Bit Manipulations

CS 491 – Competitive Programming

Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Fall 2023

Objectives

- Compute binary representations of an integer
  - standard
  - one's compliment
  - two's compliment of arbitrary integers.
- Demonstrate the properties of boolean operations *and*, *or*, *not*, *xor*.
- Use shifting operations to test, set, and toggle arbitrary bits.
- Quickly determine if an integer is a power of 2.
- Quickly determine the number of set bits in an integer.
- Quickly determine the least significant set bit in an integer.

Representation of a Positive Integer

- I think you know this very well by now....
  - Each digit is a successive power of 2
  - Let's use 6 bit integers for our examples.

$$\begin{aligned} 2 &= 000010 \\ 8 &= 001000 \\ 10 &= 001010 \\ 17 &= 010001 \end{aligned}$$

One's Compliment

- If you just "flip all the bits" you get one's compliment.
- In C++, the ~ operator will do this.

```
 2 = 000010     ~2 = 111101
 8 = 001000     ~8 = 110111
10 = 001010    ~10 = 110101
17 = 010001    ~17 = 101110
```

- We don't use one's compliment for negation though:

```
0  = 000000     ~0 = 111111
```

Two's Compliment

- Take one's compliment (flip the bits) and then add one.
- In C++, regular old negation will do this.

```
 2 = 000010     ~2 = 111101     -2 = 111110
 8 = 001000     ~8 = 110111     -8 = 111000
10 = 001010    ~10 = 110101    -10 = 110110
17 = 010001    ~17 = 101110    -17 = 101111
 0 = 000000     ~0 = 111111     -0 = 000000
```

Properties of And, Or, Not

Binary And &

- Commutative and associative.
- Identity is “all ones”.

Binary Or |

- Commutative and associative.
- Identity is “all zeros.”

Not ~

- Is its own inverse. `~(~x)` = x=

Example

```
a = 011001
b = 001010
c = 100110
a & b = 001000   a | b = 011011
b & c = 000010   b | c = 101110
a & c = 000000   a | c = 111111
```

Exclusive or

- Is true if bits are different.

```
0 ^ 0 = 0
1 ^ 0 = 1
0 ^ 1 = 1
1 ^ 1 = 0
```

- Is a good way to toggle bits:
- `(a ^ b) ^ b == a`

Shifting Operations

- Use << to shift left, >> to shift right.
  - `001010 << 2 = 101000`
  - `001010 >> 2 = 000010`
- Allows easy multiplication and division by 2.
- Allows easy bit inspection and manipulations.

Check bit *i*

```
n & (1 << i)
```

Set bit *i*

```
n |= (1 << i)
```

Toggle bit *i*

```
n ^= (1 << i)
```

Some operations

Think about how you could do these operations.

- Check if a number is divisible by 2. $\mathcal{O}(1)$
- Clear lower *n* bits. $\mathcal{O}(1)$
- Clear bits above *n*. $\mathcal{O}(1)$
- Check if *n* is a power of 2. $\mathcal{O}(1)$
  - Hint: what is `x & (x-1)`?
- Count number of set bits in *n*. $\mathcal{O}(b)$ b = number of bits.
- Get least significant set bit. $\mathcal{O}(1)$
  - Hint: you need the two's compliment.

Clear bits n and up

- To clear upper *n* bits, you need to create a bitmask that sets the lower bits.

```
mask = (1 << n) - 1;
x = x & mask;
```

  - Example

```
x = 110011  -- lets clear bits 2 and up
mask = (1 << 2) -1;
     = 000100 - 1
     = 000011
x & mask = 110011
         & 000011
         --------
           000011
```

Clear bits n and down

- To clear lower *n* bits, you need to create a bitmask that sets the lower bits, then compliment.

```
mask = ~((1 << (n+1)) - 1);
x = x & mask;
```

  - Example

```
x = 110011  -- lets clear bits 2 and up
mask = ~((1 << 3) -1);
     = ~(001000 - 1)
     = ~ 000111
     =   111000
x & mask = 110011
         & 111000
         --------
           110000
```

Check if n is odd or power of two

- If x is odd, `x|1` is 1.
- Check x is power of 2, `x & (x-1)` will be zero.

```
x   =   001000
x-1 =   000111
      & ------
        000000

x   =   001010
x-1 =   001001
      & ------
        001000
```

- Use `(x && !(x & (x-1))` to exclude when x is zero.

Check number of set bits in n

Consider $n\&(n-1)\ldots$

```
n     = 101100
n-1   = 101011
      & ------
        101000
```

so...

```
num = 0;
while (n>0) {
  num++;
  n = n & (n-1);
}
```

Get least significant bit

- `n & (-n)` will do this.

```
n    = 101100
-n   = 010100   (= 010011 + 1)
     & ------
       000100
```