Bit-wise Operators, Git

CS 2130: Computer Systems and Organization 1

September 5, 2022

Announcements

- Homework 1 due Monday 9/12/2022
- TA office hours
  - **In-person**: Olsson 001, Wed-Sun, 5-7pm
  - **Online**: Discord, Wed-Sun, varies
  - Discord is now available
- My office hours
  - Tuesday, 4-5pm, Discord/Zoom
  - Wednesday, 4:30-6pm, Rice 210 (masks requested)
  - Thursday, 11am-12pm, Discord/Zoom

Floating Point in Binary

1.0101011×2^3

We must store 3 components

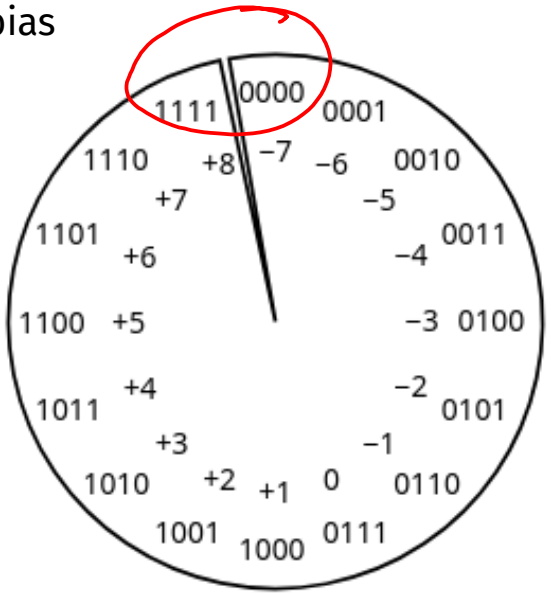
- **sign** (1-bit): 1 if negative, 0 if positive
- **fraction** or **mantissa**: (?-bits): bits after binary point
- **exponent** (?-bits): how far to move binary point

biased

Biased Integers

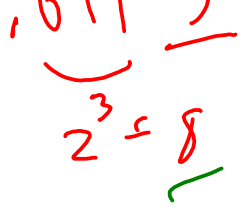
Similar to Two's Complement, but add **bias**

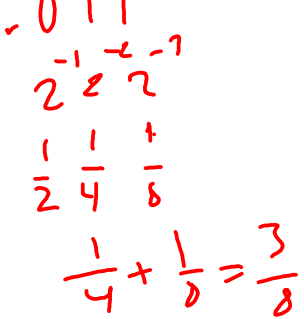
- **Two's Complement**: Define 0 as 00...0
- **Biased**: Define 0 as 0111...1
- Biased wraps from 000...0 to 111...1

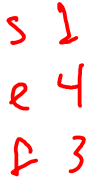


Floating Point Example

101.011_2







Floating Point Example

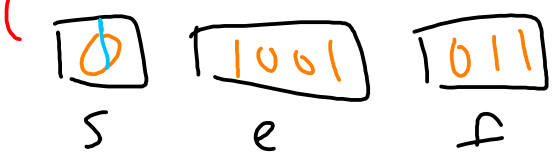
101.011_2

-1.01011×2^2

s 1
e 4
f 3

```
 1 1 10
+0 1 11
-------
 1 0 0 1
```

1.011×2^2



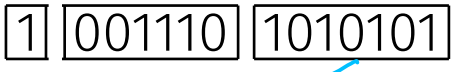
3.14159

↓

3.1416

Floating Point Example

What does the following encode?



$$-1.1010101 \times 2^{-17}$$

Floating Point Example

What does the following encode?

What about 0?

Floating Point Numbers

Four cases:

- **Normalized**: What we have seen today

$$s\ eeee\ ffff = \pm 1.ffff \times 2^{eeee - \text{bias}}$$

- **Denormalized**: Exponent bits all 0

$$s\ eeee\ ffff = \pm 0.ffff \times 2^{1 - \text{bias}}$$

- **Infinity**: Exponent bits all 1, fraction bits all 0 (i.e., $\pm\infty$)
- **Not a Number (NaN)**: Exponent bits all 1, fraction bits not all 0

Operations So Far

So far, we have discussed:

- Addition: $x + y$
  - Can get multiplication
- Subtraction: $x - y$
  - Can get division, but more difficult
- Unary minus (negative): $-x$
  - Flip the bits and add 1

Operations (on Integers)

Bit vector: fixed-length sequence of bits (ex: bits in an integer)

- Manipulated by bitwise operations

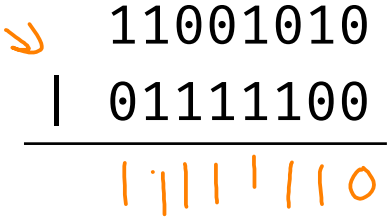
Bitwise operations: operate over the bits in a bit vector

- Bitwise not: $\sim x$ - flips all bits (unary)
- Bitwise and: $x\&y$ - set bit to 1 if x, y have 1 in same bit
- Bitwise or: $x|y$ - set bit to 1 if either x or y have 1
- Bitwise xor: $x\hat{}y$ - set bit to 1 if x, y bit differs

Example: Bitwise AND

```
  11001010
& 01111100
----------
  01001000
```

Example: Bitwise OR



Example: Bitwise XOR

```
  11001010
^ 01111100
----------
  10110110
```

Your Turn!

What is: `0x1a ^ 0x72`

Operations (on Integers)

- Logical not: $!x$
  - $!0 = 1$ and $!x = 0, \forall x \neq 0$
  - Useful in C, no booleans
  - Some languages name this one differently
- Left shift: $x << y$ - move bits to the left
  - Effectively multiply by powers of 2
- Right shift: $x >> y$ - move bits to the right
  - Effectively divide by powers of 2
  - Signed (extend sign bit) vs unsigned (extend 0)

Right Bit-shift Example

```
01011010 >> 3
```

Left Bit-shift Example

```
01011010 << 2
```

Bit-shift

Computing bit-shift effectively multiplies/divides by powers of 2

Consider decimal:

$$2130 <<_{10} 2 = 213000 = 2130 \times 100$$

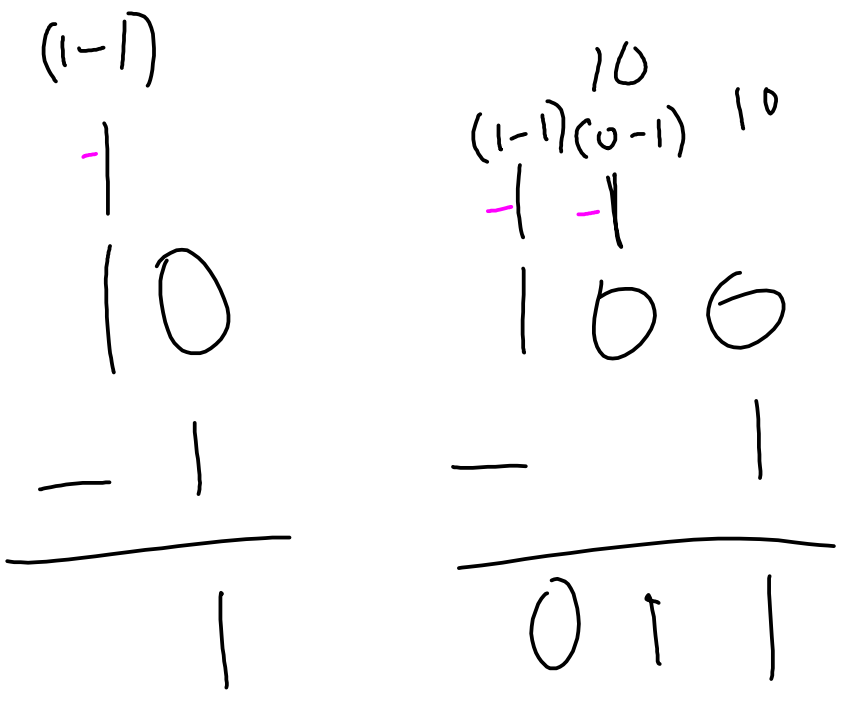
$$2130 >>_{10} 1 = 213 = 2130 / 10$$

Right Bit-shift Example

For signed integers, extend the sign bit

- Keeps negative value (if applicable)
- Approximates divide by powers of 2

```
11001010 >> 1
```


(1-1)
-1
10
-1
1
10
(1-1)(0-1) 10
-1 -1
100
-1
011

git