !, 0, T_{min}.
  - Reducing ops once you're under the threshold won't get you extra points but will help you move up the scoreboard.
  - Undefined behavior
    - Like shifting by more than 31.

Undefined Behavior (Adv. topic)

- From the Intel x86 reference:
  "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."

Operations

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

Integers - Representation

- Signed
  - The most significant bit represents the sign.
    - 0 for positive and 1 for negative.
    - On x86, the 31st bit (counting from 0)
  - Range: -2^{k-1} to $(2^{k-1}-1)$
- Unsigned
  - Represents only positive integers.
  - Range: 0 to $(2^{k}-1)$

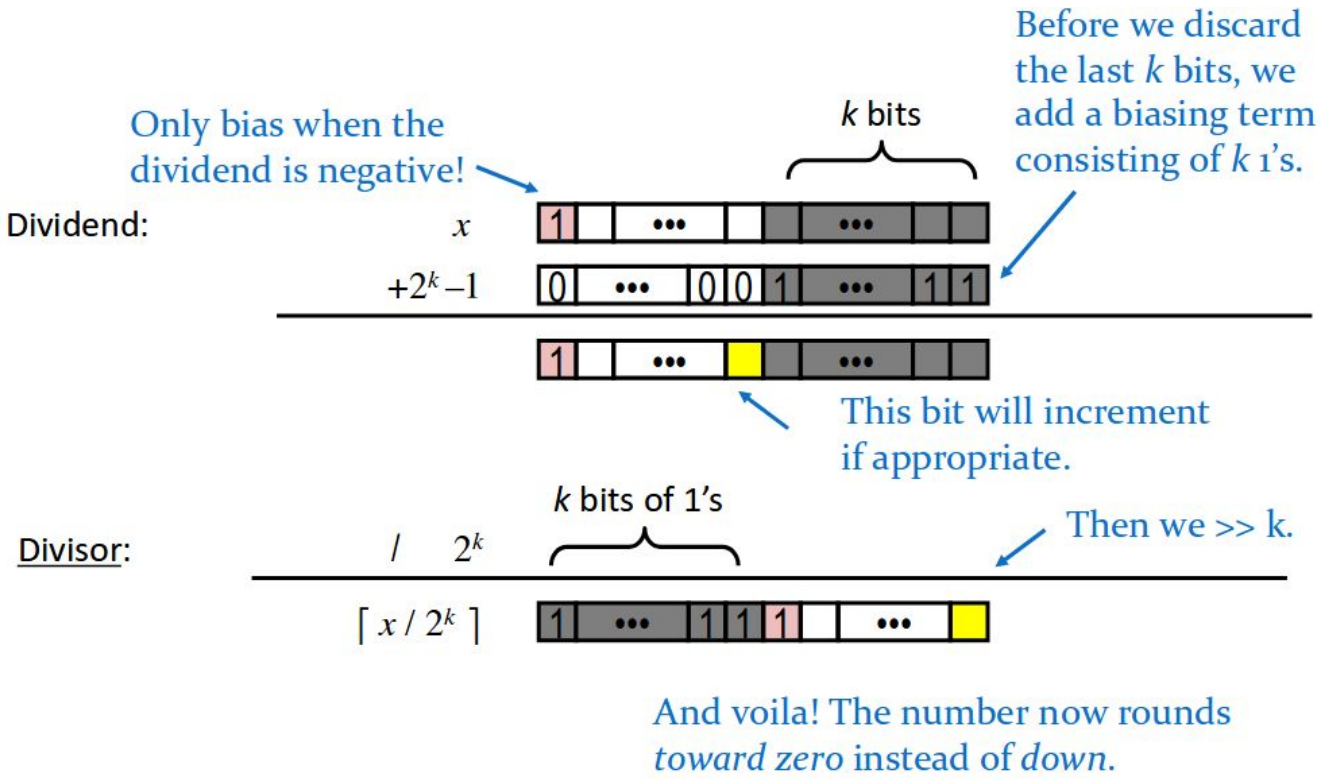
Casting

- What happens when casting between signed and unsigned ints?
- Signed ⟷ Unsigned
  - Values are reinterpreted
  - Bits remain the same
- Arithmetic between Signed and Unsigned values
  - Values are cast to unsigned first

Integers - biasing

- Multiply:
  - Left shift by k to multiply by 2^k.
- Divide:
  - Right shift by k to divide by 2^k (round towards 0).
  - Does this always hold?
  - Problem with negative numbers: rounded towards -inf.
  - Solution: Biasing!

Biasing Division

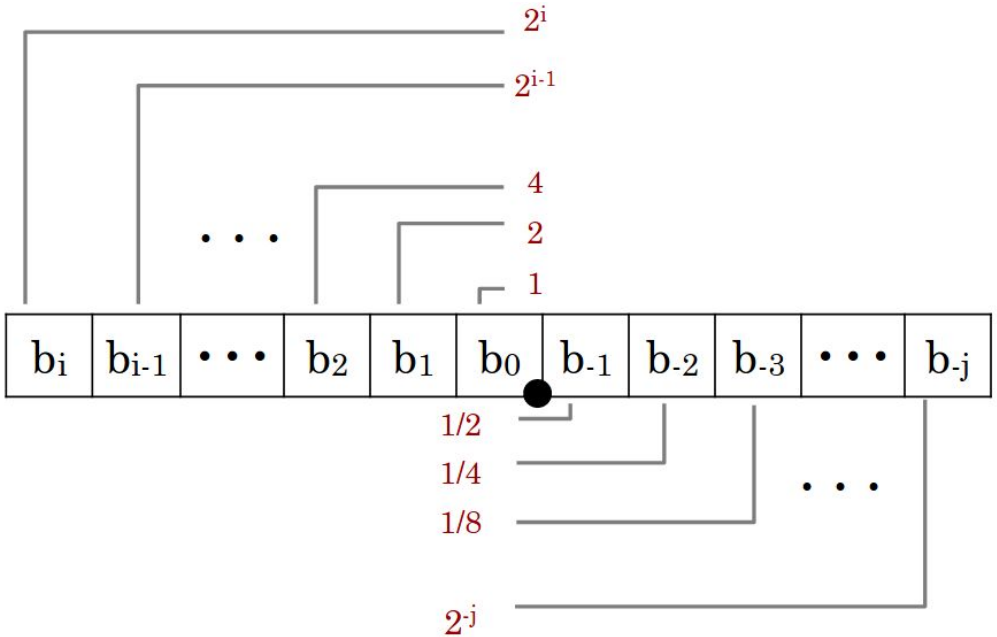


Integers - Endianness

- Describes which bit is most significant in a binary number.
- You won’t really need to work with this until Bomb lab.
- Big Endian:
  - First byte (lowest address) is the most significant.
  - This is how we typically visualize and understand binary numbers.
- Little Endian:
  - First byte (lowest address) is the least significant.
  - Intel x86 (shark/andrew linux machines) implements this.

Floating Point - Binary

- Bits to the right of the decimal point represent fractional powers of 2.



Floating Point - IEEE Standard

- Value of the floating point : $(-1)^S \times M \times 2^E$
  - S = sign, M = fractional binary number, E = exponent
- The bit representation consists of 3 parts -
  - Sign Bit (S)
  - Exponent (exp)
    - Exp = E + bias
    - Bias = $2^{k-1}-1$ (where k = number of exponent bits)
  - Fraction (f)
    - The number following the implied leading 1
- 32-bit precision floating point representation :

S 0 or 1	EXP (8 bits)	F (23 bits)

Floating point - Bit Representation (contd.)

Number	Exponent	Mantissa
0	All 0s	All 0s
NaN	All 1s	Non zero
Infinity (+ or - based on sign bit)	All 1s	All 0s
Normalised numbers	Neither all 0s, nor all 1s	-
Denormalized numbers	All 0s	Non zero

Floating Point - Denormalization

- Since floating point numbers always have a 1 preceding the mantissa, for single precision, numbers between (+/-) 2^{-127} can't be represented.
- These are represented in denormalized form.
- Format of denormalized representation :
  - Exp = 000...00
  - E = 1 - bias
  - F = the fractional representation without the implied leading 1
- Now, smallest representable single-precision floating point value is 1×2^{-149}

Floating Point - Rounding

- Round to even
  - Round to nearest value.
  - If number lies midway, then round such that the resulting value has a zero least significant bit
- Examples -
  - 1.0001 → 1.00 (lower than ½, round down)
  - 1.0011 → 1.01 (greater than ½, round up)
  - 1.0110 → 1.10 (equal to ½, round up to even)
  - 1.0010 → 1.00 (equal to ½, round down to even)

Floating Point - Bit Representation Example

Suppose we have 7-bit floating point representation with 3 exponent bits and 1 sign bit, how would we represent 3.5 ?

- $(3.5)_{10} = (11.1)_2 = (1.11)_2 \times 2^1$
- So, here S = 0, exp = 4, f = 11
  - M = 1.11, so f = 11 (with the implied leading 1)
  - Bias = $2^{3-1} - 1 = 3$
  - Exp = E + bias = 4
- So, bit Representation : (0 100 110)

Floating point - More examples

- Suppose we have 6-bit floating point representation,
  - 1 sign bit
  - k = 3 exponent bits (bias = 3)
  - n = 2 fraction bits

Value	Bit Representation	(Rounded) value
9/32	0 001 00	1/4
-3	1 100 10	-3
9	0 110 00	8
3/16	0 000 11	3/16
15/2	0 110 00	8