Bitwise Potpourri

CS429H - Spring 2011

Christian Miller

Tricky bits

- This assignment is all about knowing the intricacies of bitwise operations and the representations of numbers
- There are lots of tricks that manipulate them

Bang-Bang

- !!x will set all nonzeros to 1
- So: !!(1) = 1, !!(-378) = 1
- And: !!(0) = 0

Masking

- Using bitwise logical ops gives you control over individual bits
- Setting: 0xC0 | 0x55 = 0xD5
  - 1100 0000 | 0101 0101 = 1101 0101
- Clearing: ~0xC0 & 0x55 = 0x15
  - 0011 1111 & 0101 0101 = 0001 0101

Mask making

- Use bit shifting with |, complement with ~
- 0x55AA0000 = (0x55 << 24) | (0xAA << 16)
- 0xFFBFFDFF = ~((1 << 9) | (1 << 22))

Mask making

- Left shift will always shift in zeros
- Right shift is arithmetic, copying the top bit as it goes
- Say you have 1 or 0, and want to build 0xFFFFFFFF or 0x00000000
    - mask = val << 31 >> 31;

Faking conditionals

- Say you want to do conditional equality:
  - x = cond ? a : b;
- Evaluate both results, mask them together:
  - mask = cond << 31 >> 31;
  - x = (a & mask) | (b & ~mask);

Butterfly switch

- Say you want to toggle between two values (a and b) without using a conditional
- let c = a ^ b
- then a = b ^ c and b = a ^ c
- If you set x = a or x = b to start, then x ^= c will toggle x between a and b

Checking equality

- How do you tell if a == b without ==?
- XOR tells you whether bits are equal or not
- (a ^ b) will be zero if the two values are equal

Checking the sign

- If the top bit of an integer is set, it's negative
- You can use shifts and the XOR trick to tell if the signs of two numbers are the same

Overflow / underflow

- If you count over TMax, you'll loop around to TMin (overflow)
- Same the other direction; count below TMin, you'll loop around to TMax (underflow)
- Great way to get wrong answers
- It's impossible to over/underflow more than once during a single addition

Negating an integer

- -x = ~x + 1;
- Always works, thanks to overflow
- One special case: ~TMin + 1 = TMin
  - This is because -TMin can't be represented without an extra bit

Powers of 2

- Shift left = multiply by 2
- Shift right = divide by 2

Divide and conquer

- How do you simulate looping over bits?
- You don't, but you can sometimes exploit non-interference to fake it

Parallel add

- Say we want to add four numbers together, but we only get two adds to do it with
- As long as the numbers are small enough to fit in part of an int, we can do several adds at once
- Works only for positive numbers (negatives act like unsigneds instead)

Parallel add

```
int x = (a << 24) | (b << 16) | (c << 8) | d;
x = ((x & 0xFF00FF00) >> 8) + (x & 0x00FF00FF);
x = ((x & 0xFFFF0000) >> 16) + (x & 0x0000FFFF);
```

Parallel add

- We made it 14 ops instead of 3...
  - and we can only do 8-bit positive ints
- BUT, it was logarithmic in adds
  - we added 4 8-bit numbers with 2 adds
  - we can also do 8 4-bit numbers with 3 adds
- If you're adding a bunch of small stuff together, this is more efficient than unrolling the loop

Floating point

- Not really any tricks here
- Have a floating point reference handy
- Be sure to properly handle signs, denormal numbers, inf, and NaN

Questions

- These slides will go up on the class webpage